**Contract Number: 169-C-00-11-00102**

**Project: Sustainable Local Development Project in Serbia**

**Contractor: Chemonics Inc.**

**USAID COR: Mr. Sinisa Cadjo**

**Issuance Date: October, 2014**

**Document Title: Program Manual for ICT Academy**

**Author: ICT Cluster Academy**

**Language: Serbian**

This publication was produced for review by the United States Agency for International Development. It was prepared by Chemonics International Inc. The authors' views expressed in this publication do not necessarily reflect the views of the United States Agency for International Development or the United States Government.

TVOJA
KARIJERA
U TVOJIM
RUKAMA

# IKT Klaster Akademija

# Program priručnik

# Sadržaj


| | | |
|---|---|---|
| Osnovni predmeti | | **12** |
| Metodologije i alati za razvoj softvera | | **13** |
| Priručnik | | |
| 1. | Uvod | 15 |
| 2. | Životni ciklus procesa (*SDLC*) | 16 |
| 2.1 | Modeli životnog ciklusa procesa | 16 |
| 2.2 | Uloge u razvojnom timu | 18 |
| 3. | Modeli procesa izrade softvera | 22 |
| 3.1 | Model vodopada (*Waterfall model*) | 22 |
| 3.2 | V model | 25 |
| 3.3 | Prototipski model | 26 |
| 3.4 | Fazni razvoj: Inkrementi i iteracije | 27 |
| 3.5 | RUP model | 29 |
| 3.6 | Spiralni model | 30 |
| 4. | Specifikacija zahteva | 32 |
| 5. | Projektovanje arhitekture sistema | 40 |
| 6. | Klasične i agilne metode razvoja softvera | 48 |
| 6.1 | XP | 49 |
| 6.2 | SCRUM | 53 |
| 6.3 | KANBAN | 56 |
| 7. | Verifikacija, validacija i testiranje | 60 |
| 8. | Literatura | 67 |
| 9. | Appendix 1 | 68 |
| 9.1 | Koncepti programiranja | 68 |
| 10. | Appendix 2 | 70 |
| 10.1 | Modeli strukture | 70 |
| | | |
| Objektno-orijentisano programiranje | | **74** |
| Priručnik | | |
| 1 Uvod u objektno-orijentisano programiranje | | 76 |
| 1.1 Objektno orjentisana paradigma | | 76 |
| 1.2 Apstrakcija | | 76 |
| 1.3 Objekti | | 76 |
| 1.4 Zatvaranje (Inkapsulacija) | | 77 |
| 1.5 Klase | | 78 |
| 2 Osnovni elementi programskog jezika | | 80 |
| 2.1 Tipovi podataka | | 80 |




Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

2.2 Azbuka jezika 81
2.3 Tokeni jezika 82
2.3.1 Identifikatori 82
2.3.2 Ključne reči 82
2.3.3 Separatori 82
2.3.4 Konstante 82
2.3.5 Literali 84
2.3.6 Operatori 84
2.4 Naredbe 86
2.4.1 Definisanje promenljivih 86
2.4.2 Definisanje simboličkih konstanti 87
2.4.3 Elementarne izvršne naredbe 87
2.4.4 Strukturne naredbe 87
2.4.5 Blok naredbi 88
2.4.6 Naredbe uslovnog grananja 88
2.4.7 Programske petlje 89
3 Polja 92
3.1 Polja u programskom jeziku C++ 92
3.2 Polja u programskom jeziku C# 94
4 Klase 96
4.1 Definisanje klase 96
4.2 Funkcija main 101
4.3 Kreiranje objekata 101
4.4 Konstruktori 103
4.5 Destruktori 106
4.6 Pristup funkcijama i atributima klase 107
4.7 Funkcije za pristup atributima klase u jeziku C++ i svojstva u jeziku C# 108
4.8 Zajednički članovi klasa 110
4.9 Prijateljske funkcije i klase u programksom jeziku C++ 111
4.10 Operatorske funkcije 113
4.10.1 Operatorske funkcije u programskom jeziku C++ 113
4.10.2 Operatorske funkcije u programskom jeziku C# 118
4.10.3 Indekseri u programskom jeziku C# 120
5 Nasleđivanje klasa 122
5.1 Generalizacija 122
5.2 Nasleđivanje 122
5.3 Polimorfizam 125
5.4 Apstraktne klase 128
5.5 Interfejsi u programskom jeziku C# 129



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

6 Izuzeci 130
6.1 Koncept obrade izuzetaka 130
6.2 Sintaksa naredbi za prijavu i obradu izuzetaka 130

Specijalistički predmeti **134**

*Smer Embedded Specialist* *135*

LabView 136

Priručnik
1 Uvod 137
2 LabVIEW okruženje 138
2.1 Startup dijalog 138
2.2 Front panel i Blok dijagram prozori 140
2.2.1 Front panel prozor 140
2.2.2 Blok dijagram prozor 141
2.2.3 Meniji 142
2.2.4 Paleta sa alatima 142
2.2.5 Tools paleta 143
2.2.6 Paleta sa kontrolama 143
2.2.7 Paleta sa funkcijama 144
2.3 Help 145
2.3.1 LabVIEW Help 145
2.3.2 Context Help 145
2.3.3 Online resursi 146
3 Virtuelni instrument 146
3.1 Front panel 146
3.2 Blok dijagram 147
3.3 Programske petlje 148
3.3.1 While petlja 149
3.3.2 For petlja 150
3.3.3 Timed loop petlja 152
3.3.4 Tuneli, šift registri i povratne petlje 153
3.4 Nizovi 156
3.4.1 Autoindeksiranje 156
3.4.2 Najčešće korišćene funkcije nizova 158
3.4.3 Polimorfizam 161
3.5 Programske strukture 162



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

3.5.1 Case struktura 162
3.5.2 Flat Sequence struktura 165
3.5.3 Stacked Sequence struktura 166
3.5.4 Formula Node struktura 167

**Sistemi u realnom vremenu 168**
Priručnik
1 Uvod u mikrokontrolere 171
1.1 Definicija mikrokontrolera 171
1.2 Osnovne karakteristike MSP430 mikrokontrolera 171
1.3 Adresni prostor MSP430 173
1.4 Procesorsko jezgro MSP430 174
1.5 Sistemski reset 176
1.6 Sistem za generisanje taktova 177
1.7 Prekidi 177
1.8 Modovi niske potrošnje 178
2 Osnovne periferije MSP430 178
2.1 Digitalni ulaz/izlaz 178
2.2 Tajmeri – osnovne karakteristike 179
2.3 Watch-dog tajmer 180
2.4 A/D konverzija 180
2.5 D/A konverzija 184
2.6 Serijski portovi 185
2.6.1 UART interfejs 186
2.6.2 SPI interfejs 188
2.6.3 I2C interfejs 191
3 RTOS 194
4 Razvojna okruženja 196
4.1 IAR Workbench za MSP430 197
5 Glavni principi razvoja softvera za mikrokontrolere 201
6 Praktični primeri softvera 203
6.1 Primer korišćenja GPIO prekida 203
6.2 Primer korišćenja tajmera 204
6.3 Primer korišćenja A/D konvertora 204
6.4 Primer korišćenja UART interfejsa 204
6.5 Primer korišćenja SPI interfejsa 205
6.6 Primer korišćenja FreeRTOS 205
7 Uvod u razvoj softvera na Linuxu 205
7.1 Deset principa open-source softvera 205



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

| | | |
|---|---|---|
| 7.2 | GNU Projekat | 206 |
| 7.3 | Istorijat GNU projekta | 206 |
| 7.4 | POSIX | 207 |
| 7.5 | Razvoj softvera na Linuxu | 208 |
| 7.6 | Prevođenje programa | 209 |
| 7.7 | Makefile | 210 |
| 7.8 | Alati za baratanje fajlovima | 210 |
| 7.9 | Tekst editori za uređivanje izvornog kôda | 211 |
| 8 | Osnove administracije Linux sistema | 211 |
| 8.1 | Osobine Linuxa kao operativnog sistema | 211 |
| 8.2 | Komandni interpreter | 212 |
| 8.3 | Primeri često korišćenih shell programa | 212 |
| 8.4 | Fajl sistemi | 213 |
| 8.5 | Specijalni fajlovi | 215 |
| 8.6 | Korisnički nalozi | 216 |
| 8.7 | Procesi | 217 |
| 8.8 | Standardni ulaz i izlaz | 218 |
| 8.9 | Runleveli | 219 |
| 9 | Git, sistem za kontrolu verzija fajlova (VCS) | 220 |
| 9.1 | VCS – Version control sistemi | 220 |
| 9.2 | Git | 221 |
| 9.3 | Običan nasuprot „bare“ repozitorijumu. | 222 |
| 9.4 | Grananje projekta | 222 |
| 10 | U-Boot, univerzalni bootloader | 223 |
| 10.1 | Uloga bootloadera pri podizanju Linuxa na embedded sistemima | 223 |
| 10.2 | U-Boot | 224 |
| 11 | Arhitektura Linuxa | 225 |
| 11.1 | Tipovi jezgra (kernela) | 225 |
| 11.2 | User space i kernel space | 225 |
| 11.3 | Komponente kernela | 226 |
| 12 | Unutrašnjost Linuxa | 228 |
| 12.1 | Procesi | 228 |
| 12.2 | „Init“ proces | 229 |
| 12.3 | Multi-threading | 229 |
| 12.4 | Sistemski pozivi | 230 |
| 12.5 | Signali | 230 |
| 12.6 | Mehanizmi međuprocesne komunikacije | 231 |
| 12.7 | Upravljanje memorijom | 232 |
| 12.8 | Obrada interapta | 232 |




Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

12.9 Real-time programiranje na Linuksu 232
13 Uređaji i drajveri pod Linuksom 233
13.1 Podrška za periferijske uređaje 233
13.2 Tipovi uređaja 234
13.3 Obrada interapta 234
13.4 Komunikacija drajvera sa userspace-om 235
14 Linux na embedded platformama 236
14.1 Specifičnosti Linuxa na embedded platformama 236
14.2 Specijalne verzije C biblioteka 236
14.3 BusyBox 236
14.4 Flash memorije 237
14.5 Linux MTD 238
14.6 NFS fajl sistem 239
15 Android u embedded okruženju 239
15.1 Specifičnosti Androida u odnosu na Linux 239
15.2 Dalvik virtuelna mašina 241
15.3 ART virtuelna mašina 241
15.4 JNI, Java Native Interface 241
16 Literatura 242

**ARM 32** **244**

Priručnik

1. Namenski sistemi 246
   1.1 Zahtevi od namenskih sistema 247
   1.2 Interesantni primeri namenskih Sistema 248
   1.3 Razvoj namenskih Sistema 248
2. Specifikacija Sistema 249
3. Arhitektura sistema 250
   3.1 Izbor komponenti 251
4. Kumunikacijoni ineterfejsi – osnove 252
   4.1 Serijska komunikacija 252
   4.2 Paralelna komunikacija 252
   4.3 Poredjenje serijske i pralelene komunikacije 253
   4.4 Modovi serijske komunikacije 255
      4.4.1 Podela po smeru komunijkacije 256
5. UART (Universal Asynchronous Receiver Transmitter) 257
6. SPI (Serial Peripheral Interface) 258
7. $I^2C$ (Inter-Integrated Circuit) 262
8. DDR SDRAM memorije 266
   8.1 Trendovi u razvoju DRAM memorija 266
   8.2 DRAM 267




Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

8.3 SDRAM 268
8.4 DDR SDRAM 269
8.5. DDR2 SDRAM 269
8.6 DDR3 SDRAM 270
8.7 Primer uptrebe DDR3 SDRAM memorije 271
9. DVI Interfejs 274
10. HDMI 275
11. Touch Screen tehnologije 279
11.1 Rezistivni Touch Screen 279
11.2 Kapacitivni Touch Screen 281
12. JTAG 285
13. SD Card 288
14. Osnove high-speed dizajna 297
15. Transmisione linije 301
16. Integritet signala 305
17. Dizajn štampanih ploča 311
18. Literatura 316
19. PCI Express Magistrala podataka 317
19.1 PCI Express Arhitektura 317
19.1.1 PCI Magistrala 319
19.1.2 PCI Express Magistrala 324
19.2 PCIe Transakcije 328
19.2.1 Non-posted read transakcija 330
19.2.2 Locked non – posted read transakcija 330
19.2.3 Non – posted write transakcija 330
19.2.4 Posted write transakcija 330
19.2.5 Message transakcija 330
19.3 Slojevi na PCI Express magistrali i paketski prenos 330
19.3.1 Transaction Layer (TL) Packet 331
19.3.2 Data Link Layer (DLL) Packet 331
19.3.3 Physical Layer (PhL) Packet 332
19.3.4 Funkcije slojeva 332
19.3.5 Polja TLP paketa 332
19.3.6 Kontrola protoka 332
19.3.7 Kvalitet servisa (Qos) 332
19.3.8 Arbitracija u PCI Express svičevima 333
19.3.9 Kontrola grešaka 334
19.3.10 Fizički sloj (Physical Layer) 334
19.3.11 Trening linka na fizičkom sloju 334
19.4 Rutiranje na PCI Express magistrali 335
19.4.1 Adresno rutiranje TLP paketa 338
19.4.2 ID rutiranje TLP paketa 340
19.4.3 Implicitno rutiranje TLP paketa 344
19.5 ACK/NAK Protokol 344
19.5.1 Predajna strana DLL sloja 344




Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

19.5.2 Prijemna strana DLL sloja 346
19.5.3 Principi rada ACK/NAK protokola - Predajnik 347
19.5.4 Principi rada ACK/NAK protokola – Prijemnik 350
19.5.5 Prioriteti slanja paketa 351
19.5.6 Switch Cut-Throug mod 352
19.6 Kvalitet servisa i arbitracija 353
19.6.1 Podrška za izohrone transakcije 353
19.6.2 Virtuelni kanali (VC) i klase saobraćaja (TC) 354
19.6.3 Vrste Arbitracije 356
19.6.4 Arbitracija virtuelnog kanala 356
19.6.5 Arbitracija na portu 360
19.7 PCIe Konfiguracija 362
19.7.1 Enumeracija PCI Express uređaja 364
20. Automatsko testiranje i verifikacija dizajna 368
20.1 Verifikacija dizajna 369
20.1.1 Prilagođenje dizajna – Test tačke 369
20.1.2 Prilagođenje dizajna – Opcione komponente 369
20.1.3 Prilagođenje dizajna – Opcione komponente 369
20.1.4 Proces verifikacije – Ulazna kontrola 369
20.1.5 Proces verifikacije – Asembliranje 370
20.1.6 Proces verifikacije – Omska provera 370
20.1.7 Proces verifikacije – Provera blok napajanja 370
20.1.8 Proces verifikacije – Generatori Takta, PLL, Reset 370
20.1.9 Proces verifikacije – Funkcionalna provera 371
20.1.10 Proces verifikacije – EMC test, Temperaturni opseg 371
20.2 Automatsko testiranje u proizvodnji 371
20.2.1 Tipovi neispravnosti 371
20.2.2 Optička kontrola 371
20.2.3 In Circuit Testing (ICT) 372
20.2.4 Boundary scan 373
20.2.5 Virtuelna instrumentacija 374
21. Literatura 375

*Smer Software Developer* *376*

**Web programiranje 1** **377**

Priručnik

1. Uvod 379
2. Globalne računarske mreže 380
3. Servisi na Internetu 380
4. TCP/IP protokoli 381

Vežba br.1 - HTML osnove 385

Vežba br.2 – Slike, liste i frejmovi 416




Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Vežba br. 3 - HTML tabele 433
Vežba br.4 - HTML obrasci 455
Vežba br. 5 – HTML5 477
Vežba br. 6 – CSS 491
Vežba br. 7 – Java Scrip 510
Vežba br. 8 – Java Script (II deo) 542
5. Literatura 546

**Web programiranje 2** **563**
Priručnik
1. Uvod 566
2. Osnove asp .net tehnologije 572
3. MVC kontroleri 581
4. MVC views 589
5. MVC modeli 607
6. Validacija podataka 612
7. Ajax & jQery 617
8. Bezbednost web aplikacija i MVC 4 626
9. Testiranje i instalacija mvc 4 aplikacije 633
10. Literatura 637
Prilog 1: Vežba br. 1 638
Prilog 2: Vežba br. 2 658
Prilog 3: Vežba br. 3 685

**Razvoj mobilnih aplikacija** **700**
Priručnik
1. Uvod u razvoj Android mobilnih aplikacija 703
1.1 Mobilne tehnologije i standardi 703
1.2 Android 704
1.3 Životni ciklus Android aplikacije 709
1.4 Praktična vežba 1 - Uvod u razvoj Android mobilnih aplikacija 711
2 Android grafički korisnički interfejs i GUI layout-i 717
2.1 Mobilno korisničko iskustvo (Mobile user experience - UX) 717
2.2 Android korisnički interfejs 718
2.3 Praktična vežba 2 - Android grafički korisnički interfejs i layout-i 727
3. Interakcija sa korisnikom - Meniji, diajalozi, intent-i 736
3.1 Meni 736
3.2 Dialog box 741
3.3 Intent 744



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

3.4 Praktična vežba 3 - Napredniji dizajn mobilnih grafičkih korisničkih interfejsa i widget-i 748
4. Mobilne baze podataka (SQLite) i višenitnost mobilnih aplikacija 761
4.1 Mobilne baze podataka 761
4.2 SQLite 762
4.3 Višenitnost (Multithreading) 765
4.4 Android Handler 765
4.5 AsyncTask 766
4.6 Praktična vežba 4 - Perzistencija podataka i SQLite baza 769
5. Mobilno pozicioniranje, Google mape i lokaciono-zasnovani servisi (LBS) 778
5.1 Lokaciono zasnovani servisi 778
5.2 Mobilno pozicioniranje 778
5.3. GPS 779
5.4. Pozicioniranje zasnovano na mobilnoj mreži 780
5.5 Pozicioniranje u zatvorenom prostoru 781
5.6 Android lociranje 782
5.7 Android i mapiranje 784
5.8 Praktična vežba 5 - Rad sa mapama - Google Maps API v2 790
6. Mobilna komunikacija i servisi 805
6.1 Servisi 807
6.2 Praktična vežba 6 - Mrežna komunikacija korišćenjem HTTP protokola 808
7. Broadcast receiver-i i notifikacije 819
7.1 BroadcastReceiver 819
7.2 Notifikacija 819
7.3 Senzori 821
7.4 Praktična vežba 7 - Broadcast receiver-i i proximity Alert-ovi 825



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# Osnovni predmeti



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# Metodologije i alati za razvoj softvera

**Predavač: Miloš Kosanović**

**Ciljevi predmeta:**
Kurs treba da pokrije osnovne koncepte i metodologije u razvoju softvera, životni ciklus softvera, procese tokom razvoja softvera i alate za upravljanje softverskim projektima.

**Ishod predmeta:**
Polaznici će razumeti sve procese i aktivnosti u toku razvoja softvera i moći da učestvuju u bilo kojoj fazi razvoja softvera.

**Teorijska nastava:**

1. Pregled metodologija za razvoj softvera. Životni ciklus softvera.
2. Proces razvoja softvera. Planiranje, implementacija, testiranje i dokumentovanje, Isporuka, i održavanje.
3. Agilne metodologije za razvoj softvera (SCRUM).
4. Uprvljanje projektima (alati i pristupi)

**Praktične vežbe:**
**20% do 50%** radnih sati. Modeli za razvoj softvera. Analiza i specifikacija zahteva, dizajniranje sistema, testiranje softvera, osiguranje kvaliteta, održavanje softvera, upravljanje softverskim projektima, pregled alata za upravljanje projektima, razvoj i praćenje zadataka i bagova.

**Sadržaj kursa:** Predavač će upotpuniti kurs svojim nastavnim planom.

**Trajanje kursa:** Kurs će trajati 30 školskih časova (24 za predavanja i 6 za polaganje ispita). Predavanja će biti držana u blokovima u toku jednog meseca (4 časa petkom i 4 časa subotom)

**Kurikulum**

Prve nedelje student će čuti generalni uvod u tematiku kursa, kao i o životnom ciklusu razvoja softvera. Sadržaj i tempo predavanja će zavisiti od prethodnog znanja i iskustva kandidata. Preostale 3 nedelje će pratiti pravilo 4 časa predavanja, 2 časa praktičnih zadataka i 2 časa za testiranje). Cilj kursa je da isprati razvoj jednog relnog softverskog projekta od njegovog začeća do isporuke finalnog softvera. Od studenata će se zahtevati da kreiraju projektni tim, analiziraju zahteve, specificiraju zahteve u obliku korisničkih priča, kreiraju taskove u nekom alatu za upravljanje projektima, opišu procdure za testiranje i kontrolu kvaliteta, kao i da objasne na koji način njihov projekat rešava neki realni problem iz života.

**Ocenjivanje:**
Studenti će biti ocenjivani na osnovu sledeća 3 testa:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

1. ABCD test - koji će se polagati nakon 2 nedelje predavanja. Maksimani broj poena je 15. Da bi položili student moraju da imaju vise od 50% tačnih odgovora.
2. Dokument o korisničkim zahtevima – Studenti će kreirati dokument za demo projekat koji će im biti zadat. Dokument će biti ocenjen ocenama od 1 do 3. Da bi položili test moraju da dobiju ocene 2 ili 3.
3. Finalni ABCD test - koji će se polagati nakon 2 nedelje predavnaja. Maksimani broj poena je 20. Da bi položili student moraju da imaju vise od 50% tačnih odgovora.

Student će položiti kurs ako položi sva 3 testa

**Testovi:**

Pitanja za test i teme projektnih zadataka neće biti objavljivane zbog poverljivosti.

# 1. Uvod

Kad se govori o razvoju softvera, programiranje je svakako njegov veoma bitan deo, ali ne i jedini. Ako bismo pokušali da damo što potpuniju definiciju razvoja softvera, rekli bismo da je to proces preuzimanja skupa zahteva od korisnika, njihovog analiziranja, osmišljavanja i dizajniranja rešenja, i napokon, implementiranja tog rešenja na računaru. Sav taj proces nije nimalo lak ni jednostavan i za njegovo uspešno sprovođenje potrebne su veštine koje se najbolje „bruse“ praktičnim radom. Uprkos tome što je razvoj softvera samo deo softverskog inženjerstva, on predstavlja srce svakog softverskog projekta. Na kraju krajeva, ono što treba isporučiti korisniku je kod koji radi, a sve to poštujući vremenske rokove, okvire budžeta, i pritom se stalno suočavajući sa promenama zahteva. Taj kod obično kreira tim programera, pa ćemo zato osmotriti jedan softverski projekt od spolja kako bismo utvrdili šta je potrebno tom timu da bi jedan projekat bio uspešan.

Za uspešan razvoj softvera potrebno je sledeće:

- *Mali, dobro povezan tim.* Mali timovi imaju manje problema u komunikaciji od većih i lakše se upoznaju međusobno, što su sve bitne stavke za uspeh projekta.
- *Dobra i konstantna komunikacija između članova tima.*
- *Dobra komunikacija između tima i korisnika.* Brza povratna informacija od korisnika u vezi sa novim verzijama proizvoda je esencijalna za kontrolu izvršenja zahteva. Neke agilne metodologije zahtevaju da korisnik bude de facto deo razvojnog tima na dnevnoj bazi.
- *Proces koji svi poštuju.* Svaki projekat bez obzira na veličinu i brojnost u timu teži da bude što je moguće planski i da sledi procese u skladu s pravilima i dokumentacijom.
- *Mogućnost fleksibilnosti u okviru procesa.* Kako nijedan projekat na kraju ne izgleda onako kako je zamišljen na samom početku, potrebno je znati kako odgovori promenama zahteva, kako identifikovati rizike i isplanirati njihovo umanjenje, itd.
- *Plan kog se svi drže.* Ovaj plan obuhvata sve što je potrebno uraditi za implementaciju projekta.
- *Sinhronizacija članova tima.* Potrebno je da se zna na čemu u kom trenutku radi koji član tima, kako bi se izbegao prazan hod i kako bi se mogao proceniti tok celokupnog projekta.
- *Hrabrost da se prizna kad projekat kasni.* Uprkos početnom optimizmu, realnost je da je ponekad teško ispoštovati prvobitni plan rada. Zbog toga menadžer mora tačno da zna pravo stanje stvari za obostrani benefit.
- *Dobar alat i dobra praksa.* Iako je svaki projekat drugačiji, tri najbitnija faktora pri izboru alata su tip aplikacije, ciljna platforma i razvojna platforma. Kad su ta tri faktora poznata, može se birati alat kojim će se maksimizovati produktivnost.
- *Prihvatanje toga da neće biti poznato sve na početku projekta*. Tokom izrade projekta zahtevi će se modifikovati, pojavljivaće se novi (jednako bitni kao oni početni), tako da je upravljanje promenljivim zahtevima neophodna veština koju bi članovi tima valjalo da imaju.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 2. Životni ciklus procesa (*SDLC*)

Kad govorimo o razvoju softvera, jedan od dugoročnih ciljeva je pronaći ponovljivi, predvidivi niz procesa kojim će se produktivnost povećati a kvalitet poboljšati. Bez efektivnog upravljanja projektom, softverski proizvodi će kasniti, prekoračiti budžet ili prosto neće ispuniti očekivanja u pogledu funkcionalnosti, cene, itd. Organizacije koje se bave razvojem softvera implementiraju tzv. procesne metodologije kako bi sam proces razvoja bio olakšan. Izbor metodologije može biti na nivou organizacije, a može i klijent zahtevati da se radi po konkretnoj metodologiji.

Metodologija predstavlja *skup metoda, procesa i praksi koje se ponavljaju do isporučenja projekta*. Ona specificira šta je potrebno uraditi kako bi se upravljalo projektom od početka do kraja, opisujući detaljno svaki korak u životnom ciklusu projekta tako da je u svakom trenutku poznato koji zadatak treba uraditi, kad i kako.

Iako se metodologije ne zasnivaju na određenom alatu, u današnjem svetu koji se oslanja na softver, metodologija i softverski alat za upravljanje projektima neke organizacije tesno su povezani. Kako projektna metodologija definiše jasne procese za upravljanje projektom, potrebno ju je prilagoditi tako da odgovara okruženju. Na taj način ona će jasno ukazati razvojnom timu šta je i kojim redom potrebno uraditi da bi se projekat završio i uspešno isporučio.

Korišćenjem metodologije moguće je:

- Kreirati mapu izrade projekta
- Nadzirati vreme, troškove i kvalitet
- Kontrolisati promene i opseg
- Minimizovati rizike
- Upravljati resursima

Naravno, potrebno je koristiti metodologiju koja je najpodesnika konkretnom projektu. Za manje projekte, biće potrebno primeniti samo „lake“ procese, dok će kod velikih projekata biti neophodni „teški“ procesi kako bi se imao detaljan uvid i kontrola nad svakim elementom projekta. Ali ako se pak svakim projektom upravlja na isti način, moći će da se odredi **efikasnost** takvog pristupa, što će doprineti i lakšem radu i manjem stresu prilikom rada. Takođe, razvojnom timu će biti jasno predočeno šta se od njih očekuje, čime će se i šanse za uspeh povećati.

## 2.1 Modeli životnog ciklusa procesa

Prilikom pružanja usluga ili izrade proizvoda, uvek je potrebno slediti niz koraka da bismo izvršili skup zadataka, obično u istom redosledu. Ovakav uređeni skup zadataka možemo smatrati **procesom:** nizom koraka koji obuhvataju aktivnosti, ograničenja i resurse, a rezultiraju željenim ostvarenjem. Softverski razvojni proces se naziva životni ciklus softvera jer opisuje „život“ softverskog proizvoda od formulisanja, preko implementacije do isporuke, upotrebe, operativnog korišćenja i održavanja.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

U svakom slučaju, nezavisno od veličine programa ili broja ljudi koji rade na projektu, svaki program prolazi kroz identične korake, i to su:

1. Analiza i definisanje zahteva
2. Projektovanje arhitekture sistema
3. Projektovanje programa
4. Implementacija programa
5. Testiranje (koje se sastoji od testiranja modula, integracionog testiranja i testiranja sistema)
6. Isporuka sistema
7. Održavanje/evolucija softvera
8. Povlačenje iz upotrebe

Neki program može kombinovati dva ili više koraka u jedan, ali svi programi prolaze kroz sve korake. Iako svaki program ima životni ciklus, ima puno različitih varijanti procesa koji obuhvataju ove korake. Pa ipak, svaki model životnog ciklusa predstavlja varijaciju dva fundamentalna tipa. Po prvom tipu, projektni tim će završiti životni ciklus (ili bar korake od 2 do 7), pre nego što počnu na izradi sledeće verzije proizvoda. S druge strane, po drugom tipu, projektni tim će delimično završiti životni ciklus (najčešće korake od 3 do 5), a zatim proći kroz svaki korak nekoliko puta pre nego što pređe na sledeći korak tj. isporuku.

U idealnoj situaciji, aktivnosti se izvode vremenski jedna po jedna. Međutim, u praksi, mnogi koraci se ponavljaju. Na primer, prilikom pregledanja projekta sistema zajedno s klijentom može da se otkrije da neki zahtevi nisu još dokumentovani, pa se zajednički radi na njihovom dodavanju i mogućoj izmeni projekta sistema. Slično tome, možda će se prilikom testiranja otkriti da se uređaj ne ponaša kako je opisano u dokumentaciji. Stoga će možda biti potrebno preprojektovanje koda, ponovno razmatranje projekta sistema, ili čak novi razgovor s kupcem u vezi sa zadovoljavanjem konkretnog zahteva. Zbog svega toga, proces razvoja softvera se definiše kao svaki opis razvoja softvera koji sadrži neku od osam prethodno nabrojanih aktivnosti, organizovanih tako da zajedno proizvode proveren kod.

Razvoj softvera uključuja korisnike, klijente i razvojni tim. Ako se od nas traži da razvijemo softverski sistem za klijenta, prvi korak je sastajanje s njim radi određivanja zahteva koji opisuju sistem, jer bez poznavanja granica, entiteta i aktivnosti, nemoguće je opisati softver, kao i to kako će on biti u interakciji s okruženjem. Nakon definisanja zahteva pravi se projekat sistema radi njihovog zadovoljavanja. On se definiše iz perspektive klijenta, tj. klijentu treba predstaviti snimke ekrana koji će se koristiti, izveštaje koji će se generisati i sve drugo što će opisati interakciju u kojoj će biti korisnici s kompletiranim sistemom. Drugim rečima, akcenat je samo na izgledu i funkcionalnosti.

Tek pošto klijent odobri projekat, tj. izgled i funkcionalnost budu određeni, počinje razmatranje kodiranja. Osnovno je da se opis softverskog projekta kao sistema dobro definiše. Projekat sistema uključuje kompletan opis funkcija i interakcija. Kad se delovi programa napišu, pre nego što se povežu



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

u celinu oni se testiraju kao individualni delovi koda – ta faza testiranja se naziva testiranje modula. Kad se uverimo da ti delovi rade kako treba, povezuju se u celinu, a zatim se proverava da li rade ispravno nakon povezivanja s drugim delovima – ova faza se naziva integraciono testiranje. Finalna faza testiranja – testiranje sistema – podrazumeva testiranje celokupnog sistema radi potvrde da su pravilno implementirane funkcije i interakcije koje su zadate na početku. Razvojni tim, klijent i korisnici proveravaju da li sistem služi svojoj nameni.

Na kraju se finalni proizvod isporučuje. Tokom upotrebe, otkrivaju se neslaganja i problemi, pa ako se radi o sistemu „ključ u ruke“, klijent preuzima odgovornost za sistem. Ako se pak ne radi o sistemu tima „ključ u ruke“, onda razvojni tim ili neka druga organizacija obezbeđuju održavanje.

Upravljanje softverskim projektima u današnje vreme može da se podvede pod dva tipa: **tradicionalno planski vođene modele**, i novije **agilne razvojne modele**. Kod planski vođenih modela, proces teži da bude rigidniji u pogledu koraka i vremena objavljivanja. Ovi modeli jasnije definišu faze i iziskuju više zahteva koje treba ispuniti pre prelaska na sledeću fazu. Pogodni su za ugovore sa državnim strukturama kad su sve karakteristike koje softver treba da ima dobro definisane. Agilni modeli pak kreću od pretpostavke da učestala objavljivanja doprinose robusnosti produkta. Faze u njima se međusobno stapaju mnogo češće nego kod planski vođenih modela, i teže da sadrže što manje potrebne dokumentacije o proizvodima rada. Osnovna ideja je da je kod ono što se proizvodi pa napori pri izradi dokumentacije treba da se usmere na to.

U narednim poglavljima ćemo baciti pogled na nekoliko modela životnog ciklusa, kako planski vođenih tako i agilnih, i uporediti ih. U suštini, najbolji proces za razvoj softvera ne postoji. Svaki projekat mora da odluči koji model najbolje odgovara određenoj aplikaciji i da tu odluku zasniva na veličini projekta, iskustvu razvojnog tima i zadatom roku izvršenja.

### 2.2 Uloge u razvojnom timu

Kao što smo već utvrdili, klijent, korisnik i razvojni tim igraju glavne uloge u definisanju i stvaranju novog proizvoda. Razvojni tim se sastoji od softverskih inženjera, ali svaki inženjer može da se specijalizuje za poseban aspekt razvojnog procesa. Pogledaćemo sada detaljnije uloge članova razvojnog tima.

Prvi korak u svakom razvojnom procesu je utvrđivanje šta korisnik želi i dokumentovanje tih zahteva. Kao što smo videli, analiza je postupak razbijanja stvari na sastavne delove, kako bismo ih bolje shvatili. Stoga, razvojni tim uključuje jednog ili više **analitičara zahteva** u rad sa klijentom, da bi se ono što klijent želi razložilo u pojedinačne zahteve.

Kada zahtevi postanu poznati i dokumentovani, analitičar radi sa **projektantima** ili arhitektama na generisanju opisa funkcija sistema. Dalje, projektanti rade sa **programerima** na predstavljanju sistema tako da programeri mogu da pišu kod koji implementira ono što je formulisano u zahtevima. Ovde mo\emo raylikovati nekoliko vrsta programera: *backend* programeri, *frontend* programeri, programeri mobilnih aplikacija, programeri baza podataka itd.

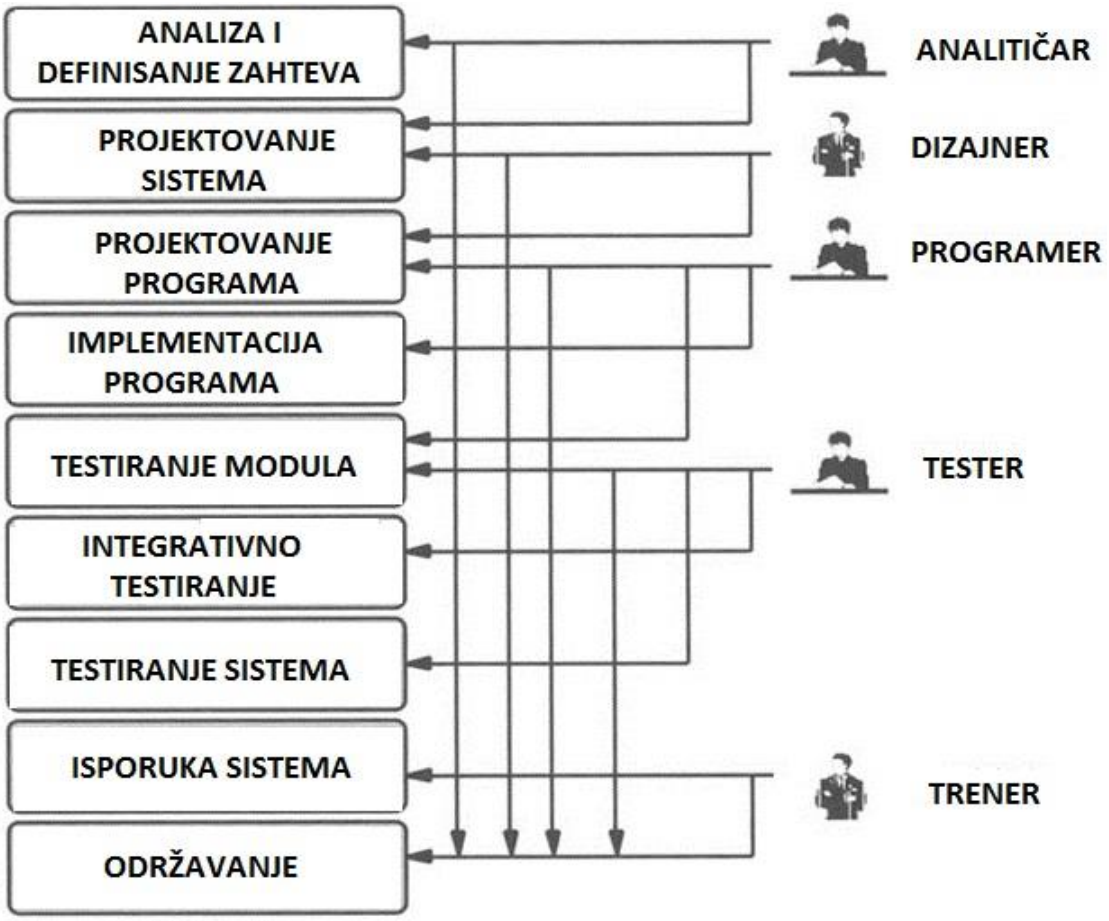
Nakon generisanja programskog koda, on mora da se testira. Prvo testiranje često obavljaju sami programeri. Ponekad se koriste drugi **testeri** kao ispomoć u otkrivanju grešaka koje previde sami programeri. Kada se jedinice koda integrišu u funkcionaine grupe, timovi za testiranje rade zajedno sa implementacionim timom na verifikovanju ispravnosti rada u skladu sa specifikacijom.

Kada je razvojni tim zadovoljan funkcionalnošću i kvalitetom sistema, pažnja se okreće **klijentu**. Tim za testiranje i klijent rade zajedno na verifikovanju celokupnog sistema, poređenjem stvarnog funkcionisanja sistema sa specifikacijom zahteva. Nakon toga, **instruktori** obučavaju korisnike za operativno koriscenje sistema.

Kao što je već spomenuto u prethodnom poglavlju, u slučaju mnogih softverskih sistema, tehnički prijem od strane klijenta ne predstavlja nužno i kraj razvoja. Ako se otkriju greške posle prijema sistema, njih ispravlja **tim za održavanje**. Pored toga, potrebe klijenta se mogu menjati tokom vremena, pri čemu na sistemu moraju da se urade odgovarajuće izmene. Stoga, u održavanju mogu učestvovati analitičari koji određuju koje su nove ili izmenjene potrebe, projektanti koji određuju šta u projektu sistema treba izmeniti, programeri za implementiranje izmena, inženjeri za testiranje radi provere da li izmenjeni sistem još uvek ispravno radi, kao i instruktori kojl objasnjavaju korisnicima kako se izmena odražava na upotrebu sistema. Slika 1 ilustruje u kojim koracima razvoja učestvuju koji pripadnici razvojnog tima.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Slika 2.2.1. Uloge razvojnog tima*

Primenimo sada ovu podelu uloga na školske projekte. Kao razvojni tim za jedan školski projekat, studenti često rade samostalno ili u malim grupama. Dokumentacija koju zahteva predavač je minimalna i od studenata se obično ne traži da pišu korisnička uputstva ili dokumente za obuku. Pored toga, zadaci su prilično nepromenljivi jer se zahtevi ne menjaju tokom životnog veka projekta. Na kraju, sistem koji je napravio student verovatno će biti obrisan na kraju godine, jer je njegova uloga bila da demonstrira problem koji se javlja kod stvarnog klijenta. Stoga, veličina programa, kompleksnost sistema, potreba za dokumentacijom i potreba za održavanjem praktično ne postoje u školskim projektima.

Međutim, u slučaju stvarnog korisnika, dimenzije sistema i kompleksnost mogu biti veliki, kao i potreba za dokumentacijom i održavanjem. Kod projekta koji ima više hiljada linija koda i mnogo interakcije između članova razvojnog tima, prilično je teško kontrolisati različite aspekte projekta. Više različitih osoba može biti uključeno u sistem na početku i tokom razvoja, radi podrške svim učesnicima u razvojnom timu.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**Bibliotekari** pripremaju i skladište dokumentaciju koja se koristi tokom životnog ciklusa sistema, uključujući specifikacije zahteva, projektne opise, programsku dokumentaciju, uputstva za obuku, podatke sa testiranja, rokove, itd. Sa bibliotekarima rade članovi **tima za upravljanje konfiguracijom.** Zadaci upravljanja konfiguracijom su održavanje korespondencije između zahteva, projekta, implementacije i testova. Unakrsne reference kazuju razvojnom timu koji program treba da se modifikuje ako se vrši izmena u zahtevima, kao i na koje delove programa se odražavaju neke predložene izmene. Osoblje koje upravija konfiguracijama takođe koordiniše različite verzije sistema koji treba sagraditi i podržavati. Na primer, softverski sistem može biti implementiran na različitim platformama ili se isporučivati u nizu verzija. Upravljanje konfiguracijom obezbeđuje konzistentnu funkcionalnost pri prelasku s jedne platforme na drugu, kao i očuvanje stepena funkcionalnosti s novom verzijom.

Razvojne uloge mogu da podrazumevaju jednu osobu ili više njih. Kod malih projekata, dva ili tri čoveka mogu da podele sve uloge. Međutim, kod većih projekata, razvojni tim se često deli na jasno odvojene grupe zasnovane na njihovim funkcijama u razvoju. Ponekad, oni koji održavaju sistem nisu isti oni koji su inicijalno projektovali i napisali sistem. Kod velikog razvojnog projekta, klijent može čak da unajmi jednu kompaniju da uradi inicijaini razvoj, a drugu da vrši održavanje.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 3. Modeli procesa izrade softvera

U literaturi iz domena softverskog inženjerstva opisani su mnogi modeli procesa. Neki predstavljaju recepte za sprovođenje procesa razvoja softvera, dok drugi pak predstavljaju opis načina na koji se razvoj sofvera stvarno odvija. Teoretski gledano, ova dva modela bi trebalo da budu ista ili slična, no u praksi, oni to nisu. Izrada modela procesa i razmatranje njegovih potprocesa pomažu projektnom timu da shvati jaz između ooga što bi trebalo da bude i onoga što stvarno jeste.

Naravno, postoje i drugi razlozi za modelovanje procesa kao što su:

- Kada grupa opiše proces projektovanja koji primenjuje, opis postaje zajedničko shvatanje aktivnosti, resursa i ograničenja uključenih u razvoj softvera.
- Modelovanje procesa pomaže razvojnom timu da pronađe nedoslednosti, viškove i izostavljene elemente u samom procesu ili njegovim delovirna. Uočavanjem i otklanjanjem tih nedostataka, proces postaje efikasniji i usmereniji na gradnju finalnog proizvoda.
- Model treba da odražava ciljeve razvoja, kao što je izrada softvera visokog nivoa kvaliteta, otkrivanje grešaka u ranim fazama razvoja i ostajanje u okvirima budžeta i formulisanih ograničenja vezanih za rokove realizacije. Nakon izrade modela, projektni tim ocenjuje aktivnosti s aspekta njihove usklađenosti s postavljenim clijevima. Na primer, tim može da uključi pregled zahteva da bi se uočili i eliminisali problemi, koji su posledica zahteva, pre početka faze izrade dizajna.
- Svaki proces treba da bude „skrojen" prema posebnoj situaciji u kojoj će se primenjivati. Modelovanje procesa pomaže da projektni tim utvrdi mesta na kojima su ta prilagođavanja potrebna.

Svaki model procesa razvoja softvera kao ulaz koristi specifikaciju zahteva, a kao izlaz isporučeni proizvod. Tokom niza godina, predloženi su mnogi takvi modeli. Svaki model ima svoje prednosti i mane, i na razvojnom timu je da adaptira onaj najpodesniji konkretnom projektu. Ponekad će recimo kombinacija više modela najviše odgovarati. U nastavku ćemo razmotriti nekoliko najpopularnijih da bismo bolje razumeli njihove sličnosti i razlike.

## 3.1 Model vodopada (*Waterfall model*)

Model vodopada prikazuje proces gde programeri treba da slede sledeće faze:

1) Analiza zahteva
2) Projektovanje arhitekture sistema
3) Projektovanje programa
4) Kodiranje
5) Testiranje delova i celine
6) Testiranje sistema
7) Završno testiranje
8) Operativni rad i održavanje

Glavna karakteristika ovog modela je što jedna faza razvoja treba da se **u potpunosti okonča pre nego što započne sledeća.** Zbog toga tek pošto su svi zahtevi klijenta iskazani, analizirani s aspekta potpunosti i doslednosti, dokumentovani u dokumentu koji sadrži tehničke zahteve, razvojni tim



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

može da pređe na sledeću fazu nastavi s aktivnostima u vezi sa dizajnom sistema. Ovaj model predstavlja veoma visok nivo apstrakcije razvojnog procesa, pri čemu se projektnom timu sugeriše redosled očekivanih događaja.

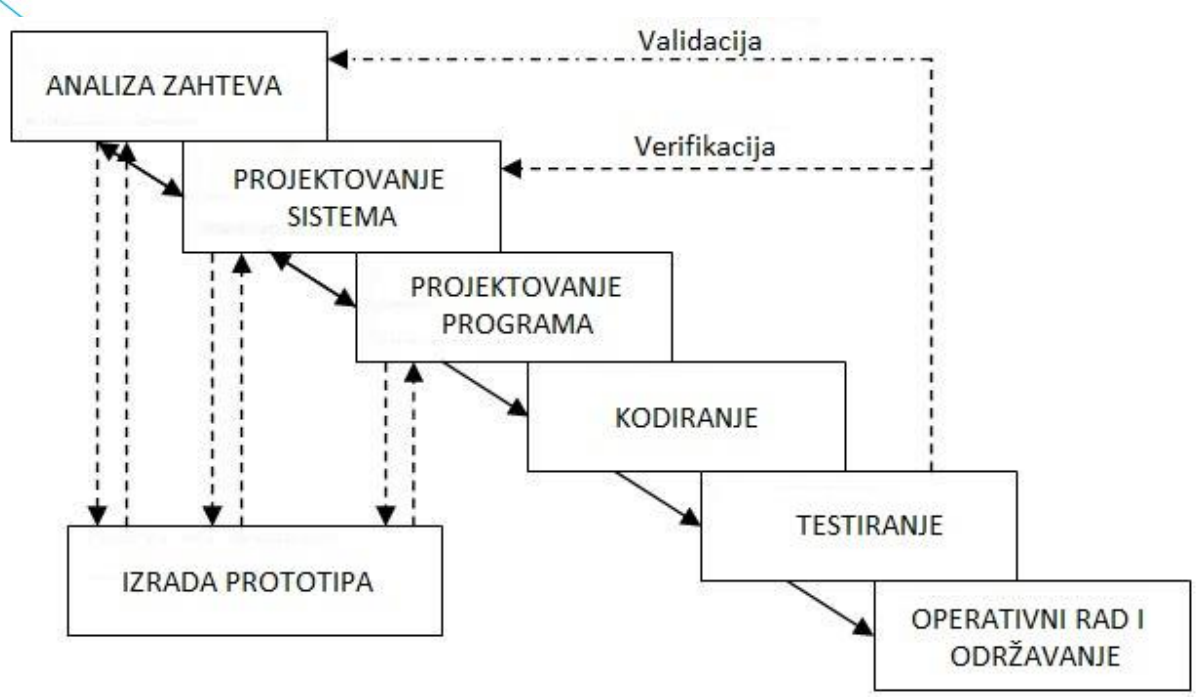
Model vodopada se koristio za propisivanje aktivnosti u procesu razvoja softvera u različitim kontekstima. Na primer, ovaj model je dugi niz godina bio osnova ugovora za razvoj i isporuku softvera u Ministarstvu odbrane SAD. Uz svaku aktivnost u sklopu procesa definisane su etape i međuproizvodi da bi rukovodioci projekata mogli da na osnovu modela mere stepen gotovosti projekta. Na primer, „testiranje delova i integrisanosti" u modelu vodopada završava se etapom „napisani, testirani i integrisani moduli", dok je meduproizvod primerak testiranog koda. Nakon toga, kod se predaje inženjerima koji testiraju sistem, gde se on prvo združuje sa ostalim komponentama sistema (hardver ili softver), a zatim testira u sklopu celine.

Model vodopada je veoma koristan kao pomoćno sreelstvo za izražavanje onoga što projektni tim treba da uradi. Njegova jednostavnost olakšava pružanje objašnjenja klijentima koji nisu upoznati sa procesom razvoja softvera jer eksplicitno ukazuje na međuproizvode koji su neophodni za započinjanje sledeće faze. Mnogi drugi, složeniji modeli, u stvari predstavljaju „ulepšani" model vodopada, kroz uključivanje povratnih sprega i dodatnih aktivnosti.

Najveći problem sa modelom vodopada je u tome što on ne odražava stvarni način na koji se kod razvija. Osim možda za vrlo jasne probleme, softver se obično razvija kroz veliki broj iteracija. Softver se često koristi za rešavanje problema koji nikada ranije nije bio rešen, ili čije rešenje mora da se unapredi da bi odražavalo promene nastale u poslovnom ili radnom okruženju. Na primer, proizvodač aviona može da zahteva softver za novi avion, koji će biti veći ili brži nego postojeći modeli, tako da se pojavljuju novi izazovi kojima se treba baviti, čak i ako tim koji radi na razvoju softvera ima mnogo iskustva u domenu aeronautičkog softvera. Ni razvojni tim ni korisnici ne znaju sve ključne faktore koji utiču na željeni ishod, pa se značajno vreme u analitičkoj fazi troši na razumevanje elemenata i procesa na koje sistem i njegov softver utiču, kao i odnosa koje sistem uspostavlja sa radnim okruženjem. Stoga, ako stvarni proces razvoja softvera nije pod kontrolom, učesnici u razvoju mogu da skaču sa jedne aktivnosti na drugu i vraćaju se nazad, u pokušajima da upoznaju problem i način na koji se on rešava. A kao što smo videli, model vodopada obeshrabruje ponovno vraćanje na neku prethodnu fazu nakon što se ona završi. Uglavnom, ovakva nefleksibilnost kod striktnog modela vodopada predstavlja glavni izvor kritika među zastupnicima drugih fleksibilnijih modela.

Proces razvoja softvera, kroz uključenje aktivnosti i potprocesa koji doprinose boljem razumevanju, pomaže u procesu kontrolisanja „škarta". Izrada prototipova je jedan takav potproces. Prototip predstavlja delimično razvijen proizvod koji omogučava naručiocima i projektantima da ispitaju neke aspekte predloženog sisterna i odluče da li je on pogodan ili potreban u sklopu gotovog proizvoda. Na primer, projektni tim može da izgradi sistem koji implementira jedan deo ključnih zahteva, da bi se ustanovila njihova doslednost, izvodljivost i praktična vrednost. Ako se ispostavi da zahtevi nisu dosiedni, izvodljivi, itd., oni se revidiraju, i time se izbegavaju mnogo skuplje revizije u fazi testiranja.

Slično tome, od delova projekta mogu da se naprave prototipovi, kao što se može videti na slici 3.1.1. Izrada prototipova u fazi projektovanja pomaže projektnom timu da oceni alternativne strategije projektovanja i odluči koja je najbolja za odredeni projekat. Projektanti mogu da dizajniraju zahteve na potpuno različite načine, da bi videli koji od njih poseduje najbolje osobine. Na primer, mreža može da se, u jednom prototipu, projektuje u topologiji prstena a u drugom u topologiji zvezde, pri čemu se ocenjuju performanse u cilju utvrđivanja koja je topologija primerenija za postizanje postavljenih ciljeva u datim ograničenjima.




*Slika 3.1.1. Model vodopada*

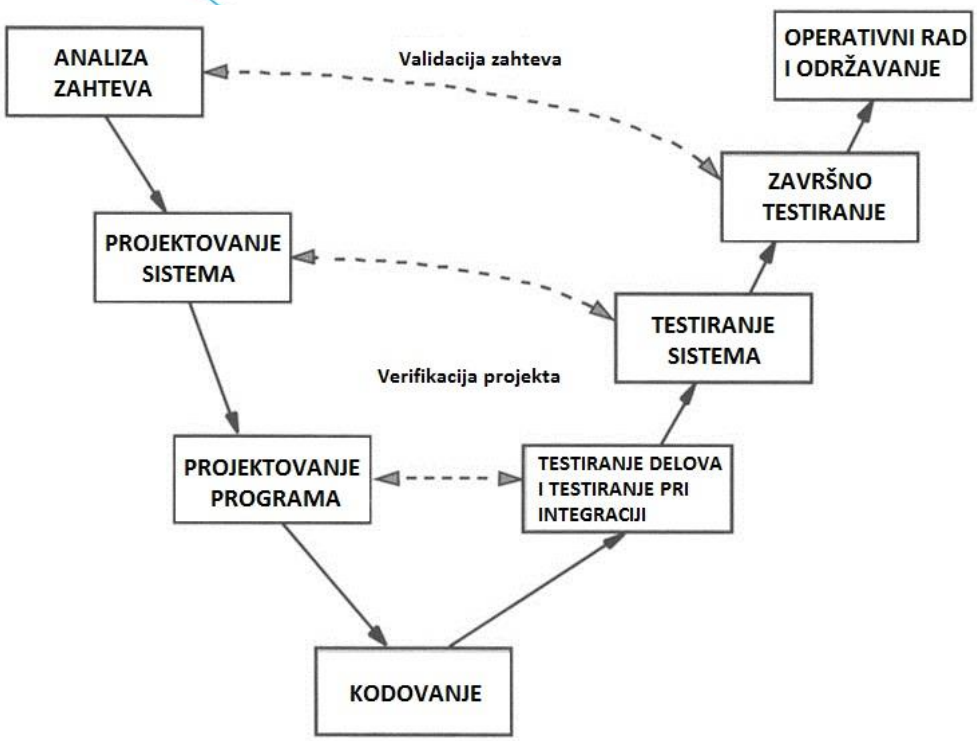
Često se korisnički interfejs gradi i testira u formi prototipa, sa ciljem da korisnici shvate izgled novog sistema a projektanti steknu bolji utisak o tome kakvu interakciju sa sistemom korisnici žele. Prema tome, osnovne poteškoće u zahtevima uočavamo i ispravljamo znatno pre nego što se one zvanično validiraju u fazi testiranja sistema. **Validacija** obezbeđuje potpunost implementacije zahteva u sklopu sistema, tako da se svaka funkcija sistema može propratiti i povezati s određenim zahtevom definisanim u specifikaciji zahteva. Testiranjem sistema zahtevi se verifikuju. **Verifikacija** osigurava ispravan rad svake funkcije. U sklopu validacije proveravamo da li je razvoj rezultirao „pravim" sistemom (usklađenim sa specifikacijom), dok verifikacija proverava kvalitet implementacije. Izrada prototipa je korisna za verifkaciju i validaciju, ali te aktivnosti mogu da se vrše i u drugim fazama razvoja softvera.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 3.2 V model

V model je modifikovani model vodopada, koji pokazuje odnos testiranja i faza analize dizajna (prema nemačkom Ministarstvu odbrane, 1992). Kao što je prikazano na slici 3.2.1, kodiranje predstavlja ishodište V modela, pri čemu su analiza i dizajn na levoj, a testiranje i održavanje na desnoj strani. Testiranje delova i testiranje pri integrisanju bave se ispravnošću programa. V model sugeriše da testiranje delova i testiranje pri integraciji mogu da se koriste za verifikovanje dizajna programa, tj. tokom testiranja delova i testiranja pri integrisanju, programeri i članovi testerskog tima treba da obezbede ispravno implementiranje svih dizajnerskih aspekata programa u kodu. Slično tome, testiranje sisterna treba da verifikuje dizajn sistema, uz potvrdu da su svi aspekti projektovanog sistema ispravno implementirani. Završni test, koji češče sprovode naručioci nego projektni tim, služi za validaciju zahteva. Pri tome se svakom elementu iz specifikacije zahteva dodeljuje po jedan korak u procesu testiranja. Završnim testiranjem se proverava da li su svi zahtevi potpuno implementirani, pre tehničkog prijema sistema i plaćanja.




*Slika 3.2.1. V model*

Veze leve strane sa desnom stranom u sklopu V modela ukazuju da, ako se pronađu problemi tokom verifikacije i validacije, tada se aktivnosti s leve strane V modela mogu ponoviti radi popravke i poboljšanja zahteva, dizajna ili koda, pre nego što se ponove koraci testiranja definisani na desnoj strani. Drugim rečima, V model čini još eksplicitnijim neke iteracije i prepravke koje su sakrivene u



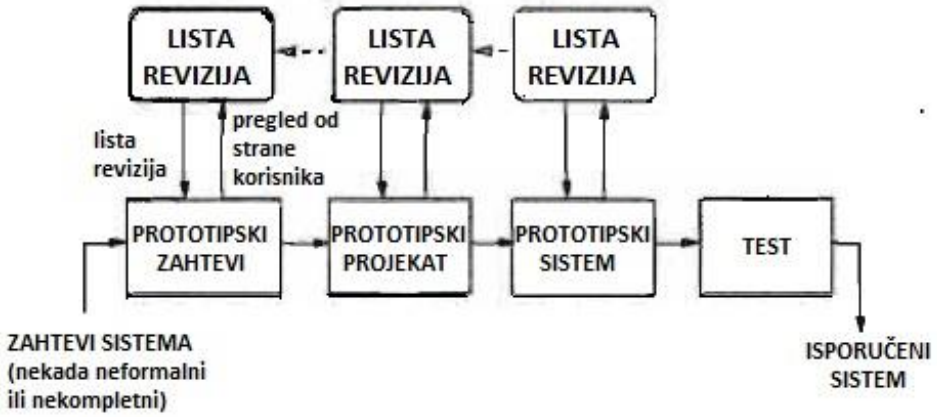
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

modelu vodopada. U središtu pažnje modela vodopada često su dokumenti i artefakti, dok se V model usredsređuje na aktivnost i ispravan rad.

Poput modela vodopada i ovaj model je kritikovan od strane zastupnika agilnih modela, budući da je i kod njega akcenat na testiranju i planiranju unapred. Kod ovog modela skripte za testiranje se pišu unapred i na taj način se testerima sugeriše šta treba da traže umesto da otkriju šta je zapravo tu. Može se reći da ovaj model bolje reflektuje upravljanje projektom iz perspektive projekt menadžera, nego li iz perspektive programera ili korisnika.

### 3.3 Prototipski model

Kao što je prikazano, model vodopada može da se izmeni dodavanjem aktivnosti izrade proto-tipova u funkciji boljeg razumevanja zahteva. Međutim, izrada prototipa ne treba samo da bude dopuna modela vodopada. Ona sama po sebi može da bude osnov za efikasno modelovanje procesa, kao što je prikazano na slici 3.3.1.




*Slika 3.3.1. Prototipski model*

Osnovna ideja je da se umesto definisanja zahteva pre kodiranja, izradi prototip kako bi se zahtevi razumeli i razjasnili. On se izrađuje na osnovu trenutno poznatih zahteva. Pomoću prototipa, klijent može da zapravo „oseti“ sistem osiguravajući tako da projektni tim, korisnik i naručilac postignu zajednički stepen razumevanja stvarnih potreba i predložene funkcionalnosti. Jedna povratna sprega ili više njih kod izrade prototipa zahteva, dizajna ili sistema mogle bi da se eliminišu, zavisno od ciljeva zbog kojih se prototip izraduje. Medutim, ukupni cilj ostaje isti: smanjenje rizika i neodređenosti prilikom projektovanja. Na primer, razvoj sistema može da započne nominalnim skupom zahteva podnetim od strane korisnika. Zatim se istražuju alternative, tako što zainteresovane strane analiziraju sve moguće izveštaje, tabele i druge izlaze iz sistema koje neposredno koriste korisnici i klijenti. Pa kako korisnik i naručilac definišu svoje želje, dolazi do revizije zahteva. Kad se postigne



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

dogovor o zahtevima projektni tim prelazi na dizajn, a zatim se istražuju varijante u dizajnu, često uz konsultaciju s korisnikom.

Inicijalni dizajn se revidira sve dok projektni tim, korisnici i naručilac nisu zadovoljni rezultatom. Ako razmatranje varijanti dizajna otkrije problem koji potiče od zahteva, projektni tim se vraća na izrađene zahteve, ponovo ih razmatra i menja specifikaciju zahteva. Napokon, sistem se kodira uz razmatranje varijanti, s mogućim ponovnim iteracijama kroz zahteve i dizajn. Ovo je odlična praksa kad se radi sa komplikovanim i velikim sistemima kod kojih ne postoji neki sistem na osnovu kog bi se zahtevi razjasnili, kao i kad postoji potreba da željeni sistem ima brojne interakcije s korisnicima.

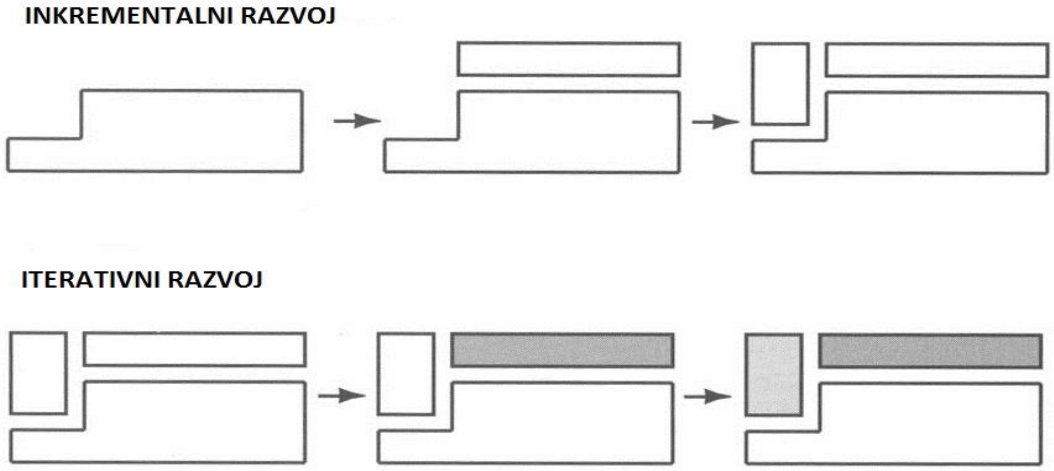
Pored svih prednosti koje ovaj model nosi sa sobom (aktivnije uključenje klijenata, brža detekcija grešaka, nefunkcionalnosti i dvosmislenih situacija), on ima i izvesne nedostatke. Jedna je ta što se on praktično svodi na implementiranje i repariranje već izgrađenog sistema. Time se kompleksnost sistema dodatno uvećava.

### 3.4 Fazni razvoj: Inkrementi i iteracije

U početnim godinama razvoja softvera naručioci su bili spremni da duže čekaju na završetak razvoja softverskih sistema. Nekada su prolazile godine od trenutka pisanja dokumenta sa zahtevima, do isporuke sistema, što je nazivano **trajanjem ciklusa**. Medutim, današnja poslovna okruženja više ne tolerišu duga čekanja. Softver pomaže da se proizvodi razlikuju na tržištu., a naručioci uvek traže nov kvalitet i funkcionalnost. Na primer, u 1996. godini, 80% prihoda Hewlett-Packarda dobijeno je od proizvoda uvedenih u prethodne dve godine. Stoga se razvijaju novi procesni modeli, koji treba da pomognu da se smanji trajanje ciklusa.

Jedan način za smanjenje trajanja ciklusa jeste fazni razvoj, kao što je prikazano na slici 3.4.1. Sistem se projektuje tako da može da se isporučuje u delovima, omogućavajući korisnicima da imaju deo funkcija, dok je ostatak još u razvoju. Tako obično postoje dva sistema koja uporedo funkcionišu: produkcioni sistem i razvojni sistern. **Operativni ili produkcioni sistem** je onaj koji trenutno koriste naručilac i korisnik. **Razvojni sistem** predstavlja sledeću verziju koja se priprema da zameni postojeći produkcioni sistem. Sisteme često nazivamo prema rednom broju njihove verzije (izdanja): razvojni tim gradi verziju 1, testira je, a zatim predaje korisnicima kao prvu produkcionu verziju. Zatim, dok korisnici koriste verziju 1, projektni tim gradi verziju 2. Tako, projektni tim uvek radi na verziji *n + 1*, dok je verzija *n* u fazi operativnog korišćenja. Zahvaljujući svim saznanjima stečenim tokom razvoja prethodnih delova sistema, razvojni tim može da ga konstantno unapređuje i da pri svakoj novoj iteraciji dodaje nove funkcionalne mogućnosti.

Postoji mnogo načina da projektni tim odluči kako da organizuje razvoj pojedinih verzija. Dva najpopularnija pristupa su **inkrementalni razvoj** i **iterativni razvoj**. Kod **inkrementalnog razvoja,** sistem, onako kako je specifikovan u specifikaciji zahteva, se deli na podsisteme, prema funkcijama. Verzije se definišu na samom početku kao mali, funkcionalni podsistemi, a zatim se svakoj novoj verziji prikliučuju nove funkcije. Na vrhu slike 3.4.1 prikazano je kako se inkrementalnim razvojem svako novo izdanje sistema polako dograđuje, sve do potpune funkcionalnosti.




*Slika 3.4.1. Inkrementalni i iterativni modeli*

Medutim, prilikom **iterativnog razvoja** isporučuje se odmah, na samom početku, potpun sistem, a zatiin se menjaju funkcije svakog podsistema, u svakoj novoj verziji. Donji deo slike 3.4.1 prikazuje tri verzije u iterativnom razvoju.

Da bismo razumeli razliku izmedu inkrementainog i iterativnog razvoja, razmotrićemo programski paket za obradu teksta. Pretpostavimo da paket treba da isporuči tri vrste funkcionalnosti: unošenje teksta, organizovanje (tj. isecanje i umetanje), kao i formatiranje teksta (upotreba različitih veličina i stilova). Za gradnju sistema inkrementalnim razvojnim putem, u verziji 1 možemo da obezbedimo samo funkcije za unos teksta, zatim u verziji 2 i funkcije za unos i za organizaciju, a na kraju u verziji 3 funkcije za unos teksta, organizaciju i formatiranje. Međutim, koristeći iterativni razvoj, mogli bismo da obezbedimo primitivne oblike sva tri tipa funkcionalnosti već u verziji 1. Na primer, možemo da unesemo tekst, a zatim da isecamo i umećemo, pri čemu je isecanje i umetanje nezgodno i sporo. Tako da u sledećoj iteraciji, tj. verziji 2, imamo istu funkcionalnost, ali poboljšan kvalitet, tako da je isecanje i umetanje lako i brzo. Svaka verzija unapređuje na neki način prethodnu. Ovakvi oblici faznog razvoja poželjni su iz sledećih razioga:

1. Obuka može da započne s prvom verzijom, čak i ako neke funkcije nedostaju. Proces obuke omogućava projektnom timu da posmatra kako se izvršavaju neke funkcije, i time sugeriše poboljšanja u kasnijim verzijama. Na isti način, projektni tim je vrlo otvoren za pitanja korisnika.

2. Na tržište može da se izađe vrlo rano, ako se radi o funkcionalnostima koje nikada ranije nisu bile nuđene.

3. Česte verzije omogućavaju projektnom timu da brzo i globalno otkloni nepredviđene probleme, čim se o tome dobije izveštaj od produkcionog sistema.

4. Projektni tim može da se usredsredi na različite oblasti usavršavanja u različitim verzijama. Na primer, u jednoj verziji može da izmeni korisnički interfejs sistema sa komandno orijentisanog na



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

zadavanje komandi mišem, na osnovu stručnog znanja tima za korisnički interfejs. Sledeća verzija može da se usredsredi na poboljšanje performansi sistema.
Prednost ovog modela je svakako mogućnost testiranja nakon svake iteracije, jer se tako lako uočavaju falinke nastale promenama u toj konkretnoj iteraciji. Takođe, inicijalni projekat je jeftiniji i njegova izrada je lakša i brža. Što se tiče nedostataka, oni se ogledaju u tome što nekad rezultujući ukupni troškovi mogu premašiti planirana sredstva. Takođe, pri dodavanju neke dodatne funkcionalnosti mogu nastati problemi u vezi s arhitekturom sistema koji nisu bili razmotreni u prethodnim prototipovima.

### 3.5 RUP model

Najbolji primer inerativno-inkrementalne metodologije je RUP model (*Rational Unified Process*) koji se pojavio 1998. To je model zasnovan na UML dijagramima koji pokriva kompletan životni ciklus. Sveukupan razvojni ciklus kod ovog modela sastoji se iz sledećih faza:

- Početak: U ovoj fazi se definišu početni zahtevi. Odredi se opseg i ciljevi projekta i identifikuju se ljudi, organizacije ili eksterni sistemi koji će interagovati sa sistemom. Takođe, u ovoj fazi se izvedu prve procene rizika i konfiguriše se inicijalna arhitektura sistema;
- Razrada: U ovoj fazi se vrši prikupljanje ključnih zahteva, razvoj i validacija arhitekture softverskog sistema, te planiranje preostalih faza projekta. U ovoj fazi je jako bitno razumeti i eliminisati sve rizike visokog prioriteta, te osigurati da su kritični procesi i standardi ispravni pre sprovođenja naredne faze;
- Konstrukcija: Sistem se implementira koristeći iterativni i inkrementalni pristup zasnovan na prethodno razvijenoj arhitekturi. Definišu se preostali zahtevi i dizajn sistema, i proverava se da li sistem zadovoljava potrebe korisnika. Takođe, u ovoj fazi se vrši optimizacija resursa kako bi se minimalizovali troškovi;
- Tranzicija: U ovoj fazi se vrši testiranje i validacija kompletnog sistema, kao i njegova integracija s postojećim sistemima. Takođe, ova faza obuhvata i obuku korisnika i isporuku sistema.

Svaka od ovih faza se može razbiti na iteracije, a svaka od njih se sastoji od devet aktivnosti koje se obavljaju u toku jedne iteracije:

- *Poslovno modelovanje:* svodi se na opisivanje poslovnih procesa i interne strukture samog posla. Nakon ove aktivnosti kreiraju se *Poslovni model slučaja upotrebe* i *Poslovni objektni model*;
- *Upravljanje zahtevima*: odnosi se na otkrivanje, organizovanje i dokumentovanje zahteva. Kreira se *model slučaja upotrebe*;
- *Analiza i dizajn*: tiče se kreiranja arhitekture i dizajna softverskog sistema. Rezultat su *dizajn model* i *analitički model*;
- *Implementacija*: odnosi se na pisanje i dibagiranje koda, te testiranje modula. Nakon ove aktivnosti dobijamo sors kod, kao i izvršne i propratne fajlove;
- *Testiranje:* obuhvata integraciono, sistemsko i završno testiranje;

- *Stacioniranje*: podrazumeva pakovanje softvera, kreiranje instalacionih skripti, pisanje korisničke dokumentacije i druge zadatke s ciljem da se softver učini dostupnim krajnjim korisnicima;
- *Upravljanje projektom*: odnosi se na planiranje projekta, raspoređivanje i kontrolu;
- *Upravljanje promenama i konfiguracijom*: vodi se računa o verziji kao i zahtevima za promenama;
- *Okruženje*: tiče se adaptiranja procesa potrebama projekta ili organizacije, kao i izbora razvojnog alata.

*Slika 3.5.1.RUP model*

Uglavnom, glavna prednost ovog modela je što je sam proces jako dobro dokumentovan i što je zasnovan na funkcionalnom i strukturnom modelovanju problema. Zato ga je lako formalno predstaviti (korišćenjem UML-a i sličnih jezika). Takođe, korišćenjem ovog modela lako je otkriti i umanjiti rizike pre samog izvršenja.

Najveći nedostatak ovog modela leži u njegovoj preteranoj kompleksnosti zbog iterativno-inkrementalne prirode procesa. Pored toga, loše je i strogo vezivanje za UML jer on sam nije savršen i može samo da dodatno pogorša problem nekonzistentnosti.

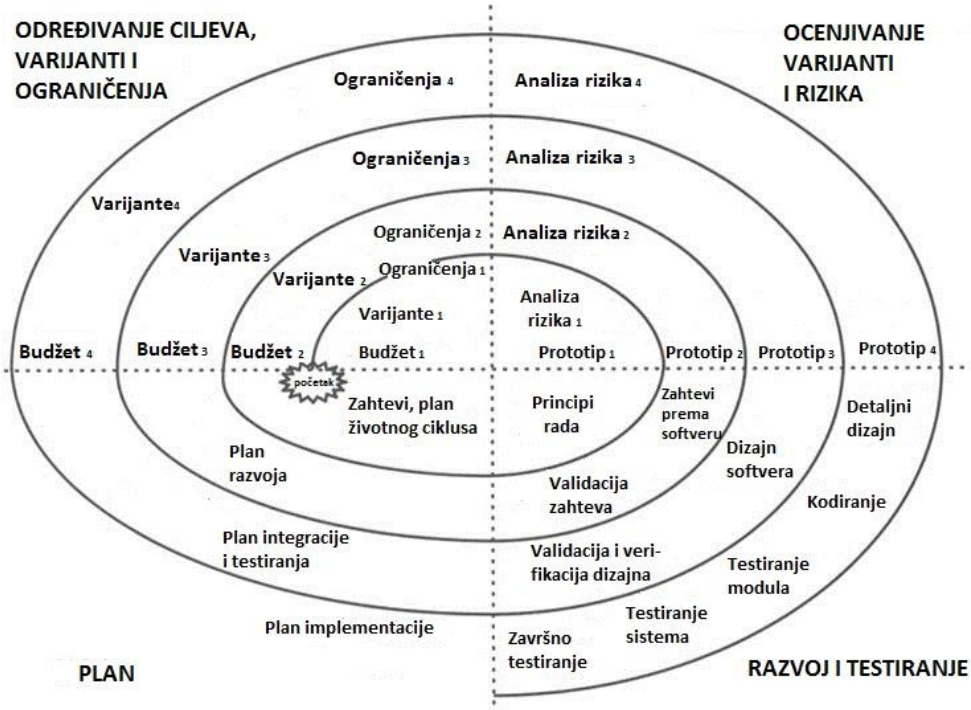
### 3.6 Spiralni model

Ključna karakteristika spiralnog modela ogleda se u **upravljanju rizicima** tokom redovnih faza razvojnog ciklusa. Bem je 1988. posmatrao razvoj softvera u svetlu prisutnih rizika, sugerišući da spiralni model može da kombinuje aktivnosti razvoja sa upravljanjem rizicima, da bi ih bilo manje i da bi se lakše kontrolisali. Spiralni model, prikazan na slici 3.6.1, na neki način je sličan modelu iterativnog razvoja datom na slici 3.4.1. Započinjući sa zahtevima i početnim planom razvoja (uključujući budžet, ograničenja i alternative u pogledu osoblja, dizajna i razvojnog okruženja), ovaj proces ubacuje korak procenjivanja rizika i prototipske alternative, pre nego što se proizvede dokument „principi rada", sa ciljem opisivanja funkcionisanja sistema na visokom nivou apstrakcije. Iz



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

tog dokumenta, definiše se i nadgleda skup zahteva da bi se potvrdili kompletnost i doslednost zahteva. Stoga je princip rada proizvod prve iteracije, dok su zahtevi glavni proizvod druge po redu iteracije. U trećoj iteraciji razvoj sistema produkuje dizajn, dok četvrta iteracija omogućava testiranje.




*Slika 3.6.1. Spiralni model*

U svakoj iteraciji, analiza rizika određuje različite varijante sa aspekta zahteva i ograničenja, dok se izradom prototipova verifikuje izvodljivost ili poželjnost, pre izbora odgovarajuće varijante. Nakon identifikacije rizika, rukovodioci projekta moraju da odluče kako da ih eliminišu ili minimizuju. Na primer, projektanti možda nisu sigurni da li će korisnicitna više da se dopada jedan korisnički interfejs ili drugi. Da bi se izbegao rizik izbora interfejsa koji bi onernogućio produktivnu upotrebu novog sistema, projektanti mogu da izrade prototip oba interfejsa i testiraju ih da bi zaključili koji je poželjniji, ili čak da oba interfejsa uključe u projekat kako bi korisnici mogli da, nakon prijavljivanja, odaberu interfejs. Ograničenja kao što su budžet i vreme isporuke pomažu kod izbora strategije upravljanja rizicima.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 4. Specifikacija zahteva

*Specifikacija* je početna faza softverskog procesa. Analiziraju se *zahtevi* za budući sistem, i kao rezultat se dobija *dokument* o zahtevima koji opisuje *šta* sistem treba da radi, po mogućstvu bez prejudiciranja kako da se to postigne. Specifikacijom se bavi razvojni tim i budući korisnici softvera.

Sami zahtevi se dele na dve vrste:

- ***Funkcionalni zahtevi***. Opisuju funkcije sistema, dakle usluge koje bi on trebalo da obavlja, izlaze koje on daje za zadate ulaze, te njegovo ponašanje u pojedinim situacijama. Drugim rečima, oni opisuju šta programeri moraju da implementiraju kako bi omogućili korisnicima da obave posao (korisnički zahtev), na taj način zadovoljavajući poslovni zahtev.
- ***Nefunkcionalni zahtevi***. Izražavaju ograničenja na funkcije sistema, na primer, traženo vreme odziva, traženi nivo pouzdanosti, dozvoljeno zauzeće memorije, obavezu korišćenja određenog programskog jezika, poštovanje određenih standarda, pa čak i različita kulturološka ili pravna organičenja.

Svaki od funkcionalnih ili nefunkcionalnih zahteva evoluira kroz različite fokuse i perspektive, definišući različite nivoe detalja u zahtevima. Svaka nenamerna praznina u zahtevima povećava rizik od kraha projekta ili dodavanja drugog defekta. Česti problemi prilikom specifikacije zahteva su što klijenti nisu sigurni šta tačno hoće, nikad nisu zadovoljni, menjaju svoja očekivanja tokom projekta i sl. Zbog toga je jako važno da klijent napravi razliku između onoga što želi i onoga što mu treba. Korišćenjem odgovarajućih standarda omogućeno je klijentu da postavi onako da ih isporučilac razume, kao i isporučiocu da analizom zahteva bude siguran da je u mogućnosti da sprovede softverski proizvod željenog kvaliteta.

Proces upravljanja zahtevima detaljnije se može podeliti u sledeće podaktivnosti:

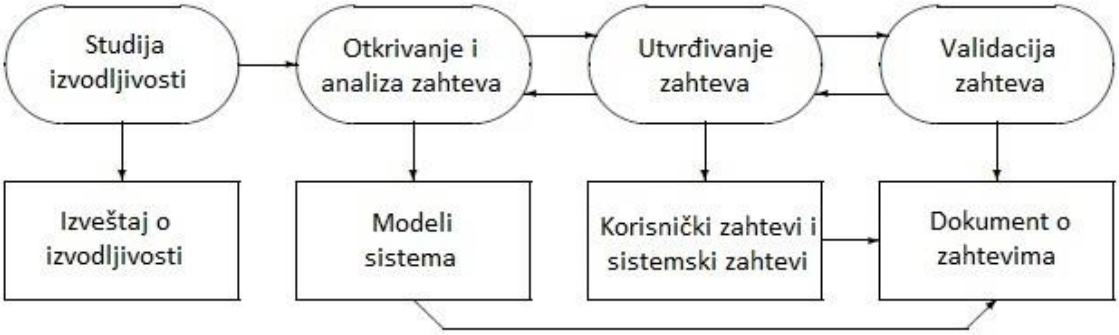
- *Studija izvodljivosti.* Procenjuje se da li se uočene potrebe korisnika mogu zadovoljiti pomoću dostupnih hardverskih i softverskih tehnologija, da li bi predloženi sistem bio isplativ u poslovnom smislu, te da li sistem može biti razvijen u okviru raspoloživog budžeta;
- *Prikupljanje i analiza zahteva.* Otkrivaju se zahtevi, tako što se posmatraju postojeći sistemi, analiziraju radni procesi, intervjuišu budući korisnici i njihovi menadžeri, itd. Na taj način stvaraju se modeli sistema, a ponekad i prototipovi, sve u cilju boljeg razumevanja sistema koji treba kreirati.
- *Specifikacija zahteva.* Informacije skupljene analizom pretvaraju se u tekstove koji definišu zahteve. Postoje dva nivoa u opisivanju zahteva: “korisnički zahtevi” i “sistemski zahtevi”.
- *Validacija zahteva.* Proverava se da li su zahtevi:
    - Realni (mogu se ostvariti raspoloživom tehnologijom i budžetom). Ovo pitanje se često odnosi na nefunkcionalne zahteve.
    - Neophodni (da li je moguće ukloniti ih bez većeg uticaja na osnovne funkcionalnosti sistema)
    - Konzistentni (nisu u međusobnom konfliktu)
    - Jasni i potpuni (uključene su sve potrebne funkcije i ograničenja).
    - Testabilni (Ovo je možda najbitnija osobina u ovoj fazu razvoja. Ukoliko ne možete da nađete način da testirate neki korisnički zahtev, onda nemožete znati da li je on



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

zaista implementiran. Većina metodologija podrazumeva da se prvo piše test za neki zahhtev ili funkcionalnost pa tek onda se vrši implementacija)

- Upravljanje promenama zahteva
  Način odvijanja ovih podaktivnosti prikazan je na slici 4.1.




*Slika 4.1. Podaktivnosti koje sačinjavaju aktivnost specifikacije*

Rezultat specifikacije su dokumenti koji opisuju zahteve. Ti dokumenti razlikuju se po svom sadržaju, načinu opisivanja, te detaljnosti. Uvode se sledeći nazivi za pojedine vrste dokumenata:

- *Korisnički zahtevi.* To je manje precizan tekst u prirodnom jeziku, propraćen dijagramima i tablicama, koji opisuje funkcije sistema i ograničenja pod kojima sistem radi. Prilagođen je krajnjim korisnicima;
- S*istemski zahtevi.* Reč je o detaljnom i preciznom opisu funkcija sistema i ograničenja. Može da služi kao temelj ugovoru između razvojnog tima i klijenta. Služi softverskim inženjerima kao polazište za dizajniranje. Pisan je u donekle struktuiranom prirodnom jeziku;
- *Modeli sistema.* To su dijagrami koji opisuju ponašanje sistema na način koji je precizniji i jezgrovitiji od prirodnog jezika. Nastaju tokom analize zahteva kao sredstvo komunikacije izmedu razvojnog tima i budućih korisnika softvera;
- *Dokument o zahtevima.* Reč je o konačnom rezultatu specifikacije, dobijenom kao unija svih prethodno opisanih dokumenata.
- *Specifikacija softverskog dizajna.* Ovaj dokument nije obavezan. Predstavlja apstraktni opis građe softvera i funkcija koje on obavlja, a pisan je u formalnom jeziku. Pretpostavlja određenu arhitekturu sistema, te na osnovu nje dalje razrađuje "sistemske zahteve". Eventualno se stvara kao most između aktivnosti specifikacije i dizajna. Služi razvojnom timu kao polazište za dalji dizajn i implementaciju.

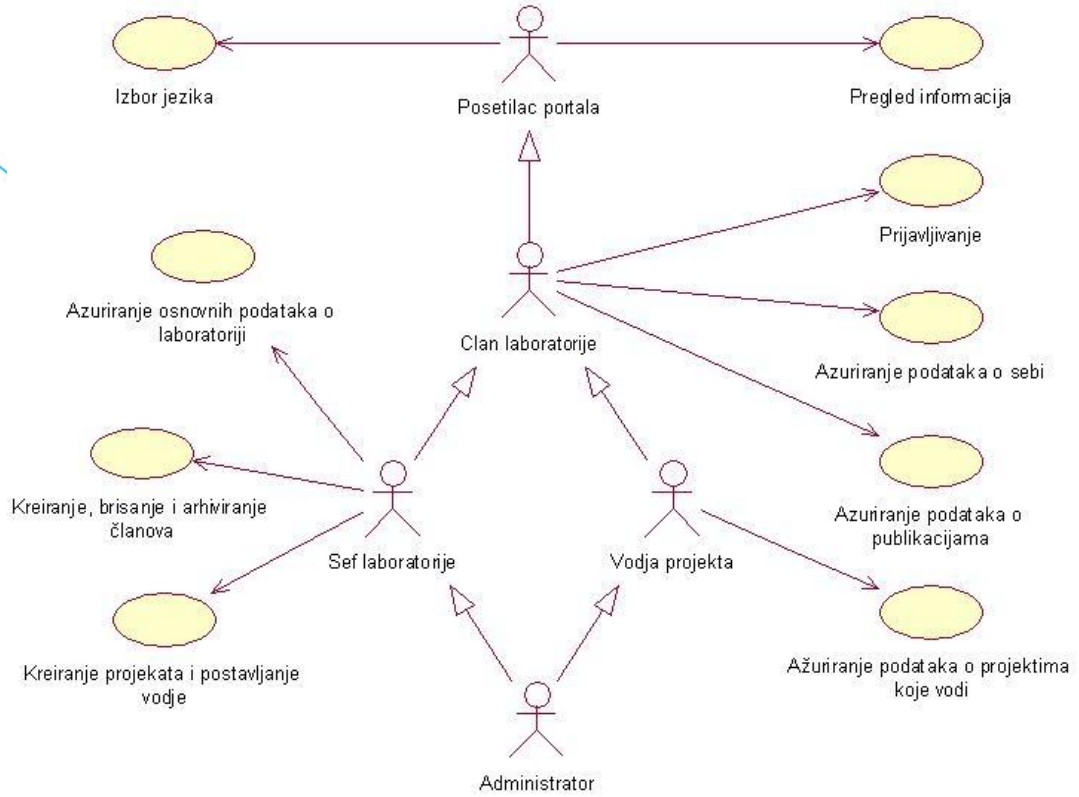
Dve najčešće korišćene tehnike za istraživanje korisničkih zahteva su **slučajevi korišćenja** (*use-case*) i **korisničke priče** (*user stories*).

UML (*Unified Modeling Language*) predstavlja standardizovani vizuelni jezik za modelovanje softvera. **Slučaj korišćenja** je opis skupa sekvenci akcija, koje sistem obavlja da bi proizveo vidljiv rezultat od vrednosti za pojedinog **aktera**. Svaka sekvenca akcija reprezentuje interakciju aktera sa sistemom



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

(jedan moguć scenario). Slučaj korišćenja specificira šta sistem (podsistem, klasa, interfejs) radi, a ne kako radi. Akter predstavlja neki skup (korisničkih) uloga i to može biti čovek (korisnik) ili neki sistem s kojim modelirani sistem interaguje. Aplikacioni sistem interaguje sa jednim ili više spoljašnjih aktera.




*Slika 4.2. Primer dijagrama slučajeva korišćenja*

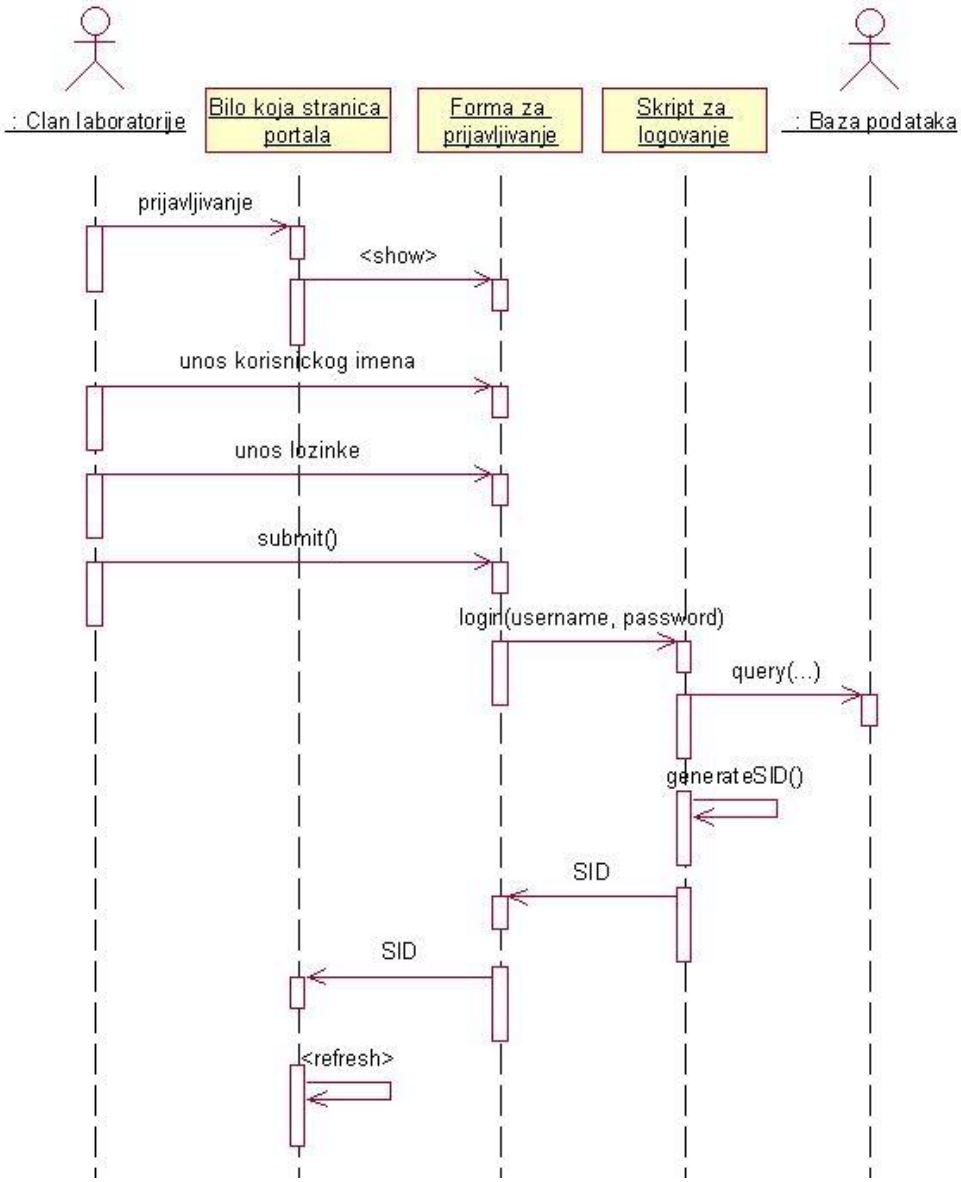
Na primeru dijagrama slučajeva korišćenja sa slike 4.2, akteri su posetilac portala, član laboratorije, šef laboratorije, vođa projekta i administrator. Mogući događaji tj. slučajevi korišćenja koje treba obuhvatiti u specifikaciji bi bili sledeći:

- Izbor jezika
- Pregled osnovnih podataka o laboratoriji
- Pregled spiska članova
- Pregled podataka o određenom članu laboratorije
- Pregled publikacija po autoru
- Pregled publika po tipu
- Pregled publikacija po godini
- Pregled spiska projekata
- Pregled podataka o određenom projektu



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Za modelovanje dinamičkih aspekata sistema najpraktičniji su dijagrami sekvenci (sequence diagrams). Oni se koriste za specifikaciju vremenskih zahteva u opisu složenih scenarija. Ako su dijagrami slučajeva korišćenja prethodno definisani, dijagram sekvenci predstavlja jednu od njihovih realizacija koja prikazuje redosled događaja i redosled operacija. Jedan dijagram sekvenci za aktivnost prijavljivanja na primeru laboratorije dat je na slici 4.3.




*Slika 4.3. Primer dijagrama sekvence*

Osnovni tok aktivnosti bi bio sledeći:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

1. Korisnik bira opciju prijavljivanje sa bilo koje stranice portala.
2. Po prikazu forme za prijavljivanje, korisnik unosi korisničko ime i lozinku, te nakon toga aktivira komandu *submit()*.
3. Uneti podaci se proveravaju [izuzetak: pogrešno korisničko ime i/ili lozinka].
4. Generiše se jedinstveni identifikator sesije (SID).
5. Osvežava se polazna stranica sa SID-om kao parametrom.

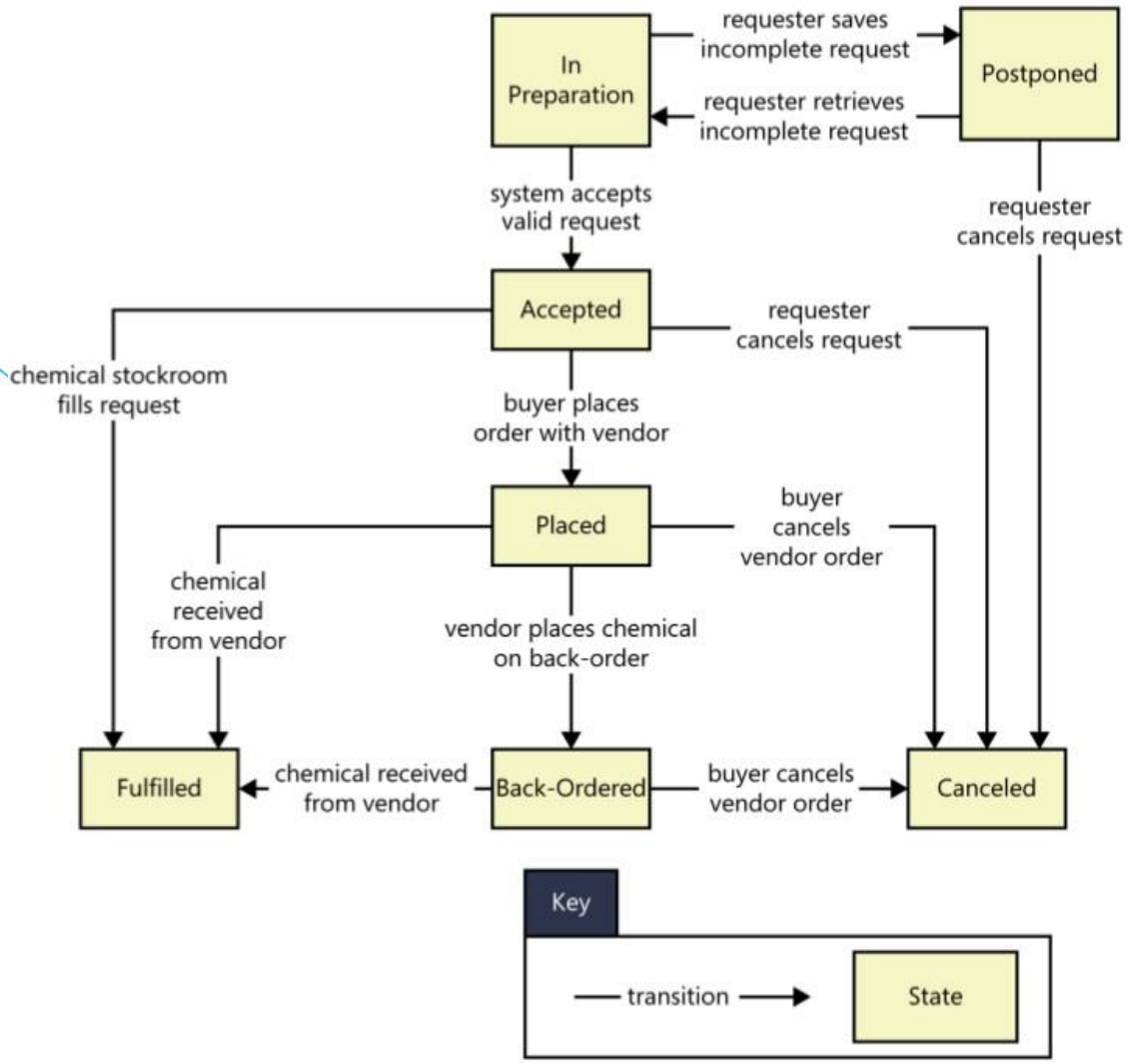
Još jedan način za prikaz specifikacije zahteva predstavlja tzv. dijagram stanja i prelaza (*State Transition Diagram*, skraćeno STD). STD sadrži tri vrste elementa:

- Moguća stanja sistema, prikazana pravougaonicima;
- Dozvoljene promene stanja ili prelaze, prikazane kao strelice koje povezuju parove pravougaonika;
- Događaje ili uslove koji uzrokuju svaku tranziciju, prikazane tekstom na svakoj tranziciji. Tekst može identifikovati kako ulazni događaj, tako i odgovarajući odgovor sistema.

STD će za objekat koji prolazi kroz definisani životni ciklus imati jedno ili više završnih stanja. Završna (terminaciona) stanja imaju samo dolazne tranzicije, ali ne i odlazne.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Slika 4.4. Primer dijagrama stanja i prelaza*

Kod specifikacije zahteva se mogu pojaviti sledeći problemi:

- Ukoliko novi sistem treba da promeni postojeći način rada, biće ga teško definisati budući da se s njim još nema iskustva;
- Razni korisnici imaju različite zahteve koji mogu biti u konfliktu. Konačni zahtevi uvek nužno predstavljaju kompromis;
- Finansijeri sistema i njegovi korisnici obično nisu isti ljudi. Klijent postavlja zahteve motivisane organizacijskim ili budžetskim ograničenjima koja su u konfliktu s korisničkim zahtevima;
- Poslovno i tehničko okruženje se menja. Time se menjaju i zahtevi. Promena zahteva je normalna stvar.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Pored UML dijagrama slučajeva korišćenja, za specificiranje zahteva mogu se koristiti i **korisničke priče**. Jedan priča predstavlja kratak i jednostvan opis funkcionalnosti posmatrano iz perspektive klijenta ili sisitema. Sastoje se iz tri aspekta:

1. Kratkog **naslova i pisanog opisa** koji se koristi za planiranje i kao podsetnik On koristi sledeći obrazac:
    a. Naslov:
    b. Opis: Kao <tip korisnika> ja želim <neki cilj> jer hoću da <neki razlog>.
2. Razgovora o korisničkoj priči koji služi da se uoče i izdvoje detalji o njoj. Akcenat je na razgovoru i diskusijji o korisničkim zahtevima umesto na pisanju (dokumentovanju) zahteva.
3. **Test prihvatljivosti (Acceptance Test):** Testova i detalja kojima se utvrđuje da je korisnička priča implementirana (*acceptance test*)

Jedan od čestih problema u razumevanju korisničkih priča je razumevanje njihove razlike u odnosu na tradicionalne specifikacije korisničkih zahteva. Najveća razlika, koja se na prvi pogled može uočiti, jeste nivo detalja. Korisničke priče pružaju onoliku količinu detalja koliko može poslužiti kao osnova procene trajanja implementacije sa razumljivo malim rizikom da dođe do greške. Kada dođe vreme da se konkretna priča implementira, programer će diskutovati sa klijentom i dobiti dovoljno detaljni opis zahteva u razgovoru uživo. Druga razlika se ogleda u tome što je fokus stavljen na potrebe korisnika. Upravo je ključ u pisanju dobre korisničke priče vrednovanje iz aspekta korisnika.

Razlika između korisničkih priča i slučajeva korišćenja je u tome što se one pišu u slobodnoj formi i ne u tehničkom žargonu. Na primer, prilikom izrade bilo kog portala za nekretnine, korisničke priče koje ga mogu opisati su:

- Kao korisnik želim da pretražujem oglase
- Kao agenti želim da postavim oglas ili više njih.

Naravno, u ovom slučaju, one su glomazne i nepraktične, i nazivaju se epskim (*epic*). Zato se preko dodatnih detalja, koji uzgred i sami predstavljaju male korisničke priče za sebe, pojašnjavaju konkretni zahtevi (Da li korisnik mora biti registrovan na portalu? Koje vrednosti korisnik pretražuje? Šta se prikazuje za pronađene stanove? itd.). Stoga se ove epske korisničke priče mogu predstaviti kao skup manjih, pri čemu treba voditi računa da se ne pretera u usitnjavanju. Ilustrujmo to primerom:

- Kao **registrovani** korisnik ja mogu da pretražujem oglase po sledećim kriterijumima: Datum postavljanja oglasa, tip oglasa..
- Kao registrovani agent je želim da izlistam sve oglase koje sam do sada postavio, vidim broj pregleda za svaki od njih i izvezem ih u excel tabelu kako bih mogao da radim dalju analizu ovih podataka.

Budući da one predstavljaju vrednost za klijenta a ne za programera, ne smeju da sadrže informacije o tome na kom jeziku će biti pisane, kakvo će biti povezivanje s bazom i druge tehničke detalje.

Korisnčkim pričama je moguće predstaviti i nefunkcionalne zahteve. Navedimo nekoliko primera:

- Sistem mora da podržava IE8, IE9 i Chrome 15 web čitače
- Kao korisnik želim da interfejs podržava engleski jezik, neki romansnki jezik, i jedan kompleksni jezik jer u tom slučaju postoji velika verovatnoća da će raditi i na ostalih 70 jezika

Još jedan od problema specifičnog za rad sa korisničkim pričama je u tome što je kod velikog broja korisničkih priča prilično teško razumeti veze između njih. Ovaj problem se može umanjiti korišćenjem uloga i držanjem korisničkih priča na srednjem ili visokom nivou dok tim ne bude spreman da počne da ih razvija. Drugi problem sa korisničkim pričama je što ih je često potrebno dodatno dokumentovati. Srećom, on se može jednostavno prenebregnuti vezivanjem dokumentacije za iteracije.

Iako su korisničke priče sjajan način da se poboljša razumevanje unutar tima, postoje problemi kada su timovi veliki. Određeni deo komunikacije jednostavno mora biti zapisan, a informacije se ne smeju menjati kroz prenošenje od jednog do drugih članova tima.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 5. Projektovanje arhitekture sistema

*Dizajn arhitekture sistema* predstavlja početnu fazu dizajniranja. Sistem se dekomponuje na nekoliko podsistema, od kojih svaki obavlja određeni skup zadataka. Način komuniciranja podsistema kao i način njihove kontrole se utvrđuju okvirno. Dizajn arhitekture često se preklapa s apstraktnim specificiranjem, odnosno s izradom detaljne specifikacije softverskog dizajna. U idealnom slučaju, specifikacija ne bi smela da sadrži informacije o dizajnu. No u praksi je to gotovo nemoguće postići, osim za vrlo male sisteme. Dizajn arhitekture je između ostalog nužno i zato da bi se bolje strukturirala sama specifikacija.

Dizajn arhitekture dobro se uklapa u klasične metode razvoja softvera. No zanimljivo je napomenuti da potrebu za tom aktivnošću prepoznaju takođe i agilne metode. Na samom početku agilnog procesa postavlja se arhitektura sistema, a dalji razvoj u obliku brzih iteracija najčešće ne menja tu arhitekturu.

*Podsistem* definišemo kao relativno samostalnu celinu, čije funkcionisanje uglavnom ne zavisi od usluga koje pružaju drugi podsistemi. Podsistem je sastavljen od svojih delova (komponente, moduli, objekti, itd.) i on ima definisani interfejs za komunikaciju s drugim podsistemima te se podvrgava određenim režimima kontrole. Prilikom dizajniranja arhitekture nekog sistema, nije dovoljno samo definisati njegove podsisteme, već treba odrediti i odnose između njih. Može se primetiti da postoje dve bitno različite vrste odnosa između podsistema: međusobna komunikacija i međusobna kontrola. Zato se u sklopu dizajna arhitekture pojavljuju sledeće dve podaktivnosti:

- *Struktuiranje sistema*. Sistem se rastavlja na podsisteme. Utvrđuje se komunikacija između podsistema, dakle razmena podataka.
- *Modelovanje kontrole*. Određuje se tok kontrole između podsistema, dakle ko kime upravlja.

Rezultat struktuiranja predstavlja se kao neformalni blok dijagram, gde su blokovi podsistemi, dok strelice označavaju komunikaciju. Pojedini blok može da sadrži podblokove. Od UML dijagrama moguće je za tu istu svrhu koristiti dijagrame klase, paketa ili komponente. U principu, za svaki sistem može da se razvije specifična arhitektura. Ipak, postoje i uobičajeni modeli strukture koji su se pokazali upotrebljivi za razne sisteme. Više o modelima možete pogledati u Appendixu 2.

Nakon dizajniranja arhitekture sistema, dalje je potrebno zasebno dizajnirati svaki od podsistema. U tu svrhu danas se obično koristi objektni pristup. Reč je o takvom načinu dizajna gde projektant prvenstveno razmišlja o objektima i pripadnim klasama, a tek onda o operacijama ili funkcijama.

Konačni rezultat objektnog dizajniranja je objektno-dizajnirani softver. On se sastoji od objekata koji su u međusobnoj interakciji, pri čemu svaki objekt pripada nekoj klasi. Interakcija objekata odvija se na način da jedan objekt (klijent) pokrene operaciju drugog objekta (servera). U tom smislu, operacije se mogu tumačiti kao usluge, a svaki objekat može da se pojavi u ulozi klijenta i u ulozi servera. Prilikom poziva operacija, objekti razmenjuju podatke (parametre i rezultate).



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

U objektno-dizajniranom softveru ne postoje globalne funkcije koje bi se izvršavale nezavisno od objekata, već je funkcionalnost izražena isključivo u terminima operacija vezanih uz objekte. Slično tome, ne postoje globalne promeljive, već se stanje posmatranog dela sistema dobija kao zbir stanja pojedinih objekata. Objekti unutar objektno-dizajniranog softvera u principu se izvršavaju paralelno. To daje mogućnost da se softver distribuira na više računara, ili da se istovremeni rad objekata simulira na jednom računaru.

Objektno-orijentisane metode za razvoj softvera donekle se razlikuju u detaljima. Ipak, sve se one slažu da je unutar objektnog dizajniranja potrebno izvršiti barem sledeće korake:

- Utvrđivanje konteksta (okoline) za posmatrani deo softvera, te interakcije softvera s njegovom okolinom.
- Dizajn na najvišem nivou, tj. podela posmatranog dela softvera na još manje sastavne delove.
- Identifikacija svih klasa koje su potrebne za realizaciju predviđenih delova.
- Razvoj rafiniranih modela strukture i ponašanja za pojedine delove ili klase.
- Detaljna specifikacija interfejsa svih delova ili klasa.

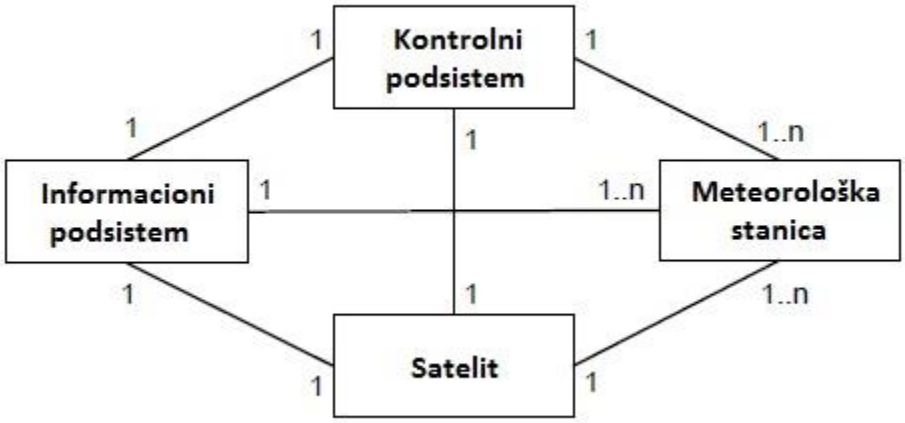
Ti koraci su u priličnoj meri isprepleteni i teško ih je obavljati u nekom fiksiranom redosledu, pa se stoga moraju ponavljati u više iteracija. Svi se oni opisuju pomoću odgovarajućih UML dijagrama (slučaja upotrebe, sekvence, klase, itd.).

Posmatraćemo primer meteorološke stanice, tj. pristup problemu kreiranja softvera za nju. Ta stanica nalazi se negde u prirodi i služi za skupljanje meteoroloških podataka. Postoji veći broj meteoroloških stanica i one su deo većeg sistema koji takođe sadrži središnji meteorološki informacioni podsistem i centralni kontrolni podsistem. Zadatak meteorološkog informacionog podsistema je da skuplja i obrađuje podatke iz svih meteoroloških stanica. Zadatak kontrolnog podsistema je da prati stanje svih meteoroloških stanica, te da ih po potrebi resetuje, pali, gasi i rekonfiguriše. Komunikacija između meteorološke stanice i meteorološkog informacionog podsistema odnosno kontrolnog podsistema odvija se radio vezom preko satelita.

Prvi korak zahteva da se odredi kontekst (okolina) za taj softver, te njegova interakcija s okolinom. Kontekst je vidljiv iz UML dijagrama klase na slici 5.1.

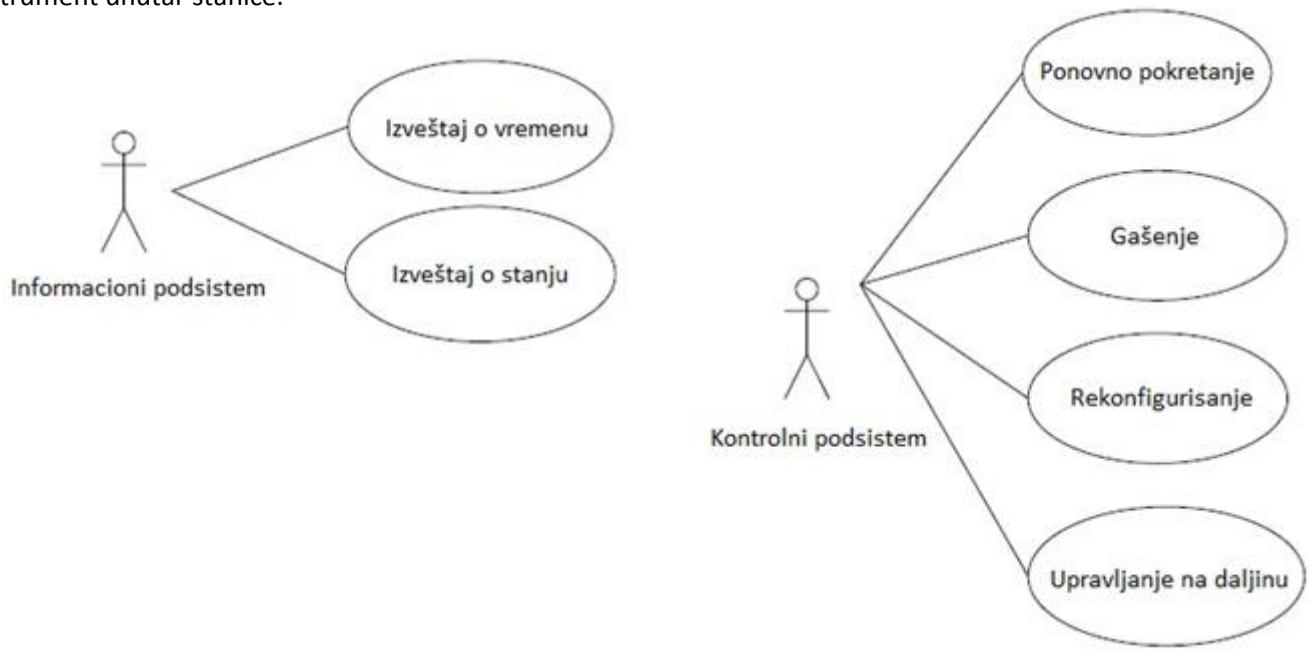


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES



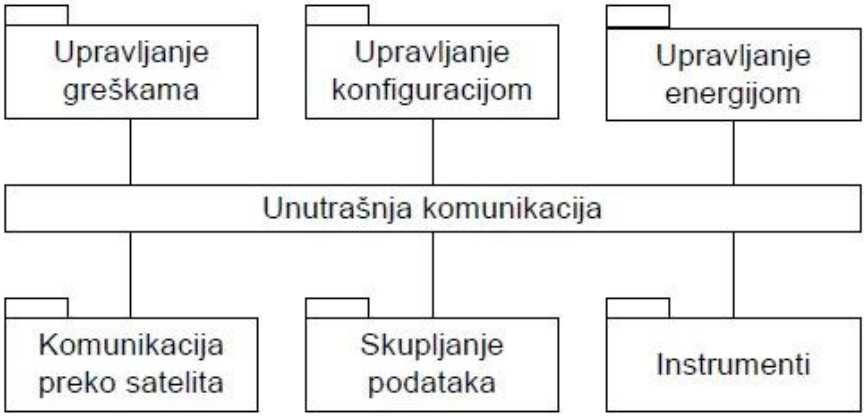

*Slika 5.1. UML dijagram klase koji opisuje kontekst meteorološke stanice*

Vidimo da je naša meteorološka stanica zapravo podsistem celog sistema. U okolini meteorološke stanice nalaze se još dva podsistema: informacioni i kontrolni. Četvrti podsistem je satelit koji omogućava međusobnu komunikaciju ostalih podsistema. Iz kardinalnosti veza vidimo da postoji samo jedan informacioni podsistem, jedan kontrolni podsistem i jedan satelit, no meteoroloških stanica ima mnogo. Interakcija meteorološke stanice s okolinom vidljiva je na UML dijagramu slučaja upotrebe na slici 8.2. Vidimo da se kao akteri pojavljuju dva vanjska podsistema: informacioni i kontrolni. Informacioni podsistem može, na primer, da od stanice zahteva da mu pošalje izveštaj sa sažetim vremenskim podacima. Kontrolni podsistem može, na primer, da resetuje određeni instrument unutar stanice.



*Slika 5.2. UML dijagram slučaja upotrebe koji opisuje interakcije meteorološke stanice s okolinom*

Drugi korak dizajniranja zahteva da se softver oblikuje na najvišem nivou, tj. da se odrede njegove komponente. U tu svrhu se koristi UML dijagram paketa (slika 5.3). Vidi se da se meteorološka stanica sastoji od nekoliko delova koji međusobno komuniciraju preko unutrašnje komunikacijske veze.

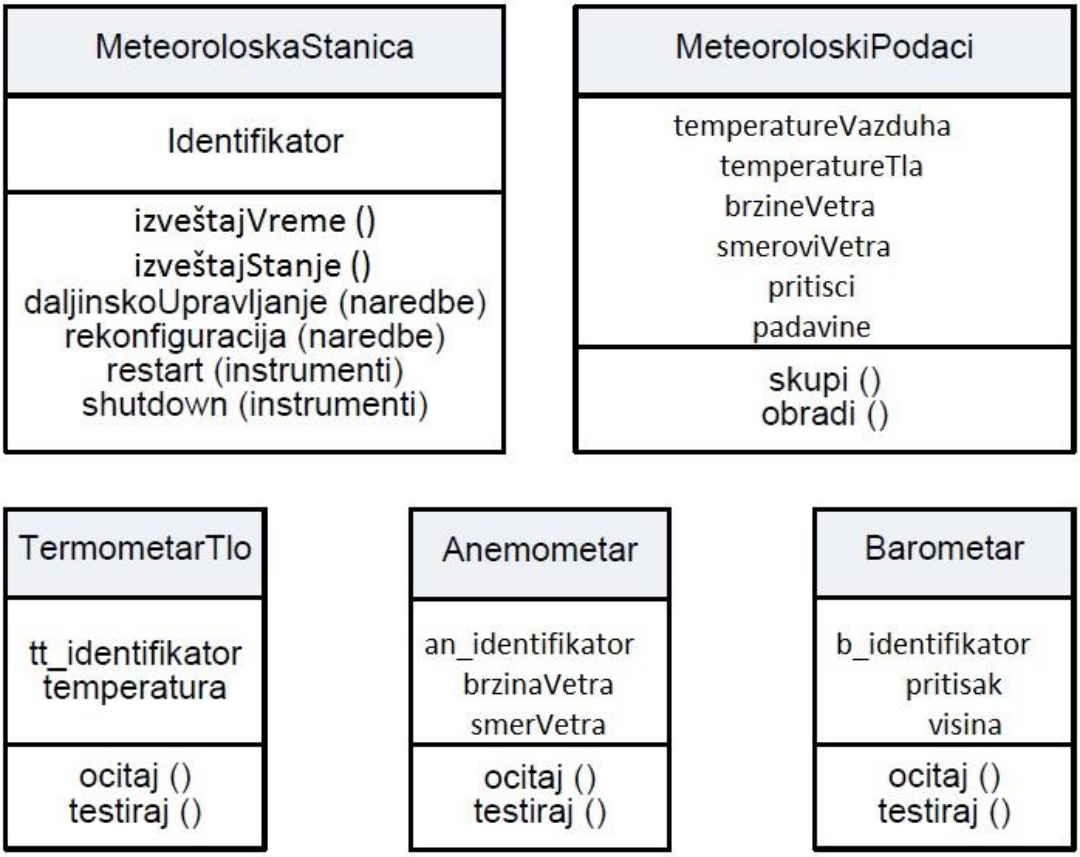



*Slika 5.3. UML dijagram paketa koji opisuje građu meteorološke stanice na najvišem nivou*

U trećem koraku, potrebno je definisati klase. Meteorološka stanica zahteva veliki broj klasa. Slika 5.4 prikazuje samo neke od njih. Vidimo da neke klase odgovaraju mernim instrumentima. Objekt iz klase MeteoroloskiPodaci služi za skupljanje izmerenih podataka pre njihovog slanja u informacioni podsistem. Objekt iz klase MeteoroloskaStanica predstavlja celu meteorološku stanicu i implementira interfejs stanice prema okolini.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

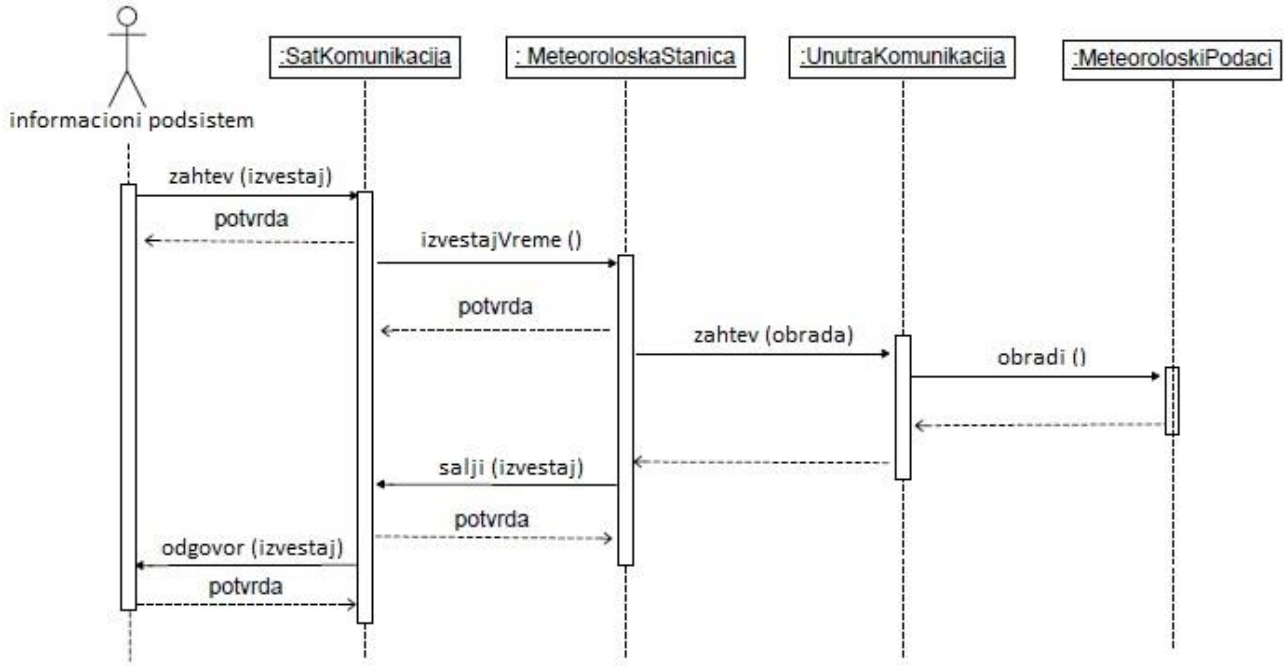



*Slika 5.4. Neke od klasa za meteorološku stanicu*

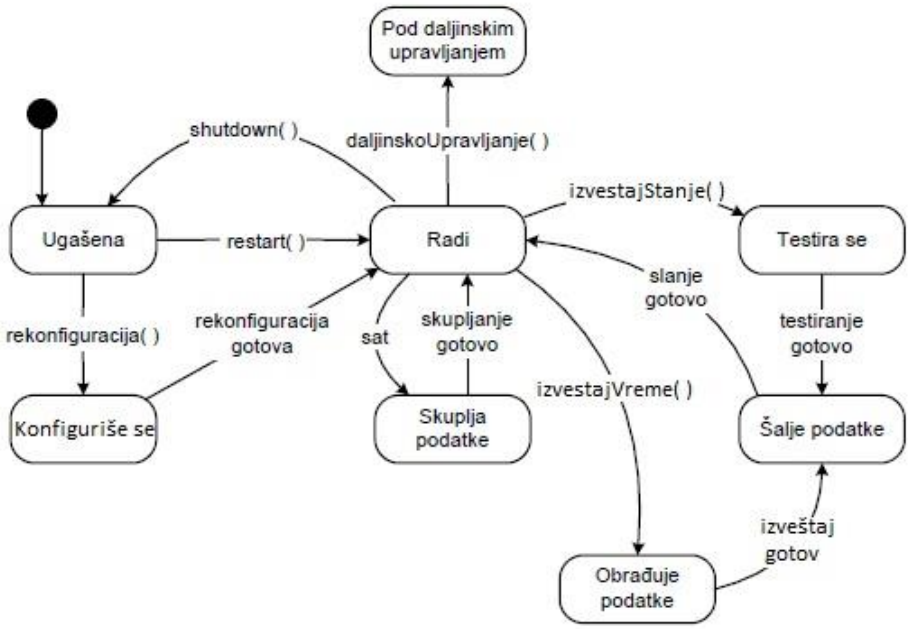
Četvrti korak dizajniranja sastoji se od razrade rafiniranih modela strukture i ponašanja. Takvih modela može biti mnogo, a ovde su priložena samo dva. Slika 5.5 sadrži UML dijagram sekvence koji opisuje realizaciju slučaja korišćenja „Izveštaj o vremenu“, tj. pokazuje kako meteorološka stanica šalje podatke informacionom podsistemu. Na dijagramu se pojavljuju objekti iz klasa s prethodne slike, ali takođe i dva nova objekta: prvi od njih enkapsulira komunikaciju stanice sa spoljnim svetom preko satelita, dok drugi enkapsulira internu komunikaciju unutar stanice. Slika 5.6 prikazuje UML dijagram stanja za meteorološku stanicu. Vidimo stanja u kojima se stanica može naći te prelaze iz stanja u stanje u zavisnosti od impulsima od spolja ili o unutrašnjim događajima.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Slika 5.5. UML dijagram sekvence koji opisuje slanje podataka iz meteorološke stanice u informacioni podsistem*

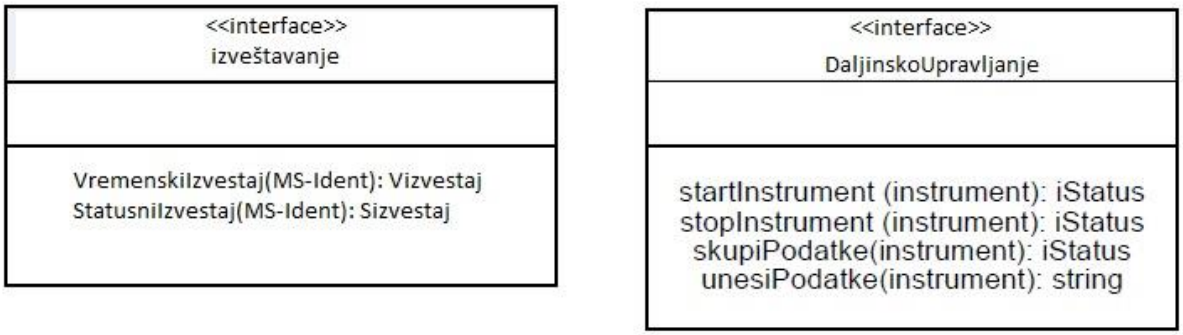



*Slika 5.6. UML dijagram stanja i prelaza za meteorološku stanicu*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Ilustracija poslednjeg koraka u dizajniranju, vidi se na slici 5.7 na kojoj su definisana dva interfejsa za meteorološku stanicu. Prvi interfejs tiče se izveštavanja, tj. slanja izmerenih podataka i statusa u informacioni podsistem. Drugi interfejs služi za udaljeno upravljanje stanicom preko satelita. Primetimo da između interfejsa i klasa ne mora postojati odnos „jedan-prema-jedan“. To je posebno naglašeno u programskom jeziku Java gde se intefejsi mogu definisati nezavisno od klasa, a razne klase mogu implementirati isti interfejs. U ovom primeru, operacije iz prvog interfejsa direktno se preslikavaju na odgovarajuće operacije iz klase MeteoroloskaStanica. Ali četiri operacije iz drugog interfejsa realizuju se različitim komandnim nizovima znakova koji se šalju jednoj te istoj operaciji DaljinskoUpravljanje() iz klase MeteoroloskaStanica.




*Slika 5.7. Interfejsi za meteorološku stanicu*

Korisnički interfejs svakako predstavlja jako bitan deo sistema jer je jedini vidljiv spoljašnjim korisnicima. Pojam komunikacija između korisnika i sistema obuhvata praktično sve: od startovanja ili prijavljivanja na sistem, preko podešavanja opcija, do dobijanja željenih informacija ili postizanja željenog cilja. Korisnički interfejs se sastoji iz delova namenjenih vođenju korisnika kroz sistem, ekrana i formi koje sadrže neke informacije za unos i prikupljanje podataka i iz izveštaja koje sistem treba da proizvodi.

Dizajniranje korisničkog interfejsa je proces definisanja načina na koji bi sistem trebalo da komunicira sa spoljašnjim entitetima, uključujući klijente, korisnike i druge sisteme. Sastoji se od tri bitna dela. Prvi je mehanizam za navigaciju, sačinjen od raznih menija i dugmića koji omogućavaju korisniku da se kreće po sistemu. Drugi je ulazni mehanizam i on predstavlja način za prihvatanje informacija dobijenih od korisnika, i sastoji se iz raznih formulara. Treći bitan deo je izlazni mehanizam. On obuhvata sve izveštaje koje sistem treba da predstavi korisnicima ili nekom drugom sistemu.

Dizajniranje korisničkog interfejsa zahteva dobro razumevanje potreba korisnika. Postoji nekoliko faza i procesa tokom dizajna, čija se bitnost razlikuje u zavisnosti od samog projekta:

- **Prikupljanje zahteva:** kreiranje lista funkcionalnosti koje sistem treba da ispunjava da bi postigao ciljeve i zadovoljio potrebe korisnika;
- **Analiza korisnika:** analiziraju se potencijalni korisnici sistema kroz razgovor i pitanja;
- **Arhitektura informacija:** informacije se struktuiraju prema kontekstu u interakciji s korisnicima ili većim bazama podataka;
- **Dizajn prototipova:** obuhvata izradu prototipa za sve važne delove sistema, uključujući navigaciju, ulazne i izlazne elemente, sve forme i izveštaje. Prototipovanje predstavlja razvoj:
  - **vajerfrejmova (*wireframe*)** – predstavlja skicu sajta sa linkovima koji rade, ali bez kompletnih grafičkih detalja. Služi da grupiše sadržaj, prikaže strukturu sajte i u nekim slučaevima vizuelizuje interakciju sa korinsikom
  - **Mokapova (Mockups) –** predstavlja statičku prezentaciju korisničkog interfejsa. Najčešće je to vizuelna skica korisničkog interfejsa sa skoro kompletnim detaljima.
  - **Prototipova –** simulira korisnički interfejs u potpunosti sa ograničenim skupom funkcionalnosti i često lažnim sadržajem.
- **Ispitivanje korisnosti:** dozvoljava se evaluatoru da ispita interfejs. Ovo je jeftinije od testiranja korisnosti i može se primeniti u ranoj fazi razvojnog procesa, jer se mogu evaluirati prototipovi ili specifikacije sistema koje se ne mogu testirati na korisnicima. Među najčešće metode ubrajaju se kognitivni prolazak (usredsređuje se na jednostavnost izvršenja zadataka za nove korisnike), heuristička evaluacija (koristi se skup heuristika da bi se identifikovali problemi korisnosti) i pluralistički prolazak (odabrana grupa ljudi prolazi kroz korak scenarija i diskutuje o problemima korisnosti);
- **Testiranje korisnosti:** testiraju se prototipovi na stvarnim korisnicima, najčešće korišćenjem tehnike „protokol glasnog razmišljanja“ gde se od korisnika traži da glasno iznosi svoje misli tokom iskustva korišćenja interfejsa;
- **Dizajniranje grafičkog interfejsa:** na osnovu zapažanja tokom testiranja korisnosti i drugih ciljeva, kreira se stvarni izgled interfejsa. Ukoliko interfejs zahteva više „skinova“, može da postoji više dizajna za jedan kontrolni panel. Ova faza je često predmet saradnje između grafičkog dizajnera i dizajnera korisničkog interfejsa.

U svakom slučaju, važno pitanje u procesu razvoja složenih softverskih sistema je njihova arhitektura, odnosno, organizacija kolekcije komponenti koje međusobno komuniciraju. Dobra arhitektura može pomoći da sistem zadovolji ključne zahteve na poljima performansi, pouzdanosti, prenosivosti i skalabilnosti. Arhitektura softvera igra ulogu mosta između specifikacije zahteva i implementacije. Obezbeđujući apstraktan opis sistema ona ističe određena svojstva sistema istovremeno skrivajući druga. Tokom poslednje decenije arhitekturi softvera, kao polju softverskog inženjerstva, posvećivana je sve veća pažnja, jer su stručnjaci shvatili da je pravilan izbor arhitekture kritičan faktor uspeha projektovanja i razvoja softverskih proizvoda.

# 6. Klasične i agilne metode razvoja softvera

Rad softverskog tima obično je organizovan u skladu s nekom metodom razvoja softvera. Svaka od metoda sledi jedan od modela za softverski proces ili kombinuje više modela. Osnovne aktivnosti iz modela podeljene su u manje aktivnosti. Propisuje se način odvijanja svake aktivnosti te način njenog dokumentovanja. U sledećim odeljcima biće navedena osnovna svojstva pojedinih vrsta metoda te detaljnije opisane danas aktualne metode: XP, SCRUM i KANBAN.

Od početka softverskog inženjerstva do danas pojavilo se nekoliko stotina metoda za razvoj softvera. One se grubo mogu podeliti u klasične i agilne metode.

*Klasične* metode pojavile su se još u 70im godinama 20. veka, a razvijaju se i danas. One se odlikuju sledećim svojstvima.

- Razvoj softvera se posmatra kao pažljivo planirani proces koji ima svoj početak i kraj. U tom procesu u potpunosti se utvrđuju zahtevi te se softver realizuje u skladu s njima.
- Sprovodi se upravljanje projektom. Određuje se precizni plan aktivnosti, po potrebi i za duži vremenski period unapred. Određuje se raspored ljudi i resursa po aktivnostima.
- Insistira se na urednom dokumentovanju svih aktivnosti i svih proizvedenih delova softvera.

Novije *objektno-orijentisane* metode predstavljaju nadgradnju objektno-orijentisanih programskih jezika (Java, C++, C#) i danas su se integrisale u zajednički standard *UP* (*Unified Process* – Booch, Rumbaugh, Jacobson – 90-e godine) koji se oslanja na grafički jezik *UML* (*Unified Modelling Language*).

Kao protivteža klasičnim metodama, početkom 21. veka pojavile su se *agilne* metode. Njihova glavna namera bila je ubrzati razvoj softvera i izbeći nepotrebnu administraciju. Agilne metode imaju sledeća svojstva.

- Razvoj softvera se posmatra kao kontinuirani niz iteracija čiji broj nije moguće predvideti. Svaka iteracija usklađuje sistem s trenutnim zahtevima, koji se stalno menjaju, što će reći da ne postoji celovita ili konačna specifikacija.
- Ne postoji upravljanje projektom u pravom smislu reči. Aktivnosti se dogovaraju s korisnicima te se odvijaju uz njihovo aktivno učešće. Planiranje je kratkoročno – jednu ili dve nedelje unapred.
- Izbegavaju se svi oblici dokumentacije osim onih koji se mogu automatski generisati iz programa. Smatra se da sam program u izvornom obliku predstavlja svoju najpouzdaniju dokumentaciju.

Najpopularnija agilna metoda je *Extreme Programming – XP*. Takođe je poznata metoda *Scrum* koja se uglavnom bavi agilnim vođenjem projekta i može se kombinovati s XP. Ima i pokušaja da se definiše agilna varijanta inače klasične metode UP.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

U literaturi su zabeležena brojna istraživanja koja mere i upoređuju učinkovitost pojedinih metoda za razvoj softvera. Prema tim istraživanjima, agilne metode imaju svojih prednosti kad je reč o malim i kratkoročnim projektima u brzo promenljivom poslovnom okruženju. No nije se još dokazalo da se agilne metode mogu uspešno skalirati na velike projekte. Zato se danas veliki softverski sistemi i dalje razvijaju klasičnim i to objektno-orijentisanim metodama. Agilne metode našle su svoje mesto na primer u razvoju web aplikacija ili u razvoju mobilnih aplikacija.

### 6.1 XP

Kao primer agilne metode razvoja softvera biće opisana metoda *Extreme Programming – XP*. Metoda je nastala oko 2000. godine i pripisuje se autoru Kentu Beku. Naziv je odabran zato što XP ideju iterativnog razvoja softvera s naglaskom na programiranje tera do ekstremnih razmera.

Razvojni proces u skladu s XP prikazan je na slici 9.1. Proces se sastoji od kontinuiranog niza vrlo brzih iteracija. Svaka iteracija stvara novo izdanje sistema koje se isporučuje korisniku. Vremenski razmak između dve isporuke nije veći od dve nedelje.

*Slika 6.1.1. Jedna iteracija metode XP*

Novo izdanje sistema može da sadrži novu funkcionalnost – tada XP podseća na model inkrementalnog razvoja za softverski proces. No moguće je da novo izdanje samo popravlja ili menja već postojeću funkcionalnost – tada XP liči na model evolucionog razvoja. Dakle, XP kombinuje inkrementalni i evolucioni model. Ipak, naglasak je na izuzetno kratkim i brzim iteracijama.

Zahtevi u XP prikazuju se kao skup kratkih kartica teksta, tzv. *korisničkih priča* (*user stories*). Na početku iteracije biraju se one priče koje će se implementirati u dotičnom izdanju – uzima se onoliko priča koliko se može implementirati u dve nedelje i to one s najvišim prioritetom. Ostale priče se ostavljaju za buduća izdanja. Odabrane priče razgrađuju se u manje zadatke koji se raspoređuju



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

programerima u skladu s procenjenom složenošću. Opisani neformalni postupak planiranja novog izdanja naziva se *planning game*.

U XP-u ne može doći do kašnjenja isporuke – ono što se do predviđenog roka isporuke uspelo implementirati to se isporučuje, a ostalo se ostavlja za buduća izdanja. Naručilac programerima plaća utrošeno radno vreme. Nema ugovora koji bi vezao plaćanje uz traženu funkcionalnost.

U metodi XP izuzetno važnu ulogu igraju korisnici. Postoji predstavnik korisnika koji je član razvojnog tima i stalno je dostupan ostalim članovima. On donosi korisničke priče, određuje prioritete, dogovara se o sadržaju idućeg izdanja, učestvuje u testiranju sistema. Takođe, predstavnik korisnika odlučuje šta će se uraditi s nerealizovanim pričama, te treba li neku od već implementiranih priča promeniti i ponovo implementirati.

Zanimljiva inovacija koju je XP doneo u programersku praksu naziva se *test-first development.* Pri realizaciji bilo kog programerskog zadatka najpre se sastavljaju testovi za dotični novi komad funkcionalnosti, a tek onda ta funkcionalnost ide na implementaciju. Čim je novi programski kod napisan, on se odmah testira. Da bi se ovakav način rada bio efikasan, nužan je odgovarajući CASE-alat koji omogućava lagano zadavanje, pamćenje, te automatsko izvršavanje testova.

Testovi dobijeni primenom *test-first developmenta* trajno se pamte. Kad god se stvori nova verzija softvera, nad njom se osim najnovijih testova takođe moraju izvršiti i svi pređašnji testovi. Time se osigurava da najnovije promene nisu narušile staru funkcionalnost.

Druga inovacija koju je XP doneo u programersku praksu je *programiranje u paru* (*pair programming*). Dakle, jedan zadatak ne obavlja jedan programer već dvojica – oni sede za istom radnom stanicom i zajedno pišu isti programski kod. Prednost takvog načina rada je da dvojica saradnika u zajedničkoj diskusiji lakše dolaze do dobrog rešenja te jedan drugom ispravljaju greške. Parovi nisu fiksirani već se menjaju od zadatka do zadatka. Time se postiže *kolektivno vlasništvo* nad programskim kodom, gde su svi upućeni u sve delove programa i sve mogu promeniti.

Još jedna osobina XP-a je da se softver projektuje tako da podrži trenutne zahteve i ništa više od toga. Takvo projektovanje je u suprotnosti s principima tradicionalnog softverskog inženjerstva gde se softver nastoji izgraditi dovoljno uopšteno da bi se u budućnosti lako mogao menjati. Pobornici XP smatraju da se promene ionako ne mogu predvideti, pa se ne isplati o njima brinuti unapred.

Opasnost kod bilo koje vrste iterativnog razvoja softvera je da se struktura softvera postepeno degradira usled stalnih promena. XP se bori protiv te opasnosti povremenim reorganizovanjem programskog koda, takozvanim *refactoringom*. Dakle, kad programer primeti da se struktura programa isuviše narušila, on je popravlja tako da pojednostavi ili premesti neke delove koda, eliminiše ponavljanje sličnih delova, i tako dalje.

Metoda XP ne propisuje nikakve oblike dokumentacije. Naime, prema pobornicima XP, jednostavan i pregledno napisani programski kod sam sebe najbolje dokumentuje. Štaviše, iz tog koda moguće je automatski generisati razne izveštaje i dijagrame kakvi se inače koriste u dokumentima. Ovakav



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

pristup je realističan u situaciji kad se nove verzije softvera proizvode gotovo svakodnevno – ažuriranje zasebne dokumentacije predstavljalo bi preveliki teret i bilo bi podložno greškama.

Iz svega izloženog vidimo da XP ima sve osobine agilne metode za razvoj softvera. Brojni softverski timovi u celosti su prihvatili XP i pokazali da agilni način rada može biti efikasan barem kod manjih projekata s vrlo promenljivim zahtevima. Ipak, još je veći broj onih timova koji su izabrali samo neko od XP-ovih svojstava, na primer *test-first development* ili programiranje u paru te ih uklopili u neku klasičnu metodu.

Metodologija se temelji na pet osnovnih vrednosti:

1. **Komunikacija:** XP podstiče i praktikuje usmenu komunikaciju s klijentima i razvojnim programerima. Vrednost komunikacije se temelji na primedbi da većina poteškoća u projektu nastane iz razloga što neko u timu nije razgovarao da bi razjasnio pitanje, sarađivao ili potražio pomoć.
2. **Jednostavnost:** dizajniranje jednostavnijeg proizvoda koji zadovoljava potrebe klijenta. Važan aspekt ove vrednosti je da se dizajniraju i programiraju samo trenutni zahtevi klijenta umesto predviđanja i planiranja nenavedenih zahteva.
3. **Povratni odgovor:** razvojni tim dobija povratni odgovor od kupca na kraju svake iteracije. Ovaj povratni odgovor je vodilja za sledeću iteraciju. Osim toga, postoji vrlo kratak povratni odgovor o dizajnu i implementaciji ugrađenim u metodologiju putem programiranja u paru i razvoja vođenog testovima (eng. *test-driven development*)
4. **Hrabrost:** navedene tri vrednosti dozvoljavaju timu da bude hrabar u njegovim akcijama i donošenju odluka.
5. **Poštovanje:** članovi tima treba da brinu jedan o drugom i o projektu.

Početna verzija XP metodologije razvoja programskog proizvoda objavljena 2000. godine sastojala se od 12 tehničkih veština orijentisanih na programera. 2005. godine, XP je promenjen na način da sadrži 13 osnovnih praktičnih tehnika i 11 posledičnih praktičnih tehnika. Osnovne veštine su namenjene da budu korisne nezavisno jedna od druge, te su opisane na sledeći način:

1. **Sedenje zajedno:** celi tim razvija u jednom prostoru.
2. **Celi tim,** koristi sve funkcionalnosti ekipe u svrhu uspeha proizvoda.
3. **Informativni radni prostor:** vidljivi zidni grafovi po celom radnom prostoru tako da članovi ekipe i drugi zainteresovani posmatrači mogu da vide u kom smeru ide projekt.
4. **Radna energija – 40-satna radna nedelja:** XP timovi ne rade prekovremeno za duži vremenski period. Cilj ove prakse je održati visok kvalitet koda jer umoran programer proizvodi više grešaka.
5. **Programiranje u paru:** [25] odnosi se na praksu gde dva programera rade zajedno na jednom računaru sarađujući u dizajniranju, kodiranju i testiranju.
6. **Priče:** tim piše kratke izjave klijenta, odnosno vidljive funkcionalnosti u proizvodu. Razvojni programeri procenjuju korisničku priču, a korisnik ih prioritizuje.
7. **Nedeljni ciklus:** početkom svake nedelje održava se sastanak da bi se utvrdio napredak na projektu i da bi klijent odabrao korisničke priče koje će se pretvoriti u zadatke kojima je cilj završetak do kraja nedelje. Na kraju nedelje se vrši test prihvatljivosti od strane kupca.
8. **Tromesečni ciklus:** celi tim odabere teme korisničkih priča za tromesečje. Navedene teme pomažu timu da se upozna sa širom slikom celokupnog projekta.

9. **Labavost:** (eng. *slack*) u svakoj iteraciji se planiraju zadaci nižeg prioriteta koji se mogu odbaciti ukoliko ih tim ne stigne odraditi na vreme.
10. **Desetominutna izgradnja projekta (eng. *build)*:** struktuirati projekt i njegove povezane testove tako da celi sistem može biti izgrađen i svi testovi mogu biti izvršeni u roku od 10 minuta.
11. **Programiranje gde se prvo izvode testovi:** (eng. *test-first programming*) sve korisničke priče imaju barem jedan test prihvatljivosti koji je (poželjno) automatizovan. Kada je test (ili testovi) prihvatljivosti izvršen bez grešaka za sve korisničke priče tada je korisnička priča završena.
12. **Kontinuirana integracija:** programeri spremaju (eng. *upload*) kompletirani kod na repozitorijum i testove koji su vezani za njega nekoliko puta dnevno. Kod se može skladištiti jedino u slučaju da su svi testovi modula (eng. *unit tests*) izvršeni bez grešaka.
13. **Inkrementalni dizajn:** umesto da se razvija preuranjen detaljni dizajn pre implementacije, u dizajn se ulaže svaki dan prema iskustvima iz prošlosti.

Posledične tehničke veštine su opisane:

1. **Stvarna uključenost klijenta:** klijent je uvek dostupan da bi razjasnio pitanja u vezi sa zahtevom te ih prioritizuje.
2. **Inkrementalni razvoj (eng. *incremental deployment*)**: postupno implementiranje funkcionalnosti uživo okruženje kako bi se izbegao rizik velikih implementacija
3. **Kontinuitet ekipe**: zadržati učinkovite ekipe zajedno
4. **Skupljanje tima:** kako raste kapacitet tima (prema iskustvu), treba zadržati njihovo radno opterećenje, ali postepeno smanjivati tim
5. **Analiza korena uzroka:** istražiti uzrok otkrivene greške pisajući testove prihvatljivosti i testove modula. Nakon toga, istražiti zašto je greška napravljena, a nije pronađena u procesu razvoja.
6. **Deljenje koda:** jednom kada se kod snimi na repozitorijum zajedno s povezanim testovima, tada je dostupan svim članovima tima.
7. **Kodiranje i testiranje:** u projektu održavati samo kod i testove.
8. **Dnevna implementacija:** implementiranje novog koda u produkciju svako veče
9. **Pregovarački opseg ugovora:** popraviti vreme, troškove i zahtevani kvalitet projekta, ali pregovarati o opsegu projekta.
10. **Plaćanje po korišćenju:** (eng. *pay-per-use*) naplatiti korisniku svaki put kada se sistem koristi za dobijanje njihove povratne informacije.
11. **Stojeći sastanci:** (eng. *Stand-up meeting*) iako nije službena XP praksa, uobičajeno svi XP timovi imaju kratke stojeće sastanke. Na tim sastancima osobe stoje u krug, a namerno na nogama kako bi sastanak trajao kratko. Svaki član ekipe tada izveštava ostale:
    - Šta je učio prethodnog dana
    - Šta planira da odradi danas
    - Ima li ikakvih prepreka ili poteškoća

Metodologija ekstremnog programiranja nije dobra kada postoji otpor ovakvom načinu programiranja, u velikim timovima i nefleksibilnim okolinama, tamo gde je potrebna opsežna dokumentacija i gde se povratne informacije dugo čekaju, te u okolini gde je prekovremeni rad uobičajen ili ljudi u timu ne žele (ili ne mogu) da komuniciraju.

### 6.2 SCRUM

Termin **Scrum** potiče iz ragbija, a znači „vraćanje lopte koja je izašla iz igre nazad u igru“ (uz pomoć timskog rada). Radi se o brzom, prilagodljivom samoorganiziranom procesu razvoja programskog proizvoda. Ovaj pristup koji povećava brzinu i fleksibilnost u razvoju programskog proizvoda, opisali su 1986. godine Hirotaka Takeuči i Ikuđiro Nonaka. Današnji *Scrum* zacrtali su početkom ’90-tih Ken Švaber i Džef Saterlend. Proces razvoja programskog proizvoda sprovodi se kroz tri faze:

- Pre igre (planiranje, dizajn/arhitektura visoki nivo apstrakcije)
- Igra (razvoj, sprintovi – iterativni ciklusi, poboljšanja, nove verzije)
- Posle igre (nema novih zahteva, sistem spreman za produkciju).

Razvoj projekta u Scrumu napreduje iterativno gde svaka iteracija (*sprint*) traje dve do četiri nedelje, ali to trajanje mora biti fiksno i unapred definisano, tokom kog se izrađuje verzija proizvoda spremna za isporuku dok celokupni projekt (*jedna igra*) traje tri do osam sprintova.

Postoje tri glavna dokumenta koje generiše Scrum tim: **zaliha proizvoda** (eng. *product backlog*), **zaliha sprinta** odnosno trenutne iteracije (eng. *sprint backlog*) i **dijagram preostalog posla** u Sprintu (eng. *Sprint burndown graph*). Svaki od njih je dostupan i vidljiv Scrum timu.

- **Zaliha proizvoda** (eng. ***product backlog***): evoluirajući, prioritizovani red poslovnih i tehničkih funkcionalnosti koje treba implementirati u sistem, kao i skup grešaka i nedostataka koje treba popraviti tokom razvoja programskog proizvoda. Za svaki zahtev u zalihi proizvoda postoji jedinstveni identifikator za zahtev, kategoriju (osobinu, poboljšanje, nedostatak (greška)), status, prioritet, procenu koliko je vremena potrebno za implementaciju. Zaliha je organizovana i čuva se u obliku tablica, ali je isto tako promenljiva na više načina. Jedan je od njih uočavanje novih funkcionalnosti koje bi proizvod trebalo da ima, a koje možda nisu bile očigledne ili dovoljno jasne u trenutku kada je projekat započeo. Drugi je u slučaju da neka od funkcionalnosti iz trenutne zalihe sprinta neće moći da se implementira za vreme trenutnog sprinta (npr. ako neki spoljni uslovi još nisu zadovoljeni za tu funkcionalnost ili jednostavno nema dovoljno vremena). U takvim situacijama zadatak mora da se vrati u zalihu proizvoda jer produžavanje sprinta izvan inicijalno zadatih vremenskih granica nije dopušteno.
- **Zaliha sprinta (zaliha trenutne iteracije):** lista svih poslovnih i tehnoloških osobina, poboljšanja i nedostataka koji su raspoređeni u trenutnu iteraciju (*sprint*). Zalihom sprinta se takođe upravlja u formatu tablica. Zahtevi su pretvoreni u zadatke i za svaki od njih postoji kratak opis, kako je nastao, ko je vlasnik zadatka, status i vreme koje je potrebno da se zadatak završi. Zaliha sprinta se svakodnevno ažurira vremenom koje je preostalo do završetka zadatka. Procena se može povećati kada član tima shvati da je posao na pojedinom zadatku potcenjen.
- **Dijagram preostalog posla u Sprintu (eng. *Sprint burndown graph*)**: prikazuje koliko je sati preostalo da bi se završili sprint zadaci, te je uglavnom prikazan Scrum timu. Primer grafikona je prikazan na slici:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

*Slika 6.2.1. Dijagram preostalog posla u Sprintu*

Uobičajeno je Scrum tim smešten zajedno, međutim, postoje Scrum timovi koji su geografski razmešteni i gde članovi tima učestvuju na dnevnim sastancima preko konferencijske veze. Scrum timovi su samousmereni i samoorganizovani. Tim se obavezuje da će izvršiti zadate ciljeve u jednoj iteraciji i da mu je data autonomija i odgovornost da odluči kako će do toga doći. Uloge koje su definirane Scrumom su:

- **Vlasnik proizvoda**: (eng. *product owner*) odgovoran za izradu i prioritizovanje zalihe proizvoda, odabire šta će biti uključeno u sledeću iteraciju (sprint) i pregledava sistem po završetku sprinta.
- **Scrum master:** (eng. *scrum master*) zna i pojačava cilj iteracije, održava dnevne sastanke (Scrum sastanci) i demonstraciju iteracije (Sprint pregled), prati napredak, uklanja prepreke i osigurava resurse. Scrum master je takođe razvojni programer i učestvuje u razvoju proizvoda, odnosno nije samo menadžer.
- **Razvojni programer:** član Scrum tima koji se zalaže za postizanje cilja sprinta i ima sva ovlašćenja da učini štagod bude potrebno da bi to postigao. Scrum tim se najčešće sastoji od 7 članova, dok je minimum 5 a maksimum 9.

Scrum proces se sastoji od nekoliko faza. Prva je planiranje sledeće iteracije (eng. ***Sprint planning***). Radi se o sastanku koji se održava s razvojnim timom, menadžerima i vlasnikom proizvoda koji predstavlja kupca. Vlasnik proizvoda izrađuje i prioritizuje zalihu proizvoda, a na sastanku planiranja bira svojstva koja će uključiti u sledeći 30-dnevni sprint vodeći se najvećom poslovnom vrednošću i rizikom. Cilj sprinta je uspostavljen i održava Scrum tim fokusiranim na široku sliku projekta, a ne samo na odabrano svojstvo. Razvojni tim shvata zadatke i zahtevane resurse da bi isporučili svojstva programskog proizvoda. Na sastanku se zajedno određuje razuman broj svojstava koja će biti uključena u sledeći sprint. U trenutku kada je skup svojstava određen ne može se raditi ponovna



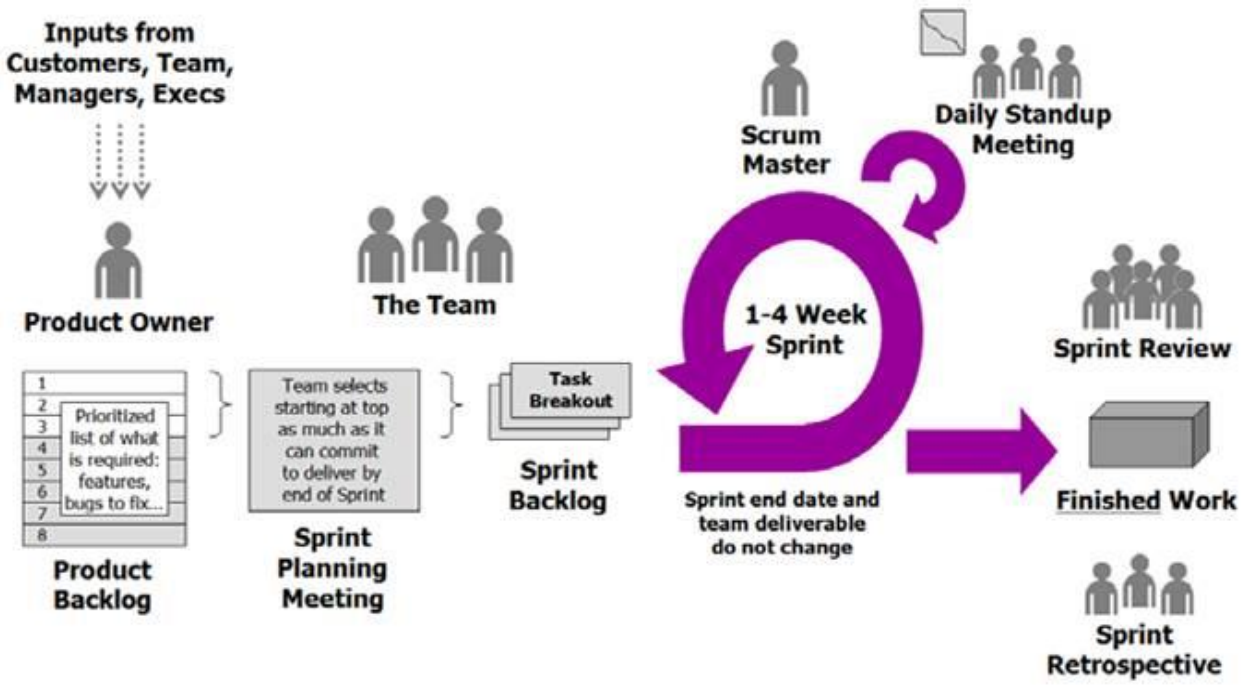
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

prioritizacija svojstava u sledećih 30 dana do kraja sprinta u kom će se svojstva dizajnirati, implementirati i testirati.

Sprint se planira tako da ima dovoljno vremena za implementaciju zadataka koji se stave u zalihu sprinta te je ovde jako bitna procena razvojnog tima. Zadaci koji se nađu u zalihi sprinta bi trebalo da bude detaljnije razrađeni od onih iz zalihe proizvoda, a procenjena količina posla po zadatku ne bi trebalo da prelazi šesnaest sati rada. Zaliha sprinta je nepromenljiva te u nju nije dozvoljeno dodavanje nove aktivnosti iz zalihe proizvoda jednom kada je sprint počeo čime je timu poverena odgovornost za procenu toga što se može načiniti i u kom vremenu, a pred sprintom se neće naći neočekivane prepreke. Tako su ljudi motivisaniji za rad, a bitne funkcionalnosti odabrane u fazi planiranja imaju veću verovatnoću da stvarno budu implementirane na vrijeme.

Svakodnevno se tokom sprinta programski kod integriše, te se nad njim vrše integracioni testovi. Takođe se svakodnevno održavaju 15-minutni sastanci (eng. ***Scrum meeting***) koji su slični XP stojećim sastancima. Sastanak počinje uvek u isto vreme na istom mestu u stojećem položaju, a s obzirom na to da dugo stajanje na nogama nije prijatno, ovim se osigurava da se članovi tima u razgovoru drže bitnih tema. Sastanak se održava u sobi s pločom tako da se zadaci i nedostaci (blokeri) mogu na nju zapisati kako bi svima bilo vidljivo. Svi mogu prisustvovati Scrum sastanku, ali samo članovi tima i Scrum master mogu govoriti. Svaki član tima odgovara na sledeća pitanja:

- Šta je radio od posledenjeg Scrum sastanka?
- Šta će raditi od sada do idućeg Scrum sastanka?
- Ima li ikakvih problema ili blokera?




*Slika 6.2.2. Scrum proces*

Scrum sastanak je osnovna komponenta metodologije te se na njemu daju obećanja koja podižu odgovornost i održavaju da projekta ide u pravom smeru. Međutim, ovi sastanci mogu biti nemogući ukoliko na njima prisustvuje previše ljudi, a preporučeno je da svaki tim ima maksimalno sedam ljudi. Kada postoje veći timovi, oni se dele u manje grupe gde svaka od njih ima svoj Scrum sastanak, a nakon toga predstavnik svake skupine učestvuje na „Scrumu Scrumova“, sastanku gde odgovara na već navedena pitanja te se na ovaj način šire informacije među podtimovima.

Na kraju Sprinta se radi pregled Sprinta (eng. ***Sprint review***) s ciljem da se utvrdi napredak, demonstriraju svojstva klijentu, menadžeru, korisniku i vlasniku projekta te da se izvrši pregled projekta iz tehničke perspektive. Na ovom sastanku, koji vodi Scrum master, a vlasnik proizvoda i drugi zainteresovani mogu prisustvovati, demonstrira se poslednja verzija proizvoda. Dakle, prezentacija u smislu slajdova nije dozvoljena. Ciklus se ponavlja sa sastankom planiranja sprinta (eng. *Sprint Planning meeting*) na kom se opet biraju svojstva koja će se implementirati u sledećem sprintu.

Scrum je jednostavan, lako razumljiv i čvrst razvojni proces te je usredsređen na bitne funkcionalnosti koje je razvojni tim sam procenio da može da implementira. Komunikacijom na svakodnevnim stojećim sastancima rešavaju se tekući problemi i daju obećanja šta će se napraviti do idućeg sastanka čime se stvara poverenje među timom. Rezultat su zadovoljni članovi tima što za posledicu ima kvalitetniji i brži razvoj, a samim time su zadovoljni naručioci programskog proizvoda.

### 6.3 KANBAN

Kanban agilna metodologija razvoja programskog proizvoda prati razvoj proizvoda s minimizacijom gubitaka (eng. *Lean software development*) „Lean“ je proizvodna filozofija koja kada je implementirana skraćuje vreme od narudžbine klijenta do isporuke gotovog proizvoda, eliminišući sve izvore rasipanja (gubitaka) u proizvodnom procesu. Rasipanja (engl. *waste*) su elementi proizvodnog procesa koji ne sadrže nikakvu vrednost, tj. to su aktivnosti koje ne donose direktnu vrednost proizvodu.

Osnovno načelo „lean“ proizvodnje je da se proizvodi tačno ono što kupac ili klijent želi, tj. vrstu, kvalitet i količinu proizvoda neposredno diktira potražnja tržišta. „Lean“ se primenjuje zbog konstantno promenljivog tržišta, visokih zahteva kupaca i konkurencije te se njegovom primenom postiže sledeće:

- Skraćuje se ukupno vreme proizvodnje
- Povećava se radni učinak
- Redukuje se papirologija
- Pojednostavljuje se planiranje
- Povećava se angažovanost zaposlenih
- Poboljšava se kvalitet
- Dobija se brži odgovor na promene u zahtevima tržišta

Sve aktivnosti u lancu vrednosti mogu da se podele na aktivnosti koje dodaju vrednost, aktivnosti koje ne dodaju vrednost, ali su neophodne i aktivnosti koje ne dodaju vrednost i nisu neophodne. Unapređenje proizvodnog procesa leži u fokusiranju na poboljšanje aktivnosti koje dodaju vrednost, te u paralelnom smanjivanju vremena trajanja aktivnosti koje ne dodaju vrednost.

*Slika 6.3.1. Kanban*

U Japanskom rječniku „kan“ znači „signal“ dok „ban“ znači „kartica“ ili „ploča“. Kanban kartica je signal koji treba da pokrene akciju i koji upućuje na kretanje ili izradu delova u „pull“ proizvodnom sistemu, izumljenom i razvijenom 1950-ih kao deo Tojotinog proizvodnog sistema (eng. *Toyota Production System – TPS*). Prema tome Kanban se odnosi na „signalne kartice“ kojima se signalizira potreba za određenim proizvodom. Taiči Ono je dobio ideju za Kanban kada je bio u poseti američkom supermarketu gde se polica dopunjava kada se količina na njoj smanji na određenu meru (*pull* mehanizam). Kanban se u proizvodnoj industriji proširio po celom svetu kao alat proizvodnje s minimizacijom gubitaka (eng. *Lean Manufacturing*), a u agilnom razvoju programskog proizvoda, kao način vizualizacije projekta postavljanjem kartica sa zadacima na ploču kojima se postiže „upravo na vreme“ (eng. *just in time – JIT*) strategija razvoja programskog proizvoda.

Tok kartica implicira da postoji glatki neprekinuti protok između niza koraka koji uključuje sve od klijenta do podrške inženjera, a ne samo razvojnog tima. Kontinuirani glatki protok vrednosti novih svojstava u razvoju je idealan krajnji rezultat. Kroz merenje podataka je moguće proceniti timsku učinkovitost, tj. koliko se vremena troši na dodavanje, odnosno nedodavanje svojstava u programski



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

proizvod. Kanban razmišljanje s minimizacijom gubitaka se generalno daleko više fokusira na to da posao bude izveden na vreme umesto da se fokusira na to ko ga je odradio. Ljudi rade zajedno, ali ne rade svi istom brzinom, nemaju ista znanja i veštine, te se moraju sinhronizovati. U Kanbanu je rad organizovan po zadatku ili procesu, te dozvoljava članovima tima da sami primenjuju tok rada na najučinkovitiji način. Pravila Kanbana su takva da se ne razmatraju svojstva programskog proizvoda koje nisu potrebna odmah; ne piše se više specifikacija nego što se može isprogramirati; ne piše se više koda nego što se može testirati, te se ne testira više nego što se može pustiti u rad.

Kanban sistem u programskom inženjerstvu izgleda tako da postoji posao koji čeka u redu i prolazi kroz faze dok ne bude završen. Kada je posao u fazi završen, prelazi se na drugu fazu. Kada je nekom potreban posao, povlači ga iz prethodne faze. Prednosti Kanbana u odnosu na tradicionalne „push“ sisteme su:

1. Jednostavan i razumljiv process
2. Pruža brze i precizne informacije
3. Niski troškovi prenosa informacija
4. Brz odgovor na promene
5. Ograničenje kapaciteta u procesu
6. Izbegavanje hiperprodukciju
7. Minimizovanje rasipanja
8. Mogućnost održavanja kontrole
9. Delegiranje odgovornosti radnicima

Osnovna načela Kanbana u programskom inženjerstvu, a koja tim kontinuirano prati su:

- Vizualna kontrola (Kanban ploča)
- Ograničenje posla u procesu (eng. *Limit Work in Process – WIP*)
- Povlačenje (eng. *pull*) kartica (s ograničenjem WIP)
- Povećanje propusnosti
- Fiksna Kanban zaliha (eng. *backlog*)
- Kvalitet ugrađen u proces

Korištenje obojenih kartica i njihovo objavljivanje za različite tipove poslova, te objavljivanje na karticama, može brzo da omogući radnom timu da:

- Vide ono na čemu rade (kartice daju vrednost određenom poslu).
- Poprave vizualni pokazatelj blokatora.
- Procene koliko je bila dobra analiza i proces osiguranja kvaliteta.
- Sprovedu jednostavno praćenje (koliko je stavki premašilo zadani rok, koliko ima blokatora, itd.)

Ukoliko postoji hitan slučaj u vidu zahteva za podršku ili greške u programskom proizvodu u produkciji, tada se koristi prazno mesto na Kanban ploči označeno kao „Hitno“. Zadaci koji se rešavaju ovim putem ne stavljaju se prvobitno u zalihu te je cilj rešiti ih što pre.

Prednosti Kanban ploče su vidljivost sledećeg:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Na čemu pojedina osoba radi
- Je li preopterećen poslom
- Gde se nalazi „usko grlo“
- Gde se stvaraju rupe u poslu
- Šta je blokirano

Ograničenja na Kanban ploči po pojedinoj koloni nisu striktno definisana nego se do njih dolazi praksom kroz rad. Kada je određena kolona po definisanom kriterijumu popunjena, tada radnik koji čeka da se kolona isprazni, tj. koji trenutno nema šta da radi, gleda postoji li negde „usko grlo“ te pomaže da se ono otkloni. Ukoliko ne postoji „usko grlo“ to može biti znak da su ograničenja po kolonama premala jer se smanjenje posla u procesu (eng. *Work In Progress – WIP*) po kolonama radi s ciljem bržeg protoka zadataka. Međutim, ako na ploči postoji veliki broj zadataka na kojima duže vreme niko ne radi, može se zaključiti da je ograničenje po kolonama preveliko. Na osnovu toga, može se zaključiti da važi:

- Premaleno Kanban ograničenje po koloni je znak da ljudi miruju, tj. nemaju posla što rezultuje lošom produktivnošću
- Preveliko Kanban ograničenje po koloni je znak da zadaci miruju što rezultuje lošim vremenom vođenja (eng. *lead time*)

Kanban ploča za razliku od ploče agilnih metodologija ima ograničenja. Iz razloga što je cilj novu funkcionalnost programskog proizvoda izdati što pre, ograničava se količina posla koja se može odraditi istovremeno. Postoje dva osnovna ograničenja:

1. **Redovi čekanja**: kolone na Kanban ploči gde se gomilaju zadaci do definisanog maksimalnog broja. Na primer, u koloni „Zadaci“ može se nalaziti pet kartica.
2. **WIP graničenje [40]**: određuje maksimalan broj radnih zadataka koji mogu biti u određenom stanju. Na primer, u koloni „Razvoj“ mogu biti maksimalno tri kartice.

Za razliku od svih drugih agilnih metodologija razvoja programskog proizvoda, Kanban metodologija nema uloge u projektu. To ne znači da ih ne sme imati, nego samo da nisu potrebne.

Glavna razlika u Kanban dnevnim stojećim sastancima je da moderator nabraja po karticama, a ne po ljudima. Pregled posla se vrši zdesna nalevo kako bi se naglasio „pull“ sistem. Ploča pokazuje ukupan status posla umesto da fokus bude na izuzecima.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 7. Verifikacija, validacija i testiranje

Verifikacija i validacija su srodni pojmovi, ali među njima ipak postoji suptilna razlika.

- *Verifikacija* je provera da li softver odgovara specifikaciji, dakle dokumentu o zahtevima.
- *Validacija* je provera da li softver zadovoljava „stvarne“ potrebe korisnika.

Razlika nastaje zato što specifikacija obično ne uspeva da u potpunosti zabeleži stvarne potrebe korisnika. Konačni cilj verifikacije i validacije je da se uveri da je softver dovoljno dobar i pouzdan da može poslužiti svojoj svrsi. To ne znači da softver mora biti sasvim bez greške, već samo da on mora biti dovoljno dobar i pouzdan za predviđeni oblik korištenja. Stépen tolerisanja grešaka zavisi od funkcije koju softver obavlja, očekivanjima korisnika, te o uslovima tržišta.

Metode za verifikaciju i validaciju se grubo mogu podeliti u dve grupe:

- *Statička verifikacija i validacija*. Svodi se na analizu i proveru dokumenata, modela sistema, dizajna, odnosno programskog koda. Odvija se *bez* stvarnog izvođenja softvera.
- *Testiranje*. Svodi se na izvođenje softvera ili njegovih delova na veštački pripremljenim podacima, uz pažljivo analiziranje rezultata.

Testiranje je uobičajena metoda, ali statička verifikacija i validacija ima nekih prednosti:

- Jedna statička provera može da otkrije mnogo grešaka. Testiranje obično odjednom otkriva samo jednu grešku budući da se nakon greške program „ruši“ ili radi s lošim podacima.
- Statička provera omogućava bolje korišćenje znanja o programiranju. Osobe koje obavljaju proveru znaju koje vrste grešaka se često pojavljuju u dotičnom programskom jeziku ili aplikaciji te se koncentrišu na takve greške.

Većina autora smatra da se statičkim metodama može uz male troškove otkriti 60-90% grešaka. Znači, dve vrste metoda su komplementarne te bi trebalo primenjivati obe. Verifikacija i validacija se prvenstveno primenjuju na programski kod, ali je poželjno da se u što većoj meri primenjuju i u ranijim fazama razvoja softvera. Statička provera se može obavljati u raznim fazama softverskog procesa, dok je testiranje primenjivo samo na program odnosno prototip. Statička provera se svodi na analizu i pregled programskog koda, bez stvarnog izvođenja programa. Program se zato najpre proverava statičkim metodama, kako bi se otkrila glavnina grešaka, a nakon toga se još podvrgava testiranju koje potom ide znatno brže, dok su ukupni troškovi verifikacije puno manji.

Da bi verifikacija i validacija dale očekivane efekte, **planovi testiranja** bi trebalo da nastanu već u fazi specifikacije i dizajna sistema. Plan za testiranje softvera je dokument sa sledećim delovima:

- opis procesa testiranja,
- popis zahteva koji će se testirati,
- popis delova softvera koji će se testirati,
- kalendar testiranja,



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- opis načina beleženja rezultata testiranja,
- popis hardvera i softvera koji je potreban za testiranje,
- ograničenja koja utiču na proces testiranja.

Tokom verifikacije i validacije, otkrivaju se greške, pa se softver menja kako bi se popravile. Iako je taj proces popravljanja tj. **dibagiranje**, povezan s verifikacijom i validacijom, on se ipak razlikuje od njih, jer dok one otkrivaju *postojanje* greške, on istu *locira* (otkriva uzrok) i *popravlja*.

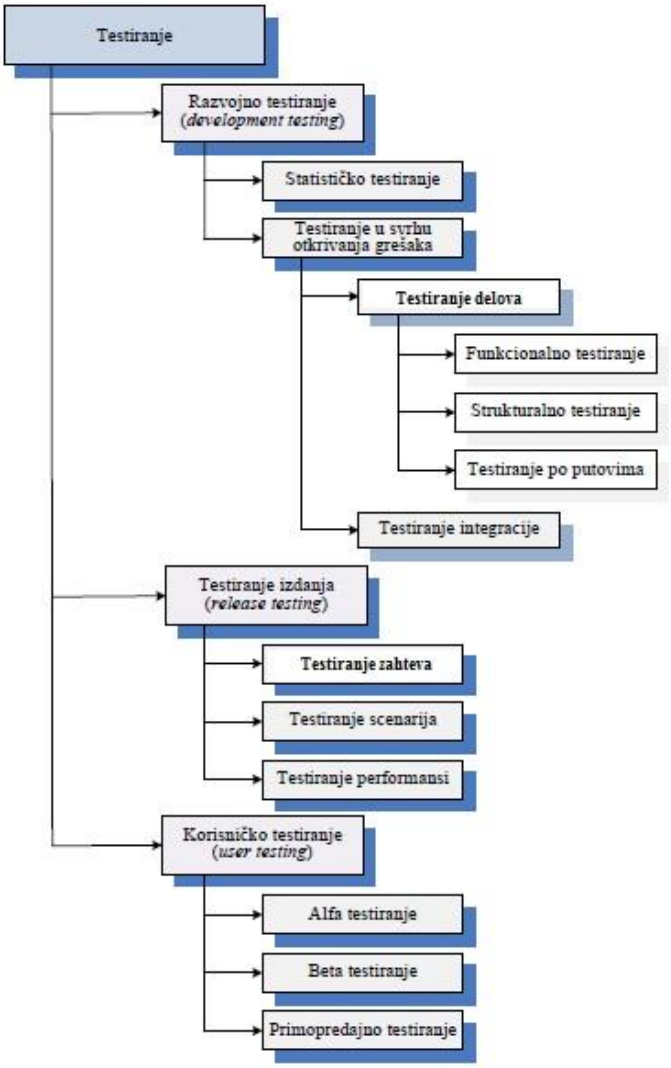
*Testiranje* je izvođenje softvera ili njegovih delova na veštački kreiranim podacima, uz pažljivo analiziranje rezultata. Svrha testiranja može biti verifikacija ili validacija.

Postoje razne vrste testiranja. Jedna moguća klasifikacija prikazana je na slici 7.1. Prvi nivo podele na toj slici napravljen je po kriterijumu ko odnosno kada učestvuje u testiranju.

- *Razvojno testiranje* sprovodi razvojni tim tokom razvoja softvera.
- *Testiranje izdanja* obavlja posebni tim za testiranje unutar softverske kuće uoči isporuke novog izdanja softvera.
- *Korisničkim testiranjem* bave se sami korisnici tokom isporuke softvera.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Slika 7.1. Klasifikacija raznih vrsta testiranja*

Svaka vrsta testiranja s prvog nivoa podele dalje se deli na podvrste. Na primer, razvojno testiranje uključuje u sebi dve podvrste koje služe za proveru da li softver koji se razvija zadovoljava zahteve, a koje se među sobom razlikuju po vrsti zahteva i načinu biranja test-podataka.

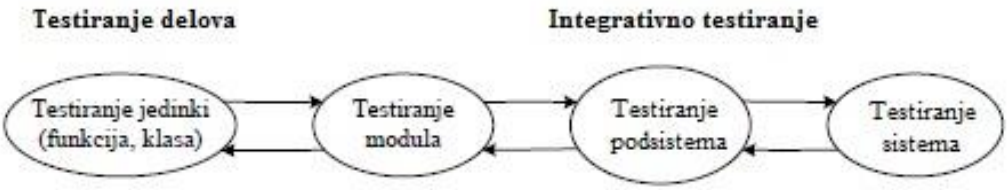
- *Testiranje u svrhu otkrivanja grešaka*. Služi za proveru funkcionalnih zahteva. Namera je otkrivanje odstupanja između softvera i njegove specifikacije. Ta odstupanja posledica su grešaka u softveru. Test-podaci su konstruisani tako da otkriju prisustvo greške, a ne tako da simuliraju normalnu upotrebu softvera.
- *Statističko testiranje*. Služi za proveru nekih oblika nefunkcionalnih zahteva. Namera je, na primer, izmeriti performanse softvera ili stepen njegove pouzdanosti. Test-podaci su odabrani tako da liče na stvarne korisničke podatke, te da odražavaju njihovu stvarnu statističku raspodelu. Obično je reč o velikom broju slučajno odabranih test primera, što omogućava statističku procenu performansi ili pouzdanosti.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Testiranje u svrhu otkrivanja grešaka se smatra uspešnim onda kad ono pokaže da softver ne radi dobro. Nakon testiranja grešaka sledi postupak dibagiranja. Naravno, testiranje može da pokaže da u softveru postoje greške, no ne može da dokaže da grešaka nema.

Proces testiranja u svrhu otkrivanja grešaka obično se sastoji od nekoliko faza, koje uzimaju u obzir da veliki sistem ima složenu građu. Detaljni prikaz faza vidi se na slici 7.2. S jedne strane testiraju se sastavni delovi softvera, a s druge, testira se da li su ti delovi na ispravan način integrisani u veće celine, te da li su te veće celine ispravno integrisane u još veće celine, i tako dalje. U fazi integracionog testiranja, testiranje se fokusira na interakcije između delova te na funkcionalnost celine. Ipak, često se dešava da se u toj fazi otkrivaju i greške u delovima. Zbog toga se faze mogu ponavljati. Testiranje delova obično sprovode osobe koje su razvile te delove, a integraciono testiranje može biti posao nezavisnog tima.



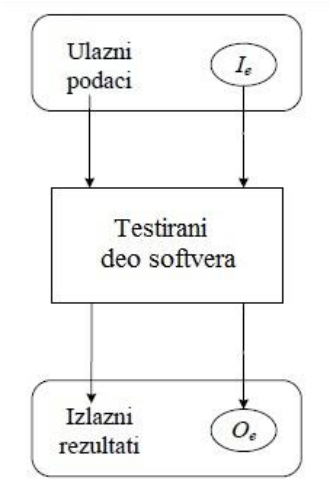

*Slika 7.2. Proces testiranja u svrhu otkrivanja grešaka*

Da bi testiranje dela softvera bilo uspešno, važno je dobro odabrati test podatke. U nastavku će biti dato nekoliko pristupa koji se razlikuju po načinu odabira test podataka:

**Funkcionalno testiranje** (*black-box testing*). Test-primeri se izvode isključivo iz specifikacije dotičnog dela softvera. Drugim rečima, softver se posmatra kao „crna kutija“ o kojoj jedino znamo da bi ona (prema specifikaciji) za zadate ulaze trebalo da proizvede zadate izlaze. Zbog postojanja grešaka, ulazi iz određenog (nepoznatog) skupa $\boldsymbol{O_e}$ izazvaće neispravno ponašanje, koje ćemo prepoznati po izlazima iz skupa $\boldsymbol{O_e}$**.** To je sve prikazano na slici 7.3. U postupku testiranja nastojimo da izaberemo ulaze za koje postoji velika verovatnoća da pripadaju skupu $\boldsymbol{I_e}$. Izbor takvih test primera najčešće se zasniva na iskustvu softverskog inženjera. Ipak, moguć je i sistematičniji pristup zasnovan na podeli skupa svih mogućih ulaznih podataka (domena) na *klase*.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES



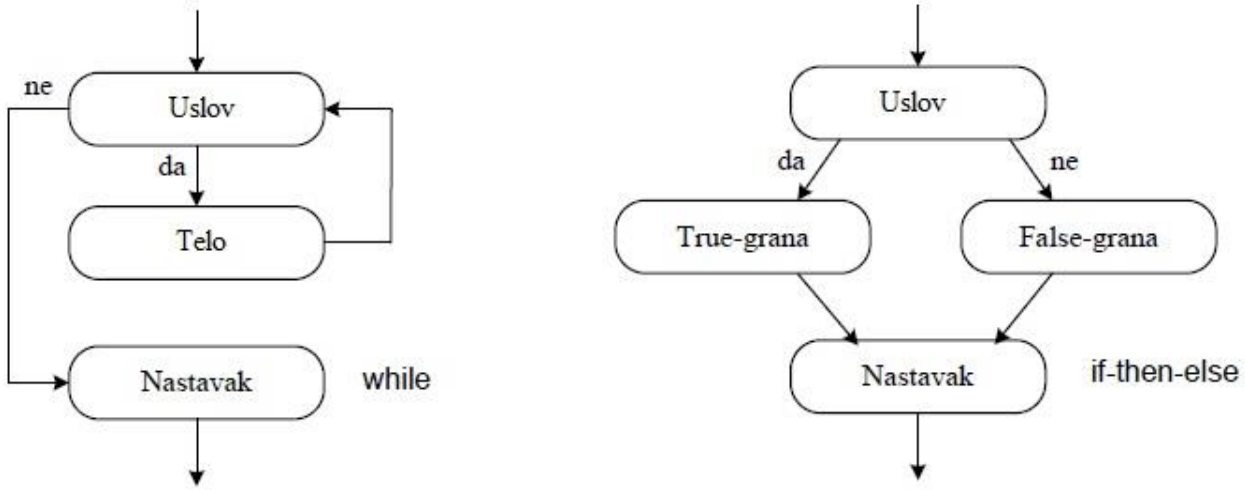

*Slika 7.3. Black box testiranje*

**Strukturalno testiranje** (*white-box testing*). Dodatni test-primeri se izvode na osnovu poznavanja interne strukture testiranog dela softvera i korišćenih algoritama. Dakle, koristi se analiza programskog koda dotičnog dela softvera da bi se odabrali dodatni test primeri, tj. da bi se bolje podelio domen na klase.

**Testiranje po putevima** (*path testing*). Reč je o posebnoj vrsti strukturalnog testiranja, gde se test primeri biraju tako da se isproba svaki „nezavisni put“ kroz dijagram toka kontrole testiranog dela softvera. Dakle najpre se mora nacrtati dijagram toka kontrole dotičnog dela softvera. Dijagram se crta tako da se naredbe while i if–then–else prikažu obrascima sa slike 7.4, a ostale naredbe se prikažu svaka po jednim čvorom. Uočavaju se čvorovi koji predstavljaju „početak“ i „kraj“ toka kontrole. Testiranje po putevima zahteva da se konstruišu takvi test primeri koji će isprobati svaki od nezavisnih puteva kroz dijagram. Nezavisni put je onaj koji ide od „početka“ do „kraja“ i prolazi bar jednim novim lukom.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Slika 7.4. Obrasci za crtanje dijagrama toka kontrole*

Integraciono testiranje sprovodi se nakon što se manji delovi softvera udruže u veću celinu. Cilj testiranja je otkriti greške koje nastaju zato što delovi na pogrešan način koriste međusobne interfejse. Test primeri se primenjuju na cjelinu, a ne na pojedine delove. Ova vrsta testiranja naročito je važna za objektno-dizajnirane sisteme. Naime, objekti su definisani preko svojih interfejsa, te se ponovo upotrebljavaju u kombinaciji s drugim objektima u drugim sistemima – tada nastaju greške čiji je uzrok interakcija objekata, a ne rad pojedinog objekta.
Postoje različiti tipovi interfejsa između udruženih delova. Nabrojaćemo četiri tipa:

- *Proceduralni interfejs.* Jedan deo softvera poziva proceduru koja pripada drugom delu softvera.
- *Parametarski interfejs*. Podatak ili referenca na njega prenosi se kao parametar u pozivu procedure iz jednog dela softvera u drugi.
- *Slanje poruka*. Jedan deo softvera zahteva uslugu od drugog dela tako što mu pošalje poruku. Povratna poruka uključuje rezultat izvršene usluge.
- *Zajednička memorija*. Blok memorije dostupan je raznim delovima softvera. Jedan deo skladišti podatke u blok, a drugi ih čita iz bloka.

Greške u korišćenju interfejsa mogu se podeliti na sledeće tri vrste.

- *Nerazumevanje interfejsa*. Na primer, kod proceduralnog interfejsa, deo koji poziva rutinu za binarno traženje ne zna da polje mora biti sortirano.
- *Pogrešna upotreba interfejsa*. Na primer, loš tip podatka u parametarskom interfejsu.
- *Greške u sinhronizaciji*. Na primer, kod slanja poruka ili kod zajedničke memorije. Delovi koji proizvode odnosno troše podatke rade na različitim brzinama.

Integraciono testiranje posebno je teško zato što se greške mogu pojaviti samo u izuzetnim situacijama (veliko opterećenje sistema) ili pak na neočekivanim mestima (u sasvim drugom delu softvera). Za integraciono testiranje važe sledeća opšta uputstva:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Pregledati programski kod svakog dela softvera i popisati sve interakcije jednog dela s drugim delovima.
- U slučaju proceduralnog interfejsa probati situaciju kad procedura vraća status greške.
- U slučaju parametarskog interfejsa oblikovati testove koji koriste ekstremne vrednosti parametara.
- Ako se prenose pokazivači (*pointeri*), isprobati primere s nul-pokazivačima.
- Ako se razmenjuju poruke, probati stvoriti znatno više poruka nego što se očekuje u normalnim okolnostima.
- Ako se koristi zajednička memorija, menjati redosled aktiviranja delova softvera koji pristupaju toj memoriji.

Softverski sistemi obično sadrže nekoliko nivoa integracije. Na primer: najmanje jedinke udružuju se u module, koji se spajaju u podsisteme, koji čine sistem. Zbog više nivoa integracije, integraciono testiranje mora se ponoviti više puta. Pritom redosled testiranja može biti jedan od sledeća dva.

- *Top-down*: najpre se testira integracija na najvišem, zatim na nižem, a na kraju na najnižem nivou. Pritom se nepostojeći ili netestirani delovi s nižeg nivoa menjaju s tzv. *stabovima*. *Stab* ima isti interfejs kao odgovarajući deo ali vrlo ograničenu funkcionalnost.

- *Bottom-up*: najpre se testira integracija na najnižem, zatim na sledećem višem, a na kraju na najvišem nivou. Nisu potrebni *stabovi* jer za svaki idući nivo integracije već imamo implementirane i testirane sve sastavne delove. Ipak, potrebni su *test drajveri* koji pozivaju manje delove softvera i simuliraju njihovu okolinu.

Testiranje dimom (eng. smoke testing) se može razmatrati i kao funkcionalno i kao testiranje modula. To je preliminarno testiranje u svrhu otkrivanja jednostavnih grešaka koje su ipak dovoljne da se odbije isporuka softvera. Odabere se i pokrene podskup testova koji pokriva najvažnije funkcionalnosti podsistema ili sistema, da bi se uvtrdilo da program radi kako treba (npr. Da li se program pokreće? Da li se otvara prozor? itd.). Cilj je utvrditi da aplikacija toliko „puca“ da sledeće (detaljnije) testiranje nije ni potrebno.

Testiranje softvera je zadatak na koji ljudi često gledaju sa nipodaštavanjem, mada je on zapravo podjednako važan kao i sâm razvoj softvera. Štaviše, od testera se očekuje da razumeju kod programa i njegovo funkcionisanje da bi mogli uspešno da predvide bagove, pa bi oni neretko mogli da zamene mesta sa programerima i naprave podjednako dobru aplikaciju. Bitno je da ljudi sa ovakvim znanjem testiraju softver, jer samo oni mogu da uoče bagove koji bi prouzrokovali loše rezultate koji običnom korisniku izgledaju validno. Iako su neki programi komplikovaniji/zahtevniji od drugih (npr. softver banke), svaki klijent ima pravo da mu naručeni program radi najbolje moguće, pa je zato testiranje uvek važno i apsolutno neophodno.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 8. Literatura


[1] Kenneth S. Rubin, ***Essential Scrum A Practical Guide to the Most Popular Agile Process***, First Edition, Pearson Education Inc., ISBN-13: 978-0-13-704329-3, 2012

[2] John Dooley, ***Software Development and Professional Practice***, First Edition, Springer, ISBN-13 (electronic): 978-1-4302-3802-7, 2011

[3] Mike Cohn, ***Succeeding with Agile Software Development Using Scrum***, Addison Wesley, ISBN-13: 978-0-321-57936-2, 2010

[4] Robert Manger, ***Softversko inženjerstvo***, First Edition, Sveučilište u Zagrebu, 2013

[5] Alan Dennis, Barbara Haley Wixom, Roberta M. Roth, ***System Analysis and Design***, Third Edition, John Wiley & Sons Inc., 2008

[6] Jeffrey L. Whitten, Lonnie D. Bentley, Kevin C. Dittman, ***System Analysis and Design Methods***, Fifth Edition, McGraw-Hill, 2001

[7] ***User Interface Design,*** https://en.wikipedia.org/wiki/User_interface_design, 2014

[8] Mike Cohn, ***User Stories Applied for Agile Software Development***, Addison Wesley, ISBN: 0-321-20568-5, 2004

[9] Ante Topić, ***Utjecaj Kanban agilne metode na razvoj programskog proizvoda***, Sveučilište u Splitu, 2012

[10] Ian Sommerville, ***Software Engineering***, Addison-Wesley, 2011

[11] Roger S. Pressman: ***Software Engineering, A Practitioner's Approach, European Adaptation***, McGraw-Hill, 2010

[12] Shari L. Pfleeger, Joanne M. Atlee, ***Softversko inženjerstvo, teorija i praksa***, prevod trećeg izdanja, CET, 2006.

# 9. Appendix 1

## 9.1 Koncepti programiranja

Koncept programiranja je način formiranja i izgradnje struktura i elemenata programa. Mogućnosti i stil različitih programskih jezika definisani su njima pridruženim konceptima programiranja. Neki jezici su dizajnirani tako da se oslanjaju samo na jedan koncept, dok drugu mogu da podržavaju više koncepata. Na isti način i svaki model razvoja softvera može (ali ne mora) da primenjuje neki od koncepata.

**Objektno-orijentisano programiranje** (**OOP**) je programski obrazac kod koga koncept programiranja čine objekti koji imaju polja podataka (atribute koji opisuju objekat) i njima pridružene procedure poznatije kao metode. Objekti, koji najčešće predstavljaju instance klasa, obično interaguju jedan s drugim kako bi dizajnirali aplikacije i programe. Stoga se OOP može posmatrati kao kolekcija interagujućih objekata, što je u suprotnosti s konvencionalnim modelom po kom se program tumači kao lista zadataka koje treba izvršiti. U OOP, svaki objekat ima mogućnost da prima poruke, procesira podatke i šalje poruke drugim objektima. Svaki objekat se može posmatrati kao nezavistan „uređaj“ sa različitom ulogom ili zaduženjem. Akcije tj. metode koje se izvršavaju nad ovim objektima su povezane s njim samim. Na primer, strukture objektno-orijentisanih podataka teže tome da „nose sa sobom“ svoje operatore (ili da ih naslede od klase ili sličnog objekta).

Objekti se mogu posmatrati kao enkapsulacije podataka unutar skupa funkcija dizajniranih da obezbede da se podaci koriste kako treba, kao i da asistiraju u tom korišćenju. Oni takođe nude i standardizovane metode za izvršenje određenih operacija nad podacima bez utvrđivanja detalja kako će ti zadaci biti urađeni. Na ovaj način se mogu izvršiti promene u okviru interne strukture ili metode objekta ne zahtevajući da se ostatak programa modifikuje. I ovaj pristup može da se primeni nad objektima različitog tipa. Na primer, nekoliko različitih tipova objekata može da sadrži metodu štampanja. Svaki tip objekta može tu metodu implementirati na drugačiji način, u skladu s različitim sadržajem koji svaki podatak nosi, ali sve ti različite metode za štampanje se mogu pozvati na isti standardni način iz bilo kog dela programa. Ova mogućnost je izrazito korisna kad na kodu radi više od jednog programera, ili kad neki kod treba nanovo koristiti u okviru drugog projekta.

**Aspektno-orijentisano programiranje (AOP)** je novi programski obrazac koji uvodi dodatne modularizacije softvera kako bi se izolovale sekundarne ili podržavajuće funkcije iz glavnog programa. Budući da programeri moraju da vode računa o svim stvarima koje treba uraditi, kako rešiti određeni problem, kako hendlovati određene interakcije i kako omogućiti da se program ponaša kako treba u pravom trenutku, to naposletku može dovesti do ozbiljnih problema tokom razvoja i održavanja aplikacije. Raspored koda za realizovanje neke funkcionalnosti postaje ključan kako zahtevi za tom funkcionalnošću evoluiraju, pa stoga održavalac sistema mora da pronađe i ažurira kod na mnogo mesta. AOP pristip se fokusira na identifikaciji, specifikaciji i reprezentaciji „rasprostranjenih



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

funkcionalnosti“ (*cross-cutting concerns*) i njihovoj modularizaciji u odvojene funkcionalne jedinice, uz naravno njihovu automatsku kompoziciju u sistem.

AOP se može koristiti u kombinaciji s funkcionalnim programiranjem, ali i s OOP. Ono predstavlja neku vrstu meta-programiranja. Sve što se može postići korišćenjem AOP, se može postići i bez njega, samo koristeći više koda. AOP praktično optimizuje proces pisanja koda. Recimo da imamo grafičku klasu koja sadrži puno „set...()“ metoda. Nakon svake „set“ metode, podaci na grafici se menjaju, pa ona mora biti ažurirana na ekranu. Pretpostavimo da se to radi pozivanjem metode „Display.update()“. Klasičnim pristupom bismo samo dodavali kod, odnosno dodali bismo ovu metodu na kraju „set“ metode:

```
void set...(...) {
   :
   :
   Display.update();
}
```

Ovo nije problem ako u kodu ima 3 „set“ metode, ali iako ih ima npr. 200 to ume da bude prilično zamoran i naporan posao i pojava bagova je postaje izvesnija. AOP ovo rešava elegantno dodavanjem aspekta:

```
after() : set() {
  Display.update();
}
```

Zbog ovakvih stvari se AOP posmatra kao lep dodatak OOP-u koji kod čini jasnijim, preglednijim i fokusiranijim na biznis logiku. Dva najveća benefita primene AOP su:

1. Logika za svaku funkcionalnost je sada na jednom mestu umesto da bude rasuta svud po kodu;

Klase su jasnije budući da sadrže samo kod za svoju osnovnu funkcionalnost dok su sekundarne funkcionalnosti preseljene na aspekte.

U poslednje vreme raste potreba za implementiranjem rasprostranjenih funkcionalnosti na različitim klasama u programu bez menjanja samih klasa. Neki dinamički jezici poput Rubya ili Pythona koriste neke stvari nalik AOP-u samo bolje integrisane u sam jezik, tako da će i AOP u dogledno vreme verovatno postati deo nekog alata koji koristimo.

Najveći nedostatak AOP koncepta se naziva anti-paternom. On se ogleda u tome što se ponašanje jednog dela programa menja usled teškoće ili nemogućnosti da se identifikuju operacije u drugom delu. Zato bi trebalo izbegavati mešanje „normalnog“ koda i AOP-a za isti aspekt.

**Servisno-orijentisano programiranje** (SOP) je programski obrazac koji koristi servise kao jedinicu računarskog rada kako bi dizajnirao i implementirao integrisane biznis aplikacije i kritične softverske programe. Poput AOP-a, i ovaj koncept se može posmatrati kao suplement OOP-a. Dok se OOP fokusira na to šta su stvari i kako su konstruisane, SOP se fokusira na to šta rade.

OOP je svakako elegantan način za kreiranje aplikacija koje izvršavaju određene zadatke, međutim njegova robusna arhitektura može da predstavlja problem. S druge strane, servisno-orijentisani pristup obezbeđuje programerima lak način za integraciju sistema. Oni su mnogo manje kompleksni, brži, manji, lakši za održavanje, i svakako mnogo lakši za konceptualizaciju.

Kod SOP pristupa programe treba dizajnirati tako da rade kao servisi. Ovo ćemo najbolje uočiti preko primera baze podataka. Naime, mnoge baze podataka rade kao servisi na serveru, tj. osluškuju konekciju, a zatim osluškuju zahteve i odgovaraju na njih. Zato je razmena poruka između programa ključni posao. Na primer, ako imamo MySQL servis i Apache servis koji rade. Zahtev se šalje Apache servisu preko Interneta, on se povezuje s MySQL servisom (preko PHP skripte) i šalje sekvencu poruka. Rezultat procesiranja tih poruka se šalje nazad kao HTML strana tamo odakle je stigao zahtev.

# 10. Appendix 2

### 10.1 Modeli strukture

Pomenućemo četiri ovakva modela:

1. **Model s repozitorijumom**. Svi zajednički podaci čuvaju se u centralnoj bazi podataka (repozitorijumu), kome mogu da pristupe svi podsistemi. Komunikacija između podsistema odvija se tako što jedan podsistem zapiše podatke u bazu, a drugi pročita te podatke. I sam repozitorijum treba shvatiti kao jedan (specijalizovani) podsistem. Ovo je uobičajena struktura za sisteme koji rade s velikim količinama podataka (na primer: informacioni sistem banke ili preduzeća).

Prednosti ovog modela su što je:

- pojednostavljena komunikacija jer je izbegnuto direktno slanje podataka iz podsistema u podsistem.
- administriranje podataka centralizovano pa nema potrebe da se podsistemi time bave.
- olakšano dodavanje novog podsistema. On samo mora da bude kompatibilan s utvrđenom logičkom organizacijom podataka.

No model s repozitorijumom ima i svoje mane:

- Podsistemi se moraju pokoriti zajedničkoj logičkoj organizaciji podataka, što može loše uticati na njihove performanse.
- Različiti podsistemi mogu imati različite zahteve u pogledu administriranja podataka, no model ih prisiljava na zajedničku „politiku“.

- Teško je distribuirati repozitorijum na više umreženih uređaja, pa repozitorijum postaje usko grlo.

2. **Model klijent–server**. U skladu s njim, sistem se sastoji od sledećih delova:
   - *serveri:* podsistemi koji nude usluge drugim podsistemima,
   - *klijenti:* podsistemi koji traže usluge koje nudi server,
   - *mreža:* omogućava komunikaciju klijenata i servera.

Klijent mora da zna imena dostupnih servera i njihove usluge, dok server ne mora da zna ni imena ni broj klijenata. Ovo je uobičajena struktura za mrežno-orijentisane sisteme. Dobar primer je world-wide-web, gde su serveri web serveri kao što je Apache, klijenti su web pretraživači poput Mozilla Firefoxa ili Google Chromea, a mreža je Internet.

Prednosti ovog modela su u tome što je:
- omogućena distribucija sistema na mrežu računara.
- olakšano dodavanje novog server ili poboljšavanje rada postojećeg.
- dopušteno da svaki server koristi svoju optimizovanu organizaciju podataka.

Dok su mane te što:
- svaki server mora sam da brine za administriranje svojih podataka.
- svaki klijent mora da zna koje usluge se nalaze na kom serveru.

3. **Slojeviti model.** Sistem je organizovan kao niz podsistema koji se zovu *slojevi*. Svaki sloj definiše „apstraktni stroj“ čiji mašinski jezik (u stvari skup usluga) služi za implementaciju idućeg sloja. Ovakva struktura pojavljuje se kod sistema gde želimo da ostvarimo što veću nezavisnost rešenja o hardverskoj platformi. Primeri su: aplikacije stvorene u jeziku Java, internetski model za mrežne protokole sa 4 sloja i sl.

Prednosti slojevitog modela su sledeće:
- Podržan je inkrementalni razvoj sistema.
- Bilo koji sloj može se promeniti ukoliko zadrži isti interfejs.
- Ako se promeni interfejs jednog sloja, pogođeni su samo susedni slojevi.
- Sistem se lako prenosi na drugu platformu, jer su specifičnosti konkretnog računara ili operacionog sistema skrivene u unutrašnjim slojevima.

Mane slojevitog modela su sledeće.
- Nekad je teško podeliti sistem u slojeve, jer osnovne usluge donjih slojeva mogu biti potrebne i znatno višim slojevima.
- Mogu nastupiti problemi s performansama, zbog višestrukog interpretiranja zahteva iz sloja u sloj.

4. **Model cevovoda i filtra (*pipe and filter*).** Sistem se sastoji od niza podsistema koji se zovu *filtri*. Razmena podataka odvija se u jednom smeru i oblikuje takozvani *cevovod*. Svaki filtar prima podatke od prethodnog filtra, transformiše ih, pa ih tako transformisane šalje idućem filtru. Na



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

taj način podaci koji ulaze u sistem doživljavaju niz transformacija da bi se na kraju pretvorili u izlazne rezultate. Ovaj model često se koristio u prošlosti za oblikovanje neinteraktivne obrade podataka, na primer računanje plata zaposlenih, ispis računa telefonskim pretplatnicima i sl. Najčešće se koristio poseban slučaj „ulaz-obrada-izlaz“ koji se sastoji od tri filtra: prvi učitava podatke, drugi računa s njima, treći ispisuje rezultate.

Prednost modela jeste taj što je jednostavan i razumljiv te da se promene u načinu obrade podataka lagano implementiraju dodavanjem novih filtara ili promenom postojećih. Prednost je i to što se opisana obrada podataka po potrebi može implementirati kao sekvencijalni sistem ili kao skup paralelnih procesa. Mana je u činjenici da prebacivanje podataka iz filtra u filtar koji put zahteva suvišne konverzije ili reinterpretacije podataka. Takođe, jednosmerna komunikacija duž cevovoda ne podržava složenije ili razgranatije vrste obrade podataka ili interaktivni rad.

Postoje dva različita pristupa kako se može ostvariti tok kontrole između podsistema:

- *Centralizovana kontrola:* jedan podsistem igra ulogu „kontrolora“, što će reći da on pokreće i zaustavlja ostale podsisteme.
- *Kontrola upravljana događajima* (*event driven*): kontrola je distribuirana u tom smislu što pojedini podsistemi samostalno reaguju na spoljne događaje. Ti događaji mogu biti generisani od strane drugih podsistema ili od okoline sistema.

Pod pojmom „događaj“ mislimo na signal koji može da poprimi zadati raspon vrednosti. To nije obični ulazni podatak, budući da vreme njegovog pojavljivanja nije pod kontrolom procesa koji ga obrađuje. Postoje standardni modeli kontrole koji su se pokazali upotrebljivim u raznim sistemima. Ovom prilikom će biti navedena četiri takva modela, od kojih prva dva spadaju među centralizovane, a druga dva među one koji su upravljani događajima. Svaki model kontrole može se kombinovati s bilo kojim modelom za strukturu sistema.

1. **Model poziva i povratka** (*call–return*). Spada među modele sa centralizovanom kontrolom. Reč je o ideji koja je implementirana u klasičnim programskim jezicima. Kontrola kreće s vrha hijerarhije podsistema (iz glavnog programa) i prosleđuje se na niže članove hijerarhije (preko poziva potprograma). Ovaj model ima smisla samo za sekvencijalne sisteme: naime u jednom trenutku samo jedan podsistem je aktivan. Prednost modela je mogućnost jednostavne analize toka kontrole. Mana je što je teško obraditi izuzetke (*exceptions)* koji remete normalni rad: ako se izuzetak dogodi negde duboko u hijerarhiji, potrebno je više puta prebacivati kontrolu na više članove hijerarhije.

2. **Model menadžera.** Jedan podsistem, takozvani menadžer, kontroliše ostale tako što im šalje signale za početak, kraj, odnosno sinhronizaciju njihovog rada. Svi podsistemi u principu rade paralelno, makar je to moguće simulirati i u sekvencijalnom sistemu. Menadžer izvršava beskonačnu petlju u kojoj ispituje promenljive stanja da bi ustanovio koje podsisteme treba pokrenuti ili zaustaviti. Ovaj model sa centralizovanom kontrolom susrećemo, na primer, u *multiprocessing* operativnim sistemima, gde jezgro operativnog sistema igra ulogu menadžera time što pokreće i zaustavlja druge procese.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

3. **Model difuzije** (*broadcast*). Spada u modele upravljane događajima. Bilo koji događaj prosleđuje se svim podsistemima. Odgovara samo onaj podsistem koji je napravljen da obrađuje taj događaj. Glavna razlika u odnosu na prethodni model je u tome što upravljanje nije prepušteno jednom podsistemu, već svaki podsistem samostalno odlučuje hoće li reagovati na događaj ili neće. Ovaj model prisutan je, na primer, u grafičkim korisničkim interfejsima, gde se događaj klika mišem emituje svim elementima interfejsa, a reaguje samo onaj element nad kojim se kursor miša nalazio. Moguća je varijanta sa selektivnim emitovanjem događaja samo onim podsistemima koji su „pretplaćeni“ na određene vrste događaja.

Prednosti modela difuzije su sledeće:

- Lako se dodaje novi podsistem.
- Svaki podsistem može da aktivira bilo koji drugi ne znajući njegovo ime i lokaciju.
- Primenljiv je u slučaju distribuiranih sistema.

Model difuzije ima sledeće mane:

- Podsistemi ne znaju da li će i kada neki događaj biti obrađen (iako su ih sami generisali).
- Mogu nastupiti konflikti ako više podsistema pokuša da istovremeno obradi isti događaj.

4. **Model upravljan prekidom** (*interrupt driven*). Takođe predstavlja model upravljan događajima. Svakoj vrsti spoljnog događaja pridružen je jedan hardverski prekid (*interrupt*). Svakom prekidu pridružen je poseban podsistem – takozvani *interrupt handler*. Kad procesor primi prekid odgovarajuće vrste, hardverski prekidač prekida trenutni proces i prebacuje kontrolu na pripadni *handler*. Taj *handler* obradi događaj, nakon čega procesor nastavlja onaj proces koji je bio prekinut. Ovaj model koristi se, na primer, za komunikaciju računara s periferijskim uređajima. Pisanje po tastaturi izaziva prekid, a odgovarajući *interrupt handler* učitava ukucane znakove.

Prednost ovog modela je u tome što se njime dobija najbrži mogući odgovor na odabrane događaje. Mana je u nepogodnosti za testiranje, kao i nemogućnosti nadogradnje ukoliko je broj prekida koje hardver podržava već potrošen.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# Koncepti objektno-orijentisanog programiranja kroz programske jezike C++ i C#

**Predavač: Suzana Stojković**

**Ciljevi predmeta:** Upoznavanje sa objektno-orijentisanim tehnikama programiranja

**Ishod predmeta:** Naučiti koncepte objektno-orijentisanog programiranja kroz programske jezike C++ i C#, savremeni pristup programiranju.

**Teorijska nastava**:

Koncept OO programiranja.Definicija klase.Članovi klase: atributi, metode, propertiji (C#) Definicija metoda. Podrazumevane vrednosti parametara Prenos parametara Statički članovi klase
Konstruktori i destruktori. Uloga konstruktora za kopiranje u C++-u Problem „curenja memorije“ u C++-u. Prijateljske funkcije i klase (C++). Preklapanje operatora. Implementacija indeksera (C#).
Nasleđivanje. Definicija izvedene klase u C++-u i C#-u. Konstruktori i destruktori izvedenih klasa. Polimorfizam. Višestruko nasleđivanje u C++-u. Pojam virtuelnih osnovih klasa. Zapečaćene klase u C#-u
Apstraktne metode i apstraktne klase. Interfejsi u C#-u. Projektni obrasci bazirani na nasledjivanju
Templejtske metode i klase. C++ STL (Standardna biblioteka templejta) .Net framework nizovi i kolekcije
Tokovi podataka. Rad sa binarnim i tekstualnim tokovima podataka u C++-u i C#-u.

**Praktične vežbe**: Software developeri rade u C#-u, a Embeded Specialisti u C++-u

**Sadržaj kursa:** Principi objektno-orijentisanog programiranja. Paralelni pregled implementacije koncepata objektno-orijentisanog programiranja u programskim jezicima C++ I C#

**Trajanje kursa:** 30 školskih časova, 2 dana nedeljno

**Kurikulum:**

Teorija 2 časa + vežbe 2 časa: Koncept OO programiranja.Definicija klase.Članovi klase: atributi, metode, propertiji (C#) Definicija metoda. Podrazumevane vrednosti parametara Prenos parametara Statički članovi klase. .NET Framework. Upoznavanje sa okruženjem MS Visual Studio (poželjno Express Edition 2010). Kreiranje projekta, klasa i članova korišćenjem vizarda. Prevodjenje, izvršenje i debagiranje programa



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Teorija 2 časa + vežbe 2 časa: Konstruktori i destruktori. Uloga konstruktora za kopiranje u C++-u Problem „curenja memorije“ u C++-u. Prijateljske funkcije i klase (C++). Preklapanje operatora. Implementacija indeksera (C#). Samostalna vežba I: kreiranje klase koja sadrži sve do sada predjene elemente

Teorija 2 časa + vežbe 2 časa: Nasleđivanje. Definicija izvedene klase u C++-u i C#-u. Konstruktori i destruktori izvedenih klasa. Polimorfizam. Višestruko nasleđivanje u C++-u. Pojam virtuelnih osnovih klasa. Zapečaćene klase u C#-u. Samostalna vežba II: Rešavanje problema sa nasleđivanjem

Teorija 2 časa + vežbe: Apstraktne metode i apstraktne klase. Interfejsi u C#-u. Projektni obrasci bazirani na nasledjivanju. Samostalna vežba III: Korišćenje apstraktnih klasa i interfejsa

I praktični test 2 časa + teorija 2 časa: Test Implementacija aplikacije koja sadrži sve do sada pređene elemente. Teorija: Obrada izuzetaka. Prijava i prihvatanje izuzetaka. Neprihvaćeni izuzeci.

Teorija + vežbe: Templejtske metode i klase. C++ STL (Standardna biblioteka templejta) .Net framework nizovi i kolekcije. Samostalna vežba IV: Razvoj aplikacije koja sadrži definiciju templejtse klase i obradu izuzetaka.

Teorija 2 časa + pitanja polaznika: Tokovi podataka. Rad sa binarnim i tekstualnim tokovima podataka u C++-u i C#-u. Rekapitulacija celokupnog gradiva

II praktični test 2 časa + finalni test: Razvoj aplikacije koja sadrži rad sa templejtskim klasama, tokovima podataka i obradu izuzetaka. Teoretski test.

**Ocenjivanje:**

Praktičnih radova: U 3 nivoa: ne zadovoljava, zadovoljava (dobar), iznad očekivanja (odličan)
Finalnog testa Svako pitanje nosi odredjeni broj poena – ukupno 100

**Testiranje:**

2 praktične provere znanja**:** Implementirati konkretan zadatak – na računaru
Finalni test: Teoretski – na papiru



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 1. Uvod u objektno-orijentisano programiranje

### 1.1 Objektno orjentisana paradigma

U svakodnevnom životu često upotrebljavamo pojam objekat, a da pri tome ni ne pokušavamo da ga definišemo. Čovek na svom putu do posla ulazi u kafić, naručuje kafu, sedne, popije je, i zatim nastavi put. U kafiću postoji više **objekata**: sam lokal, sto, kafemat, kafa, itd. Svaki objekat ima više različitih karakteristika (**atributa**) koje ga opisuju: kafić može da bude otvoren ili zatvoren; kafa može da bude gorka, srednje slatka ili veoma slatka i da ima određenu cenu; stolovi mogu da budu zauzeti ili slobodni. Takođe, neki objekti mogu da vrše i određene **funkcije**, npr. kafemat može *da pravi* kafu, konobarica može *da donosi* kafu.

Objektna orijentacija je tehnika modelovanje sistema; gde je sistem, softverski sistem ili sistem u opštem smislu, npr. kafić iz prethodnog primera. Objektna orijentacija opisuje, ili modeluje, sistem kao skup objekata koji su u međusobnoj interakciji. Objektna orjentacija je slična načinu na koji ljudi doživljavaju okolinu.

Objektno orjentisana paradigma (ili OO) je savremeni pristup programiranju i sastoji se od nekoliko koncepata, relativno novih u svetu programiranja, kao što su smanjenje složenosti razdvajanjem programa na manje module, smanjenje složenosti kroz apstrakciju (vidi 1.2), i korišćenje koncepta ponovne upotrebljivosti (reuseability) radi lakšeg i bržeg razvoja aplikacija.

### 1.2 Apstrakcija

Ljudi gledaju na fizički svet sa različitim nivoima apstrakcije. Vozač vidi kola kao transportni mehanizam sa točkovima, sedištima i rezervoarom (neki nisu čuli ni da postoji ulje i da treba da se menja). Sa druge strane, automehaničar shvata kola detaljnije: kao skup raznih mehaničkih delova koji pokreću kola.

Telefon apstrakuje detalje o načinu uspostavljanja veze – jednostavno, korisnika ne interesuje šta se dešava pri uspostavljanju veze – njemu je važno da on može da razgovara.

Apstrakcija je 76rocess ignorisanja detalja radi koncentrisanja na važnim, suštinskim karakteristikama. Apstrakcija se fokusira na najvažnijim karakteristikama nekog objekta, kao što je njegova svojstva i funkcije, i ignoriše nepotrebne detalje.

Na primer, motor ima dva točka i jedno sedište, dok automobil ima četiri točka i četiri sedišta. Ali, i motor i automobili su u suštini prevozna sredstva. Do ovog zaključka smo došli apstrakcijom detalja.

### 1.3 Objekti

Recimo da hoćemo da napravimo prodavanicu automobila. Zapažamo da automobil ima skup atributa koji ga opisuju (snaga motora, boja, klima, ...) kao i skup funkcija koje on može da izvršava



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

(startovanje motora, grejanje sedišta, ubrzavanje,...). Sve te karakteristike ćemo staviti u katalog performansi automobila. Međutim, osim gore navedenih atributa, ovaj automobil ima i XYZ šrafova i X metara žica. Da li je to potencijalnom kupcu automobila bitno? Naravno da nije. Zbog toga ignorišemo nebitne detalje, a zadržavamo samo nama bitne. Kao rezultat ove apstrakcije dobijamo objekat. Taj objekat je apstrakovana predstava automobila zato što zanemaruje nama nebitne detalje.

Objekti su apstrahovane predstave stvari. Objekat je definisan atributima i funkcijama.

Međutim, ovako apstrakovan objekat je dobar samo u kontekstu prodavnice automobila. Šta ako hoćemo da napravimo prodavnicu automobilskih delova? Da li bi nam u tom slučaju takav objekat bio adekvatan? Ne, jer smo zanemarili detalje koji su u ovom slučaju bitni!

Glavni problem u objektno-orjentisanom razvoju je nalaženje pravih objekata da bi se modelovao predstavljeni sistem. Identifikovanje objekata u kontekstu problema je umetnost, a ne nauka, tj. ne postoji jednoznačno rešenje.

Objekti imaju sledeće karakteristike:

- Objekti mogu da budu realni (stvarni) ili imaginarni (izmišljeni),
- Objekti mogu da budu jednostavni ili složeni,
- Sastoje se od atributa i funkcija,
- Nezavisni su,
- Odgovaraju kontekstu problema.

Na primer, automobil je složeni i realan objekat. Međutim, objekat ne mora uvek da ima fizičku predstavu, to može da bude i koncept. Na primer, bankovni račun je više koncept nego fizički objekat.

### 1.4 Zatvaranje (Inkapsulacija)

Da li, da bi obavili tevefonski razgovor, morate da znate kako se uspostavlja sama telefonska veza?

Ne, potrebno je samo da razumete kako da koristite telefon, tj. da razumete interfejs (**interface** – *def.* stanje ili tačka preko koje različite stvari interaguju) koji se sastoji od dugmića, zvučnika i mikrofona. Možete ignorisati interni proces uspostavljanja veze koja možda prolazi i kroz nekoliko zemalja. Ukoliko se tehnologija u uspostavljanju veze promeni, ili promenite operatera, vi ćete i dalje telefonirati na isti način.

Inkapsulacija razdvaja spoljašnje aspekte objekta, koji su dostupni drugim objektima, od internih (unutrašnjih) implementacionih detalja objekta, koji su skriveni od ostalih objekata.

Kupili smo automobil. Njegova implementacija su sva tehnička rešenja smeštena ispod haube. Nas ne interesuje kako motor radi. Njegov interfejs su volan, kvačilo, gas, kočnica i ostale sitnice u kabini. Mi koristimo objekat (automobil) preko interfejsa.

Međutim, u nekom trenutku odlučimo da ugradimo sistem za alternativno gorivo (plin). Menja se samo implementacioni deo, dok interfejs ostaje isti. Pošto je interfejs ostao isti, mi (drugi objekat koji koristi objekat automobil) ne moramo da ponovo učimo kako da koristimo automobil (drugi objekti koji koriste promenjeni objekat se ne menjaju).

Prednost inkapsulacije, koja krije implementacione detalje, je u tome da se objekat može promeniti. Ukoliko se samo dese promene u implementaciji, koja je inače skrivena, i ukoliko je novi interfejs (deo dostupan ostalim objektima) ostao isti kao original, objekti koji koriste promenjeni objekat ostaju isti.

Recimo da vozač hoće da zaustavi automobil pritiskom papučice za kočenje (operacija interfejsa) koja pomoću hidrauličnog sistema (implementacija) smanjuje brzinu automobila (privatni atribut, deo internih informacija). Vozač uopšte ne mora da zna da postoji hidraulični sistem za kočenje da bi ukočio automobil.

Objekat se sastoji iz tri dela tj. aspekta:

- **Javno dostupnog interfejsa**: spoljašnji deo koji ostali objekti koriste za interakciju sa tim objektom (papučice za kočenje, gas, volan automobila,...)
- **Implementacije:** interne funkcije i ostali implementacioni detalji; svrha objekta (motor i ostala mehanika automobila)
- **Interne informacije**: šta bi objekat trebalo da zna da bi izvršio funkciju. (brzina automobila,...)

Funkcije i atributi nekog objekta se obično nazivaju **članovima objekta**.

### 1.5 Klase

Na Slici 1.1 su prikazani različiti objekti: automobil, brod, mlaznjak, motor, jedrilica, voz, dvokrilni avion i raketa. Ovaj skup objekata je moguće **klasifikovati** (grupisati) po nekim zajedničkim karakteristikama. Na primer, motor, automobil i lokomotiva se mogu svrstati u kopneni transport. Raketa, mlaznjak i dvokrilni avion bi predstavljali vazdušni transport. Brod i jedrilica predstavljaju vodeni transport.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**Slika 0.1** Prevozna sredstva

Objekti sa istim skupom karakteristika se grupišu u klase. Klase su šabloni kojima se opisuje struktura objekata. Kažemo da objekti pripadaju klasi koja ih opisuje, da predstavljaju primerke (uzorke, istance) klase.

Moguće je sledeće poređenje: Ukoliko je klasa nacrt zgrade na papiru, onda su objekti sve zgrade izgrađene po tom nacrtu.

Može se reći da klasa predstavlja logičku apstrakciju, dok objekat poseduje fizičku reprezentaciju.

U našem primeru bi to značilo da objekti motor, automobil i lokomotiva pripadaju klasi kopneniTransport, raketa, mlaznjak i dvokrilni avion klasi vazdusniTransport, i brod i jedrilica klasi morskiTransport, Slika 1.2.

**Slika 0.2** Tri klase: za kopeneni, morski i vazdušni transport



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 2. Osnovni elementi programskog jezika

Programski jezici su veštački jezici namenjeni za komunikaciju sa računarom, tj. za pisanje programa koji će računar da izvršava. S obzirom da program čini skup instrukcija koje se izvršavaju i podataka koji se u njemu odrađuju, osnovni elementi jezika definišu način predstavljanja podataka i instrukcija u programu. Da bismo predstavili podatke u programu moramo znati koji tipovi podataka postoje u odredjenom programskom jeziku, a da bismo pisali instrukcije, prvo je potrebno znati koje simbole možemo koristiti u jeziku (azbuku jezika), kako izgledaju reči jezika i na kraju kako izgledaju složenije jezičke konstrukcije gde spadaju izrazi i naredbe.

## 2.1 Tipovi podataka

Tip podataka definiše skup vrednosti koje podaci tog tipa mogu da imaju, način njihove memorijske reprezentacije, kao i operacije koje se mogu izvršavati nad podacima tog tipa.

Svi tipovi se dele na:

- osnovne (elementarne ili primeitivne) i
- struktrune tipove podataka.

Jedna od karakretistika .Net jezika je da imaju zajednički skup osnovnih (primitivnih) tipova podataka gde spadaju:

- tipovi za predstavljanje celobrojnih podataka:

  **int** – celobrojni podatak za čije se predstavljanje koriste 4 bajta (32 bita).

  **short** – celobrojni podatak za čije se predstavljanje koriste 2 bajta (16 bitova).

  **long** – celobrojni podatak za čije se predstavljanje koristi 8 bajtova (64 bita).

  **byte** - celobrojni podatak za čije se predstavljanje koristi 1 bajt (8 bitova) – ovaj tip je definisan u programskom jeziku C#, dok se u programskom jeziku C++ za predstavljanje celih brojeve dužine 1 bajt koristi tip char.

  Sva ova 4 tipa se mogu koristiti i sa prefiksom unsigned u programskom jeziku C++ (**unsigned int**, **unstigned short**, **unsigned long**, **unsigned char**) ili dodavanjem slova „u“ na početak imena tipa u programskom jeziku C# (**uint**, **ushort**, **ulong**, **ubyte**), što označava da se radi o tipovima za predstavljanje neoznačenih celih brojeva.

- tipovi za predstavljanje realnih podataka:

  **float** – realni podatak u jednostrukoj tačnosti (32 bita),

  **double** – realni podatak u dvostrukoj tačnosti (64 bita).

- tip za predstavljanje znakovnih podataka:

  **char** - podatak dužine jednog bajta,

- tip za predstavljanje logičkih podataka:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**bool** - postoje samo 2 logičke vrednosti (tačno *true* i netačno *false*)

U struktrune tipove podataka spadaju:

- polja,
- enumeracije,
- strukture,
- klase i
- interfejsi (koji su definisani samo u programskom jeziku C#).

O pojedinim strukturnim tipovima podataka biće detaljnije reči u posebnim poglavljima.

Postoji i specijalni tip podataka koji čuva adresu drugog podatka u memoriji koji se naziva pokazivač. Pokazivači se puno koriste u programskom jeziku C++ za pristup podacima u dinamičkoj zoni memorije (u hipu).

Dakle, u programskom jeziku C++ svi podaci mogu biti smešteni u statičkoj i u dinamičkoj zoni memorije. Kada se podaci smeštaju u statičkoj zoni memorije za njih se rezerviše prostor u onom trenutku kada se podatak definiše, a iz memorije se uklanjaju kada se završi izvršenje bloka u kojem su definisani. Ako se podatak smešta u dinamičkoj zoni memorije, u statičkoj se mora definisati pokazivač koji će se koristiti za pristup tom podatku. Rezervacija prostora u dinamičkoj zoni memorije se vrši eksplicitno, korišćenjem operatora new, a uklanjanje podatka iz dinamičke zone memorije pozivom operatora delete.

U C#-u se svi tipovi podataka dele u 2 velike grupe: vrednosne i referentne tipove podataka. Podaci vrednosnog tipa se smeštaju u statičkoj zoni memorije što znači da kada se definiše promenljiva vrednosnog tipa u memoriji se rezerviše prostor za pamćenje njene vrednosti. Podaci referentnog tipa se smeštaju u dinamičkoj zoni memorije što znači da se rezervacija protora za smeštanje podataka referentnog tipa vrši dinamički operatorom new. O uklanjanju podataka iz dinamičke zone memorije u programskom jeziku C# programer ne vodi računa. Dok se izvršava C# aplikacija povremeno se startuje pomoćni program (garbage collector) koji uklanja sve podatke iz dinamičke zone memorije na koje ne ukazuje ni jedna živa referenca u programu (referenca živi u programu od tačke u kojoj je definisana do kraja bloka u kojem je definisana).

U vrednosne tipove podataka spadaju: elementarni tipovi podataka, structure i enumeracije.

U referentne tipove podataka spadaju: nizovi, klase i interfejsi.

Reference mogu biti definisane i u programskom jeziku C++. Referenca u ovom programskom jeziku predstavlja drugo ime za već postojeći podatak. Osnovna namena reference u ovom programskom jeziku je za prenos parametara o čemu će kasnije biti reči.

### 2.2 Azbuka jezika

Azbuka programskih jezika C++ i C# sadrži velika i mala slova engleske azbuke, dekadne cifre i specijalne znake.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 2.3 Tokeni jezika

Reč je niz znakova koji predstavlja elementarnu logičku celinu u jednom programskom jeziku. Programski jezici C++ I C# sadrže 6 vrsta reči (tokena):

- identifikatore,
- ključne reči,
- separatore,
- konstante,
- literale (konstantne znakovne nizove),
- operatore.

Osim reči koje imaju svoje određeno značenje u programu, postoje i delovi programa koje kompilator jednostavno ignoriše, a koji služe programeru da pojasne kod. To su komentari. U ovim programskim jezicima komentari se pišu izmedju simbola /* i */, ili počinju simbolom // i protežu se do kraja programske linije.

#### 2.3.1 Identifikatori

Identifikator (simboličko ime) je niz slova, cifara i crtica ('_' ) u kojem prvi znak ne može da bude cifra. Identifikatorima se imenuju sledeći elementi programa: promenljive, simboličke konstante, funkcije, korisnički tipovi podataka i labele.

#### 2.3.2 Ključne reči

Ključne reči su reči čije je značenje u programskom jeziku unapred definisano i ne mogu da se koriste kao identifikatori.

#### 2.3.3 Separatori

Separatori su posebni specijalni znaci koji imaju odredjeno (unapred definisano) sintaksno i semantičko značenje u programu, a ne označavaju nikakvu operaciju. Uglavnom služe za razdvajanje izvesnih programskih celina pa otuda i nose takav naziv. Ovoj grupi simbola pripadaju: {, }, (, ), ;, : …

#### 2.3.4 Konstante

*Celobrojne konstante*

Celobrojne konstante mogu da budu navedene u dekadnom, heksadekadnom ili oktalnom brojnom sistemu.

Decimalna celobrojna konstanta je niz dekadnih cifara u kojem je prva cifra različita od 0.

Oktalna celobrojna konstanta je niz oktalnih cifara u kojem je prva cifra 0.

Heksadekadna celobrojna konstanta je niz heksadekadnih cifara koji počinje prefiksom ‘0x’ ili ‘0X’ u kojem se za označavanje heksadekadnih cifrara koriste i mala i velika slova.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**P r i m e r i :**

123, 987, -765 //decimalne konstante tipa int

0456, -0547 //oktalne konstante tipa int

0x1a2, 0XA0F //heksadekadne konstante tipa int

*Realne konstante*

Realne konstante se pišu u dekadnom brojnom sistemu i mogu da budu predstavljene u fiksnom zarezu ili u eksponencijalnom obliku.

Zapis u fiksnom zarezu obavezno sadrži decimalnu tačku i to tačno tamo gde je njeno stvarno mesto u broju (npr. 34.67, 56., .98).

Za veoma velike ili veoma male brojeve koristi se eksponencijalni zapis oblika: *m*E*e,* gde *m* predstavlja mantisu broja, a *e* eksponent (npr. 2.23e2, -0.5, .2E-2,7e-23).

Implicitno se podrazumeva da su realne konstante tipa double. Dodavanjem sufiksa F ili f dobijaju se konstante tipa float

*Znakovne konstante*

Znakovna konstanta je ceo broj čija je vrednost jednaka kodu znaka navedenog između jednostrukih navodnika (npr. 'a', '>',...). Na primer, u ASCII skupu znakova, znakovna konstanta ‘5’ ima dekadnu vrednost 53, što nema veze sa vrednošću celog broja 5.

ASCII tabela znakova sadrži i izvestan broj upravljačkih simbola (simboli koji ne ostavljaju trag na ekranu ili štampaču). Za takve znakove su uvedene određene simboličke oznake.

| upravljački znakovi | simboli |
|---|---|
| prelaz u sledeći red | \n |
| horizontalna tabulacija | \t |
| vertikalna tabulacija | \v |
| znak za vraćanje (backspace) | \b |
| nova strana (form feed) | \f |
| obrnuta kosa crta (backslash) | \\ |
| znak pitanja | \? |
| apostrof | \' |
| znaci navoda (dvostruki) | \" |



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 2.3.5 Literali

Literal (konstantni znakovni niz) je sažeti način zapisivanja niza znakovnih konstanti. Npr. "Ovo je literal". U C++-u će ovakav literal biti predstavljen kao niz podataka tipa char kome se na kraju dodaje još jedan znak sa vrednošću 0 ('\0'), tzv. *null terminated symbol* (završni simbol null). U programskom jeziku C# ovakav literal će biti predstavljen kao objekat klase string.

### 2.3.6 Operatori

Operatori su simboli koji označavaju određenu operaciju i koji povezuju jedan ili više operanada u izraz. U zavisnosti od broja operanada operatori se dele na:

- unarne,
- binarne i
- ternarne.

U odnosu na funkcionalnost operatori se mogu podeliti u sledeće grupe:

- operatori za pristup elementima polja i klasa
- operator za poziv funkcije
- aritmetički operatori
- relacioni operatori
- logički operatori
- operatori dodeljivanja vrednosti
- operatori referenciranja i dereferenciranja

*Aritmetički operatori*

Aritmetički operatori imaju numeričke operande i rezultate, takođe, numeričkog tipa. Ovi operatori biće navedeni u odnosu na prioritet, od operatora najvišeg prioriteta prema operatorima nižih prioriteta.

- Unarni operqtori + i - – Unarni operator – menja znak operanda koji sledi. Unarni operator + zadržava znak operanda koji sledi (to znači da je on praktično operator bez dejstva).
- Operatori inkrementiranja (++) i dekrementiranja (--) – Ovi operatori su unarni i mogu biti prefiksni i postfiksni. Oni vrše povećanje (smanjenje) vrednosti svog operanda za 1. Rezultat izraza sa prefiksnim operatorima ++ (ili --) je nova vrednost operanda, dok je rezultat izraza sa postfiksnim operatorom, stara vrednost operanda. To znači da:
  - Ukoliko se vrednost izraza ++a (ili –a) koristi u nekom širem izrazu, prvo će se izvršiti promena vrednosti promenljive a i ta promenjena vrednost će se iskoristiti za izračunavanje vrednosti šireg izraza.
  - Ukoliko se vrednost izraza a++ (ili a--) koristi u nekom širem izrazu, izvršiće se izračunavanje vrednosti šireg izraza sa starom vrednošću promenljive a i nakon toga će se izvršiti promena vrednosti promenljive a.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Operatori *, /, % – * označava množenje; / označava deljenje (u slučaju primene nad celobrojnim podacima predstavlja celobrojno deljenje, tj. u rezultatu se vrši jednostavno odbacivanje cifara iza decimalne tačke); % (po modulu) označava ostatak deljenja i primenjuje se isključivo nad celobrojnim podacima.
- Binarni operatori + i - – + označava sabiranje, a – oduzimanje

**Primeri:**

```
int a,b,c,d;
a=3; b=6;
c=a*--b;  //ekvivalentne naredbe: b=b-1; c=a*b;
d=a*b--;  //ekvivalentne naredbe: d=a*b; b=b-1;
```

*Relacioni operatori*

Ovo su operatori za upoređivanje vrednosti operanada. Operandi u ovom slučaju mogu da budu bilo kog numeričkog tipa ili pokazivači, a rezultat je logičkog tipa. Ukoliko je poređenje tačno (tj. ukoliko je relacija zadovoljena), rezultat je true, u suprotnom rezultat je false. Ovoj grupi operatora pripadaju:

- <= (manje ili jednako),
- > (veće),
- >= (veće ili jednako),
- == (jednako),
- != (različito).

Prioritet prva četiri operatora je viši od prioriteta poslednja dva.

*Logički operatori*

Ovoj grupi operatora pripadaju:

- ! (negacija),
- && (konjukcija, tj. logičko I),
- || (disjunkcija, tj. logičko ILI).

Rezultat operacije je true ako je rezultat logičke operacije tačno, u suprotnom, rezultat je false. Operatori su navedeni od operatora najvišeg prioriteta prema operatorima nižeg prioriteta.

*Operatori dodele*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Ovo je binarni operator koji smešta vrednost desnog operanda na lokaciju čije se ime nalazi sa leve strane operatora. Levi operand mora da bude ime memorijske lokacije.

Postoje dve grupe operatora dodele:

- elementarni operator dodeljivanja (=),
- složeni operatori dodeljivanja (opšti oblik ovih operatora je <op>=) – operator <op> može biti jedan od sledećih: +, -, *, /, %.

  Opšti izraz a <op>= b se može napisati i kao a = a <op> b

**Primeri:**

```
a += 3;          //ekvivalentno: a = a+3;
b -= a;          //ekvivalentno: b = b-a;
proizvod *= a-5; // proizvod = proizvod *(a-5);
```

## 2.4 Naredbe

Naredbe mogu biti:

- Naredbe opisa (naredbe kojima se definišu promenljve, simboličke konstante ili tipovi podataka) i
- Izvršne naredbe.

Izvršne naredbe se dele na:

- elementarne naredbe i
- strukturne naredbe.

### 2.4.1 Definisanje promenljivih

Sve promenljive koje se koriste u jednom programu treba da budu definisane pre njihovog prvog korišćenja. U okviru definicije promenljive naovdi se njen tip, ime i, eventuralno, inicijalna vrednost. U opštem slučaju ove definicije su oblika:

<tip> <ime>[=<vrednost>][,<ime>[=<vrednost>]]...;

**Primeri:**

```
int a, b=5;
int i;
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

double a, b=45.988675654;

Definicija pokazivača:

<tip>* <ime>[=<vrednost>];

U C++-u mogu eksplicitno biti definisane i reference. Pošto je u ovom jeziku referenca drugo ime za već postojeći objekat, referenca se u ovom jeziku pri definisanju mora odmah i inicijalizovati:

<tip>& <ime> = <ime_postojece_promenljive>;

### 2.4.2 Definisanje simboličkih konstanti

U zavisnosti od potrebe, određenim konstantama u C programu se mogu dodeliti imena. U opštem slučaju imenovanje konstanti i njihovo definisanje je oblika:

const <tip> <ime>=<vrednost>[,<ime>=<vrednost>]...;

**Primeri:**

const int a=3, b=5;

const float Pi=3.14;

### 2.4.3 Elementarne izvršne naredbe

Elementarnim izvršnim naredbama se vrši neka elementarna obrada podataka, a dobijaju se tako što se na kraj jednog izraza doda znak ;.

**Primeri:**

a+=b*5;

Max(a,b);

i++;

### 2.4.4 Strukturne naredbe

U teoriji programiranja definisane su tri osnovna tipa programskih struktura :

- sekvenca,
- selekcija,
- iteracija.

U svim C-like programskim jezicima (gde spadaju i C++ i C#) sekvenca se implementira korišćenjem bloka, selekcija naredbama uslovnog skoka (if i switch) i naredbama bezuslovnog skoka (goto, break i continue), a iteracija naredbama za realizaciju programskih petlji: for, while i do-while.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 2.4.5 Blok naredbi

Skup programskih naredbi koje se izvršavaju onim redosledom kako su zapisane čini blok. Blok počinje simbolom {, a završava se simbolom }. U opštem slučaju ima sledeću strukturu:

```
{
    <naredba1>
    <naredba2>
      …
    <naredbaN>
}
```

Na početku bloka mogu da budu navedeni opisi promenljivih i konstanti koje su lokalne za blok.

### 2.4.6 Naredbe uslovnog grananja

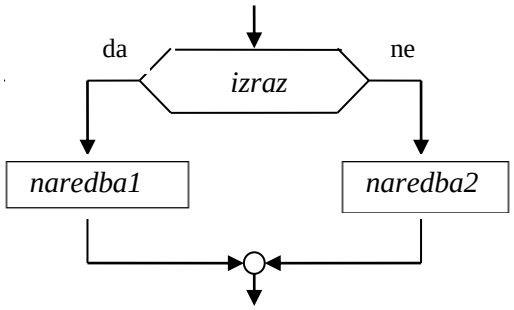
*If-naredba*

*If*-naredbom se implementira osnovi tip selekcije (grananja) kojom se vrši uslovno izvršenje jedne od dve moguće naredbe. Standardni dijagram toka sturkture grananja prikazan je na Sl. 2.1

Ekvivalentan kod u C-u je:

```
if (<izraz>)
  <naredba1>
else
  <naredba2>
```

Dejstvo: Kada je vrednost izraza različita od nule (tj. true), izvršava se naredba <naredba1>, u suprotnom, izvršava se naredba <naredba2>.



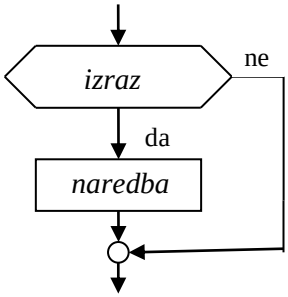

**Sl. 2.1 Dijagram toka strukture grananja**

*Skraćeni oblik If-naredbe*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Selekcijom se može implementirati i odluka da li će se jedna naredba u programu izvršiti ili ne. Dijagram toka structure skraćenog grananja prikazan je na Sl. 2.2.




**Sl. 2.2 Struktura skraćenog grananja**

Ekvivalentan kod je:

```
if (<izraz>)
  <naredba>
```

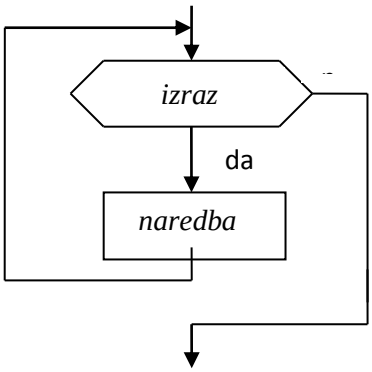
Dejstvo : Kada je vrednost izraza različita od nule (tj. *true*), izvršava se navedena naredba.

### 2.4.7 Programske petlje

Programske petlje omogućavaju višestruko ponavljanje određenog dela programa. U programskom jeziku C++ postoje tri vrste programskih petlji: *while*-petlja, *do-while*-petlja i *for*-petlja. U programskom jeziku C# postoji još i *foreach* petlja.

*While-petlja*

While petlja omogućava ponavljanje naredbe, koja predstavlja telo petlje, dok je definisan uslov zadovoljen. Dijagram toka *while*-petlje prikazan je na Sl. 2.3.




**Sl. 2.3 Dijagram toka while petlje**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Ekvivalentan kod je:

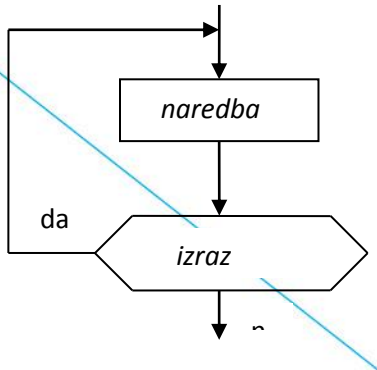
```
while (<izraz>)
        <naredba>
```

*While*-petlja služi za realizaciju iterativnih procesa (sumiranje beskonačnih redova i sl.)

*Do-while-petlja*

Do-while petlja ima slično dejstvo kao i while-petlja. Jedina razlika je u tome što se uslov za ponavljanje petlje u kodu nalazi iza njenog tela tako da se naredba, koja predstavlja telo petlje, mora izvršiti bar jednom. Dijagram toka ove struktura prikazan je na

Sl. 2.4




**Sl. 2.4 Dijagram toka do-while petlje**

Format odgovarajuće naredbe u programskim jezicima C++ i C# je:

```
do
        <naredba>
while (<izraz>);
```

*For-petlja*

For petlja se obično koristi kada u programu treba realizovati takozvanu brojačku petlju. Brojačka petlja podrazumeva da je unapred (pre ulaska u petlju) poznat (može da se izračuna) broj ponavljanja tela petlje. Dijagram toka brojačkih petlji prikazan je na slici Sl. 2.5.

Oznake na slici imaju sledeća značenja:

<brojač> - ime promenljive koja predstavlja brojač petlje,

<dg> - početna vrednost (donja granica) brojača petlje,

<gg> - konačna vrednost (gornja granica) brojača petlje,

<korak> - vrednost koja se dodaje brojaču petlje na kraju svake iteracije.

For petlja u programskom jeziku C ima širi smisao od navedene definicije brojačkih petlji. Format for-naredbe u C-u je:



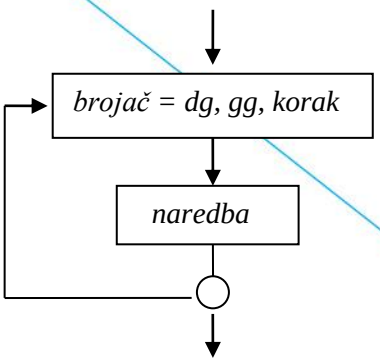
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
for (<izraz1>; <izraz2>; <izraz3>)
        <naredba>
```

gde:

- <izraz1> - vrši inicijalizaciju promenljivih koje se koriste u petlji (što može da bude postavljanje početne vrednosti brojača petlje),
- <izraz2> - predstavlja uslov na osnovu koga se odlučuje da li će se telo petlje još izvršavati ili se izvšenje petlje prekida – (uslov za izvršenje tela petlje može da bude dok brojač petlje ne postigne svoju gornju granicu)
- <izraz3> – definiše promenu vrednosti promenljivih koje se koriste u petlji. Navedena promena se vrši nakon svake iteracije (logično, tu se može definisati kako se menja vrednost brojača petlje nakon svake iteracije).

Bilo koji od izraza može da bude izostavljen, ali se znak ';' mora pisati. Dejstvo naredbe je: najpre se izračunava vrednost izraza <izraz1>, onda izraza <izraz2> i ako je on logički tačan, tj. ako je njegova brojna vrednost različita od nule, izvršava se naredba petlje. Nakon završetka svake iteracije vrši se izračunavanje vrednosti izraza <izraz3> i ponovo proverava vrednost izraza <izraz2>.




**Sl. 2.5 Dijagram toka brojačke petlje**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 3. Polja

Polja predstavljaju složenu lineranu strukturu podataka sa elementima istog tipa. Polja mogu biti jednodimenzionalna (nizovi), dvodimenzionalna (matrica),…

Pošto polja spadaju u referentni tip podataka u programskom jeziku C#, definiciju polja ćemo posebno posmatrati u programskom jeziku C++, a posebno u programskom jezicu C#.

### 3.1 Polja u programskom jeziku C++

Ukoliko se polje smešta u statičkoj zoni memorije, jednodimenzionalna polja se u programskom jeziku C++ se definišu na sledeći način:

<tip> <ime> [<veličina>];

gde je:

<ime> - ime polja;

<veličina> - broj elemenata polja;

<tip> - tip svakog pojedinačnog elementa polja.

Prvi element polja ima uvek indeks (poziciju) 0, drugi indeks 1, …, poslednji element ima indeks <veličina>-1.

**Primeri:**

```
int a[100],b[50];
float x[30];
double d[2000];
```

Jedan od načina da se pristupi elementima polja je korišćenjem indeksa (tj. pozicije elementa u polju). Za pristup elementima polja pomoću indeksa koristi se operator []. Opšti oblik izraza za pristup elementu polja je:

<ime_polja>[<izraz>]

**Primeri:**

… a[5] …

… b[i] …

… x[5*j+7]

Inicijalizacija polja se vrši tako što se lista početnih vrednosti elemenata polja se piše unutar para vitičastih zagrada.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**Primeri:**

char samoglasnik[5]={ 'A', 'E', 'I', 'O', 'U' };

char m_samoglasnik[]={ 'a', 'e', 'i', 'o', 'u' };

/*prevodilac sam izracunava broj elemenata */

int br_dana_u_mesecu[12]={31,28,31,30,31,30,31,31,30, 31,30,31};

Dvodimenzionalna polja u programskom jeziku C++ se definšu kao niz nizova:

<tip> <ime> [<broj_vrsta>][<broj_kolona>];

Pristup elementima ovako definisanog polja:

…<ime>[<indeks_vrste>][<indeks_kolone>]

Ukoliko se elementi polja smeštaju u dinamičkoj zoni memorije, za pristup elementima treba u statičkoj zoni memorije definisati pokazivač tipa elemenata polja:

` <tip> *<ime>;

<tip> - tip podataka na koji ukazuje pokazivačka promenljiva,

<ime> - ime pokazivačke promenljive.

Rezervacija prostora u dinamičkoj zoni memorije za smeštanje elemenata polja vrši se pomoću operatora **new**:

**Primer:**

Rezervisati memorijski prostor za vektor tipa int od N komponenata.

```
int *vektor, N;
cout << "unesite broj elemenata u nizu" << endl;
cin >> N;
vektor=new int[N];
if (vektor==0)
        cout << "nema mesta u mem." << endl;
else …
```

Oslobađanje dela memorijskog prostora na koji ukazuje neki pokazivač, vrši se poperatorom **delete**, na sledeći način:

delete <ime_pokazivaca>, ako pokazivac ukazuje na 1 podatak, ili

delete []<ime_pokazivaca>, ako pokazivac ukazuje na niz podataka.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 3.2 Polja u programskom jeziku C#

Jednodimenzionalna polja se u programskom jeziku C# se definišu na sledeći način:

```
<tip>[] <ime>;
```

Pošto je polje referentni tip podataka, ovim nije izvršena rezervacija prostora za njegove elemente. Rezervacija prostora se vrši naredbom:

```
<ime> = new <tip>[<izraz>] ;
```

Gde <izraz> predstavlja veličinu polja. Pristup elementima polja se I u ovom programskom jeziku vrši korišćenjem indeksa.

Dvodimenzionalna polja se u programskom jeziku C# mogu definisati na 2 načina:

```
<tip>[][] <ime>;
```

Ili :

```
<tip>[,] <ime>;
```

**Primer 1:** Dvodimenzionalno polje kao niz nizova.

```
int[][] a;

a=new int[m][n];

…

a[i][j]=…
```

**Primer 2:** Dvodimenzionalno polje kao matrica.

```
int[,]a;

a=new int[m,n];

…

a[i,j]=…
```

U programskom jeziku C# definisana je posebna programska struktra koja se koristi za obradu svih elemenata nekog polja. To je *foreach* petlja. *Foreach* petlja se osim za obradu elemenata polja može



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

koristiti i za obradu elemenata bilo koje kolekcije podataka koja implementira interfejs IEnumerable (o interfejsima će biti reči kasnije).

Format *foreach* petlje:

```
foreach (<tip> <ime> in <ime_kolekcije>)
        <naredba>
```

**Primer:** Sumiranje elemenata niza

```
int[] a = new int[n];
…
int s=0;
foreach( int x in a)
        s += x;
```

Ovakva petlja je potpuno ekvivalentna sledećoj:

```
foreach( int i=0; i<n; i++)
        s += a[i];
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 4. Klase

### 4.1 Definisanje klase

Klase su tipovi podataka kojima se opisuju objekti. Klasa opisuje osobine objekata (njihove attribute) i funkcije koje definišu ponašanje objekata.

Za sve članove klase (attribute i funkcije) definiše se pravo pristupa što odredjuje u kojim delovima programa se taj član može koristiti. U programkom jeziku C++ postoje 3 prava pristupa: private, protected i public. Privatni članovi klase su dostupni samo unutar klase u kojoj su deklarisani dok su atributi deklarisani kao *public* javni i može im se pristupiti iz bilo kog dela aplikacije. Članovi deklarisani kao *protected* se mogu koristiti u klasi u kojoj su deklarisani ali su dostupni i iz izvedenih klasa, o kojima će biti reči kasnije.

U programskom jeziku C# osim navedenih, postoje još 2 prava pristupa: internal I protected internal. Članovi sa pravom pristupa internal su dostupni u okviru asemblija u kojem je klasa definisana, a članovi sa prvom pristupa protected internal se mogu koristiti u okviru istog asemblija, kao i u izvedenim klasama koje mogu pripadati drugim asemblijima. (Asembli je datoteka, programska jedinica dobijena prevodjenjem jednog .Net projekta. Može biti izvršsni program, kada ima ekstenziju .exe, ili biblioteka klasa i metoda koja se prema potrebi povezuje sa drugim asemblijima koja ima ekstenziju .dll.)

Obično se atributi klase deklarišu kao private ili protected čime se ograničava direktan pristup atributima spolja, a za promenu njihovih vrednosti se moraju koristiti eksplicitno definisane funkcije deklarisane kao public.

Za svaku klasu u programskom jeziku C++ kriraju se po 2 fajla:

- *Header* fajl (ime.h) u kojem se nalazi defincija klase koja sadrži definicje atributa I deklaracije metoda,
- *Cpp* fajl u kojem se pišu implementacije funkcija članica klase.

U C#-u se za svaku klasu kreira 1 fajl. Iimplementacija funkcija članica se piše direktno u definiciji klase.

Definicija klase:

C++:

```
class <ImeKlase>
{
  public:
    <niz_javnih_clanova>
[ protected:
    <niz_zasticenih_clanova> ]
```

C#:

```
class <ImeKlase>
{
  <atributi>
  <metode>
}
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
[ private:
   <niz_privatnih_clanova> ]

};
```

Ispred svakog člana klase u C#-u se piše eksplicitno pravo pristupa. Ukoliko pravo pristupa nije navedeno, u programskom jeziku C++ podrazumevano pravo pristupa je private, a u programskom jeziku C# internal.

Deklaracija funkcijie u jeziku C++ sadrži: povratni tip ime, i tipove fiktivnih argumenata. Imena argumenata se mogu, ali ne moraju navoditi u deklaraciji funkcije:

```
<povratni_tip> <ime_funkcije> ( <argumenti>);
```

Implementacija funkcije članice klase će izgladati:

```
<povratni_tip>  <ime_klase>::<ime_funkcije>(<argumenti>)
{
        <telo_funkcije>
}
```

U programskom jeziku C# funkcia će biti definisana na sledeći način:

```
[<pravo_pristupa>] <povratni_tip> <ime_funkcije>(<argumenti>)
{
        ...
}
```

gde je:

- <ime_funkcije> - simboličko ime funkcije
- <telo_funkcije> - blok naredbi (može biti i prazno)
- <argumenti> - deklaracije formalnih parametara. Formalni parametri služe za prenos podataka iz pozivajućeg modula sa kojima će funkcija raditi nakon poziva. Izrazi čije se vrednosti dodeljuju fiktivnim parametrima funkcije u trenutku poziva nazivaju se stvarnim parametrima.
- <povratni_tip> – određuje tip vrednosti koju funkcija vraća. Funkcija može biti i tipa void što znači da ne vraća nikakvu vrednost.

Izvršenje funkcije se prekida naredbom return. Ova naredba može da vrati i rezultat funkcije koji preuzima pozivajuća funkcija. Rezultat koji funkcija vraća je izraz navedenog tipa.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Funkcija može da sadrži nijednu (ukoliko je tipa void), jednu ili više naredbi return.

Postoje dva oblika ove naredbe:

- return (<rezultat>);
- return; //ne vraca se nista - za funkcije tipa void

**Primer:** Definisaćemo klasu vozilo koja ima 2 atributa: brzinu i stanje kojim ćemo označavati da je motor uključen ili isključen (1 – uključen, 0 – isključen) i dve funkcije, ukljuci() i iskljuci(), za paljenje i gašenje motora, respektivno.

C++:

```
//vozilo.h
class vozilo {
  private:
    int stanje;
    int brzina;
  public:
    void ukljuci();
    void iskljuci();
};

//vozilo.cpp
void vozilo::ukljuci(){
    stanje = 1;
}
void vozilo::iskljuci(){
    stanje = 0;
}
```

C#:

```
class vozilo
{
  private int stanje;
  private int brzina;
  public void ukljuci()
  {
    stanje = 1;
  }
  public void iskljuci()
  {
    stanje = 0;
  }
}
```

Unutar jedne klase moguće je definisati i veći broj funkcija sa istim imenom koje se razlikuju po broju i tipovima argumenata. Ovakve funkcije se u literaturi sreću pod imenom *overload* funkcije. Tako u klasi vozilo moge da postoje 2 funkcije promeniBrzinu, jedna koja menja brzinu za zadatu vrednost, a druga koja brzinu uvek povećava za 1. Deklaracije ovih funkcija bi bile:

void promeniBrzinu ( );

void promeniBrzinu ( int korak );



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Parametri funkcije mogu da imaju i svoje podrazumevane vrednosti. Ako pogledajmo prethodne 2 funkcije, i jedna i druga bi imale isto dejstvo – povećavaju brzinu za neku vrednost. Prva funkcija je, praktično, specijalni slučaj druge, kada je korak = 1. U tom slučaju mžemo definisati jednu funkciju pri čemu bi argument korak imao podrazumevanu vrednost 1:

void promeniBrzinu ( int korak = 1 );

Ovo znači da, ukoliko u pozivu funkcije nije naveden korak promene brzine, njegova podrazumevana vrednost je 1.

Kada funkcija ima veći broj parametara, parametri koji imaju podrazumevane vrednosti se pišu na kraju liste argumenata.

Npr.

int f1( float a, float b, int c=0, int d=200 );

Funkcija f1 može da se poziva navođenjem 2, 3 ili 4 stvarna argumenta. Ukoliko je funkcija pozvana sa 3 argumenta, podrazumeva se da je u pozivu navedena vrednost parametra c, dok se za parametar d uzima njegova podrazumevana vrednost. Nikako u pozivu ne može da bude izostavljena vrednost parametra c, a navedena vrednost parametra d. Zbog toga je kod navođenja argumenata sa podrazumevanim vrednostima veoma važno voditi računa o njihovom redosledu.
Osnovna stvar o kojoj treba voditi računa pri kreiranju funkcija je način prenosa parametara. Parametri se u oba programska jezika mogu preneti funkciji po vrednosti i po referenci. Prenos parametara po vrednosti podrazumeva da se pri pozivu funkcije u operativnoj memoriji prave kopije za sve odgovarajuće parametre funkcije. Funkcija radi sa tim kopijama i u trenutku završetka rada funkcije te kopije se brišu iz operativne memorije. To automatski onemogućava da parametar funkcije bude promenjen u funkciji, a da to bude vidljivo u pozivajućem modulu.

**P r i m e r :**

```
void fun(float a)
{
        a = a*a + 234.12;
}

  float x;
        x=23.123;
        cout<<"Vrednost prom. x pre poziva funkcije” << x;
        fun(x);
        cout<<"Vrednost prom. posle poziva funkcije” << x;
        /* Odstampana je ista vrednost*/
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

U navedenom primeru štampa se vrednost promenljive x pre i posle poziva funkcije fun sa tim argumentom. U oba slučaja dobija se ista vrednost i pored toga što funkcija fun menja vrednost argumenta. Ove promene funkcija fun vrši nad kopijom argumenta tako da one nisu vidljive iz pozivajućeg modula.

Kada promena argumenata treba da bude vidljiva glavnom modulu, prenos treba vršiti po referenci. Ako se funkcija iz prethodnog primeradefiniše na sledeći način:

```
void fun(float& a)
{
        a = a*a + 234.12;
}
```

u trenutku poziva funkcije ne pravi se lokalna kopija za parametar, već je a drugo ime za stvarni parametar koji se prosledjuje funkciji. To znači da i glavna funkcija i funkcija fun rade sa istom memorijskom lokacijom, i sve promene koje izvrši funkcija nad parametrom, videće se i u glavnoj funkciji.

Rezultat funkcije u programskom jeziku C++, takođe, može biti prenesen po vrednosti i po referenci. Ukoliko se rezultat vraća po vrednosti, kreira se privremena kopija podatka koja se uklanja iz memorije odmah nakon što je upotrebljena u naredbi poziva. Pri prenosu po referenci, vraća se baš ona promenljiva sa kojom je i funkcija radila. Zato treba voditi računa o tome da po referenci sme da se vrati samo podatak koji postoji u memoriji pre I posle poziva funkcije, nikako promenljive koje su definisane u samoj funkciji.

U proramskom jeziku C# takodje postoji prenos parametara po vrednosti i po referenci, ali, u ovom programksom jeziku se smisao ova dva načina prenosa parametara razlikuju zavisno od toga da li se prenose podaci vrednosnog ili referentnog tipa. Doslovno, u ovomo programskom jeziku se parametri funkcije definišu kao:

- ulazni,
- izlazni (tako što se ispred definicije parametra stavi ključna reč **out**) i
- ulazno-izlazni (tako što se ispred definicije parametra stavi ključna reč **var**).

Parenos će se vršiti po referenci ako je parametar definisan kao izlazni, ili kao ulazno-izlazni.

Ukoliko je parametar vrednosnog tipa, prenos po vrednosti i po refernci će imati isto značenje kao u programkom jeziku C++. Medjutim, kod parametara referentnog tipa i prenos po vrednosti će omogućiti da funkcija menja parametar, ali ako je prenos izvršen po vrednosti nije moguće u funkciji kreirati novi objekat i dodeliti ga parametru. Kada je prenos izvršen po referenci (tj. ako je definisan kao izlazni ili kao ulazno-izlazni) u funkciji je moguće kreirati potpuno novi objekat i dodeliti ga parametru).



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 4.2 Funkcija main

Izvršenje svake C++ ili C# aplikacije počinje pozivom main metode. Main funkcija, kako joj samo ime kaže je glavna funkcija, nju poziva sam operativni sistem u trenutku poziva aplikacije.

U programskom jeziku C++:

- funkcija main se piše na globalnom nivou,
- funkcija main nema argumente ili ima 2 argumenta
  - **int argv** (predstavlja broj argumenata)**,**
  - **char** argc** (niz argumenata pri čemu se kao prvi argument nalazi ime aplikacije)**.**
- povratni tip funkcije main je **void** ili **int**.

U programskom jeziku C#:

- funkcija main je članica klase (klasa kojoj pripada funkcija main se naziva glavna klasa aplikacije),
- funkcja main je obavezno javna i statička,
- funkcija main nema argumente ili ima 1 argument
  - **strign[] args** (niz argumenata – ovaj niz ne sadrži ime aplikacije)**,**

### 4.3 Kreiranje objekata

Objekat se u programskom jeziku C++ može kreirati u statičkoj ili dinamičkoj zoni memorije. Kada se objekat smešta u dinamičkoj zoni memorije u statičkoj se mora definisati pokazivač koji će ukazivati na njega. U ovom programskom jeziku, svuda u programu gde se kreira objekat neke klase, mora da bude poznata definicija te klase, tj. mora se uključiti header fajl u kojem je klasa definisana.

**Primer:** Kreiraćemo funkciju main i u njoj definisati po jedan objekat klase vozilo u statičkoj i dinamičkoj zoni memorije.

(**main.cpp**)

```
#include ”vozilo.h”
void main()
{
  vozilo bmw;
  vozilo *yugo;
  yugo = new vozilo();
        ...
  delete yugo;
}
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Prvi način, statički:

```
vozilo bmw;
```

Klasa vozilo se koristi kao apstraktni tip podataka i kreira se objekat kao njen uzorak koji ima strukturu (atribute) opisanu klasom. Taj objekat se naziva *stalni objekat*.

Drugi način, dinamički:

```
vozilo *yugo;
```

U ovom slučaju yugo je pokazivač na objekat klase vozilo, objekat nije kreiran, već to treba eksplicitno da se uradi pomoću operatora **new**.

```
yugo = new vozilo();
```

Operator **new** služi za dinamičko alociranje (dodeljivanje) memorije (kao malloc u C-u). Tako napravljeni objekti su dinamički objekti.

Sada u memoriji postoje dva objekta klase vozilo kojima možemo manipulisati: mw i yugo.

U programu je moguće definisati i referencu (upućivač) na objekat klase vozilo. Referenca je drugo ime za neki podatak (objekat). Zbog toga reference moraju da se inicijalizuju prilikom definisanja:

```
vozilo& strano = bmw;
vozilo& domaće = *yugo;
```

U programskom jeziku C#, kao što je već rečeno, klase su referentni tipovi podataka pa će se objekti klasa uvek smeštati u dinamičkoj zoni memorije. To znači da kada definišemo promenljivu tipa klase, objekat nije kreiran, već ga treba eksplicitno kreirati operatorom new.

**Primer:** Kreiraćemo funkciju main u progamskom jeziku C# i u njoj kreirati objekat klase vozilo.

(**main.cs**)

```
public class Main
{
    public static void main()
    {
        vozilo bmw;
        bmw = new vozilo();
        ...
    }
}
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 4.4 Konstruktori

Konstruktor je posebna funkcija klase koja ima isto ime kao i klasa ( T() za klasu T), a služi da se definišu početne vrednosti podataka klase. Konstruktor se poziva u trenutku kreiranja objekta klase. Može da bude bez argumenata ili da ima jedan ili više argumenata. Takođe, moguće je da u klasi postoji više od jednog konstruktora, pri čemu svaki mora da ima različitu listu argumenata. Još jedna bitna osobina konstruktora je da nema povratni tip (ni void).

**Primer:** Proširićemo klasu vozilo tako što ćemo joj dodati i atribut za predstavljanje maksimalne brzine, kao i 2 konstruktora: bez argumenata koji maksimalnu brzinu postavlja na 180 i konstruktor koji maksimalnu brzinu postavlja na zadatu vrednost.

C++:

```
//vozilo.h
        class vozilo {
          private:
            int stanje;
            int brzina;
            intn maxBrzina;
          public:
            vozilo();
            vozilo(int maxV);
            ...
        };


//vozilo.cpp
vozilo::vozilo(){
    stanje = 0;
    brzina = 0;
    maxBrzina = 180;
}

vozilo::vozilo(int maxV){
    stanje = 0;
    brzina = 0;
    maxBrzina = maxV;
}
```

C#:

```
class vozilo
{
  private int stanje;
  private int brzina;
  private int maxBrzina;

  public vozilo()
  {
    stanje = 0;
    brzina = 0;
    maxBrzina = 180;
  }

  public vozilo(int maxV)
  {
    stanje = 0;
    brzina = 0;
    maxBrzina = maxV;
  }
}
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

U programkom jeziku C++ inicijalizacija atributa može da se navede u telu konstruktora, ali i u delu za inicijalizaciju. Deo za inicijalizaciju se piše iza zagrlavlja konstruktora, a pre njegovog tela i od zaglavlja je odvojen simbolom :. U delu za inicijalizaciju pozivaju se konstruktori atributa kao i konstruktor roditeljske klase (o tome će kasnije biti reči). Tu se obavezno inicijalizuju reference i konstantni atributi.

Moguća implementacija konstruktora u programskom jeziku C++ bila bi:

(**vozilo.cpp**)

```
...

vozilo::vozilo(int maxV)
: brzina(0), maxBbrzina(maxV), stanje(0)
{
}
```

I u programskom jeziku C# može da postoji deo za inicijalizaciju, ali se u njemu mogu samo pozivati drugi konstruktori (iste ili roditeljske klase). Za poziv konstruktora iste klase koristi se referenca this.

**Primer:** Poziv drugog konstruktora iste klase u programskom jeziku C#.

```
class vozilo
{
  private int stanje;
  private int brzina;
  private int maxBrzina;

  public vozilo()
  {
    stanje = 0;
    brzina = 0;
  }

  public vozilo(int maxV) : this()
  {
    stanje = 0;
    brzina = 0;
    maxBrzina = maxV;
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
  }
}
```

Koji će konstruktor biti pozvan određuje se na osnolu liste argumenata.

**Primer:** Pkreiranje objekata pozivom različitih konstruktora.

C++:

```
//korišćenjem konstruktora bez
//argumenata:
vozilo bmw;
vozilo *yugo = new vozilo;

//korišćenjem konstruktora sa
//argumemtom:
vozilo bmw(220);
vozilo *yugo = new vozilo(100);
```

C#:

```
//korišćenjem konstruktora bez
//argumenata:
vozilo bmw = new vozilo();

//korišćenjem konstruktora sa
//argumemtom:
vozilo bmw = new vozilo(250);
```

Ukoliko u klasi nije definisan ni jedan konstruktor prevodilac će sam definisati konstruktor bez argumenata (takozvani podrazumevani konstruktor).

Posebna vrsta konstruktora koji se u programksom jeziku C++ veoma često kojiste su konstruktori za kopiranje. Konstruktori za kopiranje prave kopiju postojećeg objekta. Konstruktor za kopiranje klase vozilo izgledao bi:

```
vozilo::vozilo(vozilo& v)
{
        this->brzina = v.brzina;
        this->max_brzina = v.max_brzina;
        this->stanje = v.stanje;
}
```

Postojeći objekat se konstruktoru za kopiranje obavezno prenosi po referenci. To je razumnljivo jer, ukoliko se parametar prenosi po vrednosti, u trenutku poziva funkcije se poziva konstruktor za kopiranje za sve objekte koji se prenose po vrednosti. Ako bi se parametar prenosio po vrednosti, dobila bi se beskonačna rekurzija koja bi na kraju dovela do „pada“ aplikacije.

Konstruktor za kopiranje se obavezno pravi kada klasa sadrži dinamičke atribute. Ukoliko nije napravljen konstruktor za kopiranje, kada treba praviti kopiju objekta svi atributii postojećeg objekta se jednostavno prepišu u atribute novog. Ako su atributi postojećeg objekta pokazivači na druge objekte, kada se napravi novi objekat, vrednosti pokazivča će se prepisati, tako da će i stari i novi objekat ukazivati na iste dinamičke članove. To će dovesti do problema pri promenama takvih



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

članova (ako se u starom objekat nešto izmeni, promena će se odraziti i na novi objekat, i obrnuto), ali će najveći problem biti pri brisanju objekata. Ako se obriše, recimo, novi objekat i on ispravno obriše i sve svoje dinamičke članove, aplikacija će ponovo da „padne“ ukoliko se iz starog objekta pokuša da pristupi tom dinamičkom članu, ili u trenutku kada se briše stari objekat pa i on pokuša da obriše svoje dinamičke članove (koji su već obrisani).

### 4.5 Destruktori

Specijalna funkcija klase koja se implicitno poziva u trenutku kada se objekat klase briše iz memorije je **destruktor** Destruktor je funkcija bez argumenata i u klasi može da postoji samo jedan destruktor.

Identifikator destruktora je isti kao identifikator klase, sa prefiksom ~ na početku (~T() za klasu T). Dodaćemo našem primeru sledeći kod:

(**vozilo.h**)

```
~vozilo();
```

(**vozilo.cpp**)

```
vozilo::~vozilo()
{
  // telo desktruktora
}
```

Ukoliko se ne definiše destruktor, podrazumeva se destruktor sa praznim telom.

Kada i zašto koristiti destruktor? U programskom jeziku C++ je destruktor, kao i konstruktor za kopiranje, obavezan ukoliko smo u okviru nekog objekta kreirali dinamičke strukture (lančane liste, stabla, ...) ili objekte. U tom slučaju se u destruktoru vrši oslobađanje memorije koju zauzimaju te dinamičke strukture ili objekti.. U praksi se dešava da dođe do gomilanja neuništenih dinamičkih struktura i objekata ako se objekat u kome se kreiraju više puta kreira i uništava, a da se pri tome ne uništavaju uredno kreirane strukture. To dovodi do toga da sistem ostane bez memorije i aplikacija “pada” (*memory leak*). Takve greške se veoma teško otkrivaju jer aplikacija neko vreme radi stabilno, a onda u nekom proizvoljnom trenutku “padne”. To se takođe, dešava nezavisno od procesorske i memorijske snage računara. Što računar ima veću memoriju, to je ovu grešku teže otkriti (jer će sistem ređe “padati”)!

Kada se destruktor automatski poziva? Razmotrićemo prvo programski jezik C++:

Vratimo se našem primeru u kome je istanciran objekat bmw:

```
void main()
{
  vozilo bmw(250);
  // proizvoljan kod
}
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Vreme života objekta bmw će biti sve dok se funkcija main izvršava. Onog trenutka kad se main završi, destruktor se automatski poziva i uništava objekat. To znači da ukoliko se neki objekat kreira u **statičkoj** zoni memorije, taj objekat će postojati sve do završetka bloka u kojem je definisan.

Operator **delete** služi za dinamičko dealociranje (oslobađanje) memorije. Pomoću njega se mogu eksplicitno uništiti dinamički kreirani objekti. Na primer, dinamički kreirani objekat jugo se može uništiti sa:

```
void main()
{
  vozilo *yugo;
  yugo = new vozilo();
  delete yugo;
}
```

Šta se dešava u programskom jeziku C#? U njemu su objekti uvek u dinamičkoj zoni memorije, a nikada se ne koristi operator delete. Dakle, u jeziku C# referenca na objekat postoji dok se izvršava blok u kojem je definisana. Objekat će se ukloniti iz memorije prilikom prvog izvršenja „garbage colectora“ nakon završetka izvršenja tog bloka i u tom trenutku će se pozvati destruktor.

### 4.6 Pristup funkcijama i atributima klase

Da bi se pristupilo funkciji nekog objekta, potrebno je navesti ime **objekta**, propraćeno tačkom (.) i zatim ime **funkcije** sa odgovarajućom sintaksom (stvarni parametri u okviru zagrada funkcije, ukoliko postoje). Navedena funkcija mora da bude deklarisana kao javna u okviru klase kojoj objekat pripada.

<objekat>**.**<operacija>

Ukoliko navodimo pokazivač na objekat, onda koristimo strelicu (**->**) umesto tačke.

<pokazivac_na_objekat> **->** <operacija>

Recimo da hoćemo da uključimo motor prvog vozila. To ćemo uraditi na sledeći način:

bmw.ukljuci();

Operacija ukljuci() je promenila samo atribut stanje objekta nad kojim je primenjena.

Uzimajući primer sa vozilo, uključićemo motore oba vozila, a zatim isključiti motor prvog vozila.

**Primer:** Poziv funkcija članica klasa.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

C++:

```
vozilo bmw;
vozilo *yugo = new vozilo;
bmw.ukljuci();
yugo->ukljuci();
```

C#:

```
vozilo bmw = new vozilo();
bmw.ukljuci();
```

Ista sintaksa važi i za pristup atribitima klase ako su deklarisani kao javni (public), na primer, ako atribut stanje definišemo kao javni, promenu stanja bismo mogli da izvršimo na sledeći način:

```
bmw.stanje = 1;
```

### 4.7 Funkcije za pristup atributima klase u jeziku C++ i svojstva u jeziku C#

Poželjno je izbegavati direktno pristupanje atributima klase iz funkcija drugih klasa. Time se narušava koncept **inkapsulacije**. Zbog toga se obično atributi klase deklarišu kao privatni, a za pristup atributima koristite se public funkcije klase za postavljanje i preuzimanje vrednosti atributa (u engleskoj literaturi poznate kao set i get funkcije).

Funkcije za pristup atributima se u programskom jeziku C++ obično prave kao **inline** funkcije. Telo inline funkcija se piše direktno u definiciji klase, ili se u definiciji klase ispred deklaracije funkcije navede ključna reč inline, a implementacija se navodi iza definicije klase (u .h fajlu). Inline funkcije štede vreme izvršenja programa na taj način što prevodilac njihov kod umeće u kod u svakoj tački njihovog poziva. Zbog toga se kao inline definišu samo funkcije koje se veoma često pozivaju, a čije je telo veoma kratko. Neki kompilatori čak nameću ograničenje da funkcije koje sadrže cikluse ne mogu biti definisane kao inline funkcije.

Klasi vozilo dodaćemo funkcije za postavljanje i preuzimanje vrednosti brzine.

(**vozilo.h**)

```
class vozilo
{
  private:
    int stanje;
    int max_brzina;
    int brzina;
  public:
          vozilo();
          vozilo(int max);
    void ukljuci();
    void iskljuci();
    void ubrzaj();
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
    void uspori();
        const int uzmiBrzinu() const {
            return brzina;
        }
        inline void postaviBrzinu( int brzina );
};
void vozilo::postaviBrzinu( int brzina) {
        this->brzina = brzina;
}
```

U ovom primeru može se uočiti još jedna novina. Ispred povratnog tipa funkcije navedena je ključna reč const, što znači da je vrednost koju funkcija vraća konstantna (ne sme se menjati dalje u programu). Iza zaglavlja funkcije navedena je takođe klučna reč const. To označava da je funkcija konstantna, tj. da funkcija ne menja objekat nad kojim je pozvana. Kad god funkcija ne menja vrednosti atributa treba je definisati kao konstantnu jer se nad konstantnim objektima klase mogu pozivati samo konstantne funkcije. To znači da u programu možemo definisati konstanti objekat:

```
const voziloNaIzlozbi(200);
```

Nad ovako definisanim objektom dozvoljeno je pozvati samo funkciju uzmiBrzinu jer je trenutno u klasi samo ona definisana kao konstantna.

U programskom jeziku C# kada spolja treba koristiti attribute klase, definišu se svojstva (property). Spolja gledano, property se vidi kao javni atribut klase, a unutar klase property čini get i/ili set metoda.

Definisaćemo property za pristup atributu brzina u klasi vozilo:

```
class vozilo
{
  private int brzina;
  ...
  public int Brzina
  {
    get {
      return brzina;
    }
    set {
      brzina = value;
    }
  }
}
```

Kao što se iz primera vidi, set metoda ima obavezni parametar (value) koji se eksplicitno ne delklariše.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Kada se propertiju dodeljuje vrednost, poziva se set metoda, a kada se koristi vrednost propertija, koristi se njegova get metoda:

```
//poziv set metode
bmw.Brzina = 100;
//poziv get metode
if (bmw.Brzina>100) …
```

### 4.8 Zajednički članovi klasa

Recimo da hoćemo da napravimo lančanu listu u kojoj svaki element sadrži podatak o ukupnom broju elemenata te liste u tom trenutku. Ukoliko bi obrisali jedan element liste, morali bi, prolaskom kroz celu listu, da ažuriramo podatak o broju elemenata u svakom elementu liste. To je prilično neefikasno i loše rešenje. Idealno bi bilo ukoliko bi svi elementi liste čitali vrednost tog podatka sa jednog mesta.U tom slučaju ne bi bilo problema sa ažururanjem liste. *Zajednički atributi klase* upravo to i rade. Ukoliko je broj elemenata liste zajednički podatak, postojaće samo jedna kopija tog podatka za sve elemente te liste. Isti prtincip važi i za *zajedničke funkcije klase*.

Stavljanjem modifikatora static na početak definicije ili deklaracije nekog člana date klase, taj član postaje zajednički za sve objekte te klase.

Zajednički atibuti su slični globalnim podacima, ali se njihove vrednosti mogu menjati samo na način koji odgovara opisu klase. Zajedničkim članovima se može pristupiti i kada ni jedan objekat klase nije kreiran.

**Primer:** Definisaćemo klasu Abc koja će imati jedan zajednički i jedan običan (pojedinačan) atribut, i jednu zajedničku i dve pojedinačne funkcije.

C++:

(**Abc.h**)

```
class Abc
{
  static int a;
  int b;
public:
  static int getA();
  int getB();
  void incA();
};
```

(**Abc.cpp**)

```
int Abc::a = 10;
int Abc::getA()
```

C#:

(**Abc.cs**)

```
class Abc
{
  private static int a=10;
  private int b;
  public static int getA()
  {
    return a;
  }
  public int getB()
  {
    return b;
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
{
        return Abc::a;
}
int Abc::getB()
{
        return b;
}
void Abc::incA()
{
        Abc::a ++;
}
```

```
  }
  public void incA()
  {
    a++;
  }
}
```

Kao što se iz primera vidi, zajednički atribut je u programskom jeziku C++ u definiciji klase samo samo **deklarisan**, ali nije i **deifinisan**. Definicija je navedena u implementacionom fajlu (Abc.cpp). Definicija atributa mora obavezno da se izvrši pre kreiranja bilo kog objekta te klase, ili pre pozivanja funkcije main().

Funkcija getA() je zajednička, i kao takva može pristupati samo zajedničkim atributima, a ne i pojedinačnim. Sa druge strane, pojedinačna funkcija može pristupati i pojedinačnim i zajedničkim atributima.

Pristup zajedničkim članovima:

C++:

```
<ime_klase>::<ime_clana>

//za konkretan primer:
Abc::getA()
```

C#:

```
<ime_klase>::<ime_clana>

//za konkretan primer:
Abc.getA()
```

Pristup zajedničkim članovima je moguć I korišćenjem objekata (na isti način kako se pristupa I pojedinačnim članovima), ali se to ne prporučuje.

### 4.9 Prijateljske funkcije i klase u programksom jeziku C++

Prijateljska funkcija klase je funkcija koja nije član klase, ali ima pristip privatnim članovima te klase. To mogu da budu globalne funkcije ili funkcije neke druge klase.

Definisaćemo prijateljsku funkciju klase vozilo koja će upoređivati maksimalne brzine dva vozila. Deklaracija prijateljske funkcije se navodi u definiciji klase tako što se na početku piše modifikator friend.

(**vozilo.h**)

```
class vozilo
{
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
...
        friend bool uporediBrzine(vozilo& v1, vozilo& v2){
                if (v1.max_brzina > v2.max_brzina) return true;
                else return false;
        }
};
```

Definicija prijateljske funkcije može da se izvši na dva načina. U oba slučaja će se smatrati globalnom funkcijom! Prvi je **inline**, tj. u definiciji klase.

Argumenti funkcije uporediBrzine su reference na objekte klase vozilo.

Recimo da hoćemo da u funkciji main uporedimo maksimale brzine dva automobila koristeći funkciju uporediBrzine:

(**main.cpp**)

```
#include <iostream>
#include "vozilo.h"
using namespace std;

void main()
{
        vozilo bmw(230), ferrari(340);
        if (uporediBrzine(bmw, ferrari))
                cout << "Bmw je brzi";
        else
                cout << ("Ferrari je brzi");
}
```

Funkcija uporediBrzine vraća true ukoliko je maksimalna brzina prvog objekta veća od maksimalne brzine drugog (što u gornjem primeru nije slučaj).

Drugi način je da se prijateljska funkcija deklariše u zaglavlju (vozilo.h) ali bez bloka naredbi (tela funkcije):

```
friend void ispisiKarakteristike(vozilo& v);
```

a zatim da se u nekom implementacionom fajlu (recimo vozilo.cpp) definiše kao globalna funkcija:

```
void ispisiKarakteristike(vozilo& v)
{
        cout << ”max. brzina: " << v.max_brzina << endl;
        cout << "trenutna brzina: " << v.brzina << endl;
        cout << "stanje motora: " << v.stanje << endl;
}
```

Zašto koristiti prijateljske funkcije klase?

Njihova najčešća upotreba je pri **preklapanju operatora** (operator overloading) o čemu će biti više rečeno kasnije. Može se reći da se njihovom upotrebom povećava **efikasnost** jer direktno pristupaju



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

privatnim elementima, izbegavajući indirektni pristup preko funkcija klase. Još jedna od pogodnosti je i to što je upotreba tih funkcija prirodnija onima koji su navikli na proceduralno programiranje. Umesto bmw.uporediBrzine(ferrari) možemo pisati uporediBrzine(bmw, ferrari). Taj efekat ne možemo postići definisanjem proizvoljne globalne funkcije jer ona ne bi mogla da pristupi privatnim članovima klase.

Zašto NE koristiti prijateljske funkcije klase?

Što se tiče negativnih strana korišćenja prijateljskih funkcija, one predstavljaju **bočne veze** u hierarhiji klasa. Vodi se polemika da li one **krše koncept inkapsulacije**, ili samo proširuju javni interfejs. Programski kod koji ih koristi će generalno biti **teži za razumevanje** i **podložniji greškama**. Zbog toga, treba biti oprezan pri korišćenju prijateljskih funkcija.

Ukoliko više funkcija druge klase treba da budu prijateljske našoj klasi, moguće je celu tu drugu klasu proglasiti za prijateljsku klasu naše klase.

## 4.10 Operatorske funkcije

Operatorskim funkcijama se definiše značenje standatdnih operatora za korisničke tipove podataka. Npr. zna se da operator + primenjen nad numeričkim tipovima podatak znači sabiranje, primenjen nad objektima klase string znači nadovezivanje... Ako želimo da omogućimo da se operator + primenjuje i nad objektima naše klase, moramo definisati njegovo značenje, odnosno kreirati operatorsku funkciju +.

Pošto se operatorske funkicje različito implementiraju u programskim jezicima C++ i C#, posmatraćemo odvojeno definisanje ovih funkcija u svakom od navedenih jezika.

### 4.10.1 Operatorske funkcije u programskom jeziku C++

Operatorske funkcije se mogu definisati ili članice klase ili kao globalne funkcije (prijateljske) koje imaju opšti oblik:

operator ***op***

gde je ***op*** simbol odgovarajućeg operatora (npr. operator+, operator--).

Kako se funkcije članice klase uvek pozivaju za neki objekat te klase, u slučaju operatorskih funkcija, one će se uvek pozivati za prvi operand u izrazu. To znači da funkcija može biti članica klase samo ukoliko je prvi operand u izrazu koji želimo da kreiramo objekat te klase. Ukoliko je operator binaran, drugi operand će biti parametar funkcije.

Pretpostavimo da imamo klasu za predstavljanje kompleksnih brojeva. Recimo da želimo da realizujemo operatorske funkcije za sledeće operacije:

- Sabiranje 2 kompleksna broja,



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Sabiranje kompleksnog i realnog broja,
- Sabiranje realnog i kompleksnog broja.

Prve 2 navedene funkcije mogu da budu članice klase za predstavljanje kompleksnih brojeva, dok poslednja mora biti realizovana kao globalna. Globalne operatorske funkcije imaju onoliko parametara kolika je n-arnost operatora (za unarne operatore 1, za binarne 2).

Realizacija navedenih operatorskih funkcija za rad sa kompleksnim brojevima:

(**Complex.h**)

```
class Complex
{
private:
  double real;
  double imag;
public:
  Complex(double r, double i);
  Complex operator+(Complex& c);
  Complex operator+(float f);
  friend Complex operator+(float f, Complex c);
};
```

(**Complex.cpp**)

```
#include „Complex.h“
Complex::Complex(double r, double i)
{
  real = r;
  imag = i;
}


Complex Complex::operator+ (Complex& c)
{
        Complex b;
        b.real = this->real + c.real;
        b.imag = this->imag + c.imag;
        return b;
}
Complex Complex::operator+ (float f)
{
        Complex b;
        b.real = this->real + f;
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
        b.imag = this->imag;
        return b;
}
Complex operator+ (float f, Complex& c)
{
        Complex b;
        b.real = f + c.real;
        b.imag = c.imag;
        return b;
}
```

Pozivi ovih opetatorskih funkcija:

```
        Complex a(2.4, 3.5), b(3.2,4.3), c(0,0);
        float x=3.4f;
        ...
        c=a+b;
        ...
        c=a+x;
        ...
        c=x+b;
```

U prethodnom primeru iskorišćen je i operator = kojim se objektu c dodeljuje objekat koji predstavlja rezultat navedenih operacija sabiranja. Na taj način se svi atributi objekta koji se nalazi sa desne strane operatora dodele dodeljuju odgovarajućim atributima objekta koji se nalazi sa njegove leve strane. Ovo je, opet, prihvatljivo samo ukoliko objekti ne sardže dinamičke atribute. U suprotnom, i operator = mora biti implementiran. Problemi koji nastaju pri realizaciji operatora = biće pokazani na primeru klase Vector.

(**Vector.h**)

```
class Vector
{
private:
        int size;
        int* elements;
public:
        Vector ( int max )
        {
                size = max;
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
		elements = new int[max];
	}
	~Vextor()
	{
		delete [] elements;
	}
	Vector& operator=( const Vector& v);
} ;
```

Prvi problem koji se javlja pri implementaciji operatora = je da vektori sa leve i sa desne strane operatora ne moraju biti iste dužine. Zbog toga u vektoru kojem se dodeljuje vrednost treba obrisati prethodni niz elemenata i kreirati novi odgovarajuće dužine. Drugi problem koji se odmah uočava je da će ovo brisanje dovesti do problema ukoliko se sa leve strane operatora nađe isti taj vektor. Zbog toga je najkorektnije operator = u ovom slučaju realizovati na sledeći način:

(**Vector.cpp**)

```
Vector& Vector::operator=( const Vector& v )
{
	if ( this != &v )
	{
		size = v.size;
		delete [] elements;
		elements = new int[size];
		for( int i=0; i<size; i++ )
			elements[i] = v.elements[i];
	}
	return *this;
}
```

U jeziku C++ za operatorske funkcije postoje ograničenja:

- Ne mogu da se pišu operatorske funkcije za operatore pretprocesora.
- Ne mogu da se pišu operatorske funkcije za operatore ?:(ternarni), .(*116orm od), .*(pokazivač na člana) ili :☹razrešenje dosega).
- Ne mogu da se izmišljaju novi operatori.
- Ne mogu da se menjaju asocijativnost, prvenstvo ili vrsta (binarni ili unarni) operatora.

Pravila za pisanje operatorskih funkcija kao funkcija članica klase su:

- *Unarni operatori* – Funkcija članica nema argumente. Jedan jedini argumet je objekat koji se koristi za poziv funkcije članice
- *Binarni operatori* – Funkcija članica ima jedan argument. Levi operand je objekat koji se koristi za poziv funkcije, a desni operand je argument funkcije članice.
- *Unarni i binarni operatori* – Operatorska funkcija mora da se pozove objektom ispravnog tipa. Prevodilac će izvršiti konverziju levog (za binarne operatore) ili jedinog operanda (za unarne



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

operatore). Uobičajena pravila za konverziju se primenjuju na desni operand (za binarne operatore).

Osobine samostalnih operatorskih funkcija:

- Za binarne operatore samostalne operatorske funkcije se ponašaju simetrično. I levi i desni operand podležu uobičajnoj konverziji i nije nužno da levi operand bude objekat.
- Samostalne operatorske funkcije mogu da se pišu i za klase iz biblioteka (klase koje niste sami kreirali).

Međutim, ne mogu sve operatorske funkcije da budu samostalne. Tako, sledeće funkcije moraju da se pišu isključivo kao funkcije članice klase: operator=(), operator()(), operator[](), operator->().

Izbor povratnih tipova operatorskih funkcija:

- Operatori koji menjaju levi operand (npr. =) treba da vrate 117ormalin117 na levi operand.
- Operatori koji biraju ili isporučuju operande (npr. [], ->, ()) obično treba da vrate 117ormalin117 na izabranu vrednost.
- Operatori koji izračunavaju novu vrednost, a ne menjaju svoje operande (operatori nad bitovima, +, -, *, /, %, &, \, ^, &&, ||, unarni (-, +), !~) obično treba da vrate kopiju lokalno kreiranog objekta, tj. da lokalne objekte vrate po vrednosti.
- Preinkrement i predekrement – obično treba da vrate 117ormalin117 na operand.
- Postinkrement i postdekrement – obično treba da sačuvaju kopiju svog operanda, izmene operand i zatim vrate kopiju.

U sledećem primeru je isti operator realizovan na dva načina, kao prefiksni i postfiksni.

```
class demo{
privat:
        int val;
public:
        demo& operator++()              // 117ormal
        {
                val++;
                return *this;
        }
        demo operator++(int)            // postfiks
        //da bi se realizovao kao postfiksni, uvodi se
        // 117ormalin argument koji se u telu funkcije i ne koristi
        {
                demo tmp;
                tmp.val=val;
                val++;
                return tmp;     //vraća objekat po vrednosti
        }
};
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 4.10.2 Operatorske funkcije u programskom jeziku C#

Operatorske funkcije u programskom jeziku C# se uvek prave kao statičke i kao javne. Jedino ograničenje koje treba poštovati je da bar jedan operand mora da bude tipa klase u kojoj je funkcija definisana.

Implementiraćemo iste opratorske funkicje za rad sa kompleksnim brojevima u jeziku C#:

(**Complex.cs**)

```
class Complex
{
  private double real;
  private double imag;

  public Complex(double r, double i)
  {
    real = r;
    imag = i;
  }

  public static Complex operator+ (Complex c1, Complex c2)
  {
        Complex b;
        b.real = c1.real + c2.real;
        b.imag = c1.imag + c2.imag;
        return b;
  }

  public static Complex operator+ (Complex c,float f)
  {
        Complex b;
        b.real = c.real + f;
        b.imag = c.imag;
        return b;
  }

  public static Complex operator+ (float f, Complex& c)
  {
        Complex b;
        b.real = f + c.real;
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
        b.imag = c.imag;
        return b;
  }
}
```

Pravila za definisanje operatorskih funkcija u C#-u:

1. Kada 2 operatora predstavljaju negaciju jedan drugog, nije moguće preklopiti jedan, a ne preklopiti drugi za neki tip podataka. Primeri takvih parova operatora su:
   - **== i !=**
   - **< i >**
   - **<= i >=**
2. U C#-u je moguće preklopiti sledeće operatore:
   - Unarne operatore: **+ - ! ~ ++ --**
   - Binarne operatore: **+ - * / % & | ^ << >> == != > < >= <=**
   - Operatore za implicitnu i eksplicitnu konverziju tipova
   - Logičke literale (**true** i **false**)
3. U C#-u nije moguće perklopiti operaore: **. = && || ?: new typeof as is**

Primer preklapanja operatora za implicitnu I eksplicitnu konverziju tipova, kao I za preklapanje logičkih 119iteral:

```
class Complex
{
        …
  public static implicit operator double(Complex c)
  {
        return Math.sqrt(c.re*c.re+c.im*c.im);
  }
  public static explicit operator double(Complex c)
  {
        return Math.sqrt(c.re*c.re+c.im*c.im);
  }
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
  public static operator false(Complex c)
  {
   return c.re==0 && c.im==0;
  }
  public static operator true(Complex c)
  {
   return c.re!=0 || c.im!=0;
  }
}
```

Ovi operatori se koriste na sledeći način:

```
Complex c(10,10);
double d;
d=c; //implicitna konverzija
d = (double) c;   //eksplicitna konverzija
if (c) ...   //konverzija u logički literal
```

### 4.10.3 Indekseri u programskom jeziku C#

Indekser je specijalni operator ([]) koji se koristi za pristup elementima niza. U programskom jeziku C++ se indeksni operator definše kao obična operatorska funkcija. Specifičnost ovog operatora je da izabrani element niza može da se menja. Dakle, dozvoljena je naredba oblika:

```
A[i] = x;
```

U C++-u je ovo jednostavno postići jer funkcija može da vrati rezultat po referenci, pa samim tim ta rezultujuća vrednost se može menjati. Da bi se u C#-u omogućilo da se podatak selektovan operatorom [] može menjati, ovaj operator se predifiniše na specifičan način – definiše se kao property. Ima svoju get i set metodu, pa će se u zavisnosti da li se selektovani podatak koristi, ili mu se dodeljuje vrednost, pozvati jedna ili druga metoda.

**Primer:**

```
public class Imenik
{
  string[] imena;
  public string this[ int indeks ]
  {
        get {
                return imena[indeks];
        }
```

```
        set {
                imena[index]= value;
        }
    }

    public string this[ string indeks ]
    {
      get {
                int i=0;
                while (i<imena.Length && !imena[i].Equals(indeks))
                        i++;
                return i==imena.Length ? null : imena[i];
      }
    }
  }
```

# 5. Nasleđivanje klasa

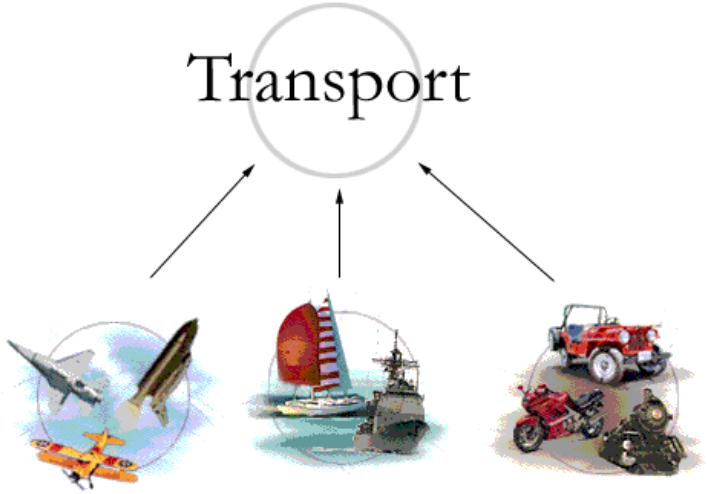
### 5.1 Generalizacija

Pri razmatranju skupa objekata, pogodno je grupisati ih po sličnim atributima i zajedničkim funkcijama. Na primer, hoćemo da napravimo klasu koja će predstavljati sve objekte koji služe za transport. Ona bi uključivala objekte aviona, objekte automobila ili čak objekte brodova. Drugim rečima, ona bi objedinjavala klase kopneniTransport, vazdusniTransport i morskiTransport. Nazovimo tu klasu Transport. Ovu klasu smo dobili procesom **generalizacije**.

Generalizacija je proces kojim se identifikuju i definišu zajednički atributi i funkcije u skupu objekata. Identifikovanjem zajedničkih delova sistema, generalizacija smanjuje redundansu (višak) u razvoju i omogućava ponovno korišćenje pojedinih delova.

### 5.2 Nasleđivanje

Klasa Transport definisana u prethodnom razmatranju a se zove **superklasa** (roditeljska klasa), dok se klase vazdusniTransport, morskiTransport i kopneniTransport nazivaju **podklase (izvedene klase)**. Dakle, izvedene klase nasledjuju svoju roditeljsku klasu. Klase povezane procesom nasleđivanja obrazuju hijerarhijskiu strukturu. Hierarhija klasa izvedenih iz klse Transport prikazana je je prikazana na sledećoj slici:




**Slika 0.1** Hierarhija klasa

Atribute klase Transport (npr. maksimalan broj putnika) kao i funkcije te klase (transportuj, stani, kreni) su **nasledile** podklase. Može se takođe reći da su podklase izvedene iz roditeljske klase.

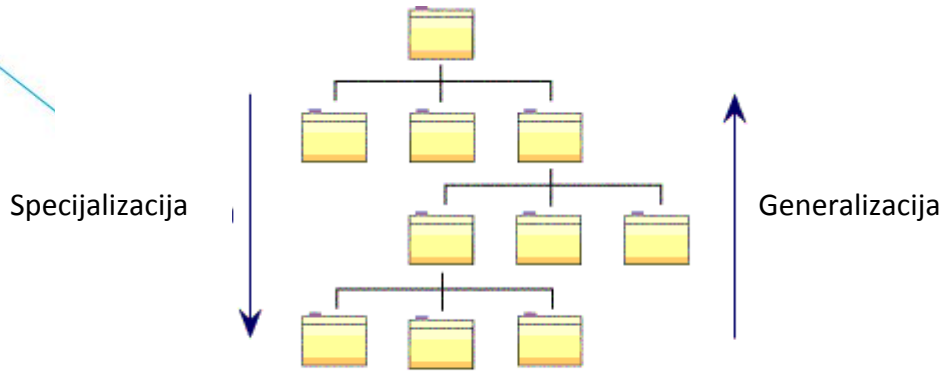


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Nasleđivanje je mehanizam definisanja novih klasa od već postojećih klasa. Tako napravljene klase dele, tj. nasleđuju članove roditeljske klase.

Zbog čega je nasleđivanje korisno? Ukoliko izvršimo bilo kakvu izmenu među članovima roditeljske klase (promena ili dodavanje nekog atributa ili funkcije), ta izmena će se automatski (tj. bez ikakve naše intervencije) odraziti i na sve podklase (potomke) koje su izvedene iz roditeljske klase.

Nasleđivanje se često predstavlja pomoću stabla. Kretanjem niz stablo, klase postaju sve više specijalizovane. Kretanjem uz stablo, klase postaju mnogo generalnije.




**Slika 0.2** Specijalizacija i generalizacija

Definicija izvedene klase:

C++:
```
class <ime_klase>
:<vrsta_nasledjivaja><superklasa>
[<vrsta_nasledjivaja><superklasa>]
…
{
  …
};
```

C#:
```
class <ime_klase>
: <superklasa>
{
  …
}
```

Kap što se iz definicije vidi, u programkom jeziku C++ klasa može biti izvedena iz više roditeljk+skih klasa, dok u C#-u, klasa može da nasledi samo jednu roditeljsku klasu. Drugo, u C++-u se definiše i način nasleđivanja koji određuje kako će se u izvedenoj klasi videti članovi roditeljske klase. Način nasleđivanja može biti: public, protected i private. Značenja ovih načina nasleđivanja prikazana su u tabeli 5.1. Iz tabele se vidi da kada se koristi javno nasleđivanje pravo pristupa članovima roditeljske klase, ostaje isto i u izvedenoj. Zbog toga, kada se koristi javno nasleđivanje, objekti izvedenih klasa u potpunosti mogu da zamene objekte roditeljskih klasa.



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Kako se u praksi najčešće koristilo javno nasleđivanje, u novijim objektno-orijentisanim jezicima (pa i u C#-u) postoji samo jedna vrsta nasleđivanja koja odgovara javnom nasleđivanju u C++-u.

**Tabela 0.1 Vrste nasleđivanja i pravo prisutpa članovima osnovne klase u izvedenoj**

| Pravo pristupa članovima u osnovnoj klasi | Pravo pristupa istim članovima u privatno izvedenoj klasi | Pravo pristupa istim članovima u zaštićeno izvedenoj klasi | Pravo pristupa istim članovima u javno izvedenoj klasi |
|---|---|---|---|
| private | *nedostupan* | *nedostupan* | *nedostupan* |
| protected | private | private | protected |
| public | private | protected | public |

Očigledno je da objekat izvedene klase sadrži sve atribute koji su definisani u roditeljskoj klasi i atribute definisane u izvedenoj. Kako se inicijalizacija atributa vrši u konstruktoru, očigledno je da konstruktor izvedene klase treba da pozove konstruktor roditeljske klase. Taj poziv se vrši u delu sa inicijalizaciju. U C++-u se konstruktor roditeljske klase poziva tako što se navede ime roditeljske klase, a zatim u zagradama parametri koji se prosledjuju konstruktoru roditeljske klase. U C#-u se umesto imena roditeljske klase, za poziv konstruktora roditeljkse klase koristi referenca base.

**Primer:** Implementacija konstruktora izvedene klase.

C++:

```
//Osnovna.h
class Osnovna {
  int b;
public:
  // konstruktor osnovne klase
  Osnovna(int);
};

//Osnovna.cpp
Osnovna::Osnovna(int i) : b(i) {}

//Izvedena.h
class Izvedena : public Osnovna {
```

C#:

```
//Osnovna.cs
class Osnovna {
  private int b;

  // konstruktor osnovne klase
  public Osnovna(int i)
  {
    b=i;
  }
}

//Izvedena.cs
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
  int d;
public:
  // konstruktor izvedene klase
  Izvedena(int,int);
};

//Izvedena.cpp
Izvedena::Izvedena (int i, int j)
: Osnovna(i),d(j) {}
```

```
class Izvedena : Osnovna {
  private int d;

  // konstruktor izvedene klase
  public Izvedena(int i,int j)
  : base(i)
  {
    d=j;
  }
}
```

**5.3 Polimorfizam**

Polimorfizam omogućava da klasa saopšti da ima isti interfejs kao i osnovna klasa, ali da je ponašanje izvedene klase različito od ponašanja osnovne klase, odnosno, polimorfizam omogućava da izvedene klase imaju svoje "verzije" funkcija deklarisanih u osnovnoj klasi.

Polimorfizam se postiže deklarisanjem željenih funkcija osnovne klase *virtualnim*. Ako izvedena klasa ne definiše virtualnu funkciju osnovne klase, ona jednostavno usvaja ponašanje osnovne klase.

Virtualna funkcija je funkcija članica za koju se očekuje da bude redefinisana u nasleđenim klasama.

Ispred deklaracije (u C++-u), odnosno definicije (u C#-u) virtuelne funkcije stoji ključna reč virtual. U programskom jeziku C# mora da bude eksplicitno obeležena i funkcija koja predifiniše virtuelnu funkciju osnovne klase navodjenjem ključne reči override. Funkcija koja predefiniše virtuelnu funkciju iz osnovne klase mora da ima isto ime, isti povratni tip i istu listu argumenata kao i odgovarajuća funkcija osnovne klase.

**Primer:** Virtuelne funkcije.

C++:

```
//Osnovna.h
class Osnovna {
  …
public:
  virtual int f(float x);
};

//Izvedena.h
class Izvedena : public Osnovna {
  …
public:
  int f(float x);
```

C#:

```
//Osnovna.cs
class Osnovna {
  …
  public virtual int f(float x) {
    …
  }
}

//Izvedena.cs
class Izvedena : Osnovna {
  …
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
};
```

```
 public override int f(float x){
   …
 }
}
```

Osobina virtuelnih funkcija je da kada se pozovu nad pokazivačem ili referencom tipa osnovne klase, prvo će se proveriti tip objekta na koji pokazivač/referenca ukazuje i pozvaće se funkcija iz one klase kojoj objekat pripada. Znamo da pokazivač/referenca tipa osnovne klase može da ukazuje na objekat osnovne klase ili bilo koje klase koja je direktno ili indirektno izvedena iz nje.

**Primer:** Poziv virtuelnih funkcija.

C++:

```
Osnovna* po = new Osnovna;
po->f(); //poziva se f osnovne klase
delete po;
po = new Izvedena;
po->f(); //poziva se f izvedene klase
delete po;
```

C#:

```
Osnovna o = new Osnovna();
o.f(); //poziva se f osnovne klase

o = new Izvedena();
o.f(); //poziva se f izvedene klase
```

Nepisano je pravilo da se destruktori klasa prave kao virtuelne funkcije. U prethodnom primeru definisan je pokazivač tipa Osnovna. U trenutku brisanja objekta (delete po), kada destruktor nije definisan kao virtuelna funkcija, poziva se destruktor klase Osnovna. U tom slučaju, da su u izvedenoj klasi postojali podaci smešteni u dinamičkoj zoni memorije, oni bi ostali neobrisani, tj. taj deo memorijskog prostora bi bio blokiran do kraja izvršenja programa. Kada se destruktor osnovne klase definiše kao virtuelan, onda se u trenutku brisanja objekta, proverava stvarni tip objekta i poziva se destruktor odgovarajuće izvedene klase. Destruktor izvedene klase na kraju implicitno poziva destruktor osnovne klase. Tako će u tom slučaju biti obrisani svi dinamički atributi i izvedene i osnovne klase.

Iz prethodnog razmatranja zaključujemo da desturktori osnovih (roditeljskih) klasa treba da budu virtuelni. Medjutim, stablo nasledjivanja nikada nije konačno. Verovatno će u nekoj drugoj primeni i izvedene klase biti dalje nasledjene. Čak i ako u postojećim klasama nema dinamičkih podataka (što je bilo i u našem primeru), to ne znači da ih neće biti ni u izvedenim klasama. Zato destruktor treba **uvek** definisati kao virtuelnu funnkciju.

Nad objektom izvedene klase se ne može pozvati funkcija roditeljske klase koja je u izvedenoj klasi predefinisana. Međutim, u funkcijama izvedene klase se može koristiti funkcija osnovne klase koja je u izvedenoj predefinisana. U tom slučaju funkcija roditeljske klase se poziva na sledeći način:

C++:

```
<imeRodKlase>::<imeFunkcije>( <stvarniArgumenti> )
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

C#:

```
base.<imeFunkcije>( <stvarniArgumenti> )
```

Ovaj mehanizam poziva funkcije roditeljske klase biće pokazan na primeru kreiranja klase Krug i iz nje izvedene klase Valjak. Funkcija povrsina() definisana u klasi Krug biće korišćena u funkcijama za zračuvnavanje površine i zapremine u klasi Valjak.

C++:

```
//Krug.h
class Krug {
	float r;
public:
	Krug( float r );
	float obim();
	virtual float povrsina();
};
//Krug.cpp
Krug::Krug( float r ) {
	this->r = r;
}
float Krug::obim() {
	return 2*3.14*r;
}
float Krug::povrsina() {
	return 3.14*r*r;
}

//Valjak.h
class Valjak : public Krug
{
	float H;
public:
	Valjak(float r, float H);
	float zapremina();
	float povrsina();
};

//Valjak.cpp
Valjak::Valjak(float r, float H)
```

C#:

```
//Krug.cs
class Krug {
  private float r;

  public Krug( float r )
  {
    this.r = r;
  }

  public float obim()
  {
	return 2*3.14*r;
  }

  public virtual float povrsina(){
	return 3.14*r*r;
  }
}

//Valjak.cs
class Valjak : Krug
{
  private float H;

  public Valjak(float r, float H)
  : base( r )
  {
	this.H = H;
  }

  public float zapremina()
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
: Krug( r ) {
        this->H = H;
}
float Krug::zapremina() {
  return Krug::povrsina()*H;
}
float Krug::povrsina() {
  return 2*Krug::povrsina()
      + obim()*H;
}
```

```
{
  return base.povrsina()*H;
}

public override float povrsina()
{
  return 2*base.povrsina() +
        obim()*H;
}
}
```

### 5.4 Apstraktne klase

Izvedena klasa prečišćava i proširuje ponašanje osnovne klase. U mnogim realnim aplikacijama, izvedene klase pored svojeg ponašanja, koriste osobine i ponašanje osnovnih (roditeljskih) klasa. Ali, postoje situacije kada je osnovna klasa jednostavno skelet ili okvir, dok se funkcionalnost dobija tek u izvedenim klasama.

Klase u kojima neke funkcije nemaju svoju implementaciju se nazivaju apstraktne klase. Funkcije koje nemaju imeplementaciju se u programskom jeziku C++ nazivaju čistim virtualnim funkcijama i u kodu se označavaju tako što se ispred njihove deklaracije navodi ključna reč virtual, a iza zaglavlja se pise =0 (što označava da funkcija nema telo). U programskom jeziku C# se funkcije koje nemaju implementaciju nazivaju apstraktnim i ispred njihovih deklaracija stoji ključna reč abstract. U programskom jeziku C# se i ispred definicije same klase navodi ključna reč abstract.

**Primer:** Definicja apstraktneklase.

C++:
```
class Apstraktna
{
public:
  virtual void f()=0;
  …
}
```

C#:
```
abstract class Apstraktna
{
  public abstract void f();
  …
}
```

U programu ne može da se kreira objekat apstaktne klase, mada može da postoji pokazivač ili referenca na apstaktnu klasu. Pokazivač tipa apstraktne klase može da ukazuje na objekat bilo koje izvedene klase.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 5.5 Interfejsi u programskom jeziku C#

Interfejsi definišu šablone za ponašanje klasa koje ih implementiraju. To je zbirka apstraktnih metoda. Definicija inerfejsa u programksom jeziku C# ima sledeći format:

```
interface <ime_imeterfejsa>
{
        <telo>
}
```

U telu interfejsa nije obavezno ispered metoda navoditi ključne reči abstract i public.Implicitno svi metodi definisani u interfejsu su apstraktni i javni. Nepisano je pravilo da sam interfejs uvek treba da bude definisan kao javan.

Klasa koja impelementira intrerfejs definiše se na sledeći način:

```
class <ImeKlase> : <ImeInterfejsa>
{
        <telo>
}
```

Klasa u C#-u može da nasledi samo jednu superklasu ali može da nasledi više različitih interfejsa. To znači da klasa nasleđuje atribute iz samo jedne hijerarhije, a ponašanja može da nasledi iz više različitih hijerarhija. Na ovaj način je delimično nadoknađeno nepostojanje višestrukog nasleđivanja u ovom programskom jeziku.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 6. **Izuzeci**

Tokom izvršenja programa mogu da nastanu situacije koje računar nije spreman da obradi (deljenje nulom, čitanje podataka iz oštećenog fajla,..).

Pre nego što je uveden sistem obrade izuzetaka, programi definitivno nisu bili bez grešaka. Postoji nekoliko metoda za obradu grešaka koje su se tada koristile, a objasnićemo samo jednu da bi mogli videti prednost sistema obrade izuzetaka.

U starijim programima, svaka važnija funkcija vraća neku vrednost kojom se označava kako je izvršavanje prošlo. Kod koji je pozvao funkciju proverava vraćenu vrednost i zavisno od toga sprovodi određene akcije. Posledica – programski kod je prepun dodatnim if–naredbama i delovima koda za obradu greške. Takođe, povratna vrednost funkcije ne može da se koristi za vraćanje ničeg drugog sem koda greške.

### 6.1 Koncept obrade izuzetaka

Globalni koncept obrade izuzetaka je jednostavan. Ideja je da se postave "zastavice za greške" (error flags) svaki put kada nešto krene naopako. Takođe postoji sistem koji non-stop nadgleda te zastavice. Ukoliko je zastavica postavljena, taj sistem poziva kod za obradu greške. Postavljanje zastavice za grešku se naziva *generisanje* (eng. raising, throwing) greške.

Kad se generiše greška, ceo sistem reaguje *hvatanjem* (eng. catching) greške. Uokviravanje bloka (dela) koda “osetljivog” na greške kodom za obradu izuzetaka se naziva *pokušavanje* (eng. trying) izvršenja bloka. Kod za obradu izuzetaka se naziva *try*-kod.

Jedna od najmoćnijih osobina obrade izuzetaka je to što greška može da se generiše i van granica funkcije. To znači da ukoliko jedna od najdubljih funkcija na steku (npr. rekurzivni pozivi funkcije) generiše grešku, ta greška može da se prosledi do funkcija iznad u hierarhiji. To omogućava programerima da postave kod za obradu greške samo na jednom mestu, npr. u glavnoj (main) funkciji.

### 6.2 Sintaksa naredbi za prijavu i obradu izuzetaka

Izuzetak se prijavljuje throw naredbom, a prihvata i obradjuje try-catch strukturom.

Sintaksa try-catch strukture:

<u>C++:</u>

```
try <block> <handler-list>
```

<u>C#:</u>



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

try <block> (<handler-list>|<finally-block>|<handler-list> <finally-block>)

gde je:

<block> - deo koda u kojem može da dođe do prijave izuzetaka,

<handler-list> - niz catch blokova koji prihvataju i obradjuju izuzetke različitog tipa,

<finally-block> - deo koda koji se izvršava poslednji, bez obzira da li je neki izuzetak prijavljen ili ne.

Sintaksa catch bloka:

```
catch (<exception-type>  <exception-name>)
{
        …
}
```

Sintaksa finally bloka:

```
finally
{
        …
}
```

Tok izvršenja programa izgleda ovako:

1. Kontrola se predaje try naredbi. Štićeni deo koda u okviru try bloka se izvršava.
2. Ukoliko ni jedan izuzetak nije generisan tokom izvršenja try bloka, izvršenje programa se nastavlja:
    a. izvršenjem prve naredbe navedene iza try-catch structure (u C++-u),
    b. izvršenjem finally-bloka ukoliko postoji, ili prve naredbe navedene iza try-catch structure ukoliko finally-blok ne postoji (u C#-u).
3. Ukoliko se generiše izuzetak tokom izvršenja štićenog dela ili u okviru bilo koje funkcije pozvane u tom delu (direktno ili indirektno), kreira se objekat izuzetka, zatim se traži odgovarajući catch blok koji ume da obradi izuzetak podgovarajućeg tipa. Catch handler-i (klauzule koje odbrađuju izuzetke) se ispituju onim redom kojim su navedeni u try-catch strukturi. Ukoliko se ne nađe odgovarajući handler, ispituje se try blok koji obuhvata trenutni try blok (ukoliko postoji).
4. Ukoliko se ne nađe ni jedan handler koji može da prihvati izuzetak odgovarajućeg tipa, poziva se predefinisana funkcija terminate.
5. Ukoliko se nađe odgovarajući catch handler, memorija se oslobađa automatski generisanih objekata, kreiranih između početka aktuelnog try bloka i mesta podizanja izuzetka. Destruktori se pozivaju u obrnutom redu u odnosu na konstruisanje (prvo se uništava poslednji kreirani objekat). Zatim se izvršava catch handler, a potom se nastavlja sa izvršenjem programa, i to od:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

a. prve naredbe napisane ispod svih catch blokova (u C++-u),
b. finally-bloka ukoliko postoji, ili prve naredbe iza svih catch blokova ukoliko finally-blok ne postoji (u C#-u).

U C++-u se umesto deklaracije izuzetka koji se prihvata u zaglavlju catch bloka mogu napisati tri tačke, što znači da takav catch blok prihvata izuzetke proizvoljnog tipa. Ukoliko u try-catch strukturi postoje catch blokovi za orihvatanje izuzetaka konkretnog tipa, i univerzalni catch blok (čiji je argument ...), univerzalni catch blok mora biti napisan kao poslednji u nizu. Kad bi on bio napisan kao prvi, on bi prihvatao sve izuzetke koji se pojave, tako da ostali nikad ne bi bili izvršeni.

Sintaksa naredbe za prijavu izuzetaka:

throw <izraz>***;***

Šta se sve može prijaviti kao izuzetak?

U programskom jeziku C++ se kao izuzetak može iskoristiti podatak bilo kog tipa. Uobičajeno je da se kao izuzetak koristi:

- znakovni niz koji predstavlja poruku o grešci,
- enum koji sadrži kodove svih mogućih grešaka koji mogu da nastanu u programu,
- objekat bibiliotečke klase exception ili kalse koja je direktno ili indirektno izvedena iz klase exception.

Klasa **exception** je definisana u standardnoj biblioteci klasa u zaglavlju <exception.h>. To je osnovna klasa (koren hijerarhije) svih standardnih izuzetaka. Metode standardnih klasa i neki operatori prijavljuju izuzetke klasa izvedenih iz klase exception. Preporučuje se da se i korisnički izuzeci izvode iz klase exception. Ovako kreirana hijerarhija izuzetaka omogućava da se pojedini tipovi izuzetaka mogu obrađivati :

- pojedinačno,
- u srodnim grupama,
- svi zajedno (ali ipak ne kao ...).

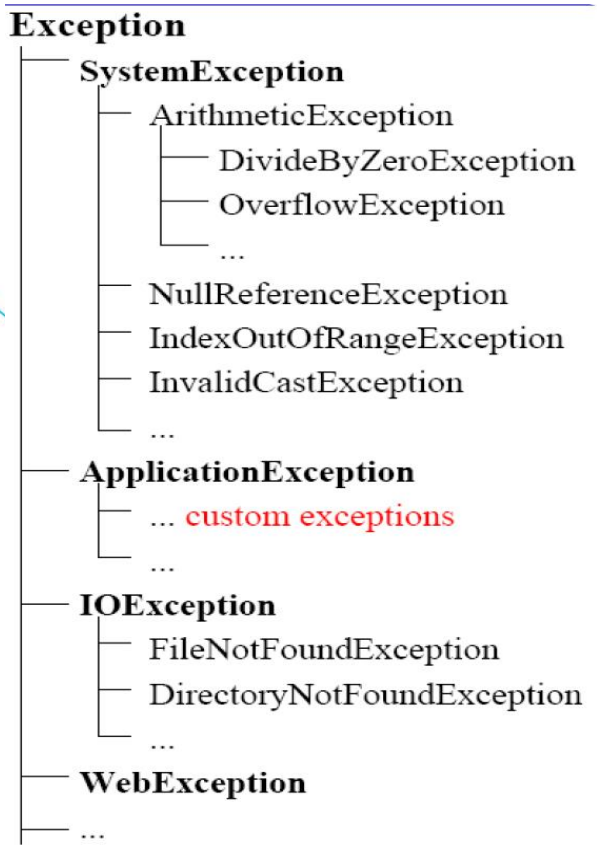
U programskom jeziku C# se kao izuzetak može koristiti isključivo objekat bibliotečke klase Exception ili klase direktno ili indirektno izvedene iz klase Exception. U standardnoj bibilioteci klasa je definisan veliki broj klasa za predsavljanje izuzetaka. Ipak, ako u aplikaciji postoje situacije kada treba prijaviti izuzetak koji nije predvidjen u standardnoj biblioteci klasa, treba praviti sopstvenu klasu za predstavljanje izuzetaka izvedenu iz klase Exception. Neke bitnije bibliotečkeklase za predstavljanje izuzetaka prikazane su na slici 6.1.

Ukoliko se piše veći broj catch blokova, treba strogo voditi računa o redosledu njihovog navođenja jer catch blok koji prihvata objekte klase XException će prihvatiti i izuzetke svih klasa izvedenih iz



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

XException. To znači da kada treba posebno prihvatati i izuzetke tipa roditeljske i izvedene klase, catch blok koji prihvata izuzetke tipa izvedene klase mora da bude naveden pre catch bloka koji prihvata izuzetke roditeljske klase. U skladu sa tim, catch blok koji prihvata izuzetke tipa Exception se uvek navodi kao poslednji (on se ponaša kao catch blok sa ... u programkom jeziku C++, prihvata sve moguće izuzetke).




**Slika 0.1** Hierarhija standardnih izuzetaka u programkom jeziku C#



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# Specijalistički predmeti



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

TVOJA KARIJERA U TVOJIM RUKAMA

# *Embedded Specialist*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# LabVIEW

**Predavač:** Marko Dimitrijević

**Ciljevi kursa:** Kurs obuhvata koncepte virtuelne instrumentacije, akvizicije, obrade i prezentovanja podataka u LabVIEW okruženju

**Ishod kursa:** Sticanje potrebnih znanja za implementaciju virtuelnih instrumenata u LabVIEW razvojnom okruženju

**Teorijska nastava:**

Koncept virtuelne instrumentacije i data-flow programiranje, Osnovne akvizicije podataka (A/D, rezolucija, brzina semplovanja), Virtuelni instrumenti, LabVIEW razvojno okruženje

**Praktične vežbe**:

- 50% celokupne nastave biće praktična nastava u LabVIEW okruženju, a u zavisnosti od tehničkih mogućnosti rad sa akvizicionom opremom (USB moduli, PXI platforma).
- Praktična nastava će obuhvatati osnovne tehnike realizacije virtuelnih instrumenata, kao i napredne tehnike mrežnog povezivanja, bazama podataka i upravljanja događajima.

**Sadržaj kursa:**

Materijal za nastavu će obezbediti predavač prema kurikulumu kursa

**Trajanje kursa:**

Kurs će biti realizovan u 30 školska časa (4 časa dnevno, dva dana u nedelji, 45 minuta). Struktura kursa je 12+12+6 (teorijska nastava + praktična nastava + testovi)

**Kurikulum**:

Prva nedelja: Osnovni pojmovi akvizicije podataka i merenja. Koncept virtuelnog instrumenta
Druga nedelja: Programske strukture (nastavak) i programske petlje. Test 1
Treća nedelja: Akvizicija i generisanje podataka u LabVIEW okruženju.Tipovi podataka (nastavak) Test 2
Četvrta nedelja: Napredne tehnike programiranja. Završni test

**Ocenjivanje:** 2 testa biće ocenjivanja poenima do 100, dok će završni test – projektni zadatak biti ocenjivan ocenama od 1 do 5

**Testiranje:**

U toku trajanja kursa biće održana dva testa (dva školska časa), i jedan završni test



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 1 Uvod

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) je razvojno okruženje za razvoj aplikacija u grafičkom programskom jeziku G. Najčešće se koristi za akviziciju podataka, automatizaciju industrijskih procesa, kontrolu instrumenta i analizu dobijenih podataka, mada se mogu razvijati aplikacije opšte namene. LabVIEW je podržan na različitim softverskim platformama kao što su Microsoft Windows, Linux, različite verzije UNIX-a i Mac OS X. Osim navedenih operativnih sistema, realizovane aplikacije se mogu kompajlirati za operativne sisteme za rad u realnom vremenu (PharLap, VXWorks) a podržane su i hardverske platforme bazirane na FPGA ili ARM mikrokontrolerima.

Virtuelna instrumentacija je pojam koji označava integraciju hardvera i softvera, obuhvatajući procese merenja, akvizicije, obrade i prezentacije dobijenih podataka. Virtuelna instrumentacija je trend u tehnici i nauci zbog rastuće kompleksnosti inženjerskih poslova, potrebe za velikim brojem specijalizovanih i skupih instrumenata i softvera, potrebe za visokokvalifikovanim ljudima za rad sa tim instrumentima kao i organizovanja podele rada sa instrumentima i softverom. Osnovna paradigma ovog koncepta je *virtuelni instrument*.

Virtuelni instrument je višeslojni hardversko-softverski sistem. Njegova namena je identična funkciji klasičnog instrumenta. Razlika je u fleksibilnosti i funkcionalnosti koju je moguće menjati, za razliku od klasičnog instrumenta koji ima nepromenljivu funkcionalnost, određenu od strane proizvođača. Prednosti virtuelnih instrumenata nad klasičnim su i veća mogućnost obrade i prezentacije rezultata merenja. Fizički, odnosno hardverski sloj virtuelnog instrumenta čini računar sa odgovarajućom mernom, akvizicionom i dodatnom opremom. Ekspanzija proizvodnje personalnih računara, naročito intenzivna u prethodnih dvadeset godina, podstakla je razvoj uređaja za akviziciju i obradu podataka sa odgovarajućim interfejsima za računar. Veliki broj klasičnih mernih uređaja takođe poseduje interfejs za povezivanje sa računarom. Ovi uređaji se mogu povezati eksterno preko serijskog RS232/RS422/RS485 interfejsa, GPIB, paralelnog interfejsa, ili preko bržih USB 1.1/2.0/3.0 i Ethernet interfejsa. Najnoviji trendovi u povezivanju su bežične senzorske mreže (*Wireless Sensor Networks*), koje koriste različite implementacije bežičnih protokola (WLAN, *Bluetooth, ZigBee*) i autonomne metode napajanja. Ethernet interfejs je naročito pogodan, jer pruža mogućnost povezivanja velikog broja različitih uređaja i računara u heterogenu mrežu. Komunikacija sa uređajima se realizuje preko standardnih OSI i TCP/IP modela. Ovakva realizacija osim skalabilnosti omogućava i povećanje fizičke distance između uređaja i korisnika. Ukoliko se primenjuje TCP/IP model, razdaljina je praktično neograničena.

Softverski sloj virtuelnog instrumenta čine drajveri za rad sa uređajem i softver višeg nivoa za akviziciju i obradu podataka – aplikacija virtuelnog instrumenta. Često u literaturi termin virtuelni instrument predstavlja sinonim za aplikaciju virtuelnog instrumenta. Funkcija aplikacije je definisanje



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

signala koji se generišu, obrada podataka dobijenih akvizicijom i njihovo prezentovanje. Obrada podataka obuhvata različite funkcije kao što je analiza u vremenskom i frekventnom domenu, različita izračunavanja fizičkih veličina koja se posredno mere pretvaranjem u električne, itd. Aplikacije virtuelnih instrumenata imaju iste osobine kao i druge aplikacije. Mogu se realizovati kao multithread aplikacije, klijent/server ili web aplikacije, postoji i mogućnost njihovog izvršavanja u obliku sistemskog servisa. Omogućena je i jednostavna prezentacija i razmena podataka u obliku standardnih i široko rasprostranjenih formata.

Predstavljanje podataka je najčešće u grafičkom obliku, funkcijama i kontrolama grafičkog interfejsa koje svojim izgledom podsećaju na realne merne instrumente. Interfejs je interaktivan i ima dvosmernu funkciju: osim predstavljanja pruža mogućnost kontrole procesa merenja, definisanjem parametara signala koji se generišu (talasni oblik, amplituda, frekvencija, faza, modulacija,...) ili čak definisanjem topologije mernog kola primenom matrica prekidača.

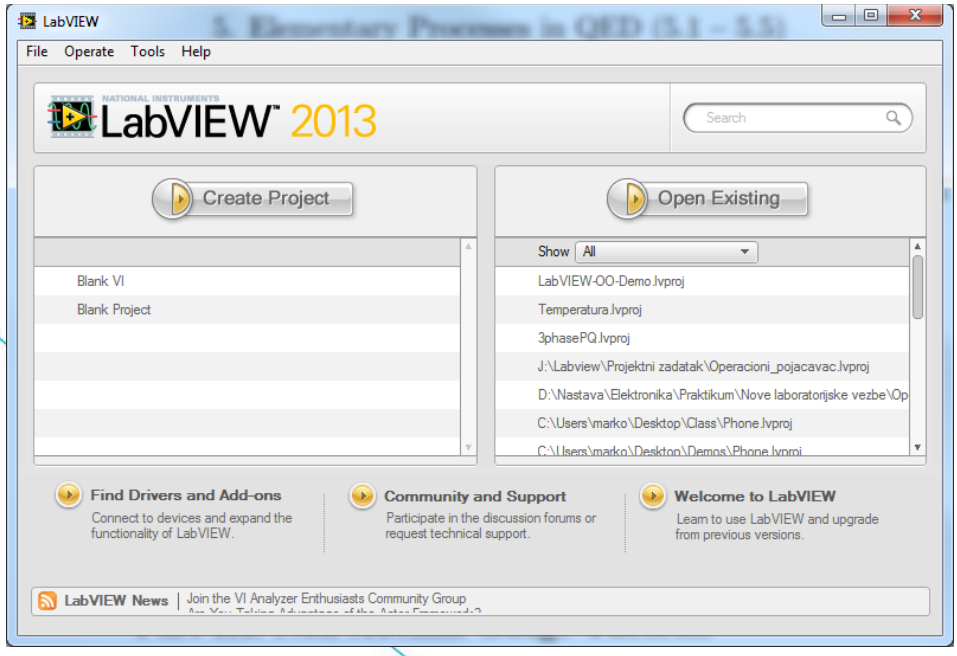
Postoje različiti razvojni alati i okruženja za realizaciju aplikacija virtuelnih instrumenta. Ovi alati mogu biti dodaci za već postojeća okruženja za razvoj softvera (uglavnom C ili C++) ili kao posebni alati za ovu namenu (LabVIEW). Posebni alati za razvoj aplikacija virtuelnih instrumenata omogućuju intuitivan i jednostavan razvoj bez potrebe za programerskim znanjem. Razvoj aplikacija je na taj način omogućen izvođaču merenja, pružajući posebnu fleksibilnost. Virtuelni instrumenti realizovani na taj način mogu imati opštu namenu ili biti specijalizovani, namenjeni za *ad hoc* merenja.

Pomenuti koncept virtuelne instrumentacije pruža mnogobrojne prednosti u odnosu na klasične metode merenja. Dodatna obrada podataka, povezivanje sa bazama podataka, izvođenje procesa merenja sa velike udaljenosti kroz web interfejs su samo neke mogućnosti. Najznačajnija prednost je vizuelizacija podataka. Vizuelizacija podataka, odnosno mogućnost njihove vizuelne analize i prezentacije predstavlja najbrži put da se sagleda celokupna situacija, uoče problemi i pronađu odgovarajuća rešenja.

# 2 LabVIEW okruženje

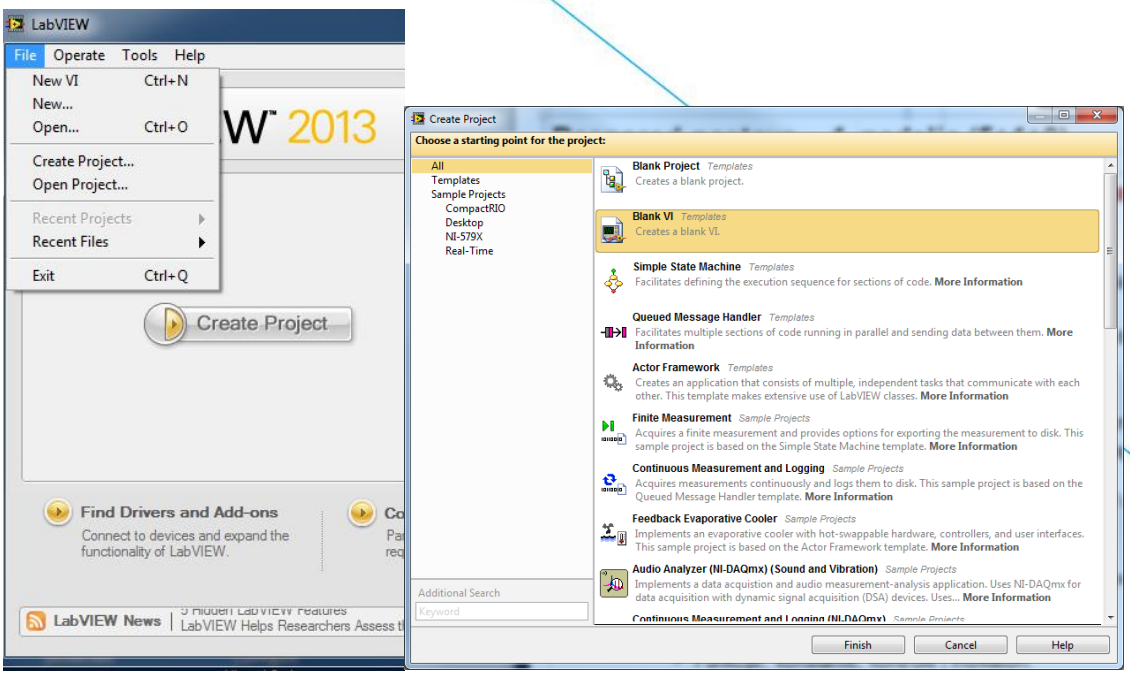
### 2.1 Startup dijalog

Okruženje se sastoji od više elemenata koji su aktivni u različitim fazama realizacije virtuelnog instrumenta ili projekta. Prilikom pokretanja razvojnog okruženja, aktivan je *startup dijalog* prikazan na slici 2.1. Osnovne funkcije dijaloga su započinjanje novog virtuelnog instrumenta/projekta ili otvaranje postojećeg. Pored osnovnih funkcija, dijalog pruža mogućnost pristupa *online* resursima, pretraživanje primera, interaktivnu pomoć i pretragu drajvera i opcionih dodataka. Iz dijaloga se može pristupiti drugim alatima za konfiguraciju okruženja, debagiranje i kompajliranje virtuelnih instrumenata.




**Slika 2.1 Startup dijalog**

Novi virtuelni instrument se može kreirati na osnovu datih šablona ili kao prazan (Slika 2.2).



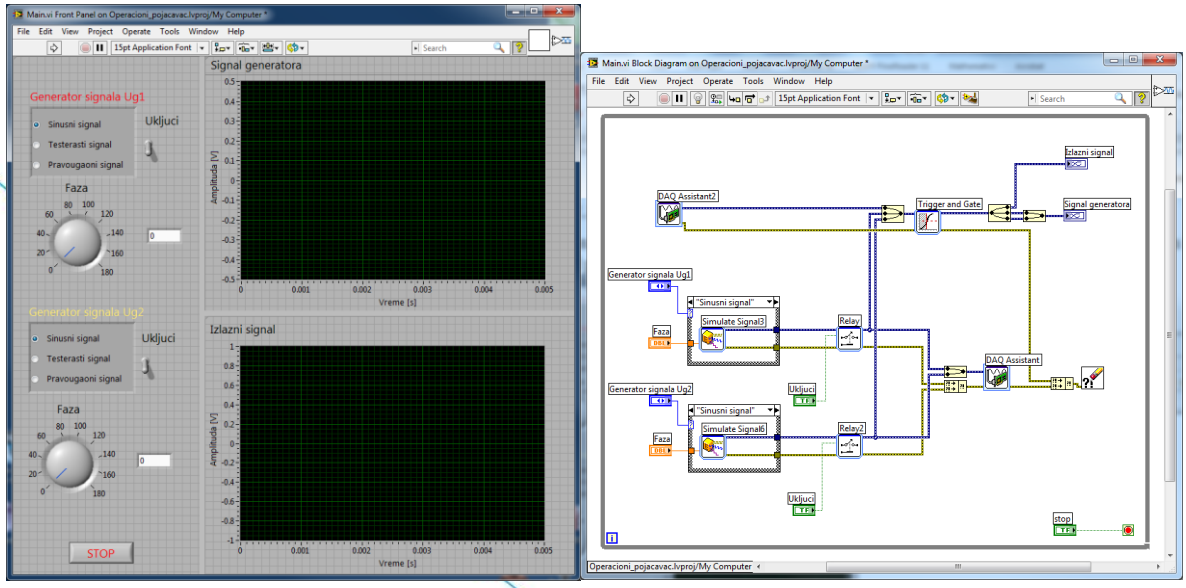

**Slika 2.2 Kreiranje novog virtuelnog instrumenta – meni (levo) i dijalog sa postojećim šablonima (desno)**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

## 2.2 Front panel i Blok dijagram prozori

Nakon otvaranja/kreiranja novog virtuelnog instrumenta, aktivni su blok dijagram i front panel prozori (Slika 2.3)




**Slika 2.3 Front panel (levo) i blok dijagram (desno)**

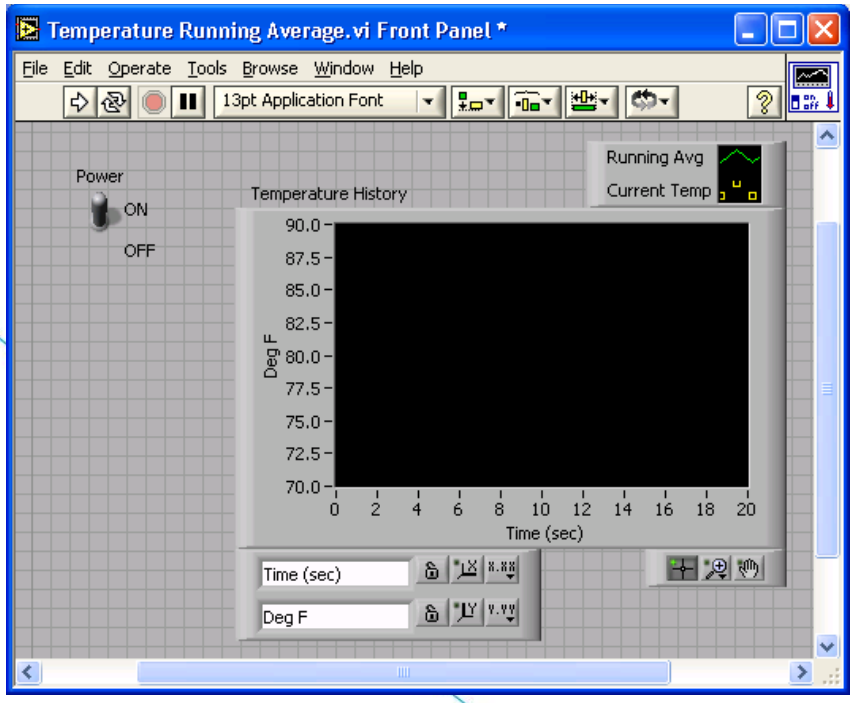
Prozori sadrže elemente virtuelnog instrumenta i alate za manipulaciju.

### *2.2.1 Front panel prozor*

Front panel prozor je prikazan na slici 2.4. U gornjem delu prozora se nalaze meni i osnovna paleta sa alatima. Donji deo prozora sadrži elemente korisničkog interfejsa virtuelnog instrumenta (front panel). Ovi elementi se mogu dodati sa **Controls** palete i predstavljaju terminale (kontrole i indikatore) virtuelnog instrumenta, koji omogućuju interakciju korisnika i aplikacije.



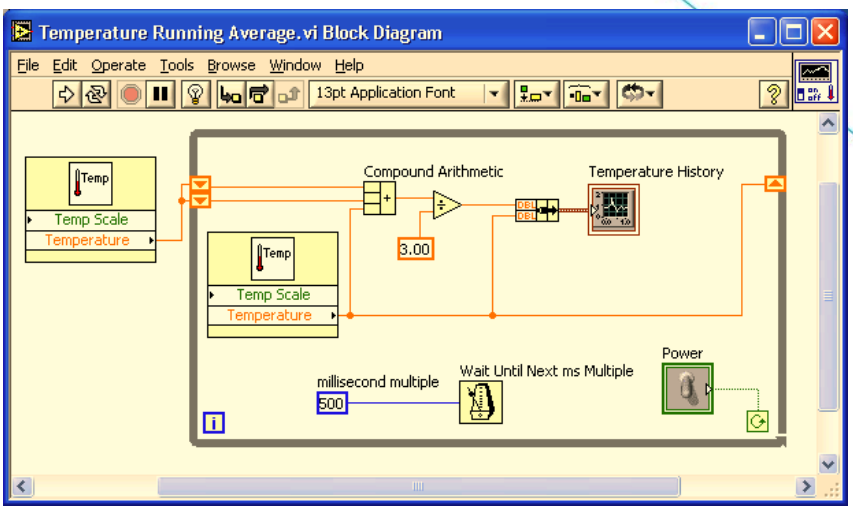
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 2.4 Prozor Front panela**

*2.2.2 Blok dijagram prozor*

Blok dijagram prozor je prikazan na slici 2.5.




**Slika 2.5 Blok dijagram prozor**



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

U gornjem delu prozora se nalaze meni i paleta sa alatima. Donji deo prozora sadrži elemente blok panela, koji predstavljaju kôd aplikacije. Elementi blok dijagrama se mogu proizvoljno dodavati sa palete **Functions** i povezivati odgovarajućim vezama.

*2.2.3 Meniji*

Na vrhu prozora se nalaze meni sa uobičajenim funkcijama. Glavni meni sadrži sledeće menije:

• Meni **File** sadrži opcije za osnovne operacije sa fajlovima, kao što su otvaranje, zatvaranje, snimanje, štampanje i slično.

• **Edit** meni objedinjuje opcije za rad sa virtuelnim instrumentima i njegovim elementima.

• **View** meni sadrži opcije vezane za izgled radnog okruženja, aktiviranje paleta, hijerarhiju virtuelnih instrumenata, klasa, eksternih komponenti i slično. U meniju

• **Project** se nalaze opcije za otvaranje, kreiranje i rad sa LabVIEW projektima.

• **Operate** meni objedinjuje kontrole za pokretanje, kontrolu toka izvršavanja i debagiranje virtuelnih instrumenata.

• **Tools** meni sadrži kontrole za konfigurisanje okruženja, virtuelnog instrumenta, projekta, drajvera ili drugih komponenti.

• **Window** služi za podešavanje izgleda radnog okruženja, prozora i paleta.

• **Help** je skup opcija za pomoć u realizaciji virtuelnih instrumenata, pristup dokumentaciji, online resursima i primerima, kao i tehničkoj podršci.

*2.2.4 Paleta sa alatima*

Ispod menija se nalazi paleta sa alatima (**Toolbar**), prikazana na slici 2.6.

**Slika 2.6 Toolbar**

Paleta sa alatima Front panela sadrži sledeće komande (s leva na desno):

• dugme **Run**, za izvršavanje virtuelnog instrumenta,

• dugme **Continuous Run**, za višestruko pokretanje virtuelnog instrumenta – nakon izvršavanja VI, LabVIEW će ponovo pokrenuti izvršavanje aktivnog VI,

• dugme **Abort**, za zaustavljanje izvršavanja VI,

• dugme **Pause/Continue** za pauziranje i nastavljanje izvršenja VI,

• Drop-down meni za izbor fonta i veličine,

• Opcije za ravnanje, raspodelu i dimenzionisanje elemenata,



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

• Dugme za podešavanje redosleda elemenata,

• Dugme kojim se aktivira kontekstni help.

U prozoru blok dijagrama, paleta ima dodatne komande, prikazane na slici 2.7,

**Slika 2.7 Dodatne komande Blok dijagram palete**

koje služe za debagiranje virtuelnog instrumenta. Prva opcije je **Execution Highlight**, koja naglašava čvorove blok dijagrama koji se trenutno izvršavaju, kao i tok podataka na vezama između čvorova. Sledeće opcije služe za izvršavanje virtuelnog instrumenta korak-po-korak.

U slučaju da virtuelni instrument ima grešaka, **Run** dugme će promeniti izgled u **Broken Run** , što ukazuje na postojanje grešaka koje se moraju izmeniti. Klikom na **Broken Run**, LabVIEW će prikazati listu identifikovanih grešaka.

*2.2.5 Tools paleta*

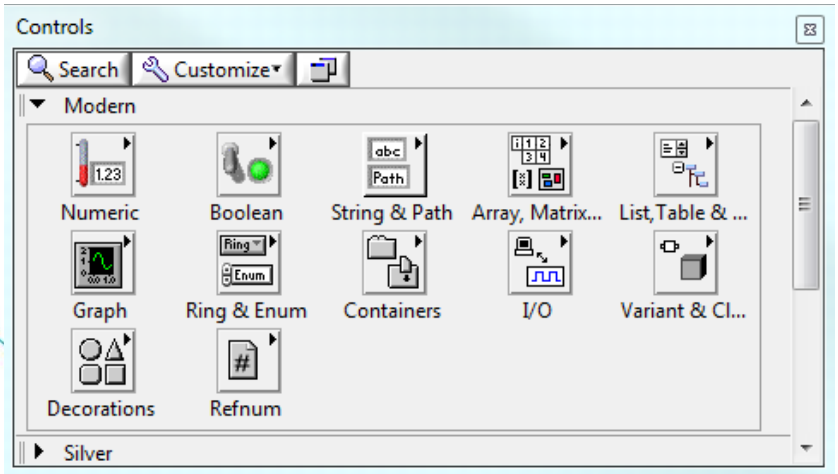
**Tools** paleta (slika 2.8) je dostupna u Front panel i Blok dijagram prozorima. Aktiviranjem opcija iz palete se menja môd rada sa objektima, što je indicirano promenom izgleda pointera. Tools paleta se može aktivirati dostupna izborom iz menija **VIEW » Tools Palette**.

**Slika 2.8 Tools paleta**

Ukoliko je selektovana opcija **Automatic Tool Selection,** koja se nalazi na vrhu palete, priliko, pomeranja kursora preko objekta, LabVIEW automatski selektuje odgovarajući alat iz palete.

*2.2.6 Paleta sa kontrolama*

Paleta sa kontrolama (**Control palette**) je dostupna na Front panel prozoru (slika 2.9). Paleta se može aktivirati desnim klikom ili iz menija **VIEW » Control Palette**. Paleta sadrži kontrole i indikatore, elemente korisničkog interfejsa virtuelnog instrumenta. Kontrole i indikatori su grupisani prema kategorijama i tipu. Paleta sadrži opcije za pretragu kontrola i indikatora po imenu.

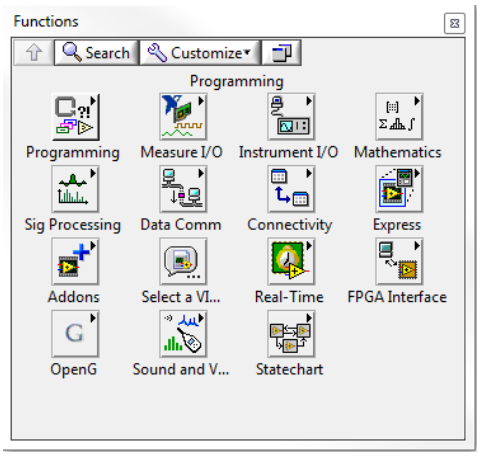



**Slika 2.9 Paleta sa kontrolama – najčešće korišćene kontrole**

Elementi front panela se postavljaju prevlačenjem odgovarajuće kontrole ili indikatora sa palete na prozor front panela.

*2.2.7 Paleta sa funkcijama*

Paleta sa funkcijama je dostupna na blok dijagram panelu (Slika 2.10). Paleta sadrži sve funkcije, programske strukture i module dostupne u LabVIEW. Paleta se može aktivirati desnim klikom ili iz menija **VIEW » Functions Palette**.




**Slika 2.10 Paleta sa svim funkcijama – prikazane su sve raspoložive funkcije i programske strukture**

Elementi blok dijagrama se postavljaju prevlačenjem odgovarajućih elemenata – čvorova, programskih struktura ili konstanti sa palete na blok dijagram. Elementi su na paleti grupisani prema tipu i nameni.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**Functions** paleta može biti fiksirana ili plivajuća. Subpalete – delovi glavne palete – se mogu posebno izdvojiti, u cilju lakše manipulacije elementima koji se često koriste. Paleta sadrži **Search** – opciju za pretragu funkcija po imenu. **Customize** opcija omogućava podešavanje prikaza elemenata na paleti.

### 2.3 Help

U okviru LabVIEW razvojnog okruženja korisnicima je na raspolaganju više opcija za pružanje pomoći. Na raspolaganju su **LabVIEW Help, Context Help, Explain Error, Find Examples, Web Resources** i drugi.

#### *2.3.1 LabVIEW Help*

**LabVIEW Help** je osnovni help razvojnog okruženja. Sadrži različite objašnjenja kontrola, indikatora, funkcija, programskih struktura i modula. Osnovni help je offline tipa. Help omogućuje pretragu prema kategorijama ili po nazivu elementa. Može se pristupiti direktno iz menija **Help » LabVIEW Help**, ili iz kontekstnog helpa.

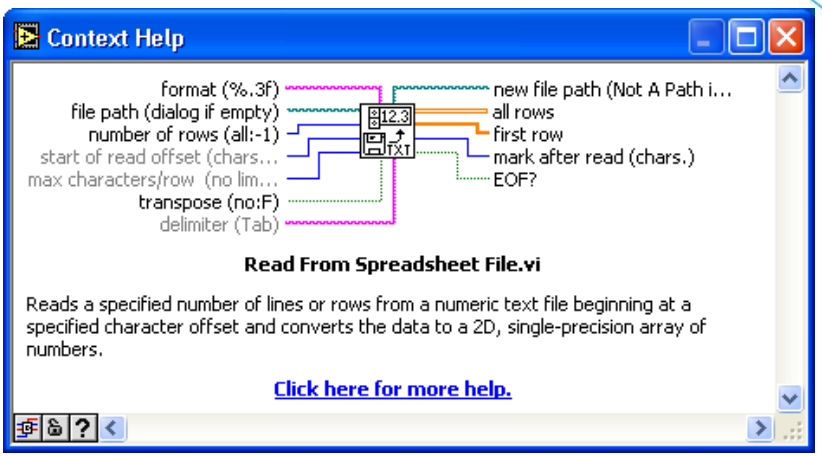
Help sadrži detaljne opise većine paleta, menija, alata, VI, funkcija, instrukcija za korišćenje LabVIEW opcija, linkove ka online resursima, tutorijalima, PDF verzijama priručnika i tehničkoj podršci.

#### *2.3.2 Context Help*

**Context Help** omogućuje interaktivnu pomoć pri realizaciji virtuelnog instrumenta. Poziva se opcijom **Help » Show Context Help**, pritiskom <Ctrl-H> tastere, ili aktiviranjem **Show Context Help Window**

dugmeta na paleti sa alatima. Kada je aktivan, u prozoru

**Context Help**-a je prikazan selektovani čvor sa nazivom, pripadajućim ulazima i izlazima i kratkim objašnjenjem funkcije (Slika 2.11).




**Slika 2.11 Kontekstni help**

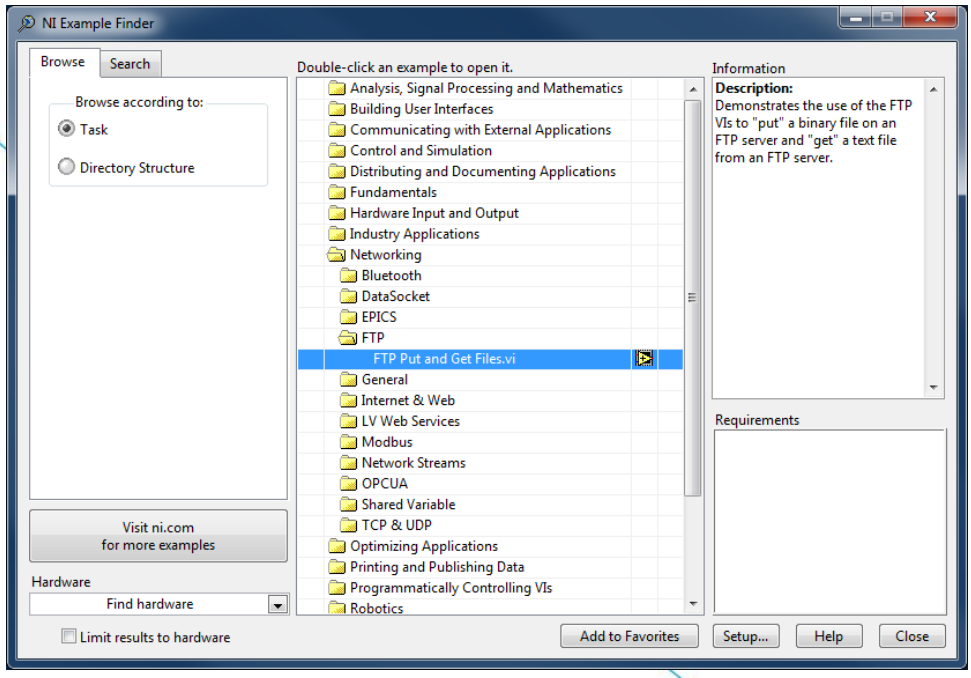
Iz **Context Help**-a je moguće pozvati detaljni help.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

*2.3.3 Online resursi*

Iz LabVIEW okruženja je moguće pristupiti *online* resursima pomoću opcije **Help » Web Resources**. Opcija **Help » Find Examples** pokreće posebnu aplikaciju, **NI Example Finder**, (slika 2.12) koja omogućuje pretragu primera aplikacija i rešenja u različitim online i offline resursima.




**Slika 2.12 NI Example Finder**

# 3 Virtuelni instrument

Aplikacije realizovane u LabVIEW okruženju se nazivaju *virtuelni instrumenti* (skraćeno VI). Za razliku od tradicionalnih programskih jezika sa tekstualnim kôdom kod kojih je redosled izvršavanja tekstualnih programskih jezika određen redosledom unetih dekleracija, programiranje u LabVIEW okruženju se izvodi povezivanjem čvorova koji predstavljaju određene funkcije. Redosled izvršavanja je definisan tokom podataka – *data flow*.

Virtuelni instrument se sastoji od nekoliko komponenti, od kojih su najznačajnije *front panel* i *blok dijagram*.

## 3.1 Front panel

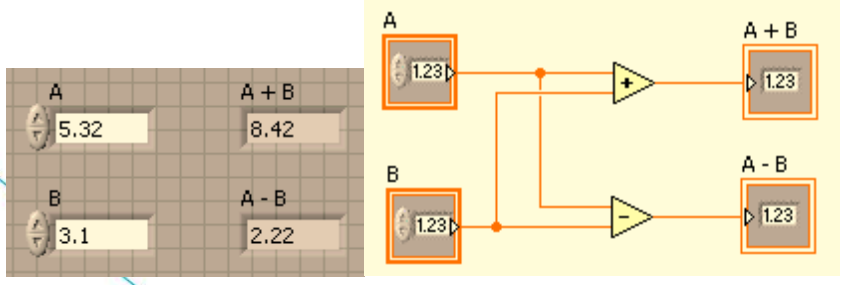
Front panel predstavlja korisnički interfejs aplikacije i sadrži elemente koji omogućuju interakciju između korisnika i aplikacije (slika 3.1). Ulazni elementi front panela se nazivaju kontrole, izlazni indikatori. Front panel može sadržati i druge neinteraktivne i interaktivne elemente – dekoracije, menije, tabove, itd. Kontrole i indikatori imaju grafički izgled sličan realnim instrumentima i



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

kontrolama. U zavisnosti od tipa podatka koji reprezentuju, elementi mogu biti logički, numerički, string, nizovi ili složeniji.

Front panel




**Slika 3.1 Front panel (levo) i blok dijagram (desno) virtuelnog instrumenta**

Elementi front panela se postavljaju prevlačenjem iz **Controls** palete. Osobine elementa se mogu modifikovati kontekstnim menijem i dijalogom sa opcijama: izgled, tip i reprezentacija podatka, vidljive komponente elementa, opis itd.

Kontrola se može pretvoriti u indikator istog tipa, i obratno, izborom opcije **Change to indicator/control** iz kontekstnog menija.

### 3.2 Blok dijagram

Blok dijagram sadrži grafički kôd aplikacije, sačinjen od terminala, čvorova, veza i programskih struktura (slika 3.1).

Terminali blok dijagrama (slika 3.2) predstavljaju ulaze i izlaze blok dijagrama i definišu tok podataka u virtuelnom instrumentu – od ulaznih ka izlaznim terminalima. Mogu korespondirati kontrolama (terminali ulaznog tipa) i indikatorima (terminali izlaznog tipa) postavljenim na front panel. U terminale spadaju i konstante – terminali ulaznog tipa, kao i labele, koje mogu imati funkcije ulaza i izlaza.

**Slika 3.2 Terminali**

Čvorovi blok dijagrama su elementi koji procesiraju podatke, analogni su deklaracijama i funkcijama tekstualnih programskih jezika. Čvorovi mogu biti elementarne funkcije ili drugi virtuelni instrumenti.

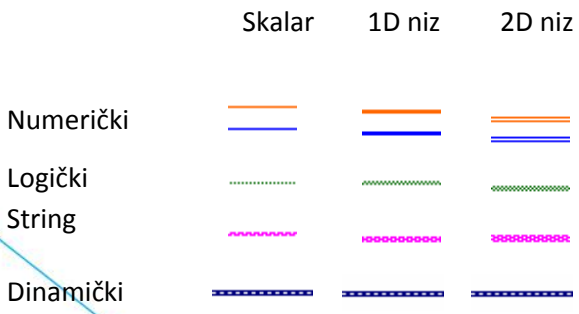
Funkcije su fundamentalni elementi u LabVIEW i predstavljaju osnovne gradivne blokove svakog virtuelnog instrumenta, koje je nemoguće razložiti.

Virtuelni instrumenti se mogu instancirati u drugim virtuelnim instrumentima, čime se postiže modularnost i preglednost grafičkog kôda i definiše hijerarhija u lancu virtuelnih instrumenata. Instancirani virtuelni instrumenti (SubVI) mogu biti standardni i tzv. Express VI, koji imaju definisanu funkciju i mogu se modifikovati kroz unapred definisan korisnički dijalog.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Elementi blok dijagrama se povezuju **vezama**. Boja i tekstura veze označavaju tip i dimenzionalnost podatka. Primer nekih tipova je dat na slici 3.3.




**Slika 3.3 Tipovi podataka**

Čvorovi blok dijagrama se izvršavaju zavisno od toka podataka – *data flow* koncept programiranja. Izvršavanje počinje kada su na svim ulaznim terminalima čvora dostupni podaci. Rezultati izvršavanja su dostupni na izlaznim terminalima nakon izvršenja čvora. Redosled izvršavanja i prosleđivanja podataka između čvorova se može pratiti aktiviranjem **Execution Highlighting** (odeljak 2.2.4) opcije sa palete sa alatima blok dijagram prozora.

U cilju otklanjanja grešaka, izvršavanje virtuelnog instrumenta se može zaustaviti na određenom mestu pomoću prekida (**Breakpoints**). Prekid se postavlja otvaranjem kontekstnog menija (desni klik) veze i izborom opcije **Breakpoint/Set Breakpoint**. Izvršenje će biti zaustavljeno u tački prekida.

Vrednost podatka na vezi se može pratiti pomoću sonde (**Probe**). Sonda se može postaviti otvaranjem kontekstnog menija (desni klik) veze i izborom opcije **Probe**, čime se otvara prozor sa prikazom trenutne vrednosti podatka na vezi.

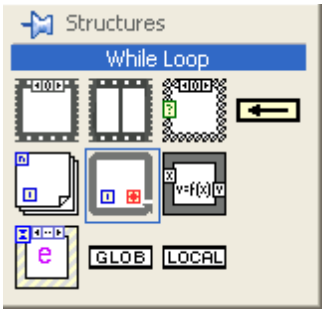
Uslovna sonda (**Conditional Probe**) predstavlja hibrid sonde i prekida. Postavlja se aktiviranjem kontekstnog menija (desni klik) veze i izborom opcije **Custom probe/Conditional Double Probe**. Uslovna sonda proverava podatak prema unapred zadatom kriterijumu i prekida izvršavanje ukoliko je ispunjen.

### 3.3 Programske petlje

U realizaciji aplikacija je često neophodno određeni deo kôda izvršiti više puta. Standardni programski jezici imaju različite mehanizme kojima je ovakvo izvršavanje omogućeno – programske petlje. LabVIEW podržava nekoliko tipa programskih petlji: **WHILE** petlja, **FOR** petlja, uslovna **FOR** petlja i **Timed loop** petlja. Programske petlje u LabVIEW okruženju predstavljaju skup programskih struktura, i mogu se postaviti na blok dijagram virtuelnog instrumenta iz palete **Programming/Structures** (slika 3.4).



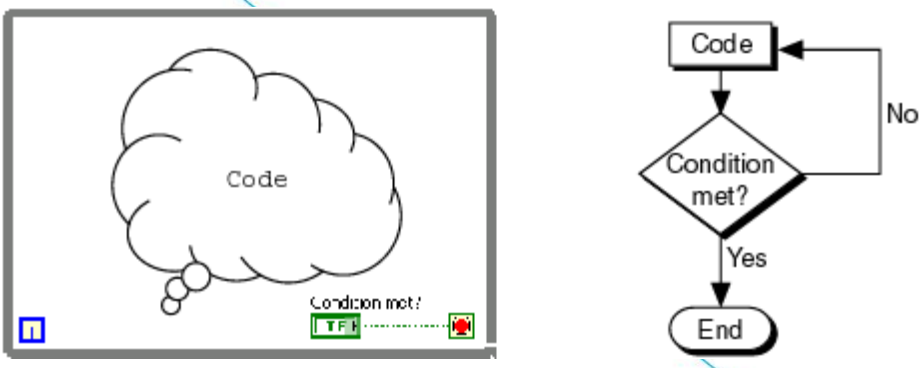
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 3.4 Structures paleta**

*3.3.1 While petlja*

While petlja je programska petlja koja izvršava sekvencijalno grafički kôd koji se nalazi unutar sve dok se ne ispuni neki logički uslov. Izgled strukture i odgovarajući algoritam su prikazani na slici 3.5.

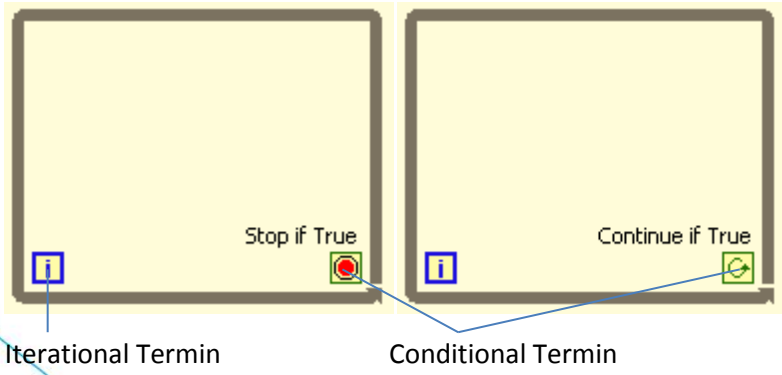



**Slika 3.5 While loop struktura (levo) i algoritam (desno)**

Elementi petlje su okvir, **iteration (i)** terminal i **condition** terminal. Okvir strukture određuje granice njenog domena, odnosno uokviruje kôd koji se izvršava. Iteration terminal je inicijalno postavljen sa leve strane strukture, predstavlja celobrojni numerički terminal izlaznog tipa čija vrednost se inkrementira nakon svakog izvršenja petlje. Inicijalna vrednost terminala je nula. Na desnoj strani se nalazi condition terminal, ulaznog boolean (logičkog) tipa, kojim se kontroliše izvršavanje petlje. Podrazumevani način izvršavanja je **Stop if True**, koje zaustavlja petlju kada se na terminal dovede logička vrednost **True**. Način izvršavanja petlje se može promeniti otvaranjem kontekstnog menija terminala (desni klik na terminal) i izborom opcije **Continue if True**, čime petlja prekida izvršavanje kada se na terminal dovede logička vrednost **False** (slika 3.6).



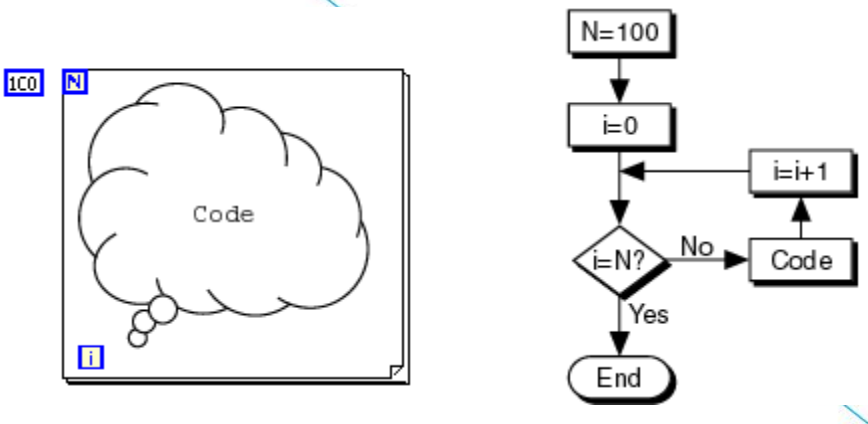
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 3.6 Načini izvršavanja petlje**

*3.3.2 For petlja*

For petlja je programska petlja koja izvršava deo kôda unapred određen broj puta. Izgled strukture je prikazan na slici 3.7. Petlja se može postaviti na blok dijagram iz palete **Programming/Structures**.




**Slika 3.7 For petlja (levo) i algoritam (desno)**

Elementi For petlje su okvir, **iteration (i)** terminal i **count (N)** terminal. Okvir petlje određuje granice njenog domena, odnosno uokviruje kôd koji se izvršava. Iteration terminal, kao i kod While petlje, je inicijalno postavljen sa leve strane strukture, predstavlja celobrojni numerički terminal izlaznog tipa čija vrednost se inkrementira nakon svakog izvršenja petlje. Inicijalna vrednost terminala je nula.

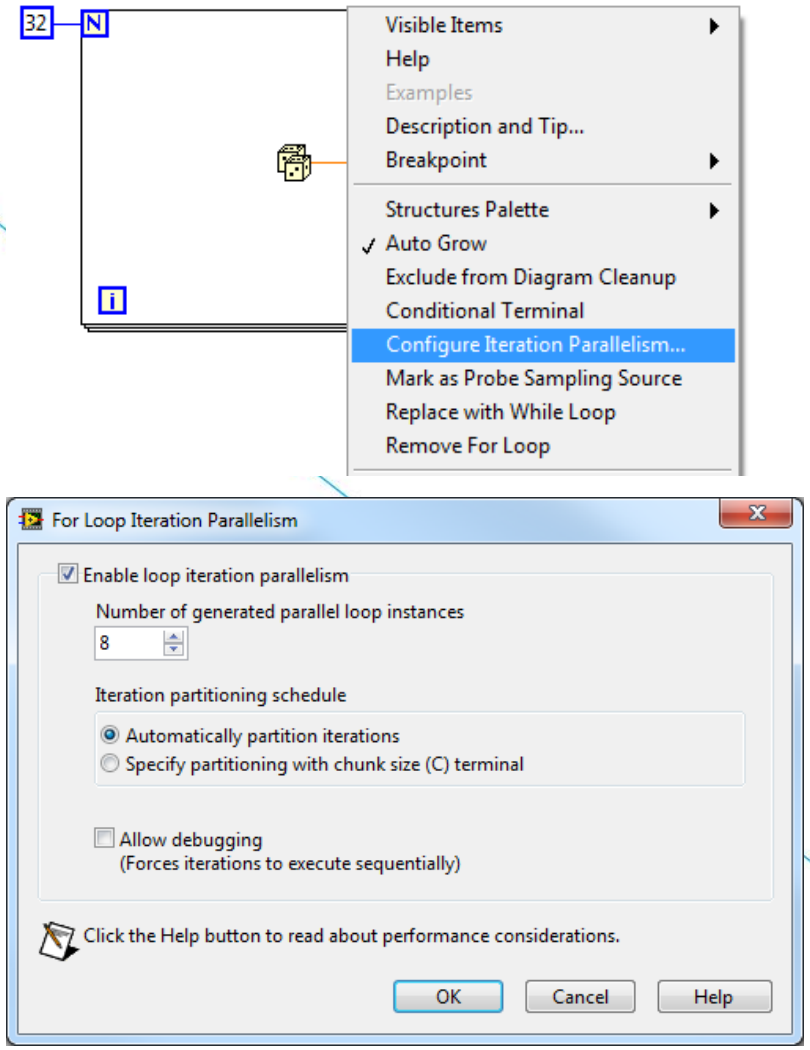
Count terminal se nalazi u gornjem levom uglu strukture. Terminal je ulazni, celobrojnog numeričkog tipa. Broj koji se dovodi na terminal određuje broj ponavljanja petlje.

For petlja se može konfigurisati za paralelno izvršavanje, pri čemu se kôd izvršava u različitim *thread*-ovima. Ovim se postižu bolje performanse, pogotovo na višeprocesorskim platformama. Prilikom korišćenja paralelizma, treba imati u vidu broj procesorskih jezgara kako bi se postigle optimalne performanse. Takođe treba uvažiti činjenicu da u većini slučajeva nije moguće paralelizovati kôd kod koga se u iteraciji petlje koriste podaci koji su rezultat izvršavanja prethodnih iteracija.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Konfiguracija paralelizma se ostvaruje pomoću dijaloga koji se pokreće iz kontekstnog menija For petlje, izborom opcije **Configure Iteration Parallelism**, slika 3.8.



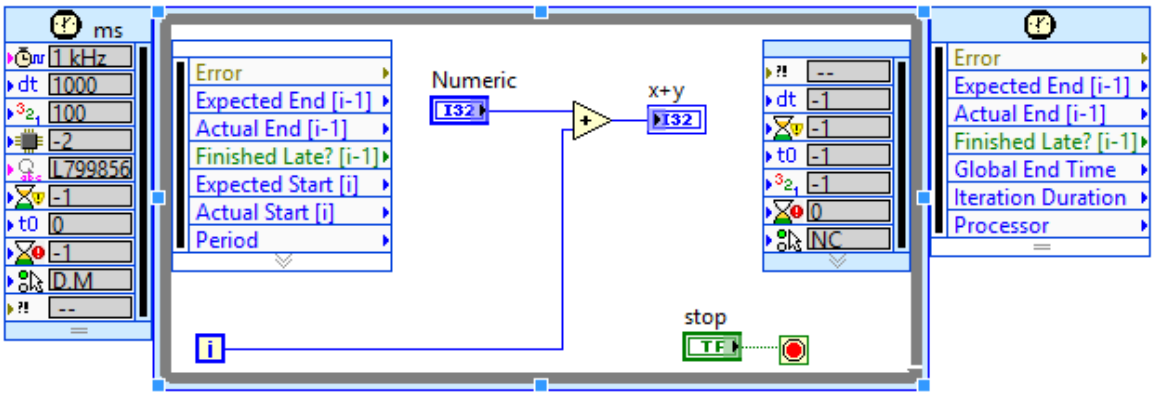

**Slika 3.8 Kontekstni meni For petlje i dijalog za konfiguraciju**

Nakon konfigurisanja i zatvranja dijaloga, ispod count terminala će biti dostupan **Parallelism (P)** terminal, koji se može programski menjati. Petlja će biti paralelizovana u onoliki broj thread-ova, koliki je minimum između broja dovedenog na parallelism terminal i broja unetog u dijalod (**Number of generated parallel loop instances**).

Paralelizam petlje je moguće precizno kontrolisati opcijom **Specify partition with chunk size (C) terminal**, čijim izborom postaje dostupan **Chunk (C)** terminal. Povezivanjem numeričkog niza na navedeni terminal precizno se može odrediti broj iteracija petlji koje izvršava svaki thread-a. Povezivanjem numeričke konstante na terminal, definiše se najmanji broj iteracija koje preuzima jedan thread.

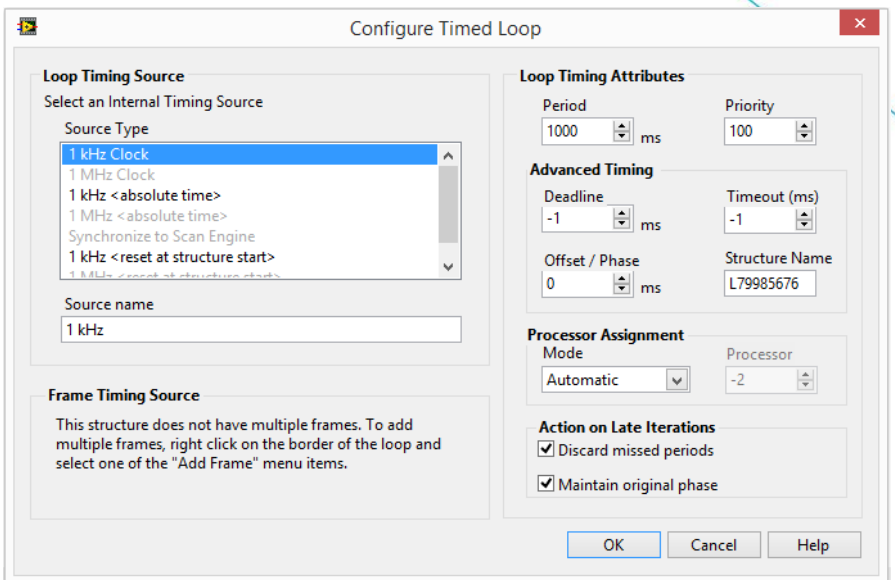
*3.3.3 Timed loop petlja*

Vremenski definisane petlje **(Timed Loop)** omogućavaju precizno vremensko izvršavanje kôda, sinhronizaciju i multithreading, izbor CPU-a na kome će se thread izvršiti (slika 3.9).




**Slika 3.9 Timed Loop petlja**

Vremenski definisana petlja je slična While petlji, način izvršavanja kôda i algoritam su identični. Iteration i condition terminal imaju istu funkciju i ponašaju se na identičan način. Vremenski definisana petlja se može konfigurisati pomoću dijaloga koji se aktivira iz kontekstnog menija (slika 3.10). Prilikom konfiguracije, moguće je izabrati izvor sinhronizacije (source type), pomoću koga se postiže sinhronizacija više različitih petlji u okviru virtuelnog instrumenta.




**Slika 3.10 Dijalog za konfiguraciju vremenski definisane petlje**



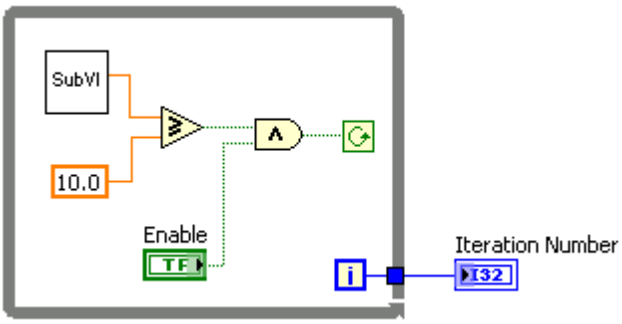
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

U dijalogu se zadaju parametri koji određuju period izvršavanja, kašnjenje u odnosu na izvor sinhronizacije (offset), vremensko prekoračenje (timeout), maksimalno vreme izvršavanja (deadline) i prioritet izvršavanja petlje. Dodeljivanje thread-ova može biti automatsko ili eksplicitno određeno. Svaka timed loop petlja ima jedinstveno ime.

Vremenski definisane petlje su neophodne kod procesa koji zahtevaju preciznu sinhronizaciju i vremenska ograničenja, kao kod *real-time* procesa.

*3.3.4 Tuneli, šift registri i povratne petlje*

Programske petlje obrađuju podatke koji su prethodno generisani ili obrađivani u aplikaciji, a rezultati obrade u petlji se prosleđuju aplikaciji nakon njenog izvršenja. Za prosleđivanje podataka unutar i van petlje služe **tuneli**. Tunel je prikazan kao kvadrat na ivici strukture, čija boja odgovara boji veze vezanoj za tunel, odnosno tipu podatka koji se prosleđuje (slika 3.11). Tunel će biti automatski generisan kada se veza dovede do ivice strukture.




**Slika 3.11 Tunel na izlazu While petlje**

Tuneli mogu biti ulaznog i izlaznog tipa, u zavisnosti od toka podatka. Kod tunela ulaznog tipa, petlja počinje sa izvršavanjem tek nakon što je podatak postavljen na ulaz. Kod tunela izlaznog tipa, podatak je dostupan nakon izvršenja petlje.

Tuneli su univerzalan mehanizam razmene podataka između programskih struktura u LabVIEW grafičkom kôdu.

Ukoliko je u iteraciji programske petlje potrebno koristiti podatak dobijen obradom prethodne iteracije, koriste se šift registri (**Shift Register**) i povratne petlje (**Feedback Loop**).

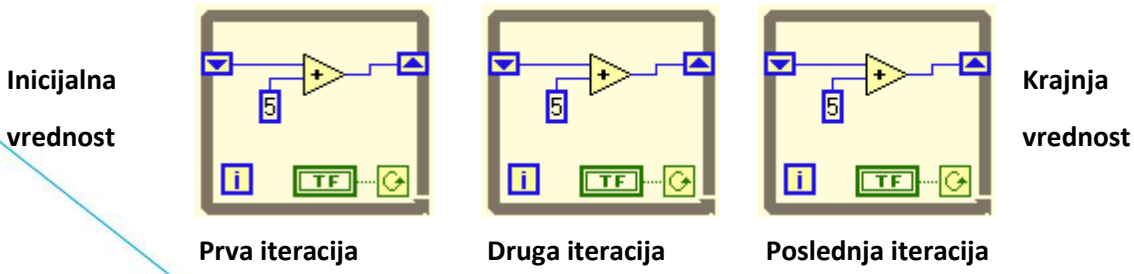
*Šift registar* je ulazno/izlazni tunel koji omogućuje prenos podataka između iteracija petlje. Razmotrimo petlju koja u svakoj iteraciji dodaje konstantnu vrednost sumi, prikazanoj na slici 3.12. Šift registar je predstavljen kao kvadrat sa strelicom nadole (ulazni tunel), odnosno strelicom nagore (izlazni tunel). Boja šift registra odgovara tipu podatka, u konkretnom slučaju celobrojnom numeričkom tipu. Prilikom prve iteracije,

vrednosti ulaznog tunela šift registra se dodaje konstantna vrednost (5), koja se prosleđuje ka izlaznom tunelu. U narednoj iteraciji se vrednosti prethodne iteracije dovedenoj sa ulaznog tunela



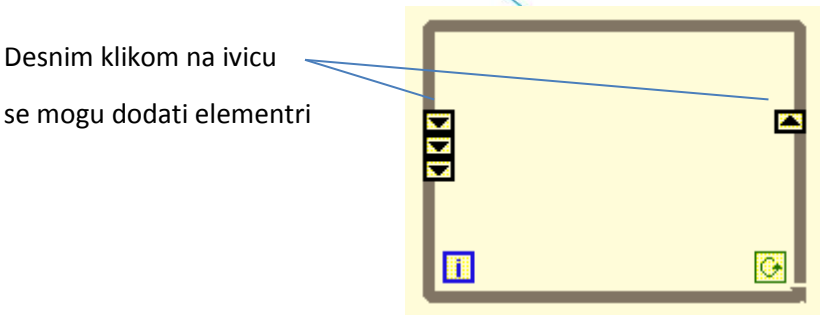
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

dodaje konstantna vrednost i rezultat prosleđuje ka izlaznom tunelu. Ovaj proces se ponavlja dok logička vrednost dovedena na condition terminal ima vrednost True. Nakon završetka izvršavanja petlje, krajnja vrednost je dostupna na izlaznom tunelu.




**Slika 3.12 Petlja sa šift registrom**

Šift registar se može postaviti opcijom **Add Shift Register** iz kontekstnog menija otvorenog na levoj ili desnoj ivici petlje. Ukoliko je registar već postavljen, pomoću kontekstnog menija šift registra se mogu dodati elementi koji predstavljaju vrednosti iz proizvoljnog broja prethodnih iteracija (slika 3.13).

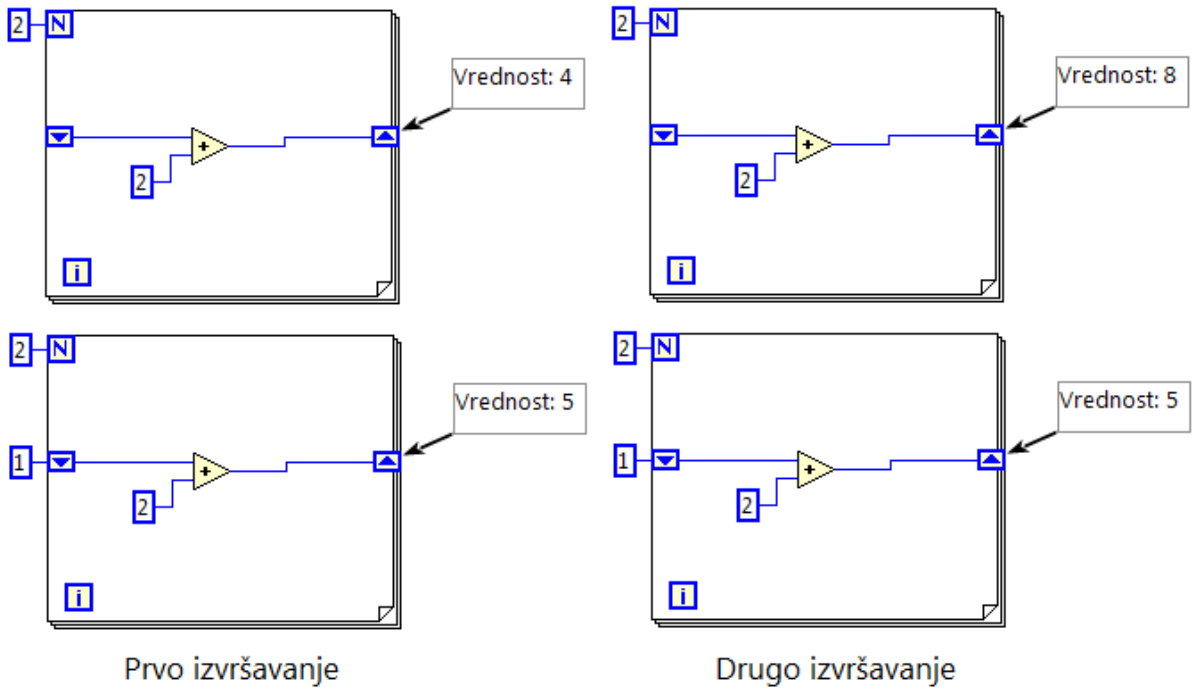



**Slika 3.13 Dodavanje elemenata prethodnih petlji**

Ukoliko na ulazni terminal nije vezan podatak, podrazumeva se vrednost prethodno izvršene petlje ili osnovna (default) vrednost podatka ukoliko petlja nije prethodno izvršavana (slika 3.14).

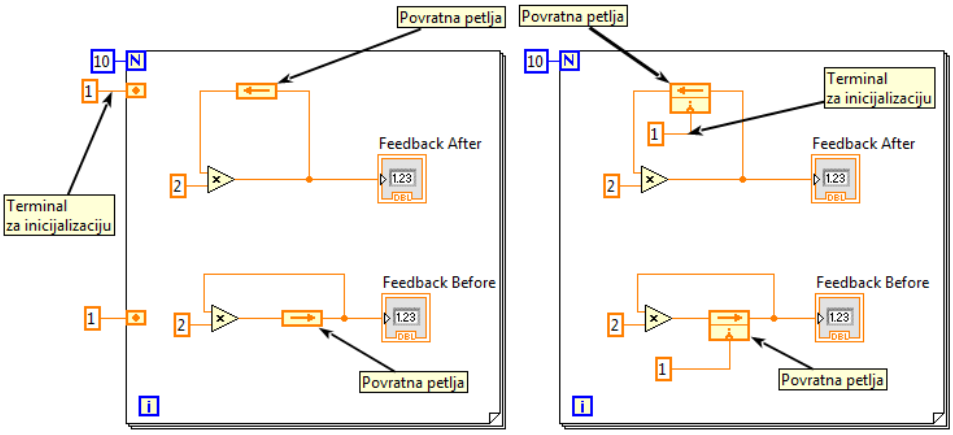


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 3.14 Inicijalne vrednosti šift registra**

Povratne petlje će se automatski javiti ukoliko se u petlji poveže izlaz čvora ili grupe čvorova na ulaz istog čvora, odnosno grupe čvorova. Povratna petlja čuva vrednost podatka do okončanja izvršenja iteracije petlje, i prosleđuje podatke sledećoj iteraciji. Može da prosledi podatak bilo kog tipa, pri čemu boja povratne petlje zavisi od tipa podatka. Povratna petlja se može postaviti i iz palete **Programming/Structures**.




**Slika 3.15 Primer povratne petlje**



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Primer povratne petlje prikazan je na slici 3.14. Obe petlje imaju identičnu funkciju. Sastavni deo petlje je terminal za inicijalizaciju, koji se može postaviti na ivicu strukture radi bolje preglednosti opcijom **Move Initializer One Loop Out**. Ovom opcijom je moguće precizno definisati u odnosu na koju programsku petlju povratna petlja čuva vrednosti. Vezivanjem podatka za terminal za inicijalizaciju, određuje se početna vrednost povratne petlje. Ukoliko terminal nije vezan, podrazumeva se vrednost prethodno izvršene petlje ili osnovna (default) vrednost podatka ukoliko petlja nije prethodno izvršavana, slično kao kod šift registra.

## 3.4 Nizovi

Niz je skup pobrojanih elemenata istog tipa. Određen je tipom podatka, brojem elemenata, dimenzionalnošću i samim elementima. Niz može biti jednodimenzionalan ili višedimenzionalan. Svakom elementu niza se pristupa preko indeksa, pri čemu prvi element niza ima indeks nula, poslednji element niza ima indeks N-1, gde je N broj elemenata niza. Tip elemenata niza je proizvoljan, a broj elemenata niza je ograničen memorijom.

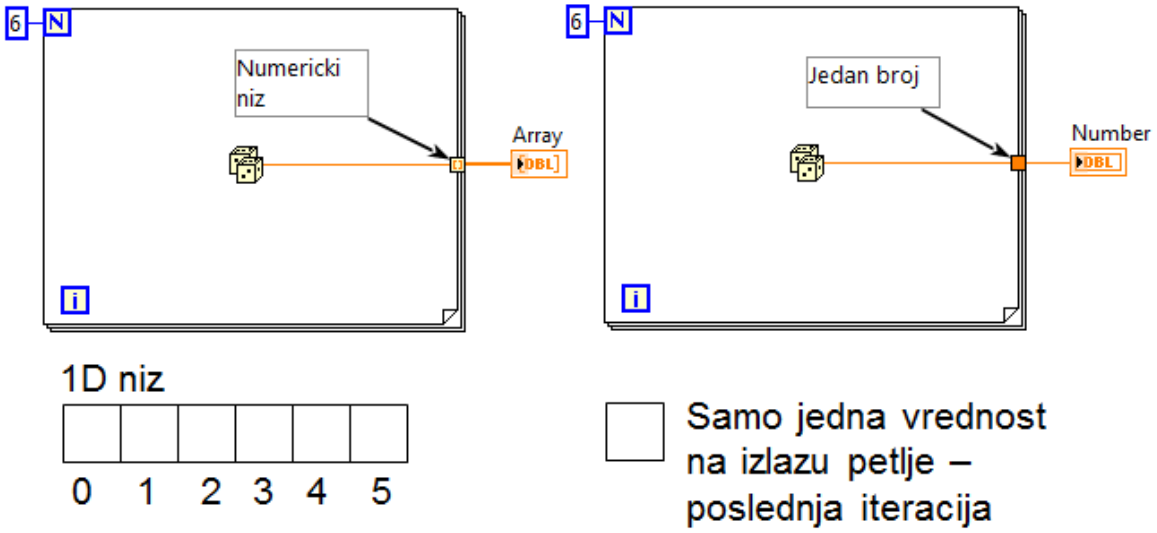
U LabVIEW grafičkom jeziku, nizovi se tretiraju kao i ostali tipovi podataka, što implicira postojanje odgovarajućih kontrola, indikatora i konstanti nizova. Kontrole i indikatori nizova se mogu kreirati na front panelu izborom **Array** kontrole iz **Controls/Array, Matrix & Cluster** palete i postavljanjem elementa (kontrole ili indikatora) konkretnog tipa. Kreirani niz će biti istog tipa podatka kao i umetnuti element, kontrola ili indikator u zavisnosti od samog umetnutog elementa. Dimenzionalnost niza se može povećati opcijom **Add Dimension** iz kontekstnog menija. Na blok dijagramu je moguće realizovati konstantni niz izborom **Array Constant** iz **Programming/Array** palete i postavljanjem konstante elementa čiji tip odgovara željenom tipu niza.

### *3.4.1 Autoindeksiranje*

Za manipulaciju i obradu elemenata nizova se najčešće koriste programske petlje, mehanizmom koji se zove *autorindeksiranje*. Najjednostavniji primer autoindeksiranja je generisanje elemenata niza, prikazan na slici 3.16.



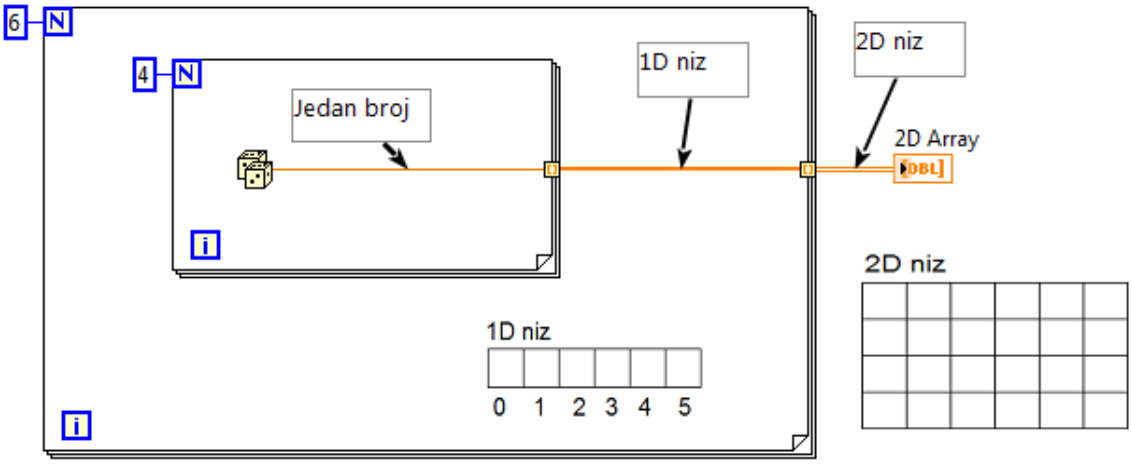
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 3.16 Autoindeksiranje**

Kada se veza dovede do ivice For petlje, biće kreiran autoindeksni tunel, koji akumulira vrednosti iz svih iteracija u niz. U slučaju While petlje, biće kreiran klasičan tunel koji prosleđuje vrednost poslednje iteracije. Ponašanje tunela se može regulisati kontekstnim menijem, izborom podmenija **Tunnel Mode**.

Višedimenzionalni nizovi se mogu realizovati višestrukim petljama. Svaka petlja sa autoindeksnim tunelom dodaje jednu dimenziju nizu (slika 3.17).

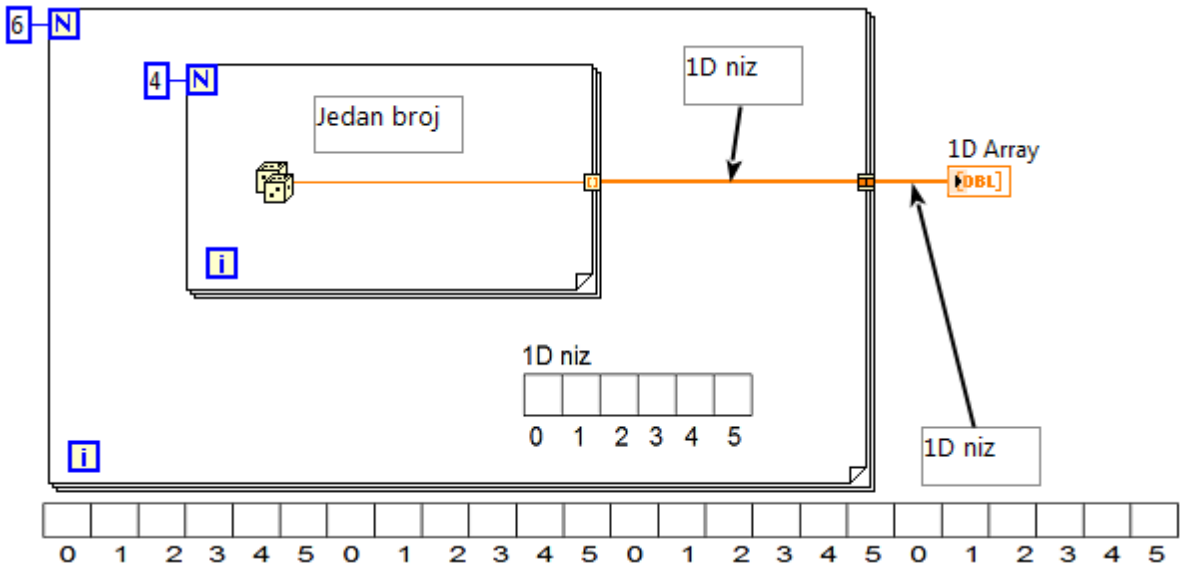



**Slika 3.17 2D niz kreiran For petljama**

Osim kreiranja višedimenzionalnih nizova, izborom opcije **Tunnel Mode/Concatenating** moguće je za svaku iteraciju spoljne petlje nastaviti jednodimenzionalni niz (slika 3.18).

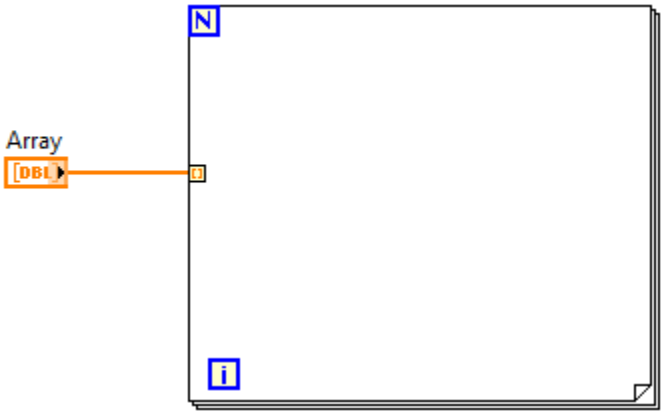


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 3.18 Nastavljanje (konkatenacija) nizova**

Mehanizam autoindeksiranja se može iskoristiti i kod obrade pojedinačnih elemenata niza, ukoliko se niz dovede na ulaz For petlje (slika 3.19).




**Slika 3.19 For petlja sa ulaznim nizom**

U ovom slučaju, nije potrebno odrediti vrednost count terminala, već će se petlja izvršiti onoliko puta koliko ima elemenata niza.

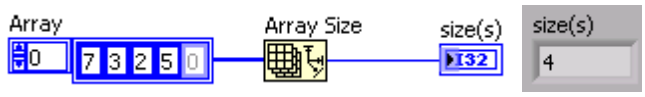
*3.4.2 Najčešće korišćene funkcije nizova*

3.4.2.1 Array Size

Funkcija **Array Size** se poziva iz palete Programming/Array. Ulazni podatak je niz proizvoljnog tipa i dimenzionalnosti. Ukoliko je ulazni niz jednodimenzionalan, izlazni podatak je celobrojna numerička vrednost koja predstavlja broj elemenata niza (slika 3.20).



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 3.20 Array Size funkcija**

U slučaju višedimenzionalnih nizova, izlazni podatak je numerički niz čije su vrednosti jednake veličini svake dimenzije niza. U tom slučaju, broj elemenata ulaznog niza je jednak proizvodu elemenata izlaznog numeričkog niza.

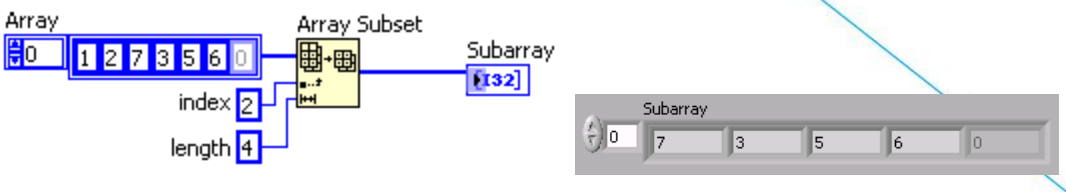
3.4.2.2 Initialize Array

**Initialize Array** funkcija kreira niz čiji su elementi jednaki. Ulazni parametri su vrednost elementa i broj elemenata (slika 3.21). Funkcija Initialize Array je *polimorfna*, tako da je moguće dodavanjem većeg broja ulaza (razvlačenjem funkcije na dole) kreirati višedimenzionalni niz. U ovom slučaju, osim vrednosti elementa, ulazne parametre čine veličine svake dimenzije niza.

**Slika 3.21 Initialize Array funkcija**

3.4.2.3 Array Subset

**Array Subset** funkcija izdvaja podniz iz datog niza (slika 3.22). Ulazni parametri su niz, indeks elementa od koga počinje izdvajanje i broj elementa koji se izdvajaju. Funkcija je polimorfna, tako da dovođenjem višedimenzionalnog niza na ulaz menja oblik i zahteva indeks elementa i broj elementa za svaku dimenziju.




**Slika 3.22 Array Subset**

3.4.2.4 Build Array

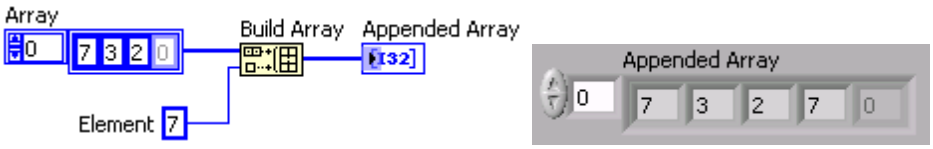
**Build Array** takođe spada u polimorfne funkcije. Ova funkcija može poslužiti za kreiranje niza od elemenata, dodavanje elemenata nizu, spajanje nizova ili dodavanje dimenzije nizu. Funkcija može prihvatiti proizvoljan broj ulaznih parametra, razvlačenjem funkcije na dole, koji mogu biti nizovi ili skalari (pojedinačni elementi).

Ukoliko su ulazni parametri niz i pojedinačni elementi, build array će kreirati niz koji se sastoji od elemenata niza i dodatih elemenata, sa redosledom elemenata koji odgovara redosledu ulaznih



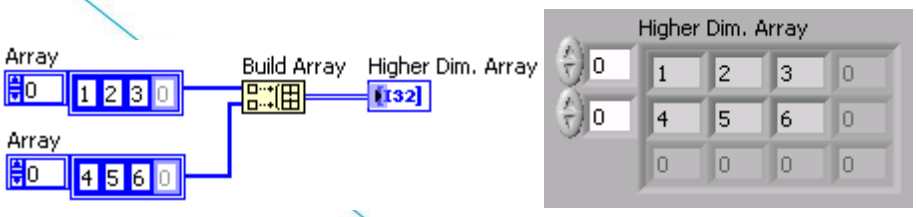
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

parametara odozgo nadole (slika 3.23). Ukoliko su ulazni parametri pojedinačne vrednosti, biće kreiran niz čiji će elementi odgovarati vrednostima parametara.



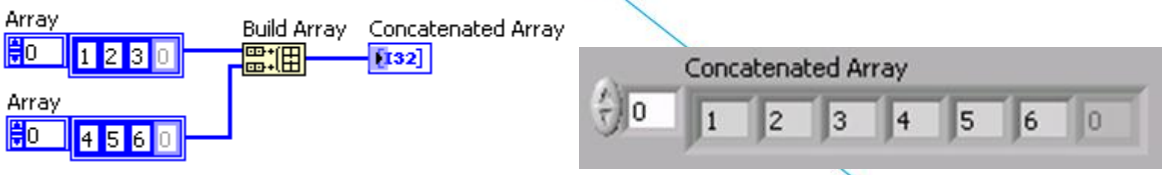

**Slika 3.23 Dodavanje elemenata nizu**

Ukoliko su parametri funkcije nizovi, Build Array će kreirati višedimenzionalni niz (slika 3.24). Dimenzionalnost izlaznog niza je jednaka broju ulaznih parametara, tj. nizova.




**Slika 3.24 Dodavanje dimenzije nizu**

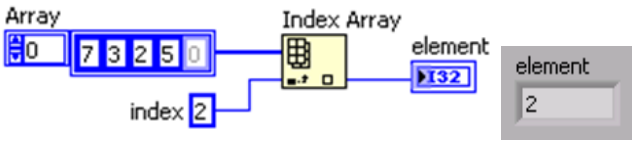
Ponašanje Build Array funkcije se može promeniti izborom opcije **Concatenate Inputs** iz kontekstnog menija, pri čemu će na izlazu biti kreiran niz iste dimenzionalnosti, sačinjen od elemenata ulaznih nizova prema redosledu odozgo nadole (slika 3.25).




**Slika 3.25 Spajanje nizova**

3.4.2.5 Index Array

Index Array funkcija služi za ekstrakciju elemenata iz niza. Ulazni parametri funkcije su niz i indeks elementa koji se izdvaja (slika 3.26).

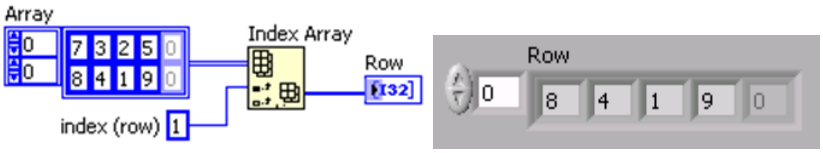



**Slika 3.26 Izdvajanje elementa iz jednodimenzionalnog niza**

Ukoliko je niz višedimenzionalan, moguće je izdvojiti jednu dimenziju (red) iz niza (slika 3.27).

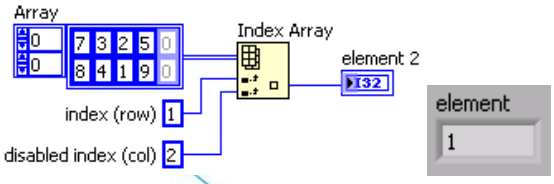


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 3.27 Izdvajanje reda**

Prilikom izdvajanja elementa kod višedimenzionalnog niza, ulazne parametre čine indeksi elementa koji se izdvaja po svim dimenzijama ().

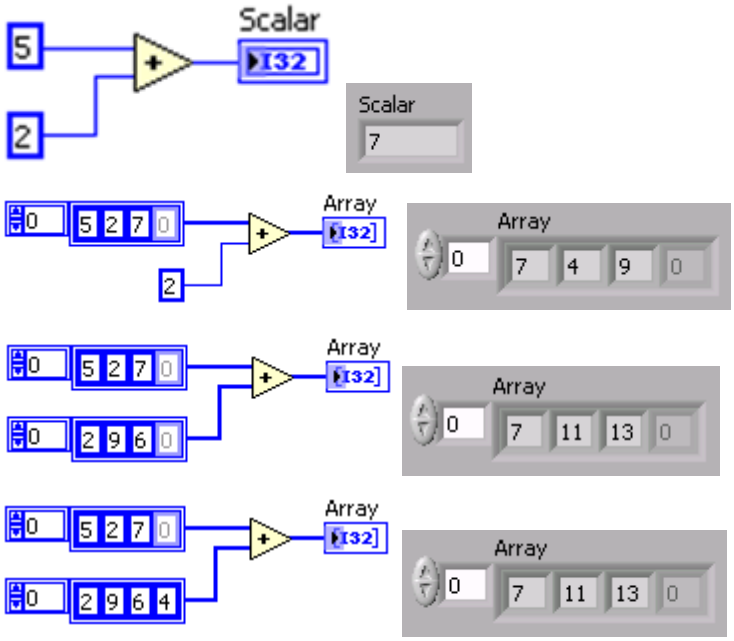



**Slika 3.28 Izdvajanje elementa iz dvodimenzionalnog niza**

Funkcija indeks array se može proširiti razvlačenjem nadole, pri čemu je moguće izdvojiti više od jednog elementa niza. Ukoliko se dovedu ulazni parametri za indekse elemenata koji se izdvajaju, biće izdvojeni elementi po redosledu od poslednjeg navedenog indeksa.

*3.4.3 Polimorfizam*

Većina elementarnih funkcija u LabVIEW kôdu je polimorfna, tj. prihvata različite tipove ulaznih podataka. Najznačajnije polimorfne funkcije su aritmetičke funkcije, koje kao ulazne parametre mogu prihvatiti različite numeričke tipove, kao i nizove numeričkih vrednosti. Primer funkcije sabiranja je dat na slici 3.29.






Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**Slika 3.29 Funkcija sabiranja sa različitim ulaznim parametrima: sabiranje dva skalara, niza i skalara, dva niza istog broja elemenata i dva niza različitog broja elemenata**

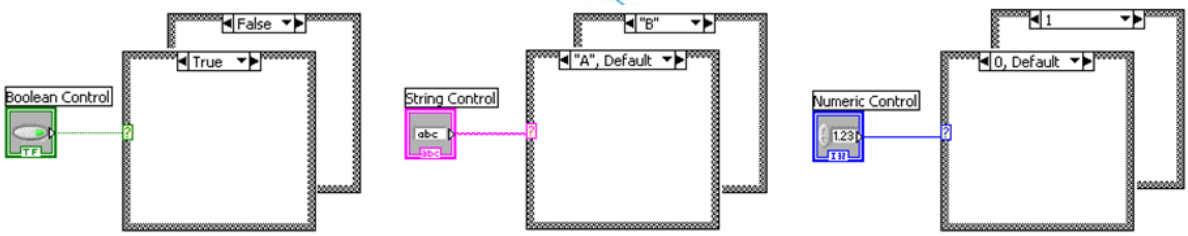
### 3.5 Programske strukture

Programske strukture služe za kontrolu toka aplikacije, programska grananja i precizno definisanje redosleda izvršavanja pojedinih funkcija, odnosno delova kôda.

#### *3.5.1 Case struktura*

**Case** struktura omogućava izvršavanje kôda u zavisnosti od nekog logičkog uslova. U tekstualnim programskim jezicima sličnu namenu imaju Case i If dekleracije. Struktura se može postaviti na blok dijagram izborom iz **Programming/Structures** palete. Slično kao kod programskih petlji, Case strukturom se može obuhvatiti postojeći kôd. Case struktura se sastoji od okvira, selektorskog terminala i labele. Okvirom strukture je oivičen kôd raspoređen u *subdijagramima*, kojih može biti dva ili više. Subdijagrami su poređani kao špil karata, tako da se u jednom trenutku vidi samo jedan. *Labela subdijagrama* označava logički uslov pod kojim će prikazani subdijagram biti izvršen. Klikom na strelice na vrhu strukture bira se prethodni ili naredni subdijagram, respektivno. Samo se jedan subdijagram izvršava, zavisno od logičke, numeričke ili znakovne vrednosti, koja je povezana za *selektorski terminal* (označen sa ?).

Ukoliko je **Boolean** vrednost povezana za selektorski terminal struktura ima dva subdijagrama: **FALSE** i **TRUE** (slika 3.30).




**Slika 3.30 Case strukture sa logičkom (Boolean), znakovnom i numeričkom selektorskom varijablom**

U slučaju da je numerički ili znakovni tip podatka povezan na selektorski terminal, struktura može imati beskonačan1 broj subdijagrama. Prilikom postavljanja strukture, dostupna su samo dva subdijagrama, ali se taj broj može povećati dodavanjem novih. Opcijama **Add Case After** i **Add Case Before** iz kontekstnog menija strukture dodajemo nove subdijagrame. Opcijom **Duplicate Case** se kopira, a opcijom **Delete Case** se briše trenutno aktivan subdijagram.

Moguće je odrediti više od jedne vrednosti dodavanjem varijabli u labelu subdijagrama, pri čemu će se on izvršiti ukoliko je bar jedna dovedena na selektorski terminal. Case struktura mora imati definisan subdijagram koji će se izvršiti ukoliko se na selektorski terminal dovede vrednost koja nije obuhvaćena nijednim drugim subdijagramom. Ovaj subdijagram je označen kao **Default** subdijagram.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Opcijom **Make This the Default Case** iz kontekstnog menija bira se Default subdijagram. Ukoliko je selektorska varijabla numerička, labela subdijagrama može biti

• jedan celi broj (0 ili 1, kao na slici 3.30),

• lista brojeva odvojenih zarezom (na primer 1,2,5,15,35),

• opseg brojeva (**10..20** – opseg od 10 do 20, **..10** – brojevi manji ili jednaki 10, **10..** – brojevi veći ili jednaki 10).

Labele subdijagrama moraju biti celobrojne.

3.5.1.1 Enumeracije i prstenovi

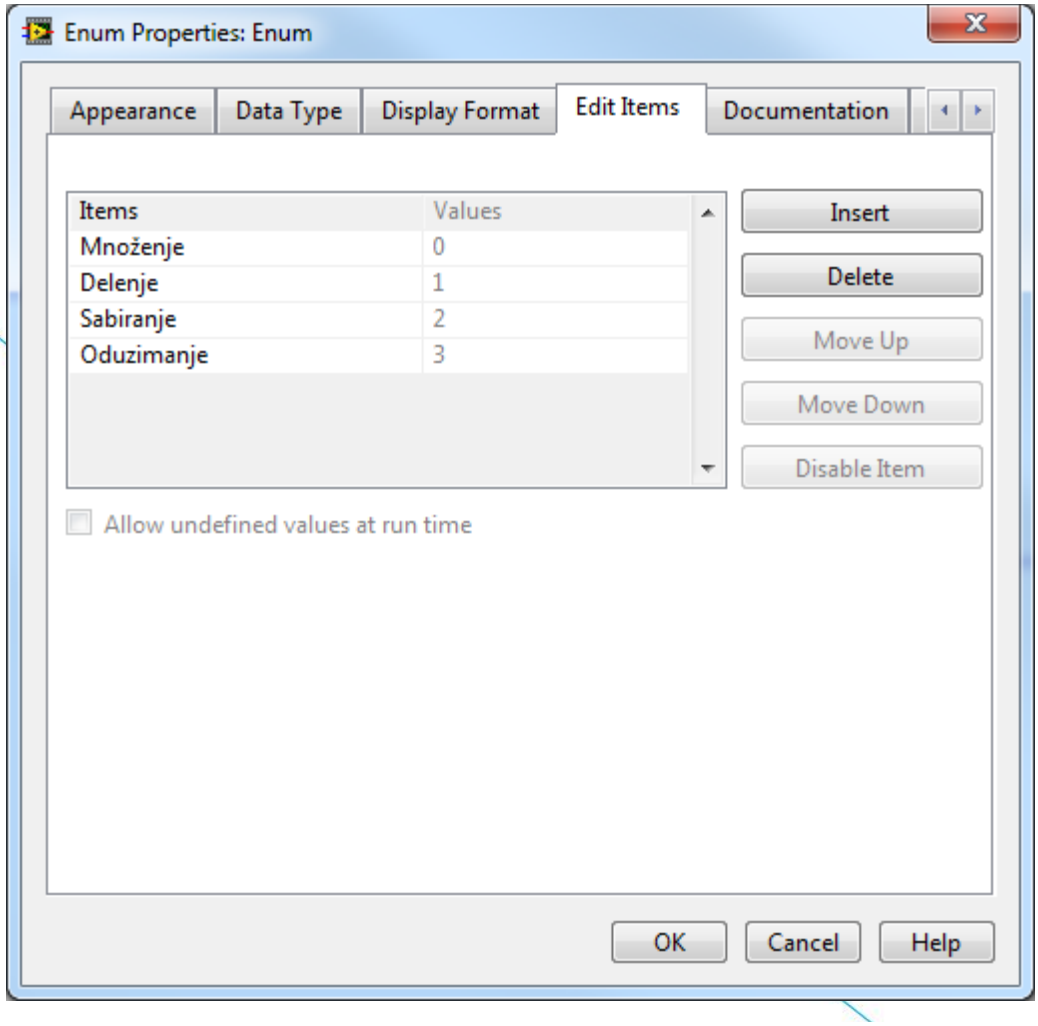
Selektorske varijable mogu biti i *enumeracije*. Enumeracije (Enum) su tip celobrojnih promenljivih u određenom, konačnom opsegu. Mogu se dodati kao kontrole na front panel iz palete **Ring & Enum**, ili na blok dijagramu kao konstanta izborom **Programming /Numeric/Enum Constant.** Svakoj vrednosti se dodeljuje celi broj sekvencijalno. Članovi enumeracije se mogu dodati u **Properties/Edit Items** dijalogu (slika 3.31). Prilikom vezivanja enumeracije za selektorski terminal Case strukture, dobija se mogućnost selekcije subdijagrama na osnovu članova enumeracije.

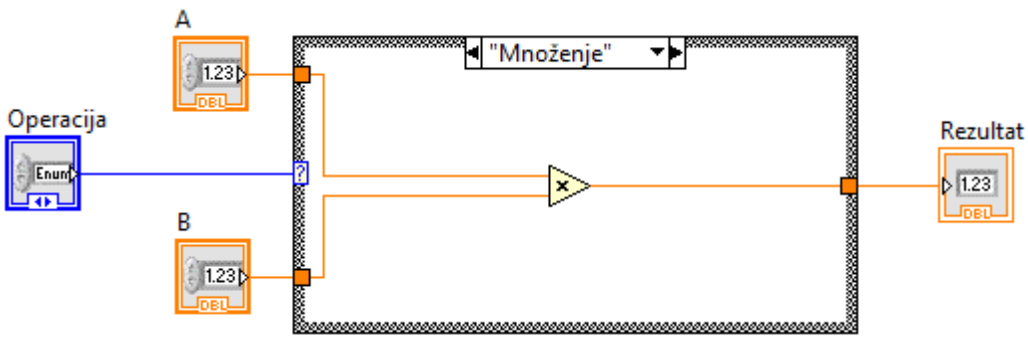
1 Ovo ne treba shvatiti doslovno. Broj subdijagrama je ograničen opsegom koji određeni tip podataka ima i pre svega, memorijom.

Prsten (**Ring**) je tip podataka sličan enumeraciji, s tom razlikom što je moguće imati različite (nesekvencijalne) vrednosti asocirane članovima. Prsten imati necelobrojnu vrednost ili znakovni podatak.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 3.31 Enumeracija i Case struktura kontrolisana enumeracijom**

3.5.1.2 Tuneli

Tuneli kod Case strukture se ponašaju slično kao kod petlji. Pojaviće se automatski ukoliko se dovede veza do ivice strukture. Ukoliko je tok podataka od spolja ka strukturi, tunel je ulaznog tipa. Podaci na



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT

svim ulaznim terminalima Case strukture (tuneli i selektorski terminal) su dostupni za sve subdijagramne. Nije neophodno da svi subdijagrami koriste ulazne podatke. Ukoliko je tok podataka iz strukture ka spolja, tunel je izlaznog tipa. Ukoliko bar jedan subdijagram daje vrednost izlaznom tunelu, i ostali moraju dati izlaznu vrednost. Kada povežemo izlaz u jednom subdijagramu, tunel se pojavljuje na istoj lokaciji u svim slučajevima u obliku kvadrata sa belom unutrašnjošću. **Broken Run**

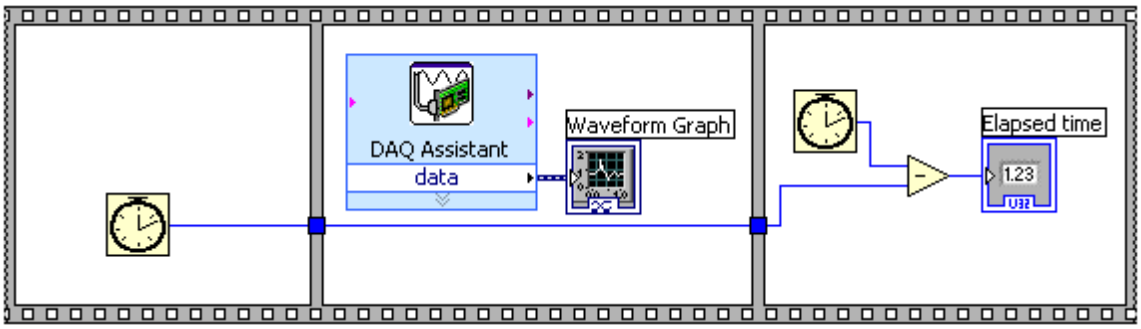
dugme će biti prisutno dok se izlazni tunel ne poveže u svakom subdijagramu. Desnim klikom na tunel se može odrediti podrazumevana vrednost opcijom

**Use Default If Unwired**, čime se eliminiše potreba za povezivanjem u svakom subdijagramu.

*3.5.2 Flat Sequence struktura*

U tekstualnim programskim jezicima redosled dekleracija određuje redosled izvršavanja kôda. U LabVIEW, redosled nije toliko trivijalan. **Flat Sequence** strukture se mogu iskoristiti za precizniju kontrolu toka programa. Kôd se može rasporediti u specificirane sekvence i time odrediti redosled izvršavanja. **Flat Sequence** struktura se može postaviti izborom iz **Programming/Structures** palete.

Prilikom postavljanja, struktura ima jedan subdijagram – *frejm*. Kôd koji se izvršava sekvencijalno je raspoređen po subdijagramima koji se nazivaju *frejmovi*. Cela struktura liči na filmsku traku, gde se frejmovi prikazuju jedan za drugim (slika 3.32). Izvršavanje frejmova je s leva na desno.




**Slika 3.32 Flat Sequence struktura sa tri frejma**

Frejmovi se mogu dodavati Sequence strukturi desnim klikom na ivicu strukture i izborom opcije **Add Frame Before/After**. Frejm se može izbrisati desnim klikom na ivicu frejma i izborom opcije **Delete This Frame**.

Virtuelni instrument prikazan na slici 3.32 koristi **Tick Count** funkciju, koja vraća broj milisekundi očitan sa internog tajmera. Interni tajmer se resetuje prilikom butovanja računara. Prethodni primer može poslužiti za određivanje potrebnog vremena za izvršavanje nekog kôda, u konkretnom primeru meri se vreme akvizicije neke veličine. **Tick Count** funkcija se nalazi u paleti **Programming/Timing**.

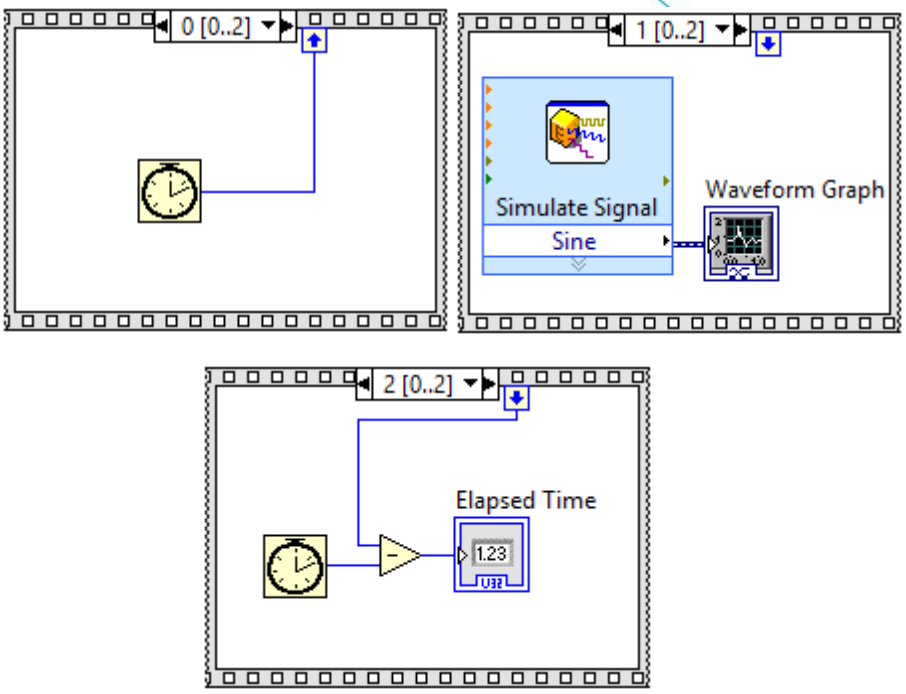
*3.5.3 Stacked Sequence struktura*

**Stacked Sequence** strukture se mogu iskoristiti za precizniju kontrolu toka programa. Kôd se može rasporediti u specificirane subdijagrame i time odrediti redosled izvršavanja. **Stacked Sequence** struktura se može postaviti izborom iz **Programming/Structures**. Prilikom postavljanja, struktura ima jedan subdijagram.

**Stacked Sequence** struktura ima sličnu funkciju kao i **Flat Sequence** struktura, pri čemu su subdijagrami postavljeni kao kod **Case** strukture, jedan iznad drugog. Kôd koji se izvršava sekvencijalno je raspoređen po subdijagramima. Cela struktura liči Case strukturu, pri čemu se izvršavaju svi subdijagrami prema redosledu. Na vrhu strukture se nalazi labela koja označava redni broj subdijagrama koji je prikazan, kao i ukupan broj subdijagrama. Samo je jedan subdijagram vidljiv, može se izabrati iz padajućeg menija sa vrha strukture.

Subdijagrami se mogu dodavati strukturi izborom opcije **Add Frame Before/After** iz kontekstnog menija strukture. Subdijagram se može izbrisati izborom opcije **Delete This Frame**. Subdijagram se može iskopirati u novi izborom opcije **Duplicate Frame**.

Podaci se prosleđuju preko posebnih tunela, takozvanih lokala, čija strelica ukazuje da li podaci ulaze ili izlaze iz subdijagrama. Lokal se može dodati izborom opcije **Add Sequence Local** iz kontekstnog menija strukture. Strelica lokala prikazuje smer toka podataka, i određuje se vezivanjem za ulaz/izlaz funkcija, kontrola ili indikatora unutar subdijagrama. Lokal koji je vezan kao izlazni lokal nekog subdijagrama, dostupan je kao ulazni lokal samo subdijagramima koji se kasnije izvršavaju.






Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**Slika 3.33 Stacked Sequence struktura. Na gornjoj ivici strukture se vide lokali celobrojnog numeričkog tipa**

Na slici 3.33 je prikazan virtuelni instrument realizovan pomoću **Stacked Sequence** strukture sa identičnom funkcijom kao virtuelni instrument na slici 3.32.

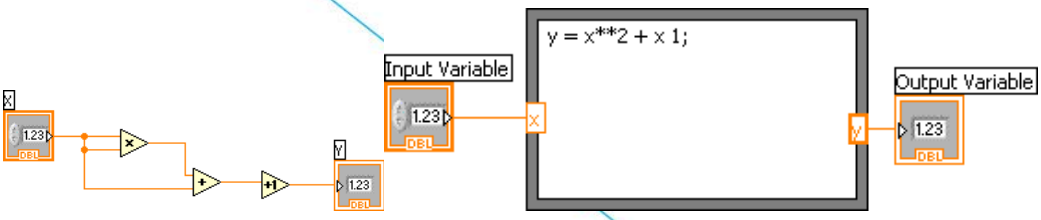
*3.5.4 Formula Node struktura*

**Formula Node** struktura služi za iplementiranje komplikovanijih aritmetičkih formula, pri čemu se dobija na preglednosti. Struktura se nalazi u **Programming/Structures** paleti. Promenljive se kreiraju

na ivicama strukture. Promenljive su

*case-sensitive*. Svaka dekleracija se završava tačka-zarezom (;). Formula Node raspolaže velikim skupom funkcija, koje se mogu pronaći u LabVIEW Help-u (odeljak 2.3.1).

Primer virtuelnog instrumenta realizovanog elementarnim funkcijama i Formula Node strukturom je prikazana na slici 3.34.




**Slika 3.34 Virtuelni instrument realizovan elementarnim funkcijama i Formula Node strukturom**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# Sistemi u realnom vremenu

**Predavači:** Marko Nikolić i Ivan Todorović

**Ciljevi kursa:**

Kurs obuhvata glavne koncepte i metodologije razvoja softvera za mikrokontrolere sa prosečnim resursima, sa ili bez operativnog sistema, kao i za ARM Cortex-A procesore pod operativnim sistemima Linux i Android.

**Ishod kursa:**

Polaznici kursa će biti u stanju da razumeju glavne principe programiranja mikrokontrolera, da se bliže upoznaju sa Linux kernelom i da razumeju osnovne tehnike sistemskog programiranja pod Linux i Android operativnim sistemom za namenske (embedded) platforme.

**Teorijska nastava:**

Kurs je podeljen u dva dela. Prvi deo kursa je posvećen razvoju softvera za mikrokontrolere sa skromnijim i prosečnim resursima, dok se drugi deo kursa bavi glavnim principima sistemskog programiranja pod operativnim sistemima Linux i Android za ARM Cortex-A klasu procesora.
Prvi deo kursa je baziran na korišćenju mikrokontrolera iz familije MSP430 Texas Instruments-a. Kurs obrađuje nekoliko tema:

- Definicija mikrokontrolera. Arhitektura MSP430 mikrokontrolera.
- Glavni principi razvoja softvera za mikrokontrolere.
- Razvojna okruženja za MSP430.
- Pregled periferija MSP430 mikrokontrolera (digitalni ulaz/izlaz, tajmeri, serijski portovi, A/D konvertori, itd.) sa odgovarajućim primerima softvera.
- Glavni principi operativnih sistema u realnom vremenu (RTOS).
- Pregled RTOS za MSP430 sa naglaskom na FreeRTOS.

Drugi deo kursa posvećen je operativnim sistemima Linux i Android na namenskim (embedded) računarskim sistemima, sa primerima na Texas Instrumentsovom sistem-na-čipu AM3984/DM8168, zasnovanom na procesorskom jezgru ARM Cortex-A8. Ovaj deo kursa obrađuje sledeće teme:

- Kros-platformsko programiranje za ARM Cortex platforme na Linux PC-u koristeći GNU alate
- Korišćenje GIT sistema za upravljanje revizijama
- Korišćenje i modifikovanje univerzalnog bootloadera U-Boot
- Osnovni kurs administracije Linuxa
- Proširenje jezgra operativnog sistema Linux, uključujući upravljački softver (drajvere) i ekstenzije za rad u realnom vremenu
- Specifičnosti Linuxa za namenske (embedded) platforme



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Android u okruženju namenskih (embedded) sistema

**Praktične vežbe:**

30% to 50% časova je posvećeno praktičnim vežbama.
Prvi deo kursa
Praktične vežbe su pokrivene primerima softvera za sve glavne periferije MSP430 mikrokontrolera (digitalni ulaz/izlaz, tajmer, serijski portovi konfigurisani u UART, SPI i I2C modu). Na kraju kursa biće dat primer korišćenja FreeRTOS-a.
Svi softverski primeri će biti demonstrirani na eksperimentalnoj ploči MSP430F5438. Koristi se MSP-FET430UIF programator (JTAG emulator) i razvojno okruženje *IAR Workbench Kickstart Free IDE* sa ograničenjem veličine koda.
Nakon završenih predavanja i vežbi (12 časova), studenti će dobiti praktični projektni zadatak (2 časa). U okviru projektnog zadataka studenti treba samostalno da razviju softver koji treba da pokaže koliko su dobro razumeli osnovne principe *embedded* programiranja. Nakon praktičnog testa studenti će raditi finalni test za prvi deo kursa (1 čas).
Drugi deo kursa
Praktične vežbe obavljaju se na Linux PC-u i na EPP-Q7-DM8168 modulu zasnovanom na Texas Instrumentsovom sistem-na-čipu DM8168. Studenti će naučiti kako da prilagode jezgro (kernel) operativnog sistema Linux za razne potrebe, uključujući upravljački softver, kao i da povežu kernelske funkcije sa aplikativnim softverom.
Svi primeri demonstriraju se koristeći softver otvorenog koda. Nakon završenih predavanja i vežbi (12 časova), studenti će dobiti praktični zadatak (2 časa) koji se sastoji od prilagođenja Linux jezgra, razvoja pomoćne aplikacije, i prilagođenja korenog sistema fajlova za upotrebu na namenskom sistemu. Nakon praktičnog zadatka, studenti dobijaju završni test za drugi deo kursa (jedan čas).

**Sadržaj kursa**: Predavači treba da kompletiraju kurs iz odabranih oblasti sa svojim planom kursa.

**Trajanje kursa:** Kurs traje 30 časova (24 časa za predavanje i vežbe i 6 časova za testove). Predavanja i vežbe će biti organizovane po blok sistemu i trajaće jedan mesec (4 časa petkom, 4 časa subotom).

**Kurikulum:**

Ceo kurs biće podeljen u dva jednaka dela. Prve dve nedelje su posvećene razvoju softvera za mikrokontrolere (TI MSP430), dok će u poslednje dve nedelje biti prikazani glavni principi sistemskog programiranja pod Linux operativnom sistemom na namenskim (embedded) platformama.

**Ocenjivanje:**
Za svaki deo kursa:

- Softverski projekat (praktični test) biće ocenjen ocenama 1-3. Prelazne ocene su 2 i 3.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Finalni test biće ocenjen ocenama 1-3. Prelazne ocene su 2 i 3.
- Student je položio kurs ako je svaki od testova položio sa prelaznom ocenom.

**Testovi**

Sadržaj finalnog testa i projektnog zadatka nije uključen iz razloga poverljivosti.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 1. Uvod u mikrokontrolere

## 1.1 Definicija mikrokontrolera

Teško je dati preciznu definiciju mikrokontrolera s obzirom na konstantni razvoj poluprovodničke tehnologije. Današnji prosečni mikrokontroleri su znatno moćniji od prvih mikrokontrolera koji su se koristili u svemirskim misijama. Prvi komercijalno dostupan mikroprocesor je bio Intel 4004 (1971. godina). Intel 4004 je bio 4-bitni mikroprocesor, u pakovanju sa 16 pinova (nožica), sa maksimalnom radnom frekvencijom od 740 kHz.

Koncept programabilnog sistema na čipu (System on Chip – SoC) prvi je započeo Texas Instruments. Prvi mikrokontroler Texas Instruments bio je TMS1000 (4-bitni) koji je osim mikroprocesorskog jezgra integrisao i memoriju i ulaze/izlaze.

Mikrokontroleri u sebi integrišu: mikroprocesorsko jezgro, fleš/ROM memoriju, RAM memoriju, digitalne ulaze/izlaze, reset kolo, watch-dog tajmer, tajmere, serijske portove (UART, SPI, I2C, USB, CAN), A/D i D/A konvertore. Integrisanjem svih ovih komponenti dobija se mikrokontroler koji omogućava: znatno smanjenje dimenzija uređaja u koji je ugrađen, mnogo veću pouzdanost i znatno nižu cenu. Mikrokontroleri se mogu razlikovati po: veličini internog RAM-a ili fleša, broju nožica (pinova), broju tajmera, broju i tipu serijskih portova, broju kanala i preciznosti A/D ili D/A konvertora. Osnovni način povezivanja mikrokontrolera sa eksternim kolima je preko ulazno/izlaznih linija podeljenih po grupama (najčešće 8 linija u grupi), koje se nazivaju portovi. Ulazno/izlazni pinovi se obično mogu pojedinačno konfigurisati tako da pripadnu željenoj periferiji mikrokontrolera (npr. jedan pin se može koristititi kao pin opšte namene ili pin koji pripada serijskom portu). Smer pina se takođe može konfigurisati. Moguć je slučaj da je na jedan pin multipleksirano više periferija, tako da postoji mogućnost da samo deo periferija može biti iskorišćeno (kod složenijih mikrokontrolera).

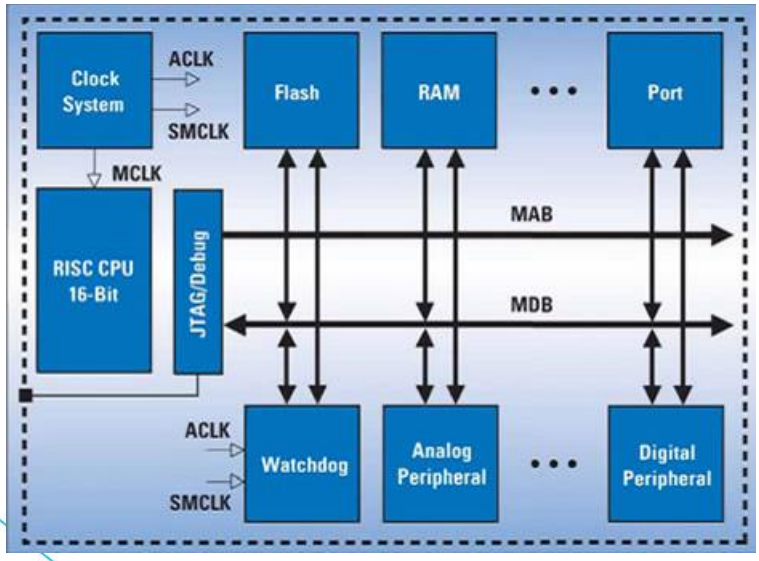
## 1.2 Osnovne karakteristike MSP430 mikrokontrolera

Svi MSP430 mikrokontroleri, bez obzira na familiju sadrže16-bitni procesor sa redukovanim skupom instrukcija (RISC arhitektura).Obrada u okviru jedne instrukcije može biti na nivou reči, bajta ili bita. Magistrala podataka je uvek 16-bitna. Postoji 27 instrukcija i 7 modova adresiranja. Prekidi su vektorski i mogu se podeliti na prekid za reset, nemaskirajuće(NMI) i maskirajuće prekide.

MSP430 ima Von Neumann arhitekturu, što znači da se koristi zajednička adresna magistrala za program i podatke, odnosno sve memorije i periferije su mapirane u jedinstven adresni prostor. Razdvojeni taktovi za procesor (MCLK) i periferije (SMCLK i ACLK) što omogućava različite modove niske potrošnje.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Blok* šema *arhitekture MSP430*

MSP430 sadrži digitalne i analogne ulaze/izlaze (otuda naziv *mixed-signal processor*) – sadrži A/D i D/A konvertore, komparatore, operacione pojačavače.

MSP430 kontroleri se odlikuju niskom potrošnjom, a imaju i više različitih modova niske potrošnje (LPM). Tipična potrošnja u aktivnom modu je 100 µA/MHz, <1 µA u RTC modu, 0.1 µA u modu održavanja RAM. Buđenje iz LPM traje manje od 1 µs. Napon napajanja je u opsegu 1,8 V – 3,6 V Gotovo svi MSP430 kontroleri imaju *brownout* reset kolo sa nultom potrošnjom.

Interni fleš može biti kapaciteta do 512 kB, interni RAM ide do 66 kB, dok je maksimalna radna frekvencija 25 MHz. Najveće pakovanje MSP430 ima 100 pinova.

Za spuštanje i debagovanja koda se koristi JTAG emulator.

MSP430 je dostupan u verzijama sa flešom (F u nazivu) ili ROM memorijom (C u nazivu). Prednost verzije sa flešom je veća fleksibilnost, jer se može obrisati i ponovo programirati (mana viša cena). Početna adresa fleša ili ROM-a zavisi od veličine i menja se od jedne do druge verzije mikrokontrolera. Svi tipovi memorija, uključujući interni RAM i fleš/ROM, registre za specijalne funkcije (SFR) i registre periferija su mapirani u jedinstven adresni prostor. Svaka memorijska lokacija sadrži jedan bajt (ujedno najmanja adresibilna jedinica). CPU može da adresira 8-bitne ili 16-bitne podatke. Instrukcije su 16-bitne, pa su one uvek smeštene na parnim adresama.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**1.3 Adresni prostor** MSP430

| Memory Address | | Description | Access |
|---|---|---|---|
| End: | 0FFFFh | Interrupt Vector Table | Word/Byte |
| Start: | 0FFE0h | | |
| End: | 0FFDFh | Flash/ROM | Word/Byte |
| Start *: | 0F800h | | |
| | 01100h | | |
| End *: | 010FFh | | |
| | 0107Fh | Information Memory | Word/Byte |
| Start: | 01000h | (Flash devices only) | |
| End: | 0FFFh | Boot Memory | Word/Byte |
| Start: | 0C00h | (Flash devices only) | |
| End *: | 09FFh | | |
| | 027Fh | RAM | Word/Byte |
| Start: | 0200h | | |
| End: | 01FFh | 16-bit Peripheral modules | Word |
| Start: | 0100h | | |
| End: | 00FFh | 8-bit Peripheral modules | Byte |
| Start: | 0010h | | |
| End: | 000Fh | Special Function Registers | Byte |
| Start: | 0000h | | |

*Memorijska mapa jednog MSP430 mikrokontrolera*

Tabela vektora prekida se nalazi u opsegu adresa 0FF80h-0FFFFh (kraj adresnog prostora od 64 kB). Svakom vektoru prekida odgovara jedna 16-bitna reč u ovom opsegu, koja predstavlja adresu prekidne rutine (programski brojač se postavlja na ovu vrednost kada se desi odgovarajući prekid). Vektoru prekida sa većim prioritetom odgovara viša adresa u tabeli.

Fleš se može koristiti za program i podatke. Tabele se mogu čuvati u flešu ili ROM-u. Fleš je podeljen na segmente veličine 512 B. Da bi se upisao bajt/reč u neki segment potrebno je da prethodno segment kome pripada bude obrisan (obrisan segment u svim lokacijama ima upisano FFh). MSP430F5438A ima 256 kB fleša.

| | | MSP430F5419A<br>MSP430F5418A | MSP430F5436A<br>MSP430F5435A | MSP430F5438A<br>MSP430F5437A |
|---|---|---|---|---|
| Memory (flash)<br>Main: interrupt vector<br>Main: code memory | Total Size<br>Flash<br>Flash | 128 KB<br>00FFFFh–00FF80h<br>025BFFh–005C00h | 192 KB<br>00FFFFh–00FF80h<br>035BFFh–005C00h | 256 KB<br>00FFFFh–00FF80h<br>045BFFh–005C00h |
| Main: code memory | Bank D | N/A | 23 KB<br>035BFFh–030000h | 64 KB<br>03FFFFh–030000h |
| | Bank C | 23 KB<br>025BFFh–020000h | 64 KB<br>02FFFFh–020000h | 64 KB<br>02FFFFh–020000h |
| | Bank B | 64 KB<br>01FFFFh–010000h | 64 KB<br>01FFFFh–010000h | 64 KB<br>01FFFFh–010000h |
| | Bank A | 41 KB<br>00FFFFh–005C00h | 41 KB<br>00FFFFh–005C00h | 64 KB<br>045BFFh–040000h<br>00FFFFh–005C00h |

*Fleš – adresni opseg za pojedine kontrolere MSP430F5xx*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Informacionu memoriju sadrže samo verzije MSP430 sa flešom. Ova memorija je zapravo sastavni deo fleša, samo što je veličina segmenta manja i iznosi 128 bajtova. Može se koristiti kako za program tako i za podatke – ovaj deo fleša je pogodan za čuvanje podataka, jer su segmenti manji. Broj segmenata se razlikuje (npr. MSP430F5438A ima ukupno 4 segmenta informacione memorije).

Boot memorija (takođe sadrže samo verzije sa flešom) sadrži fabrički kod koji omogućava programiranje fleša (glavni način za programiranje je preko JTAG emulatora). Ovom delu memorije korisnik ne može pristupiti, pa ga nije moguće izbrisati. Bootstrap loader obavlja pojedine funkcije JTAG interfejsa korišćenjem TI protokola za UART komunikaciju sa 9600 baud-a.

Početak RAM memorije i njena veličina se razlikuju od jedne do druge verzije MSP430 (npr. MSP430F5438A ima 16 kB, podeljena u 4 sektora po 4 kB, početna adresa je 001C00h). Početna adresa i veličina namenjena registrima periferija zavise od konkretnog mikrokontrolera (MSP430F5438A ima 4 kB za periferije, počevši od adrese 000000h).

| | | MSP430F5419A<br>MSP430F5418A | MSP430F5436A<br>MSP430F5435A | MSP430F5438A<br>MSP430F5437A |
|---|---|---|---|---|
| RAM | Size | 16 KB | 16 KB | 16 KB |
| | Sector 3 | 4 KB<br>005BFFh–004C00h | 4 KB<br>005BFFh–004C00h | 4 KB<br>005BFFh–004C00h |
| | Sector 2 | 4 KB<br>004BFFh–003C00h | 4 KB<br>004BFFh–003C00h | 4 KB<br>004BFFh–003C00h |
| | Sector 1 | 4 KB<br>003BFFh–002C00h | 4 KB<br>003BFFh–002C00h | 4 KB<br>003BFFh–002C00h |
| | Sector 0 | 4 KB<br>002BFFh–001C00h | 4 KB<br>002BFFh–001C00h | 4 KB<br>002BFFh–001C00h |

**MSP430 arhitektura mikroprocesora**

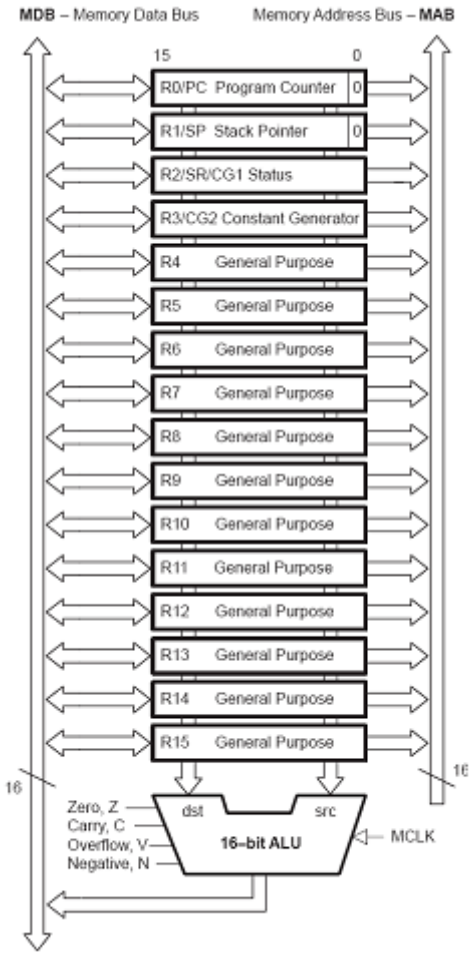
| | | | | |
|---|---|---|---|---|
| Peripherals | Size | 4KB<br>000FFFh–000000h | 4KB<br>000FFFh–000000h | 4KB<br>000FFFh–000000h |

### 1.4 Procesorsko jezgro MSP430

RISC arhitektura je zasnovana na smanjenom broju instrukcija (27), korišćenjem trostepenog pipeline-a za dekodovanje instrukcija. Postoje i 24 emulirane instrukcije – nisu prave mašinske instrukcije, ali se prevode u odgovarajuće mašinske instrukcije. CPU ima 16-bitnu aritmetičko-logičku jedinicu (ALU), 4 namenska registra i 12 radnih registara koji omogućavaju značajne performansi. Često korišćene promenljive se mogu smestiti u radne registre, čime se postiže maksimalna brzina obrade (instrukcije koje operišu nad registrima se brže izvršavaju nego one koje imaju operande u memoriji). Ortogonalni skup instrukcija omogućava da se za bilo koji tip instrukcije može koristiti bilo koji mod adresiranja. Ovim se postiže velika efikasnost kompajlera za više programske jezike, poput C-a.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Blok dijagram mikroprocesorskog jezgra MSP430*

Aritmetičko logička jedinica (ALU) podržava operacije sabiranja, oduzimanja, poređenja i logičke operacije (I, ILI, isključivo ILI). ALU operacije utiču na bitove statusnog registra (Z, N, C, V). Hardverski množač je realizovan kao poseban modul.

Procesorsko jezgro MSP430 poseduje 4 registra posebne namene (R0 – R3) i 12 registara opšte namene (R4 – R15). R0 (PC) je programski brojač ipokazuje na sledeću instrukciju koja će biti pročitana iz memorije i izvršena od strane CPU.R1 (SP – Stack Pointer) je pokazivač na vrh steka. Stek se koristi za privremeno smeštanje promenljivih koje će naknadno biti korišćene (instrukcije PUSH i POP). Tipični primeri korišćenja steka:

- Prilikom poziva funkcije na steku se čuva adresa povratka (naredbom RET na kraju funkcije se u programski brojač upisuje adresa povratka privremeno smeštena na steku).
- Prilikom prekida prvo se snima na stek trenutna vrednost programskog brojača, a zatim i vrednost statusnog registra. Po povratku iz prekidne rutine prvo se vraća stara vrednost



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

statusnog registra koja je privremeno bila na steku, a zatim i stara vrednost programskog brojača.

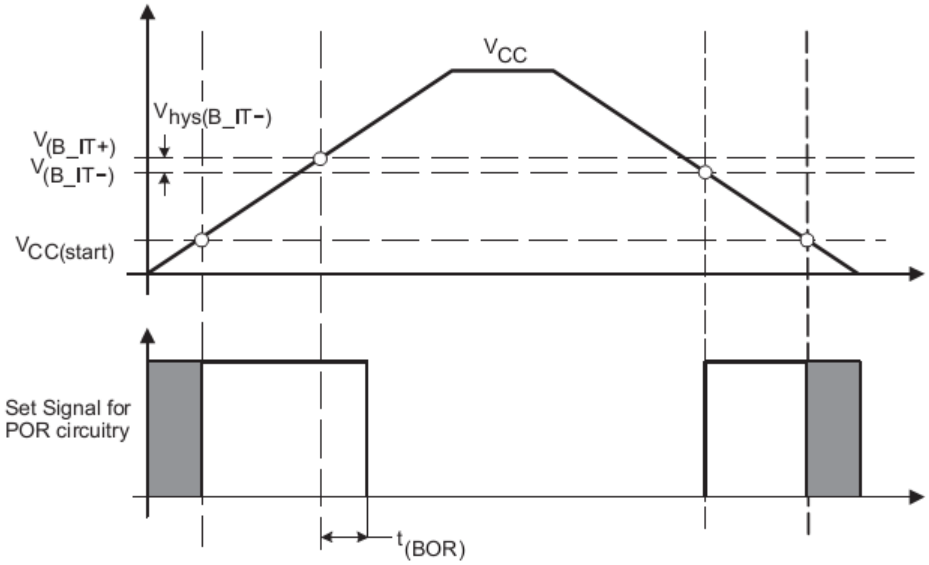
- Smeštanje argumenata i lokalnih promenljivih funkcija.

R2 je statusni registar.

Glavna razlika novije MSP430x u odnosu na prvobitnu MSP430 arhitekturu je to što je adresni prostor povećan sa 64 kB na 1 MB. Adresna magistrala je proširena sa 16 na 20 bita. Zbog toga su delimično promenjeni modovi adresiranja i uvedena su dva nova tipa instrukcija. Jedan od nijh omogućava pristup celom adresnom prostoru, a drugi je uveden za računanje adresa. Magistrala podataka je i dalje 16-bitna. CPU omogućava memorijski pristup na nivou 8, 16 ili 20 bita. Zadržan je broj registara i njihova funkcija kao kod MSP430, ali su prošireni na 20 bita, osim SR. ALU je proširena na 20 bita, a takođe i registri opšte namene. Upis bajta u neki od ovih registara briše bite 19:8, dok upis reči (16 bita) u neki od ovih registara briše bite 19:16.

**1.5 Sistemski reset**

Većina mikrokontrolera MSP430 sadrži *brown-out* reset kolo. Reset kolo zapravo „posmatra“ napon napajanja i na osnovu toga daje izlaz. Funkcija reset kola je da spreči rad mikrokontrolera kada je napon napajanje van dozvoljenih granica. Sve dok je napon napajanja van dozvoljenog opsega rada mikrokontrolera izlazni signal reset kola, BOR (Brown-Out Reset) biće u stanju logičke jedinice




*Prikaz logike reset kola (korišćenje histerezisa)*

BOR signal je aktivan i kada je: RST/NMI pin u stanju logičke nule, kada je konfigurisan kao reset pin, nakon buđenja iz pojedinih modova niske potrošnje (LPMx.5), ili ako je aktiviran softverski BOR



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

događaj (upisom u odgovarajući registar). Poslednja dva načina aktiviranja BOR reseta ne postoje kod svih familija MSP430

Pored BOR reset signala, postoji i POR (Power On Reset) reset signal, koji je uvek aktivan kada je aktivan BOR signal. Pored toga, on je aktivan kada kolo za nadgledanje napona napajanja detektuje neregularan napon napajanja (Supply Voltage Supervisor) ili ako je aktiviran softverski POR događaj (ne postoji kod svih familija).

Pored BOR i POR signala, postoji i PUC (Power-Up Clear) signal. BOR i POR signali mogu da izazovu aktiviranje PUC signala. Osim toga, PUC signal može da aktivira: istek watch-dog tajmera, loš upis u registre watch-dog tajmera, fleš kontrolera ili Power Management Modula. Može se uslovno reći da su BOR i POR hardverski reset signal, dok je PUC softverski reset signal.

Kada je POR signal aktivan u statusni registar se upisuje nula, a u programski brojač se upisuje 16-bitna vrednost iz memorijske lokacije 0FFFEh. Svi registri periferija se postavljaju u početno stanje (power-up stanje). Slično važi i za PUC signal, sa tom razlikom da se samo neki registri periferija postavljaju u početno stanje.

**1.6 Sistem za generisanje taktova**

Arhitektura MSP430 omogućava da CPU i periferije koriste različite taktove. MSP430 je prilagođen aplikacijama sa baterijskim napajanjem. Postoji više izvora signala takta (broj izvora nije isti za sve MSP430). Zajedničko za sve MSP430 kontrolere je da uvek postoje tri izlazna signala takta: MCLK, SMCLK i ACLK. MCLK (Master clock) se koristi za CPU i periferije velike brzine. SMCLK (Sub-system main clock) se koristi kao alternativni izvor za periferije. ACLK (Auxiliary clock) se obično podešen da daje niže frekvencije (npr. 32768 Hz). Razdvajanjem ovih taktova su zapravo omogućeni modovi niže potrošnje, jer je moguće posebno isključiti CPU, a posebno periferije.

**1.7 Prekidi**

Prekidi se mogu definisati kao spoljni događaji koji menjaju programski tok i izazivaju izvršenje potprograma – prekidne rutine (Interrupt Service Routine). Po završetku prekidne rutine izvršenje programa se nastavlja na mestu gde je prekinut. Prekidi obezbeđuju brzu reakciju na neki događaj (npr. primljen je bajt preko serijskog porta, ili istekao je tajmer, završena A/D konverzija). Zadržavanje u prekidnoj rutini treba da bude minimalno. Kašnjenje prekida (interrupt latency time) se definiše kao vreme koje protekne između trenutka kada se desio prekid i ulaska u prekidnu rutinu. Iznosi 6 ciklusa za sve MSP430 mikrokontrolere. Kada se desi prekid, programski brojač i statusni registar se snimaju na stek. Zatim se statusni registar briše, čime se zabranjuju novi maskirajući prekidi (GIE = 0). Na kraju prekidne rutine se vraćaju stare vrednosti statusnog registra i programskog brojača (RETI instrukcija).

Prekidi se mogu podeliti u tri grupe: reset, nemaskirajuće prekide (NMI) i maskirajuće prekide. Maskirajući prekidi su zabranjeni kada je vrednost GIE bita u statusnom registru jednaka nuli. Prekidi imaju različite prioritete. Ako se desi dva ili više prekida u istom trenutku, opslužuje se prekid najvišeg prioriteta. Glavna razlika između NMI i maskirajućih prekida, što se NMI ne mogu zabraniti



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

pomoću GIE bita u statusnom registru. NMI je prekid visokog prioriteta i koristi se, na primer, za preventivno isključenje uređaja. NMI se dozvoljava/zabranjuje pomoću individualnih bita:

- NMIE – aktivira se NMI pri promeni stanja RST/NMI pina,
- ACCVIE – aktivira se NMI pri pogrešnom upisu u fleš,
- OFIE - aktivira se NMI u slučaju lošeg signala takta.

Maskirajući prekidi koji generišu periferije se mogu zabraniti i postavljanjem bita dozvole prekida u odgovarajućem registru periferije.

### 1.8 Modovi niske potrošnje

Postojanje modova niske potrošnje je jedna od glavnih odlika MSP430 mikrokontrolera. Čak i u aktivnom modu, potrošnja MSP430 je mala i iznosi oko 1mW / MIPS. Kod baterijski napajanih uređaja, modovi niske potrošnje se moraju koristiti. Osim aktivnog moda postoji 5 modova niske potrošnje. U aktivnom modu su svi taktovi aktivni (MCLK, SMCLK i ACLK), što znači da su CPU i sve periferije aktivne. U modu niske potrošnje LPM0 isključen je CPU (MCLK), dok su sve periferije i dalje aktivne.U LPM1 se isključuje i FLL (SCG0 = 1). Počevši od LPM2 se isključuje SMCLK, dok se ACLK isključuje tek od LPM4.

# 2. Osnovne periferije MSP430

### 2.1 Digitalni ulaz/izlaz

MSP430 kontroleri imaju više portova. Broj portova zavisi od konkretnog mikrokontrolera. Portovi uglavnom imaju po 8 pinova, ali ima i onih sa manje pinova (npr. PJ ima 4 pina). Svaki od pinova u nekom portu se može koristiti kao pin opšte namene (general purpose input/output) ili pin specijalne namene, pridružen odgovarajućoj periferiji mikrokontrolera. Ako je bit **i** u registru PxSEL postavljen na 0, pin **i** Px porta će se koristiti kao pin opšte namene, dok će se u suprotnom koristiti u okviru periferije, kao pin specijalne namene. Svaki pin opšte namene može biti ulazni ili izlazni, što je određeno PxDIR registrom. Ako je bit i registra PxDIR jednak 0, pin će biti ulazni, dok je u suprotnom izlazni pin. Ako je pin izlazni, vrednost pina se definiše odgovarajućim bitom PxOUT registra (ako je vrednost bita nula, biće generisana logička nula i obrnuto).

Portovi P1 i P2 imaju mogućnost prekida (moraju biti konfigurisani kao ulazni pinovi).Prekid se dešava na ivicu. Aktivna ivica (uzlazna ili silazna) se definiše registrom PxIES. Prekid na ivicu se dozvoljava pomoću registra PxIE. Ulazni pinovi mogu imati interni pull-up ili pull down otpornik (definisano PxREN regitrom – ako se koristi odgovarajući bit mora biti jednak jedinici). Izbor između internog pull-up ili pull down otpornika je omogućen korišćenjem PxOUT registra.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**2.2 Tajmeri – osnovne karakteristike**

Praktično je nemoguće zamisliti korišćenje mikrokontrolera bez upotrebe tajmera. Njihova osnovna uloga je da generišu događaje u pravilnim vremenskim intervalima. Tajmeri se mogu koristiti za generisanje osnovnog intervala za operativne sisteme (OS). OS na osnovu toga može da generiše niz softverskih tajmera. Drugi primer korišćenja je generisanje pravilnih vremenskih intervala potrebnih za A/D konverziju. Tajmeri takođe omogućavaju buđenje procesora iz moda niske potrošnje. Mogu se koristiti u slučajevima kada je potrebno uneti kašnjenje, umesto korišćenja procesorskih resursa (while petlja u softveru). Umesto toga, CPU može da se isključi, a da se iskoristi prekid od tajmera za buđenje procesora. Tajmeri omogućavaju generisanje PWM signala (*compare* mod). *Capture* mod omogućava merenje trajanja signala (snimanje brojačkog registra na ulaznu ili silaznu ivicu signala).

U pokazanim primerima korišćen je Tajmer A iz familije. Tajmer A sadrži 16-bitni brojački registar koji se inkrementira ili dekrementira (zavisno od moda) na uzlaznu ivicu takta. Tajmer A ima do 7 capture/compare registara (CCR). Broj CCR zavisi od konkretnog mikrokontrolera. Mikrokontroler može imati više tajmera A tipa (MSP430F5438A ima Tajmer A0 sa 5 CCR i Tajmer A1 sa 3 CCR). Svaki CCR može biti konfigurisan u compare ili capture modu. Za svaki tajmer je moguće izabrati izvorni takt (SMCLK, ACLK ili spoljni takt). Delitelj ulaznog takta može imati vrednosti 1, 2, 4 ili 8. Svakom capture/compare registru se može pridružiti pin (u slučaju compare moda izlazni pin, a u slučaju capture moda ulazni pin). Tada nije nephodno u softveru da se upravlja odgovarajućim pinom, već sam tajmer automatski menja vrednost pina.

Brojački registar može raditi u jednom od tri režima (promena na ulaznu ivicu takta):

- Brojanje naviše od 0 do vrednosti TAxCCR0 (MC = 1),
- Kontinualno brojanje od 0 do 0FFFFh (MC = 2),
- Brojanje naviše od 0 do vrednosti TAxCCR0, a zatim naniže do 0 (MC = 3).

Nije preporučljivo menjanje režima brojanja u toku rada, već je prethodno potrebno zaustaviti brojač. Kontinualni mod brojanja omogućava nezavisno generisanje intervala za svaki od CCR u compare modu. Postavljanje pina može biti automatski. Svakom CCR odgovara po jedan kontrolni registar (TAxCCTLn, gde je x broj tajmera, a n broj CCR) koji definiše način njegovog rada.

*Compare* mod je odabran kada je CAP bit u kontrolnom registru jednak nuli. Koristi se za generisanje PWM signala. Kada se brojački registar (TAxR) izjednači sa vrednošću registra TAxCCRn postaviće se prekidni fleg CCIFG u kontrolnom registru i javiće se prekid ako je dozvoljen prekid (bit CCIE). Takođe će interni signal EQUn postati jednak jedinici. Ako je CCR pridružen izlazni pin, on će se u trenutku kada EQUn postane jednak jedinici postaviti na odgovarajuću vrednost, u skladu sa vrednošću OUTMOD bita (3 bita – 8 različitih modova). Promenu pina izvodi sam hardver (nije potrebna intervencija u softveru).

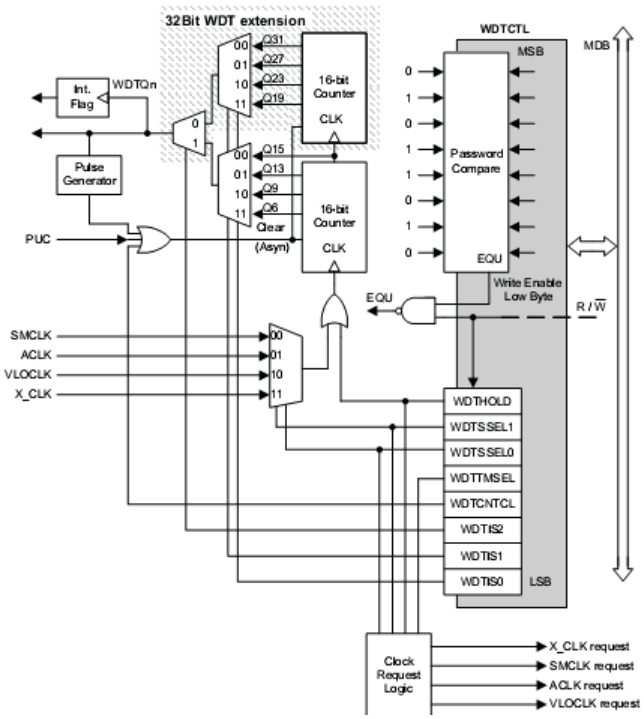


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

*Capture* mod je postavljen kada je vrednost CAP bita jednaka jedinici. Capture mod se koristi kada je potrebno izmeriti trajanje nekog signala. Početak/kraj događaja je uzlazna ili silazna ivica signala. Signal je doveden na ulazni pin koji je pridružen CCR. Postoji mogućnost izbora između dva pina (CCIxA ili CCIxB). CM biti kontrolnog registri definišu aktivnu ivicu (ulazna, silazna ili obe). Kada se desi aktivna ivica na dodeljenom ulaznom pinu, snimiće se trenutna vrednost brojačkog registra u TAxCCRn.

### 2.3 Watch-dog tajmer

Osnovna funkcija WDT tajmera je da spreči „zaglavljivanje“ softvera, npr. usled probijanja steka. WDT tajmer ima 32-bitni brojački registar koji broji naviše (ako je dozvoljeno brojanje). Kada brojač dobroji do programabilne vrednosti (definisana WTDIS bitima) aktiviraće se PUC ili WDTIFG ako WDT radi kao običan tajmer. Tipična upotreba watch-dog tajmera se svodi na to da se u glavnoj petlji resetuje brojač. Drugim rečima, ako se ne prolazi stalno kroz glavnu petlju doći će do isteka WDT i reseta mikrokontrolera (PUC reset).




*Blok šema watch-dog tajmera u familiji MSP430F5xx*

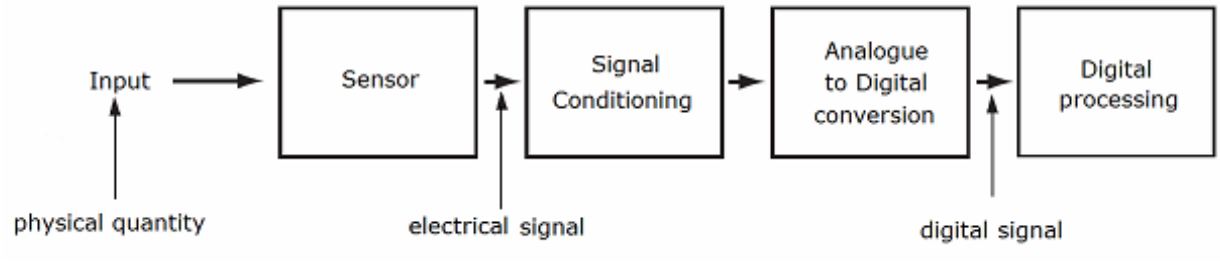
### 2.4 A/D konverzija

Jedna od značajnih periferija mikrokontrolera je analogno-digitalni konvertor. Prisutan je kod velikog broja mikrokontrolera. A/D konvertor omogućava da se analogni signal pretvori u digitalni signal. Na digitalni signal je moguće primeniti razne tehnike digitalne obrade signala (digitalno filtriranje, brzu Furijeovu transformaciju, itd.). S obzirom da A/D konvertori imaju više kanala, oni imaju u sebi

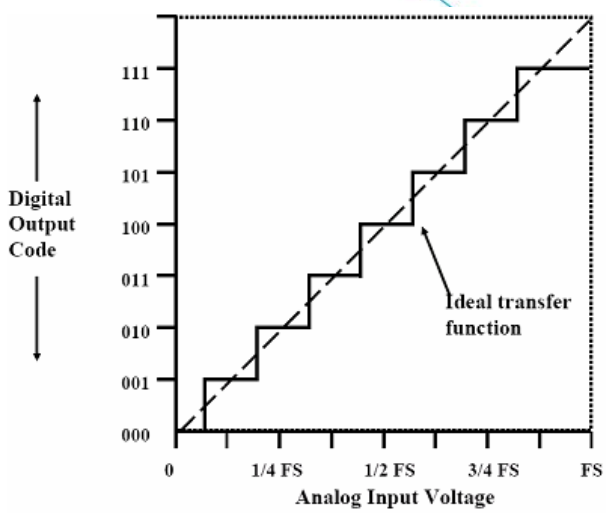


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

analogni multiplekser i prati-pamti kolo (sample and hold circuit). Prilikom promene aktivnog kanala dolazi do prelaznog procesa, tako da signal na izlazu iz prati-pamti kola konvergira ka odabranom ulaznom signalu. Da bi konverzija dala pravi rezultat potrebno je da vreme odabiranja signala bude dovoljno dugo. Teorema o odabiranju kaže da frekvencija odabiranja (*sampling frequency*) analognog signala treba da bude barem dvostruko veća od maksimalne učestanosti u spektru analognog signala. Svi realni analogni signali imaju ograničen spektar, pa je moguće na njih primeniti A/D konverziju (npr. govorni signal u telefoniji se ograničava na oko 4 kHz, dok audio signal ima značajne komponente do oko 20 kHz). Analogni signal koji se meri može biti temperatura, vlažnost, osvetljaj, pritisak, brzina, položaj, itd. Uglavnom su analogni signali neelektrične veličine. Njih je prvo potrebno pretvori u električnu veličinu pomoću odgovarajućih senzora da bi mogla da se primeni A/D konverzija. Obično su analogni signali dobijeni od senzora nedovoljne amplitude, pa je neophodno pojačati signal i dovesti ga u radni opseg A/D konvertora. Takođe je neophodno filtrirati signal, ako je potrebno primeniti digitalnu obradu signala.




*Tipičan lanac obrade analognog signala*



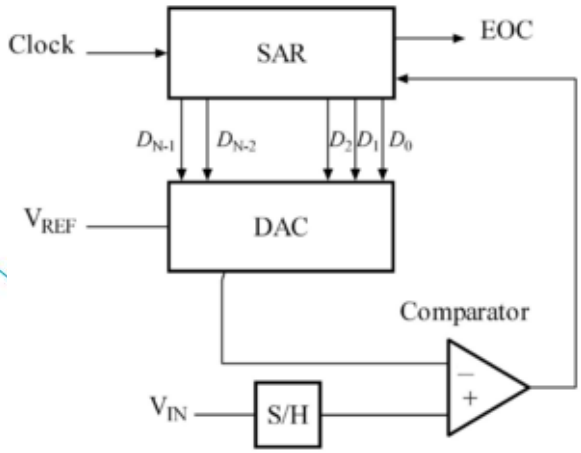

*Karakteristika idealnog A/D konvertora*

Tipovi A/D konvertora koji se najčešće koriste u mikrokontrolerima i signal procesorima su A/D konvertor sa sukcesivnim aproksimacijama (SAR) i sigma-delta A/D konvertor.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

U daljem tekstu ćemo detaljnije opisati A/D konvertor sa sukcesivnim aproksimacijama koji se koristi u familiji MSP430F5xx (12-bitni konvertor). SAR ADC radi iterativno. On generiše interni analogni signal koji poredi pomoću komparatora sa ulaznim analognim signalom. U zavisnosti od izlaza komparatora, interni analogni signal se povećava ili smanjuje. U svakoj iteraciji je promena internog signala manja (finija), tako da ovaj signal postepeno konvergira ka vrednosti ulaznog signala (zapravo SAR radi na principu binarnog pretraživanja). Obično se DAC realizuje pomoću kondenzatorske mreže kod koje svaki naredni kondenzator ima dvostruko veću vrednost od prethodnog.




*Blok dijagram A/D konvertora sa sukcesivnim aproksimacijama (SAR)*

A/D konvertor iz familije MSP430F5xx ima maksimalnu frekvenciju odabiranja do 200 ksps i ugrađeno prati-pamti kolo sa programabilnim trajanjem odabiranja. Start konverzije se može zadati softverski ili po isteku tajmera. Postoji maksimalno 12 spoljnih ulaznih kanala. Moguće je merenje signala sa internog temperaturnog senzora. A/D konvertor ima internu naponsku referencu: 1,5 V i 2,5 V, a kod nekih verzija i 2 V.Ulazni signal treba da se nalazi u opsegu između $V_{R-}$ i $V_{R+}$. Na slici 28-1 u dokumentu SLAU208M (str.724) je prikazan blok dijagram 12-bitnog A/D konvertora za kontrolere koji imaju REF modul.

ADC pretvara ulazni analogni signal u 12-bitnu digitalnu veličinu i snima rezultat u memoriji namenjenoj za odbirke signala (maksimalno 16 odbiraka).

Ako je ulazni napon veći ili jednak $V_{R+}$ dobijena vrednost digitalnog signala biće 4095 (= $2^{12}$ - 1). Ako je ulazni napon manji ili jednak $V_{R-}$ dobijena vrednost digitalnog signala biće jednaka nuli. Kada je ulazni napon V veći od $V_{R-}$, a manji od $V_{R+}$ dobijena vrednost digitalnog signala biće jednaka:

$$4095 * (V - V_{R-})/ (V_{R+} - V_{R-})$$

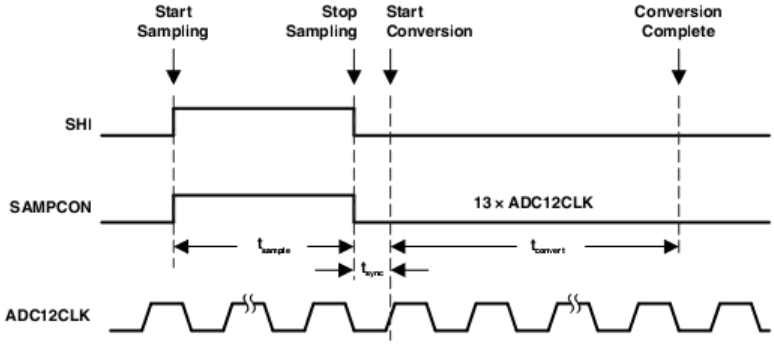
Ulazni signal bi trebalo uvek trebalo da se kreće u ovom opsegu. Izvor takta je konfigurabilan. To može biti SMCLK, ACLK ili MCLK. Programabilni delitelj frekvencije ide od 1 do 32. Kao naponska referenca $V_{R+}$ se može koristiti AVcc, izlaz REF modula ili spoljna naponska referenca povezana na odgovarajući pin mikrokontrolera. Naponska referenca mora biti precizna. Ako se koristi AVcc mora



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

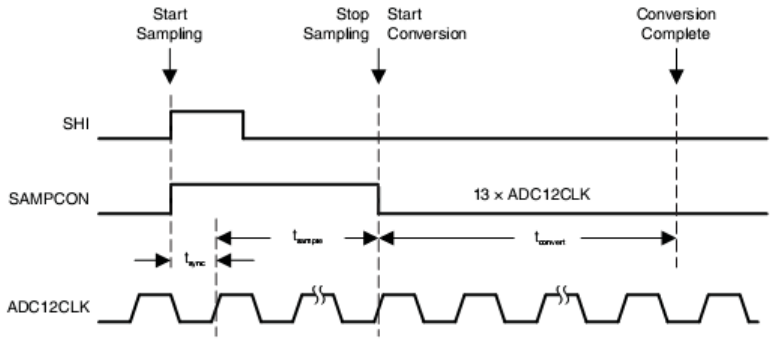
se voditi računa da ovaj napon praktično nema nikakvo odstupanje od nominalne vrednosti. Kao naponska referenca $V_{R-}$ obično se koristi Avss.

A/D konverzija se započinje usponskom ivicom signala SHI. Ova ivica se može generisati softverski ili kao izlaz nekog od tajmera. Rezolucija konverzije može biti 8, 10 ili 12 bita (ADC12RES biti u kontrolnom registru definišu rezoluciju). Postoje dva moda odabiranja: produženi mod i impulsni mod odabiranja. Kod produženog moda odabiranja signal SHI direktno definiše trajanje odabiranja – dok god je aktivan signal traje odabiranje.




*Produžen mod odabiranja (odabiranje se završava nakon silazne ivice signala SHI)*

Kod impulsnog moda odabiranja trajanje odabiranja je definisano bitima ADC12SHT0/1 i ne zavisi od trajanja SHI impulsa.

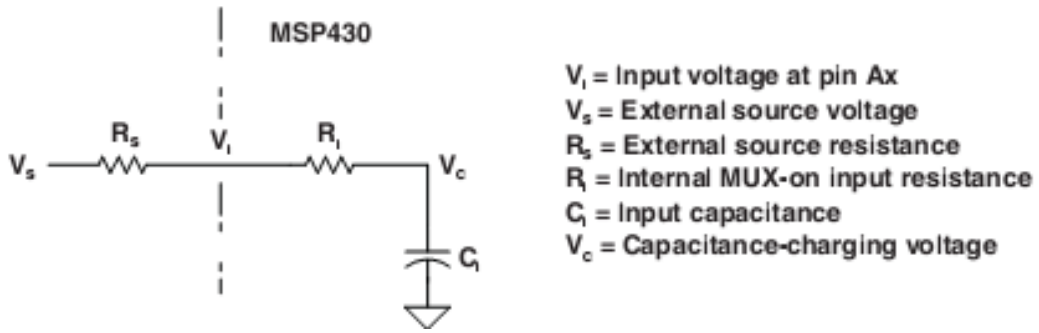



*Impulsni mod odabiranja (trajanje odabiranja ne zavisi od trajanja impulsa SHI signala)*

U oba slučaja je trajanje konverzije iznosi 13 ciklusa takta ADC.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Ekvivalentno analogno ulazno kolo A/D konvertora*

Minimalno trajanje odabiranja iznosi zavisi od spoljne redne otpornosti na ulazu u pin A/D konvertora:

$$t_{sample} > (R_S + 1.8\ k\Omega) \times \ln(2^{n+1}) \times 25\ pF + 800\ ns$$

Za Rs = 200Ω i n=12, vreme odabiranja je 1,25 us.

ADC ima 16 registara ADC12MEMx za smeštanje rezultata konverzije. Rezultat konverzije se automatski smešta u odgovarajući registar. Svakom registru ADC12MEMx je pridružen kontrolni registar ADC12MCTLx pomoću koga se definiše: broj analognog kanala za koji se vrši A/D konverzija (INCHx), naponske reference $V_{R+}$ i $V_{R-}$ (SREFx), kao i to da li se nastavlja konverzija za sledeći „logički“ kanal (ADC12EOS). Ako je ADC12EOS = 0, A/D konverzija se obavlja za sledeći logički kanal. Ako se dođe do 15.kanala i za njega važi ADC12EOS = 0, nastavlja se konverzija za logički kanal 0. Vrednost bita ADC12CSTARTADDx u registru ADC12CTL1 definiše broj logičkog kanala od koga počinje konverzija. Ako se vrši konverzija za sekvencu kanala (jednom ili periodično) broj logičkog kanala će se inkrementirati posle svake konverzije sve dok ADC12EOS ne bude 1.

Postoji 4 moda konverzije (određeni bitima ADC12CONSEQx):

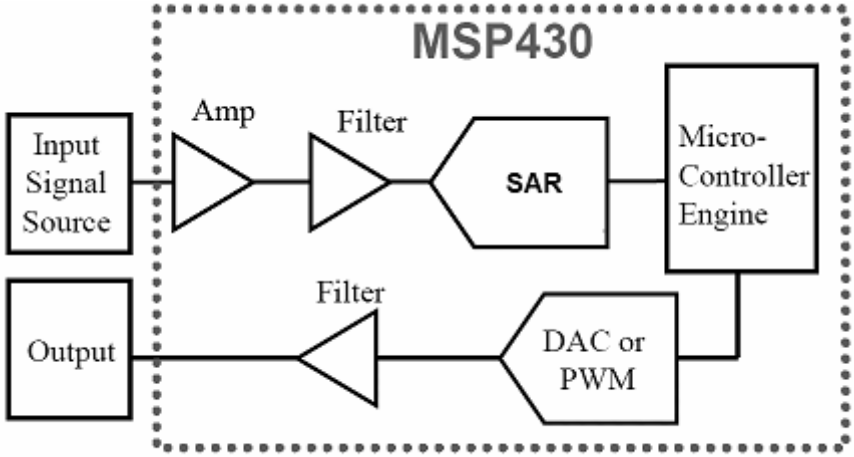
- Pojedinačna konverzija jednog kanala (00) – Referenca [1], slika 28-7,
- Pojedinačna konverzija sekvence kanala (01) – Referenca [1], slika 28-8,
- Periodična konverzija jednog kanala (10) – Referenca [1], slika 28-9 i
- Periodična konverzija sekvence kanala (11) – Referenca [1], slika 28-10.

**2.5 D/A konverzija**

Inverzna operacija A/D konverziji. Koristi se najčešće kada je potrebno izvršiti obradu signala u digitalnom domenu. Tada se prvo vrši A/D konverzija, zatim digitalna obrada signala, da bi se na kraju signal vratio u analogni domen D/A konverzijom.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Tipična primena D/A konverzije u mikrokontrolerima*

**2.6 Serijski portovi**

Mikrokontroleri omogućavaju komunikaciju sa spoljnim kolima i uređajima korišćenjem ugrađenih serijskih ili paralelnih portova. Spoljni uređaj može bit npr. serijska fleš memorija u kojoj se nalazi konfiguracija kompletnog uređaja ili časovnik realnog vremena. Pomoću komunikacionih interfejsa je moguća i komunikacija sa standardnim PC računarom. Razlika između paralelnog i serijskog interfejsa u tome što paralelni interfejs sadrži više linija podataka, dok kod serijskog interfejsa postoji samo jedna linija za podatke (paralelni interfejs prenosi više bita informacije istovremeno). MSP430 mikrokontroleri imaju periferije koje omogućavaju serijsku i paralelnu komunikaciju. U daljem tekstu razmatraćemo samo serijske interfejse, jer se mnogo češće koriste kod mikrokontrolera.

Serijski interfejsi se mogu podeliti na sinhrone i asinhrone. Sinhroni serijski interfejsi imaju dodatnu liniju za takt. Komunikacija je po principu master-slave, a master je onaj koji generiše takt. Asinhroni interfejsi nemaju dodatnu liniju takta, pa je neophodno da postoji start i stop bit (npr. UART). Serijski interfejs može imati razdvojene linije za različite smerove komunikacije (npr. SPI, UART) ili zajedničku liniju (I2C). Kod MSP430 mikrokontrolera nalazimo UART, SPI, I2C i USB interfejse. USB postoji samo kod novijih familija (MSP430x5xx i MSP430x6xx).

Različiti MSP430 mikrokontroleri imaju različite tipove serijskih portova (USART, USCI, USI).



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

| USART | USCI | USI |
|---|---|---|
| **UART:**<br>- Only one modulator<br><br>- n/a<br>- n/a<br>- n/a | **UART:**<br>- Two modulators support n/16 timings<br>- Auto baud rate detection<br>- IrDA encoder & decoder<br>- Simultaneous USCI_A and USCI_B (2 channels) | |
| **SPI:**<br>- Only one SPI available<br><br>- Master and Slave Modes<br>- 3 and 4 Wire Modes | **SPI:**<br>- Two SPI (one on each USCI_A and USCI_B)<br>- Master and Slave Modes<br>- 3 and 4 Wire Modes | **SPI:**<br>- Only one SPI available<br><br>- Master and Slave Modes |
| **$I^2C$:** *(on '15x/'16x only)*<br><br><br>- Master and Slave Modes<br>- up to 400kbps | **$I^2C$:**<br>- Simplified interrupt usage<br>- Master and Slave Modes<br>- up to 400kbps | **$I^2C$:**<br>- SW state machine needed<br>- Master and Slave Modes |

**USART** (Universal Synchronous/Asynchronous Receiver/Transmitter) modul se koristi u familijama MSP430x1xx, MSP430x2xx i MSP430x4xx. Može se konfigurisati da radi u UART ili SPI modu (posebno kod MSP430xF15x/F16x može biti konfigurisan i kao I2C port, u ostalim slučajevima je I2C zaseban port).

**USCI** (Universal Serial Communication Interface) port se nalazi kod familija MSP430x2xx, MSP430x4xx, MSP430x5xx i MSP430x6xx.Sastoji se iz dva bloka: USCI_A bloka koji podržava UART ili SPI mod komunikacije i USCI_B bloka koji podržava I2C ili SPI mod komunikacije. Ako je USCI konfigurisan da radi u UART modu, on može da podrži LIN ili IrDA protokol. I2C i SPI mogu biti setovani u master ili slave modu

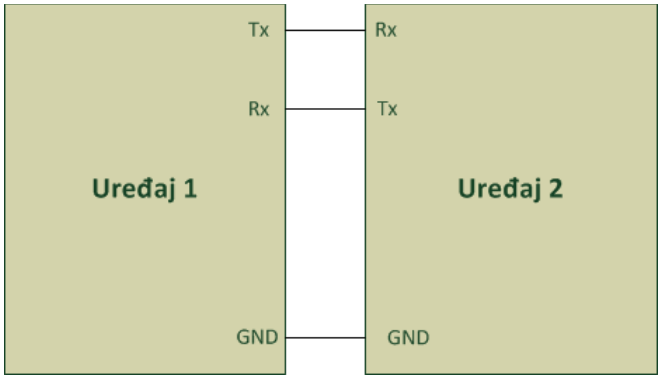
**USI** serijski port postoji samo kod MSP430x20xx familije.Podržava SPI ili I2C mod komunikacije.

U daljem tekstu ćemo se ograničiti na USCI modul, jer se on koristi kod novijih familija MSP430.

Prilikom inicijalizacije USCI porta potrebno je držati modul u resetu (bit UCSWRST u registru UCxCTL1). Zatim se registri USCI porta postavljaju na željene vrednosti (uz očuvanje vrednosti bita UCSWRST). UCSWRST se postavlja na nulu, čime se port oslobađa iz reseta. Na kraju se dozvoljavaju prekidi (po potrebi).

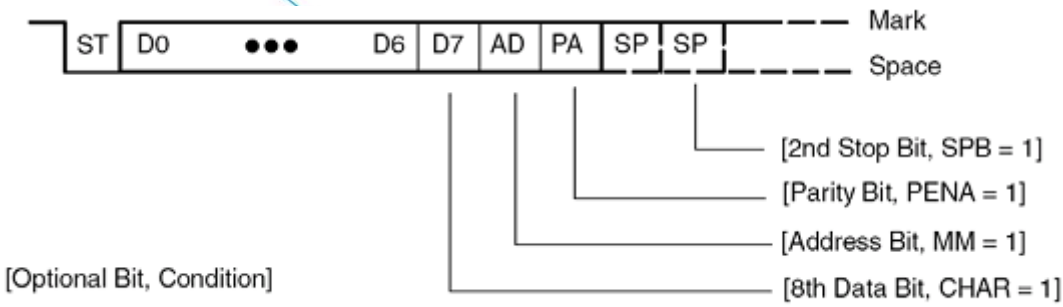
### 2.6.1 UART interfejs

UART je skraćenica od *Universal Asynchronous Receiver/Transmitter*. Spada u asinhrone interfejse. Koristi dve linije za podatke – za svaki smer po jednu liniju. Potrebna je i zajednička naponska referenca. Uglavnom se koristi u slučaju komunikacije tačka-tačka.




*Povezivanje dva uređaja koji komuniciraju koristeći UART interfejs*

Mirno stanje UART linije je logička jedinica. Prvi bit (start bit) služi da označi početak reči koja se prenosi. Dužina reči može biti 7 ili 8 bita. Opciono se koristi bit parnosti koji služi za proveru ispravnosti prenete reči. Postoji jedan ili dva stop bita koji označavaju kraj prenete reči.




*Format poruke kod UART interfejsa*

Kada je USCI_Ax port konfigurisan kao UART, on izlazi na dva pina kontrolera: UCAxTXD (predajni pin) i UCAxRXD (prijemni pin). UART mod je aktivan kada je obrisan bit UCSYNC. USCI_Ax u UART modu nezavisno šalje i prima bajtove. Bitska brzina je fiksna i ista za oba smera komunikacije (postoji mogućnost automatske detekcije bitske brzine, ali se ređe koristi). Moguće je definisati redosled slanja bita (prvo LSB ili MSB – pomoću flega UCMSB u registru UCAxCTL0). Pored osnovnog moda, gde u komunikaciji postoje samo dva učesnika, postoje dva moda koja dozvoljava više od dva uređaja:

- Višeprocesorski mod sa korišćenjem prazne linije, kod koga se na početku svake poruke (grupe bajtova) prvo šalje bajt adrese,
- Višeprocesorski mod sa dodatnim bitom za označavanje adrese/podatka.

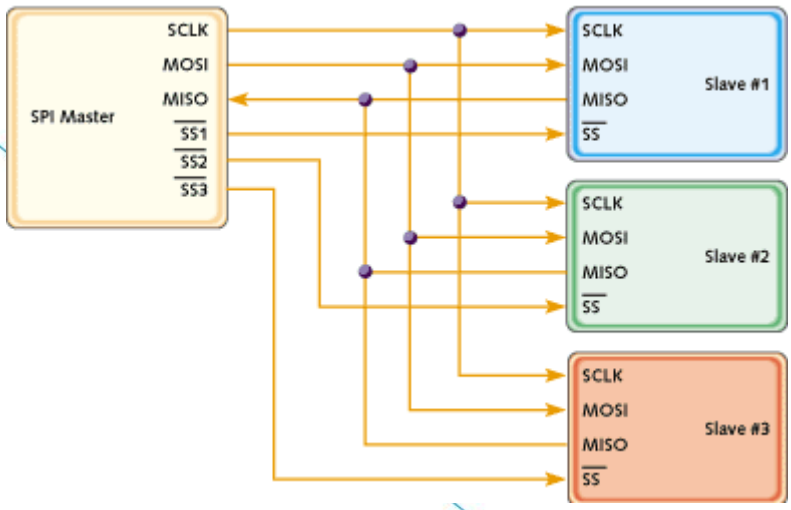
Protokol iznad UART-a treba da vodi računa da ne dođe do kolizije na magistrali.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 2.6.2 SPI interfejs

SPI je sinhroni serijski interfejs.Komunikacija je po principu master-slave, a master generiše signal takta. Postoje dve linije za podatke – za svaki smer po jedna. SPI magistrala se sastoji od najmanje 3 linije – dve linije za podatke (SIMO, SOMI) i linija takta (CLK), uz zajedničku naponsku referencu. Najčešće se koristi i chip select (CS) linija – za svaki slave uređaj potrebna je posebna CS linija. Kod SPI komunikacije se razmena podataka odvija istovremeno u oba smera (master i slave šalju istovremeno po jedan karakter). Master u komunikaciji inicira razmenu karaktera. Signali na magistrali su push-pull.




*Primer povezivanja uređaja na SPI magistrali*

Dužina karaktera može biti 7 ili 8 bita (obično se koristi 8 bita). Može se birati redosled slanja bitova u okviru jednog karaktera (prvo bit najmanje težine ili prvo bit najveće težine).Postoje modovi sakorišćenjem 3 ili 4 pina. Koriste se nezavisni pomerački registri i baferi za slanje i prijem. Polaritet takt signala i faza signala su programabilni. Bitska brzina u master modu je takođe programabilna. Nezavisno se zabranjuju/dozvoljavaju prekidi za prijem i predaju. U slave modu je moguć rad i u LPM4.Kada je USCI port konfigurisan u SPI modu on izlazi na 3 ili 4 pina mikrokontrolera. Tri pina se uvek koriste:

- UCxSIMO (slave in, master out) – kada je SPI u slave modu to je ulazni pin, dok je izlazni u slučaju master moda,
- UCxSOMI (slave in, master out) – kada je SPI u slave modu to je izlazni pin, dok je ulazni u slučaju master moda,
- UCxCLK – SPI takt – izlazni pin mastera, a ulazni za slave uređaj.

Četvrti signal je opcioni:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

– UCxSTE (slave transmit enable) – ovo je uvek ulazni pin. Korišćenjem ovog pina se daje mogućnost da postoji više mastera na SPI magistrali. Aktivna vrednost signala UCxSTE je programabilna.

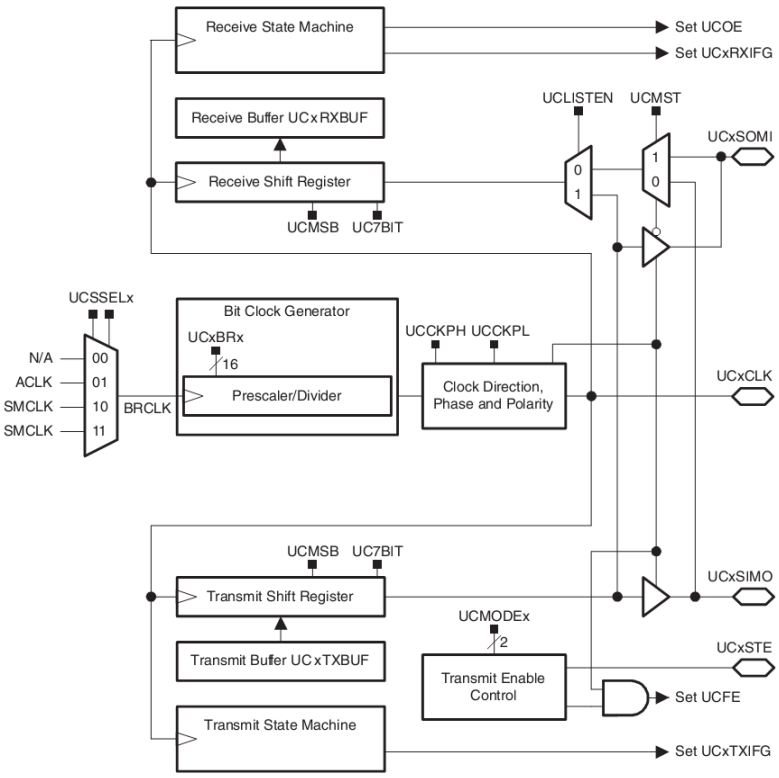



**Figure 35-1. USCI Block Diagram – SPI Mode**

*Blok dijagram USCI serijskog porta u SPI modu*

Inicijalizacija USCI modula u SPI modu se obavlja na sličan način kao i u UART modu.

Na narednoj slici je prikazan MSP430 USCI blok konfigurisan u SPI master modu.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

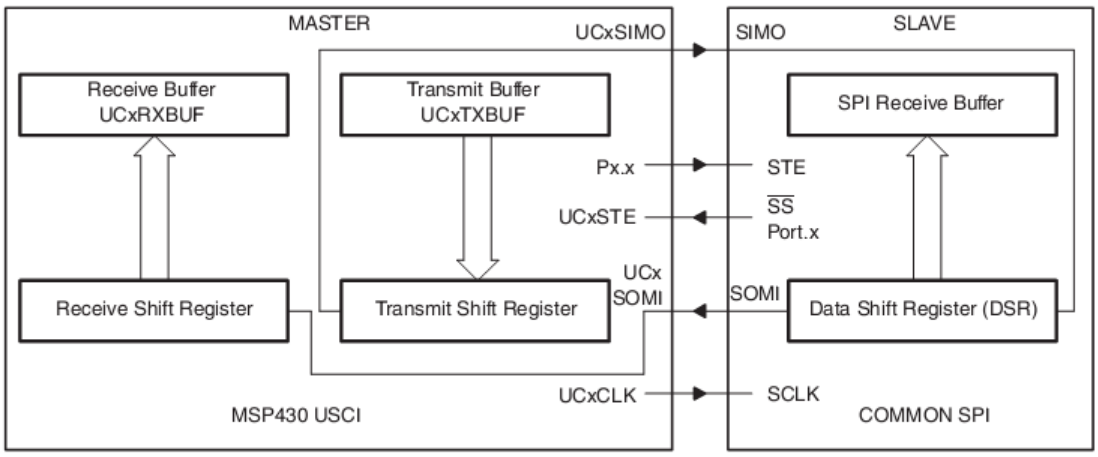



**Figure 35-2. USCI Master and External Slave**

*Blok šema USCI modula u SPI master modu*

Master inicira razmenu karaktera upisom u predajni bafer (UCxTXBUF). UCxTXBUF mora biti prazan pre upisa. Predajni bafer je prazan ako je prekidni fleg UCTXIFG na jedinici. Ako je dozvoljen prekid za predaju on će se javiti čim se oslobodi predajni bafer. Nakon upisa bajta u predajni bafer, USCI port prepisuje njegov sadržaj u pomerački registar iz koga se zatim na aktivnu ivicu takta emituje po jedan bit informacije.Uporedo sa slanjem jednog bita, na suprotnu ivicu takta se ubacuje u prijemni pomerački registar po jedan bit informacije koju šalje slave uređaj. Kada se primi svih 7 ili 8 bita informacije aktiviraće se prijemni prekidni fleg UCRXIFG. Ako je dozvoljen prekid za prijem javiće se prekid. Treba voditi računa o tome da aktiviranje predajnog prekidnog flega ne znači da je ceo karakter poslat drugom uređaju, već da je oslobođen predajni bafer, tj. da je njegova vrednost prepisana u pomerački registar za predaju. Takođe, pin Px.x nije deo USCI porta, već je „običan“ pin opšte namene. Za svaki slave uređaj je potreban po jedan ovakav pin. STE pin se koristi samo ako postoji više mastera na SPI magistrali, što je redak slučaj u praksi. Ako se koristi ovaj mod, u slučaju da vrednost pina STE bude neaktivna, svi izlazni pinovi se postavljaju kao ulazni, obustavlja se slanje ako je bilo u toku, mašina stanja se drži u resetu, postavlja se frame error fleg. Ako se upiše bajt u predajni bafer dok je STE u neaktivnom stanju, on će se poslati čim STE postane aktivan. Sa druge strane, ako STE postane neaktivan u toku samog slanja karaktera, moraće ponovo da se upiše u predajni bafer.

Na sledećoj slici je prikazana blok šema USCI bloka u SPI slave modu.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

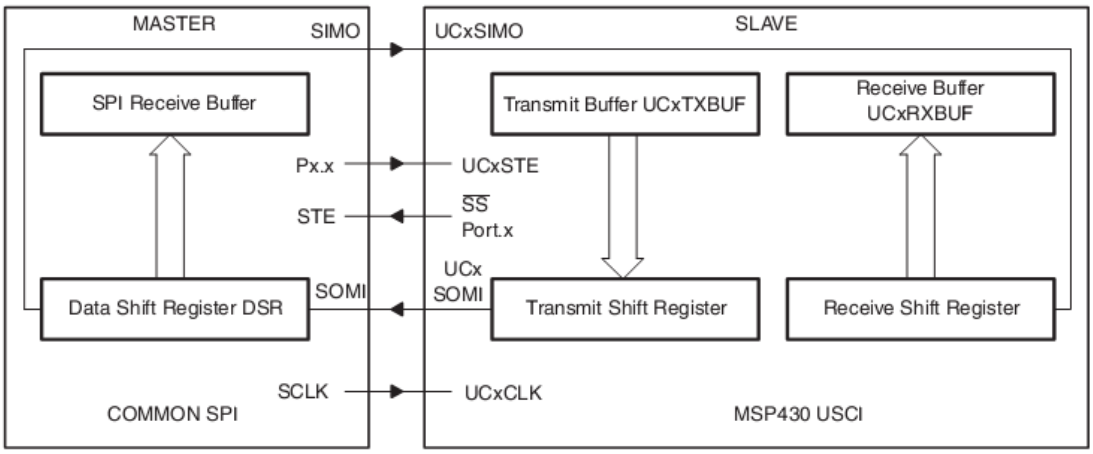



**Figure 35-3. USCI Slave and External Master**

*Blok šema USCI modula u SPI master modu*

Može biti konfigurisan u 3-pinskom ili 4 –pinskom modu. U oba slučaja je UCxCLK pin ulazni. Master mora da generiše takt, a interni generator takta slave-a se ne koristi. Nakon upisa karaktera u predajni bafer, njegov sadržaj se kopira u pomerački registar, ali se bitovi emituju na liniju tek kada master aktivira signal takta. I ovde se na suprotnu ivicu takta ubacuju biti u prijemni pomerački registar. Nakon prikupljanja svih bitova, postavlja se fleg za prijem UCRXIFG.

Master generiše serijski takt pomoću ulaznog takta BRCLK (izbor između SMCLK ili ACLK) i delitelja. Delilac se definiše pomoću registra UCBRx (najmanja vrednost ovog registra je 1). Frekvencija serijskog takta je jednaka $f_{BRCLK}$/ UCBRx. Modulacija se ne koristi u SPI modu, pa registar UCAxMCTL treba da bude obrisan. Faza i polaritet signala su programabilni.

### 2.6.3 I2C interfejs

I2C interfejs spada u serijske sinhrone interfejse. Komunikacija je po principu master-slave. Postoje dve signalne linije na magistrali: linija takta i linija za podatke. Na zajedničku liniju za podatke mogu da emituju master ili slave. Obe linije su open-drain (nula se generiše pomoću drajvera, a jedinica pomoću pull-up otpornika na magistrali). Standardni mod predviđa serijske brzine do 100 kHz. Brzi mod I2C komunikacije omogućava bitske brzine do 400 kHz. Najčešće je na jednu I2C magistralu povezan jedan master i više slave uređaja. Kada je I2C magistrala neaktivna obe linije su u stanju logičke jedinice.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

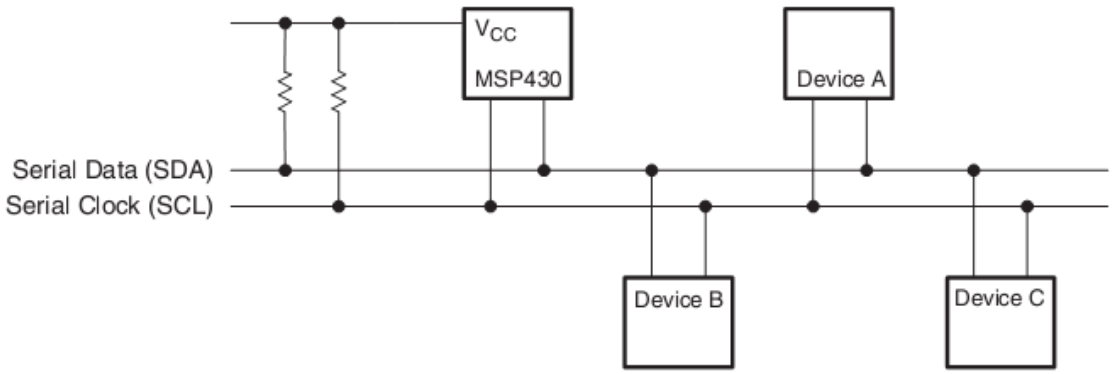
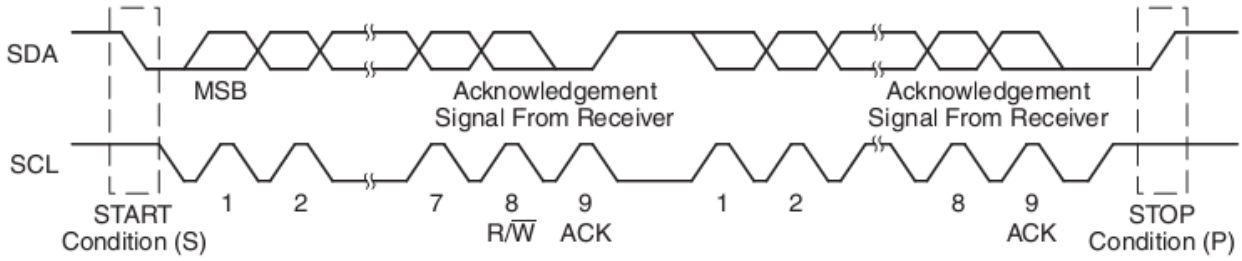



**Figure 36-2. I²C Bus Connection Diagram**

*Način povezivanja uređaja na I2C magistrali*




*Ciklus komunikacije na I2C magistrali*

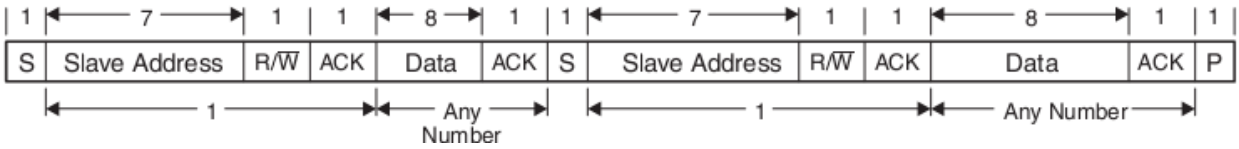
Komunikacija započinje uslovom za start, a završava se uslovom za stop. Master generiše uslov za start tako što generiše silaznu ivicu na SDA liniji dok je SCL u stanju logičke jedinice, dok uslov za stop postavlja generisanjem silazne ivice na SDA liniji dok je SCL u stanju logičke jedinice. U toku razmene informacija SDA linija ne sme da menja stanje dok je SCL linija u stanju logičke jedinice, jer bi se to tumačilo kao start ili stop bit.

Prvi bajt koji šalje master sadrži dve informacije: 7-bitnu adresu slave uređaja kome se obraća i 1 bit koji definiše da li je u pitanju ciklus upisa ili čitanja (1 – ciklus čitanja, 0 - ciklus upisa). Nakon emitovanja ovog bajta master isključuje drajver na SDA liniji i očekuje da slave uređaj koji je prepoznao svoju adresu emituje nulu (ACK bit). Ako se ispravno primi ACK bit, razmenjuju se informacioni bajtovi. Uređaj koji prima bajtove treba da generiše ACK bit posle svakog primljenog bajta.U slučaju 10-bitnog adresnog moda, adresa se šalje u okviru 2 bajta.

Moguće je da se pre stop uslova pošalje ponovo start uslov – ponovljeni uslov starta. Tada se ponovo emituje adresa uz bit R/W. Ovo se obično radi kada treba da se očita stanje lokalnog registra nekog slave uređaja. Tada se započinje sa ciklusom upisa u okviru kojeg se šalje adresa lokalnog registra. Zatim se šalje ponovljeni start bit i započinje se sa čitanjem lokalnog registra.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Ponovljeni uslov za start*

USCI_B port može da radi u I2C modu, bilo kao master ili slave. U svakom od ovih modova može da radi kao predajnik ili prijemnik informacija. Moguć je i rad kada postoji više master uređaja na I2C magistrali (redak slučaj u praksi).

Kada se USCI_B konfiguriše u I2C master modu i to u režimu predaje, pre generisanja start uslova, neophodno je definisati adresni mod (7-bitni ili 8-bitni), postaviti adresu slave uređaja (UCBxI2CSA registar) i postaviti serijski port u mod predaje (bit UCTR se postavlja na 1). Pre započinjanja ciklusa potrebno je proveriti da li je I2C magistrala u mirnom stanju. Zatim se generiše start bit – serijski port emituje adresu i R/W bit i javiće se prekid za predaju (UCTXIFG = 1), ako je dozvoljen. Tada se upisuje prvi bajt informacije u predajni bafer UCBxTXBUF. Kada slave uređaj prepozna svoju adresu treba da emituje ACK bit. Nakon toga se ponovo javlja prekid za predaju (ako je dozvoljen) i tada je moguće upisati novi podatak u predajni bafer. Da bi se završio ciklus upisa potrebno je postaviti bit UCTXSTP na 1 u toku slanja poslednjeg bajta. Treba voditi računa kod slanja samo jednog bajta informacije, jer ako se bit UCTXSTP postavi pre nego što započne slanja bajta, dešava se da se uopšte ne pošalje bajt. Moguće je i generisati ponovni start uslov postavljanjem bita UCTXSTT i definisanjem UCTR bita u toku slanja poslednjeg bajta.

Kada se USCI_B konfiguriše u I2C master modu I to u režimu prijema, pre generisanja start uslova, neophodno je definisati adresni mod (7-bitni ili 8-bitni), postaviti adresu slave uređaja (UCBxI2CSA registar) i postaviti serijski port u mod prijema (bit UCTR se postavlja na 0). Pre započinjanja ciklusa potrebno je proveriti da li je I2C magistrala u mirnom stanju. Kada slave uređaj prepozna svoju adresu treba da emituje ACK bit. Nakon toga, slave uređaj emituje bajt informacije, koji master potvrđuje emitovanjem ACK bita. Po prijemu svakog bajta javiće se prekid za prijem bajt (UCRXIFG = 1), ako je dozvoljen. Ciklus se nastavlja sve dok se ne postavi UCTXSTP bit u toku prijema poslednjeg bajta. Tada serijski port automatski nakon prijema poslednjeg bajta šalje NACK i generiše uslov za stop. Slično kao i kod ciklusa predaje, moguće je generisati i ponovni uslov za start.

Kada serijski port USCI radi u slave modu neophodno je pri inicijalizaciji postaviti sopstvenu adresu (registar UCBxI2COA). Serijski port koji radi u I2C slave modu kada prepozna svoju adresu i kada je bit R/W = 1 počinje da radi u modu predaje (UCTR = 1). UCTR bit postavlja automatski sama periferija. Prekidni flegovi za detekciju start uslova UCSTTIFG i predaju UCTXIFG se postavljaju odmah po prepoznavanju adrese. Ako je dozvoljen prekid za predaju postaće aktivan. U prekidnoj rutini treba upisati podatak u predajni bafer. Potvrda bajta adrese pomoću ACK bita se dešava kada se upiše podatak u predajni bafer. Sve dok se upiše blokira se magistrala držanjem SCL linije na nuli. Čim se prepiše predajni bafer u pomerački registar ponovo će postati aktivan UCTXIFG. Po prijemu svakog



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

bajta master treba da generiše ACK bit, osim kod poslednjeg bajta, kod koga master šalje NACK bit, a zatim uslov za kraj ciklusa. Slave detektuje uslov za kraj i postavlja UCSTPIFG na 1, a briše flegove UCSTTIFG i UCTXIFG. Kada je USCI periferija konfigurisana da radi u slave modu, ona ulazi u mod prijema kada detektuje sopstvenu adresu na I2C magistrali i bit R/W na 0. Tada se automatski šalje ACK bit i postavlja prekidni fleg UCSTTIFG. Nakon prijema svakog bajta od mastera, USCI port generiše ACK bit i postavlja prekidni fleg za prijem UCRXIFG na 1. Ako je dozvoljen prekid za prijem, javiće se prekid. Tada je potrebno očitati primljeni bajt iz prijemnog bafera. Sve dok se to ne uradi biće blokirana magistrala. Na kraju ciklusa, kada master generiše uslov za kraj ciklusa, USCI port će postaviti prekidni fleg UCSTPIFG.

# 3. RTOS

Osnovni gradivni blok RTOS su taskovi. Pisanje taskova je u većini jednostavno. Zapravo, svakom tasku odgovara jedna funkcija koja se izvršava kada su tasku dodeljeni procesorski resursi. Startovanje taska se obavlja na početku programa. Prilikom kreiranja taska njemu se dodeljuje prioritet i stek. Kod većine RTOS svaki task (proces) ima svoj stek.Svaki task se može nalaziti u jednom od tri stanja:

- **Aktivno stanje (running)** – mikroprocesor izvršava instrukcije koje pripadaju ovom tasku. Kod jednoprocesorskog sistema samo jedan task može biti u ovom stanju.
- **Spreman (ready)** – neki drugi task je aktivan, dok je posmatrani task čeka da mu nudu dodeljeni procesorski resursi. Proizvoljan broj taskova može biti u ovom stanju.
- **Blokiran (blocked)** – task čeka da se desi određeni događaj da bi prešao u stanje spreman (npr. čekanje da se pritisne neki taster, ili da se primi informacija preko serijskog porta). Proizvoljan broj taskova može biti u ovom stanju.

Osnovna funkcija raspoređivača (schedulera) RTOS je da prati stanje svakog od taskova i da odredi koji od taskova treba da postane aktivan. Samo task koji nije blokiran može da postane aktivan. Scheduler posmatra sve neblokirane taskove i proglašava aktivnim onaj task koji ima najviši prioritet. Sve dok je task blokiran, on ne može postati aktivan, već čeka da se desi određeni događaj da bi prešao u stanje „spreman“. Posao schedulera se svodi na to da menja stanje taskova iz stanja „spreman“ u stanje „aktivan“ ili obrnuto. Taskovi mogu da blokiraju sami sebe, prekidi ili drugi taskovi mogu da ih prevedu u stanje „spreman“, dok samo scheduler može da ih proglasi aktivnim.

RTOS poseduje niz funkcija koje omogućavaju uvid scheduleru da li je proizvoljni task blokiran i koji događaj ga može odblokirati. Zadatak schedulera je da signalizira tasku da se događaj koji čeka desio. U tom slučaju će se scheduler „vrteti“ u petlji (idle task) i čekaće da se desi događaj koji će odblokirati neki od taskova. Ovo je, u principu, greška programera. Programer mora voditi računa da se ovo ne desi.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Kod nekih RTOS je zabranjeno da dva taska imaju isti proritet, dok se kod drugih RTOS taskovi istog prioriteta naizmenično aktiviraju. Postoje i RTOS kod kojih tek kad se blokira jedan od taskova aktiviraće se task istog prioriteta.

Kod *preemptive* RTOS kada se odblokira task najvišeg prioriteta on postaje aktivan, dok kod *non-preemptive* RTOS trenutno aktivan task mora prethodno da se blokira. Nadalje ćemo posmatrati samo preemptive RTOS.

Čest je slučaj kada dva taska treba da pristupaju istoj promenljivoj (objektu, strukturi). Pošto taskovi mogu imati različite prioritete, task većeg prioriteta može prekinuti task nižeg prioriteta.

Semafori sprečavaju koliziju kada dva taska konkurentno pristupaju zajedničkim podacima. Zajedničkoj promenljivoj se pridružuje semafor koji govori da li je ona dostupna ili ne. Task mora pre pristupanja promenljivoj (bilo da je reč o upisu ili čitanju) da proveri pomoću semafora da li je dostupna, pa ako nije da sačeka dok se ne oslobodi. Nakon što se završi sa obradom promenljive ona se oslobađa (semafor). Funkcija za zauzimanje semafora obično u nazivu ima *take* ili *pend*, dok funkcija za oslobađanje semafora ima u nazivu *release* ili *post*.

Redovi se obično realizuju kao cirkularni FIFO baferi. Jedan task upisuje u red, dok drugi čita iz reda.Kreiranje svih redova treba da se završi pre nego što se startuje RTOS. Funkcija za kreiranje reda obično vraća njegovu adresu u memoriji (pokazivač). Prilikom upisa u red vrši se kopiranje podataka, tako da inicijalni podaci koji su kopirani u red mogu da menjaju vrednost. Sa druge strane, ako želimo da uštedimo memoriju i procesorsko vreme, umesto podataka se može slati pokazivač na podatke, ali u tom slučaju treba voditi računa da se podaci na koje ukazuje pokazivač ne smeju menjati sve dok ih drugi task ne pročita.

Dok su redovi pogodni za slanje ili prijem sirovih bajtova, mejlboksovi su zgodni za strukturirane podatke (poruke). Obično mejlboksovi mogu da čuvaju više poruka, mada u nekim RTOS mogu da čuvaju samo jednu poruku. Memorija se dodeljuje prilikom kreiranja mejlboksa. U nekim RTOS mogu postojati različiti prioriteti poruka.

Navešćemo samo neke od RTOS koji se mogu portovati na MSP430: µC/OS-II™, Contiki, embOS, FreeRTOS, TinyOS, B.lu BOS i SYSBIOS.

Stranica posvećena RTOS za MSP430: http://processors.wiki.ti.com/index.php/MSP430_Real_Time_Operating_Systems_Overview

FreeRTOS je besplatan *open-source* RTOS. Portovan je na 34 platforme. Može se koristiti i na mikrokontrolerima sa skromnim resursima, zahvaljujući tome što je realizovan minimalan skup funkcija koje omogućavaju osnovnu funkcionalnost RTOS. Broj taskova nije ograničen, samo ukupna memorija predstavlja ograničenje. Realizovan je pravi *multi-tasking*. Različiti prioriteti taskova, taskovi istog prioriteta dele procesorsko vreme ako nisu blokirani. Implementirani se redovi, binarni i brojački semafori i *mutexi*.

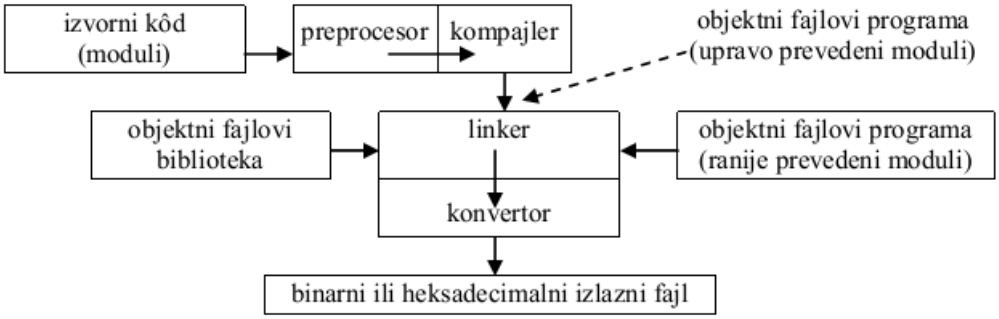
# 4. Razvojna okruženja

Integrisana razvojna okruženja (Integrated Development Environment, skraćeno IDE) obuhvataju sledeće komponente : editor teksta, kompajler, linker i dibager (debugger).

Kompajler je računarski program koji prevodi programski kôd napisan u jednom programskom jeziku u drugi programski jezik, obično u mašinski jezik (dobijeni kôd se naziva objektni kôd). Mašinski kôd se sastoji od niza instrukcija koje CPU može direktno da izvršava. Programski kod koji se prevodi zove se izvorni kod. Moguće je pisati program koristeći asembler (assembly language). Tada se koriste asemblerske instrukcije koje su direktno povezane sa instrukcijama mikrokontrolera. Asemblerski kod se pomoću specijalnog kompajlera prevodi u mašinski kod koji CPU može da izvrši.

Svaka procesorska arhitektura ima svoj asembler. Prednost pisanja koda u asembleru je što se rezultujući kod obično brže izvršava (ako je dobro napisan!). Mana pisanja koda u asembleru je to što se taj kod ne može iskoristiti na nekoj drugoj procesorskoj arhitekturi. Uglavnom se softver za mikrokontrolere razvija u višim programskim jezicima, a samo neki kritični delovi koda (intenzivna obrada signala) se moraju pisati u asembleru. Za većinu mikrokontrolera postoje kompajleri za C programski jezik koji imaju više nivoa optimizacije, pomoću kojih je moguće ostvariti kod koji se brže izvršava u odnosu na neoptimizovani kod. Uz dobar C kompajler i pažljivo pisanje programa, vrlo je teško napisati efikasniji program u asembleru (osim kod signal procesora). U svim primerima koristićemo izvorni kod napisan u C programskom jeziku.

Linker je poseban program čiji je zadatak da poveže niz objektnih fajlova u jednu celinu – izvršni kôd, tj. kôd koji će se izvršavati na mikrokontroleru. Osim objektnih fajlova nastalih prevođenjem izvornog kôda, u konačni izlazni fajl linker uključuje i inicijalizacioni kôd, kao i standardne biblioteke koje su definisane u okviru projekta.




*Kompletan proces prevođenja*

Dibager je računarski program koji se koristi za otklanjanje grešaka u napisanom programu mikrokontroler/mikroprocesor i proveru ispravnosti programa. Dibager omogućava napredne funkcije kao što su pokretanje programa korak po korak (*single-stepping*), zaustavljanje ili pauziranje izvršavanja programa na takozvanom *breakpointu*, dohvatanje stanja promenljivih i registara



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

mikrokontrolera. Povezivanje sa mikrokontrolerom, ubacivanje izvršnog koda i debagovanje koda je moguće ostvariti preko emulatora.

MSP430 podržava JTAG interfejs prema emulatorima. JTAG interfejs je sličan SPI interfejsu (master generiše signal takta, a postoje odvojene linije za predaju i prijem podataka). Novije verzije MSP430 podržavaju i Spy-Bi-Wire interfejs.

*Izgled JTAG emulatora MSP430-FET430UIF Texas Instruments-a*

Najčešće korišćena razvojna okruženja za MSP430 su:

- Code Composer Studio
- IAR Workbench za MSP430
- Open Source GCC

U daljem tekstu ćemo detaljnije opisati osnovna uputstva za korišćenje IAR Workbench za MSP430.

**4.1 IAR Workbench za MSP430**

Instalacioni fajl ovog programa dostupna je na sledećoj web stranici: http://www.iar.com/Products/IAR-Embedded-Workbench/TI-MSP430/

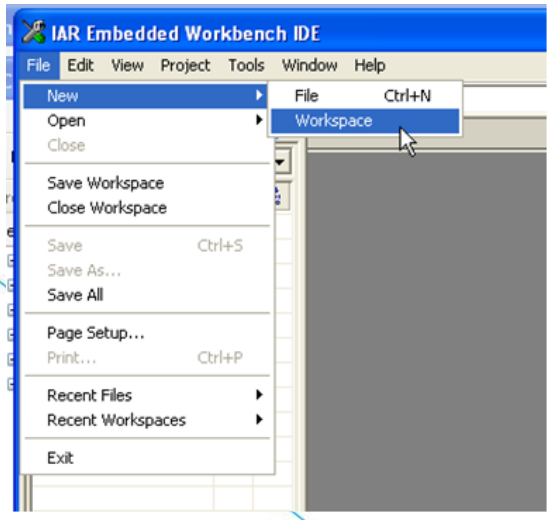
Kliknuti na *download* i skinuti instalacioni fajl. Postupak instalacije je jednostavan. Instalacija obuhvata i drajver za JTAG emulator. Nakon uspešne instalacije i pokretanja programa, treba uneti licencni kod. Da bi se dobio licencni kod potrebna je online registracija, pri kojoj se unose lični podaci. Odabrati opciju sa ograničenjem veličine koda (size-limited evaluation). Nakon registracije dobićete



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

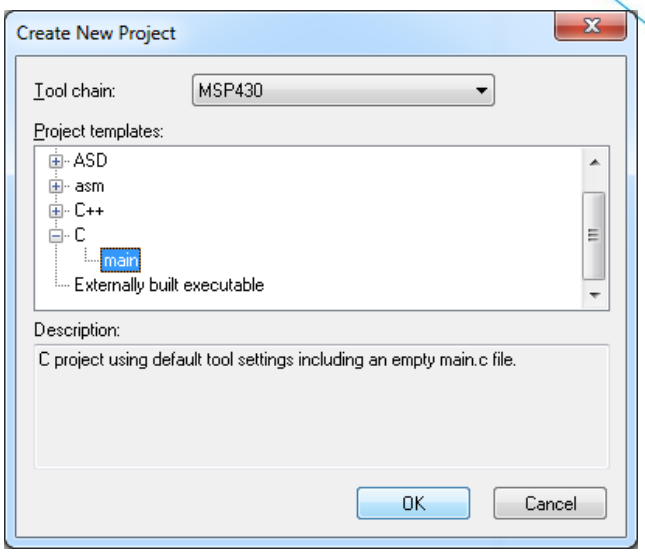
automatski povratni mejl u kome je sadržan licencni (aktivacioni) kod. Sledi unošenje licencnog koda, nakon čega možete da koristite ovo razvojno okruženje.

Više projekata mogu da budu grupisani u jedan workspace. Ipak, u slučaju IAR Workbencha bi trebalo **izbegavati** ovu mogućnost, jer je lako moguće pomešati fajlove iz dva različita projekta u istom *workspace*-u. Dakle, trebalo bi da se svaki projekat nalazi u posebnom *workspace*-u. Prvo kreirati folder gde će smestiti projekat i workspace. Startujte IAR Workbench. Kliknite **FILE -> New -> Workspace**.




*Otvaranje novog workspace-a*

Zatim kliknite na **Project -> Create new project**. U polju Tool Chain treba da bude“MSP430”. Izaberite ‘C’ opciju i u okviru nje selektujte “main”. Ovim će se kreirati nov projekat sa praznim main.c fajlom (zapravo postoji jedna linija koda kojom se stopira watch-dog tajmer).

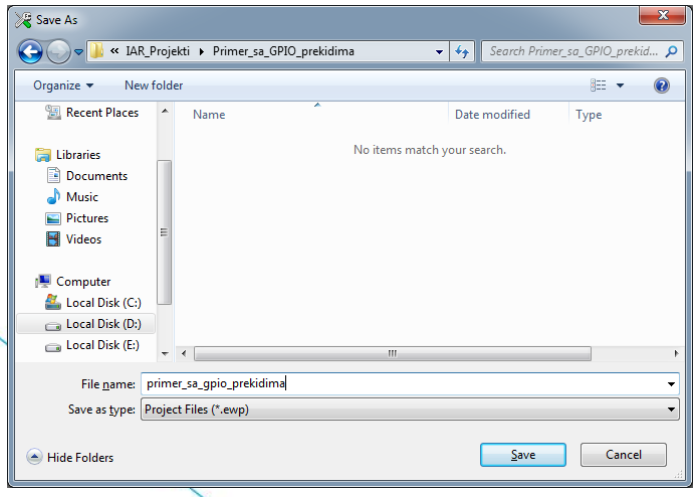





Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

*Otvaranje novog projekta*

Nakon ovoga, otvoriće se dijalog boks u kome treba da unesete ime projekta i da izaberete putanju do foldera koji je prethodno formiran. Stavite intuitivno ime za naziv projekta. Ekstenzija ewp biće automatski dodata.



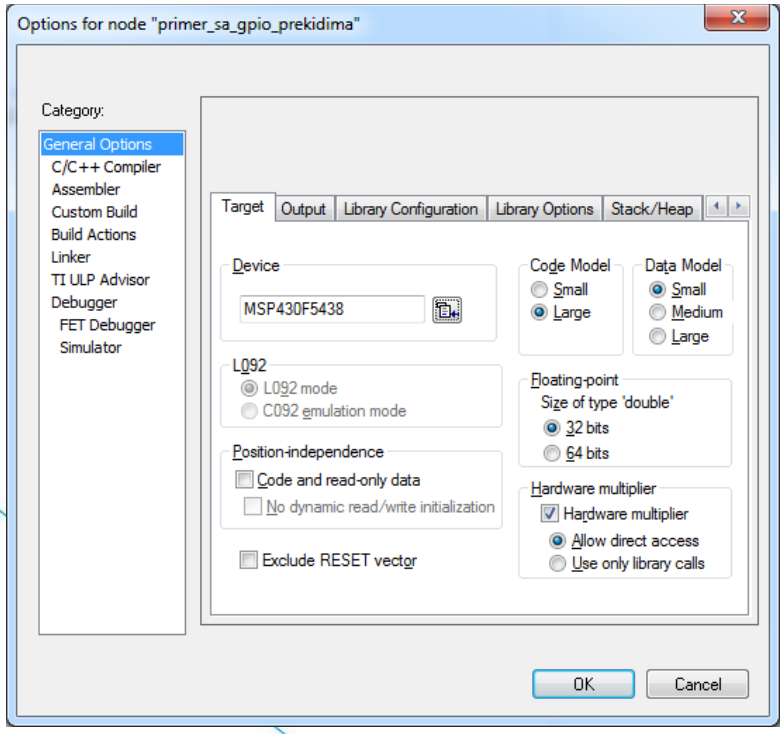

*Snimanje novog projekta*

Zatim kliknite na **File -> Save Workspace**. Stavite identično ime workspace-a kao i za projekat. Program će sam dodati ekstenziju eww.

Klikom na **Project -> Options** otvara se dijalog boks u kome je moguće postaviti željene opcije projekta. Trebalo bi na samom početku uneti pravi naziv mikrokontrolera koji će se koristiti (u našem slučaju to je MSP430F5438). Da bi se ovo uradilo potrebno je izabrati tab **General Options** i **Target**.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Izbor odgovarajućeg MSP430 mikrokontrolera*

Takođe je potrebno podesiti veličinu steka i heap-a. Potrebno je selektovati **General Options** i tab **Stack/Heap**. Podrazumevana veličina steka je 160 bajtova, što je dovoljno za jednostavnije programe (dugačke nizove ili glomazne strukture ne bi trebalo držati na steku).

Da bi se odabrao željeni debugger treba odabrati tab **Debugger**. Moguć je izbor između **Simulatora** i **FET Debugger**-a. Ako želimo da spustimo programski kod na mikrokontroler treba odabrati **FET Debugger**. Izbor konkretnog FET Debugger-a se može izvršiti aktiviranjem taba **FET Debugger**. U našem slučaju treba izabrati **Texas Instruments USB-IF**.

Da bi se dodali .c ili .h fajlovi treba ih prvo kreirati kao prazne fajlove sa ekstenzijom .c ili .h u projektnom direktorijumu, a zatim se izabere opcija **Project -> Add Files**. Otvoriće se dijalog boks u kome treba odabrati željeni fajl ili fajlove.

Pojedinačno kompajliranje fajlova se ostvaruje desnim klikom miša na željeni fajl, a zatim se izabere opcija **Compile**.

Linkovanje se vrši klikom na **Project->Make** (inkrementalno linkovanje – za one izvorne fajlove koji nisu menjani koristiće se postojeća verzija objektnog koda). Izborom opcije **Project->Rebulid All** će se svi izvorni fajlovi prvo kompajlirati, bez obzira da li su menjani, a linkovanje u izvršni fajl će se obaviti nakon toga.

Spuštanje koda na mikrokontroler se vrši selektovanjem **Program->Download and Debug**.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 5. Glavni principi razvoja softvera za mikrokontrolere

Jedan od glavnih principa razvoja softvera je sakrivanje informacija ili enkapsulacija (*information hiding, encapsulation*). Čitljivost koda se može poboljšati odgovarajućim korišćenjem belih znakova. Obavezno treba koristiti komentare u softveru koji povećavaju razumljivost koda. Softver mora da bude jasno napisan, kako bi se olakšalo njegovo korišćenje i eventualne modifikacije. Imena promenljivih treba da budu takva da su intuitivna i da ih jasno opisuju. Kad god je moguće, treba koristiti ANSI C kod. Portabilnost koda je jako bitna – softver treba da bude napisan tako da može da se portuje na drugu platformu.

Programi su podeljeni u izvorne fajlove(.c) i zaglavlja (.h) na logičan način. Zaglavlja sadrže interfejse funkcija koji se koriste u drugim softverskim modulima (fajlovima). Softverski modul koji koristi neku funkciju drugog modula treba samo da zna **šta ta funkcija radi, ali ne i kako**. Fajl treba da bude organizovan tako da sadrži deo za definisanje promenljivih i deo za definisanje (implementaciju) funkcija. Unutar jednog fajla grupisati funkcije koje čine jednu logičku celinu. Korišćenje struktura je moćno sredstvo u programiranju – pristup pomoću pokazivača. Grupisati srodne skalarne promenljive u strukturu. Izbegavati promenljive vidljive u više datoteka (extern promenljive). Umesto toga realizovati funkcije za pristup tim promenljivama (set i get funkcije).

Korišćenje belih znakova (razmak, tabovi, prazne linije) povećava čitljivost koda. Obavezno koristiti razmake u izrazima. Pokazalo se u praksi da je optimalna veličina taba treba da bude širine 4 karaktera. Vitičaste zagrade treba da budu u posebnom redu.

Komentari omogućavaju bolju razumljivost koda. Koriste se na različitim nivoima. Na programskom nivou treba generisati README fajl koji objašnjava šta program radi i sadrži opis organizacije programa. Na nivou fajlova treba da postoji prolog (na početku fajla) koji treba da opiše svrhu tog fajla. Na nivou funkcija treba da postoji prolog funkcija (neposredno pre samog koda) koji opisuje šta funkcija radi i njene argumente. Komentari unutar funkcija treba da postoje radi boljeg objašnjenja delova koda. Uz lokalne promenljive treba da postoji komentar koji opisuje njihovu namenu.

Odabrati imena promenljivih tako da precizno označavaju njihovu namenu (ako je potrebno koristit i duža imena). Izbegavati promenljive poput i, j. Umesto toga koristiti counter, buffer_index ili slično. Treba izbegavati da se imena dve promenljive razlikuje samo po jednom karakteru. Poželjno je da se unutar istog fajla ne koriste dve promenljive sa istim imenom – izbegavati sakrivanje promenljivih.

Pre izrade softvera treba definisati konvenciju za nazive promenljivih, da li se koriste isključivo mala slova ili kombinacija malih i velikih slova. Simboličke konstante se označavaju velikim slovima uz korišćenje donje crte, ako je potrebno.

Imena promenljive se mogu pisati malim slovima uz korišćenje donje crte između reči:

```
int max_volume;
```

Alternativno se mogu koristiti početna velika slova:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
int MaxVolume;
```

Za imena funkcija se mogu koristiti početna velika slova:

```
void VolumeUp(void);
```

Treba naglasiti da je izbor forme za naziv promenljivih i funkcija stvar konvencije. Bitno je da se unutar jednog projekta poštuje usvojena konvencija, tj. da svi fajlovi imaju istu strukturu.

Dobra je praksa da se umesto naziva standardnih tipova (int, char, float, itd.) koriste korisnički tipovi podataka koji sugerišu veličinu promenljive u bajtovima. Na MSP430 je promenljiva tipa int 16-bitni podatak, dok je na primer na mikrokontroleru TMS570LS3137 32-bitni podatak. Korišćenjem korisničkih tipova podataka umesto standardnih tipova je omogućeno lako prebacivanje softvera na drugu platformu.

```
typedef unsigned char uint8_t;
typedef signed char sint8_t;
typedef unsigned int uint16_t;
typedef int sint16_t;
typedef unsigned long uint32_t;
typedef long int sint32_t;
```

Ceo program se sastoji od niza datoteka:

- Heder fajlova (standardnih i korisničkih),
- Standardnih biblioteka (stdio, stdlib, math),
- Jednog ili više programskih fajlova,
- Proizvoljnog broja modulskih fajlova.

U slučaju jednostavnijih programa postoji samo jedan programski fajl, dok u slučaju složenijih programa može biti više programskih fajlova. Kada postoji više programskih fajlova treba koristiti princip enkapsulacije. Programski fajl može sadržati realizaciju nekog procesa (taska). Samo u jednom programskom fajlu se nalazi main funkcija.

Uloga README fajla je da objasni kako program radi i kako je organizovan. Može dodatno sadržati uslovne flegove za kompilaciju i njihova značenja. Putanje do onih delova softvera koji se mogu iskoristiti i u drugim programima.
Heder fajlovi sadrže:

- interfejse funkcija,
- deklaracije promenljivih (ako je moguće izbegavati),
- simboličke konstante,



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- definicije korisničkih tipova podataka (typedef),
- nabrojive tipove podataka (enum).

Pomoću heder fajlova se ostvaruje princip enkapsulacije. Heder fajlovi se uključuju u izvorni fajl (.c fajl) pomoću pretprocesorske naredbe #include. Deklaracije posebnih podsistema grupisati u posebne heder fajlove. Ako postoji deo heder fajla zavisan od platforme treba ga izdvojiti u poseban heder fajl.

Fajlovi modula sadrže logički povezane funkcije, podatke, konstante, deklaracije. Slični su programskim fajlovima, osim što ne sadrže main funkciju.

MISRA preporuke se često koriste prilikom razvoja softvera za mikrokontrolere. MISRA je skraćenica od *The Motor Industry Software Reliability Association*. Preporuke se prvenstveno odnose na softver za auto industriju, ali se mogu primeniti i na druge softver za druge uređaje. C programski jezik je pogodan za korišćenje u embedded sistemima, ali on sa druge strane ima i loše strane koje mogu da dovedu do katastrofalnih grešaka. Cilj MISRA preporuka je da spreče da do takvih grešaka dođe. MISRA®-C:2004 sadrži 142 preporuka svrstanih u 21 kategoriju. Za svako pravilo je definisano da li je zahtevano (R -required) ili samo savetodavno (A - advisory). Treba napomenuti da mnoga razvojna okruženja imaju u svom kompajleru implementiranu opciju za uključivanje/isključivanje MISRA preporuka, tako da postoji automatska provera poštovanja MISRA pravila.

# 6. **Praktični primeri softvera**

U svim praktičnim primerima softvera korišćena je eksperimentalna ploča MSP-EXP430F5438 koja sadrži mikrokontroler MSP430F5438, a kao razvojno okruženje je korišćen IAR Workbench sa ograničenjem u pogledu maksimalne veličine koda (8 kB). Jedino je primer za korišćenje FreeRTOS portovan korišćenjem Code Composer Studija. Kao JTAG emulator je u svim primerima korišćen MSP-FET430UIF.

### 6.1 Primer korišćenja GPIO prekida

Svrha ovog primera je da demonstrira korišćenje ulaznih pinova kontrolera koji imaju mogućnost prekida. Taster S1 na eksperimentalnoj ploči je povezan na pin P2.6, dok je taster S2 povezan na pin P2.7 mikrokontrolera. Pošto nije definisan naponski nivo pinova kada tasteri nisu pritisnuti, potrebno je koristiti interne *pull-up* otpornike, tako da pinovi koji se koriste za tastere budu u stanju logičke jedinice kada tasteri nisu pritisnuti. U inicijalizaciji se konfigurišu svi neophodni pinovi i dozvoljavaju se prekidi za pinove P2.6 i P2.7, a zatim se dozvoljavaju maskirajući prekidi. U glavnoj petlji procesor odlazi u LPM4, a budi se samo na pritisak nekog od tastera S1 ili S2. Kada se desi prekid, menja se stanje pridružene diode. Tasteru S1 je pridružena dioda LED1, dok je tasteru S2 pridružena dioda LED2. Nakon prekida, CPU ostaje u aktivnom modu u kome se prvo zabranjuju maskirajući prekidi,



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

zatim se čeka određeno vreme, da bi se potom izbrisali prekidni flegovi za pinove P2.6 i P2.7, ponovo se aktiviraju maskirajući prekidi i na kraju CPU ponovo odlazi u LPM4.

Kompletan projekat sa dodatnim objašnjenjima se nalazi u Dropbox folderu:

**NiCat_ItAcademy\Real Time sistemi 1. deo\IAR_Projekti\Primer_sa_GPIO_prekidima**

### 6.2 Primer korišćenja tajmera

U konkretnom primeru za demonstraciju jedne od mogućih primena tajmera, on se koristi za generisanje perioda potrebnog za skeniranje i debaunsiranje tastature. Tajmer je podešen tako da generiše prekid na 1 ms. Procesor nije aktivan sve dok ga ne probudi prekid od tajmera. Kada se desi prekid od tajmera, vrši se skeniranje i debaunsiranje tastature. Kada se ustanovi pritisak tastera (S1 ili S2) uključuje se odgovarajuća dioda.

Kompletan projekat sa dodatnim objašnjenjima se nalazi u Dropbox folderu:

**NiCat_ItAcademy\Real Time sistemi 1. deo\IAR_Projekti\Primer_sa_tajmerom**

### 6.3 Primer korišćenja A/D konvertora

Za ilustraciju rada i primene A/D konvertora iskorišćen je akcelerometar koji generiše dva analogna signala. Ti signali predstavljaju ubrzanje po x i y osi, tj. u horizontalnoj ravni. Na početku se vrši kalibracija, radi dobijanja srednje vrednosti signala. Kada se završi kalibracija, ulazi se u glavnu petlju gde se neprestano vrši A/D konverzija. Nakon startovanja A/D konverzije, aktivira se LPM3, jer A/D konvertor radi pomoću ACLK takta. Kada se detektuje ubrzanje priblizno 1 g po jednoj od osa uključuje se odgovarajuća LED dioda.

Kompletan projekat sa dodatnim objašnjenjima se nalazi u Dropbox folderu:

**NiCat_ItAcademy\Real Time sistemi 1. deo\IAR_Projekti\Primer_ADC**

### 6.4 Primer korišćenja UART interfejsa

U primeru softvera za korišćenje UART interfejsa iskorišćen je USB-UART most na eksperimentalnoj ploči. Na strani računara se ova veza vidi kao virtuelni COM port. Na računaru je neophodno pokrenuti HyperTerminal ili neki sličan program i podesiti parameter serijske komunikacije (brzina 115200 b/s, ne koristi se bit parnosti, koristi se jedan stop bit). Na početku glavnog programa na MSP430 se vrši inicijalizacija UART-a koji je povezan na računar. Realizovan je eho, tako da svaki bajt koji MSP430 primi pošalje nazad računaru, što se izvršava u prekidnoj rutini za UART port. U prozoru aplikacije na računaru ispisuje se tekst koji se kuca na tastaturi. Istovremeno se po prijemu svakog bajta menja stanje crvene diode. Procesor je u celoj glavnoj petli u neaktivnom stanju (LPM0), jer UART radi pomoću SMCLK takta koji je aktivan u LPM0 modu.

Kompletan projekat sa dodatnim objašnjenjima se nalazi u Dropbox folderu:

**NiCat_ItAcademy\Real Time sistemi 1. deo\IAR_Projekti\Primer_sa_UARTom**

### 6.5 Primer korišćenja SPI interfejsa

U primeru za SPI interfejs demonstrirano je korišćenje SPI portova na MSP430F5438 kontroleru kako u master, tako i u slave modu. U konkretnom primeru se USCI_A3 port postavlja u SPI master, a USCI_A0 u SPI slave mod. Naravno, potrebno je spojiti odgovarajuće pinove kratkospojnicima. Koristi se 3-pinski mod, bez korišćenja STE pinova. Na početku koda se konfigurišu oba porta i pripreme bajtovi za slanje, a zatim se inicira slanje. Iz okruženja je moguće proveriti da li su dobijeni ispravni bajtovi.

Kompletan projekat sa dodatnim objašnjenjima se nalazi u Dropbox folderu:

**NiCat_ItAcademy\Real Time sistemi 1. deo\IAR_Projekti\Primer_SPI**

### 6.6 Primer korišćenja FreeRTOS

U ovom primeru je predstavljeno korišćenje operativnog sistema FreeRTOS na mikrokontroleru MSP430F5438. Korišćen je već gotov primer koji se može skinuti u okviru instalacije FreeRTOS, dostupne na Internetu:

http://sourceforge.net/projects/freertos/

Postoji primer za konkretnu platformu – eksperimentalnu ploču MSP-EXP430F5438 koja je korišćena i u svim prethodnim primerima. Postoje dve verzije, jedna za IAR Workbench, a druga za Code Composer Studio. U okviru primera demonstrirano je kreiranje i korišćenje taskova, komunikacija između taskova, korišćenje funkcija operativnog sistema u prekidnoj rutini, itd.

# 7. Uvod u razvoj softvera na Linuxu

### 7.1 Deset principa open-source softvera

1. Softver je slobodan za distribuciju
2. Pored izvršnih fajlova, distribuira se i izvorni kôd
3. Open-source licenca dozvoljava korisniku da eksperimentiše nad programskim kôdom i da distribuira svoje modifikacije
4. Korisnik zna ko stoji iza softvera koji koristi
5. Nema diskriminacije prema tipu korisnika (određenim osobama ili grupama)
6. Open-source licenca nikog ne ograničava da koristi softver u željenom polju primene
7. Ne treba tražiti dodatnu licencu za korišćenje i dalju distribuciju softvera
8. Open-source licenca nije specifična za proizvod
9. Open-source licenca ne ograničava drugi softver
10. Open-source licenca je tehnološki neutralna

Postoji na desetine open-source licenci, http://opensource.org/licenses. Dva „ekstrema“: GPL (General Public Library) sa najviše ograničenja i obaveza za autora izvedenog rada, BSD (Berkeley Software Distribution) sa najmanje ograničenja i obaveza pri objavljivanju svog derivata. Još oko deset licenci u širokoj upotrebi: Lesser GPL, Apache, MIT, Mozilla i Eclipse Public License…

### 7.2 GNU Projekat

Započet 27. septembra 1983. na univerzitetu MIT u Masačusetsu. Richard Stallman je bio pokretač i glavni aktivista. Cilj: napraviti kompletno softversko okruženje (operativni sistem i alate) koji će biti slobodni za menjanje i distribuciju. Vlasnički operativni sistem Unix poslužio je kao glavna inspiracija za koncepte funkcionisanja GNU sistema. GNU je „rekurzivni akronim“ od „GNU’s Not Unix”.

*Logo GNU projekta – stilizovan portret afričke životinje gnu, slične antilopama*

Komponente GNU projekta:

1. GNU Hurd (jezgro, „kernel“)
2. GNU Compiler Collection (GCC)
3. GNU C Library (glibc)
4. GNU Core Utilities (skup osnovnih sistemskih programa)
5. GNU Binary Utilities (dodatni sistemski alati)
6. Bash shell
7. GNOME Desktop (tek od 1999. godine)

### 7.3 Istorijat GNU projekta

Godine 1989. projekat je imao sve osim jezgra operativnog sistema. Godine 1987. Stallman predložio Mach mikrokernel (razvijen na univerzitetu Carnegie Mellon kao zamena za BSD Unix kernel) kao osnovu operativnog sistema GNU. Razvoj jezgra GNU Hurd (koristeći implementaciju GNU Mach) započet tek 1990. godine jer se čekalo da CMU objavi Mach pod prihvatljivom licencom. Razvoj je tekao sporo i loše.

Godine 1991. Linus Torvalds počeo razvoj jezgra operativnog sistema u istraživačke svrhe na univerzitetu u Helsinkiju, inspirisan knjigom „Operating Systems: Design and Implementation“ i

operativnim sistemom MINIX od profesora Andrewa Tanenbauma. MINIX i GNU Herd su zasnovani na arhitekturi mikrokernela. Linux predstavlja klasičan monolitni kernel.

*Richard Stallman i Linus Torvalds*

Linux kernel + GNU softver = GNU/Linux OS („distribucija“). Postoji ogroman broj dostupnih distribucija, od serverskih operativnih sistema, preko hobi projekata, do distribucija za embedded platforme. Linux jezgro je izrazito skalabilno: iz istog izvornog kôda moguće je kompajlirati kernel za veliki broj primena (od embedded sistema do superračunara).

Linux kernel je napisan u C-u, uz „kritične“ delove u asembleru (zavisno od ciljne arhitekture). Linux kernel je specifično pisan za buildovanje pomoću GNU Toolchaina i GCC kompajlera. Intel je nudio skup patcheva za kernel koji omogućava buildovanje koristeći Intel kompajler. LLVMLinux (llvm.linuxfoundation.org) je projekat sa ciljem eliminisanja zavisnosti od GCC ekstenzija i omogućavanja buildovanja kernela koristeći Clang, frontend za LLVM kompajler koji nudi bolje performanse zahvaljujući naprednijim optimizacijama. Preporuka u ovom stadijumu: držati se GNU Toolchain-a i GCC-a.

Linux podržava veliki broj procesorskih arhitektura (minimum je 32-bitni CPU sa MMU jedinicom):

DEC Alpha, Analog Devices Blackfin, ARM, Atmel AVR32, Texas Instruments C6X, Freescale/Motorola 68k, IBM System/390, IBM System Z, Intel x86, AMD64, Intel Itanium, MIPS, OpenRISC, IBM PowerPC, SPARC, SuperH (Sega Dreamcast), Tilera…

### 7.4 POSIX

POSIX je familija IEEE standarda za kompatibilnost između operativnih sistema nalik Unixu („Single UNIX Specification“). Definiše osnovno funkcionisanje operativnog sistema (kreiranje procesa, međuprocesnu komunikaciju, operacije nad fajlovima...), funkcionisanje komandnog interpretera i skup osnovnih sistemskih komandi. Linux je u najvećoj meri kompatibilan, iako ne postoji zvaničan sertifikat.

**7.5 Razvoj softvera na Linuxu**

„Toolchain" je naziv za skup alata za razvoj softvera: kompajler, linker, biblioteke (kao interfejs ka operativnom sistemu i njegovim uslugama, za statičko ili dinamičko linkovanje), debugger.

Specifičnosti GNU Toolchaina: GCC kao skup kompajlera za razne programske jezike (C, C++, Fortran, Objective-C, Ada...), GNU Make, automatizacija kompajliranja i linkovanja.

Većina kros-kompajlera za ARM Linux zasnovana na GNU Toolchainu zahvaljujući tome što se distribuira pod GPL licencom.

Softver za Linux (uključujući Linux kernel) najugodnije je razvijati na samom Linuxu, ali nije obavezno u velikom broju slučajeva. Kompajler i svi ostali build alati (toolchain) obično dolaze preinstalirani uz Linux distribucije.

Projekat Cygwin predstavlja Unixoliko okruženje sa bibliotekama i komandnim interpreterom koje se izvršava na Windowsu, uz visok stepen integracije.

Cross-development predstavlja razvoj Linux softvera za jednu hardversku platformu na nekoj drugoj platformi. Cross-platform toolchainovi obično ne dolaze preinstalirani uz GNU/Linux distribucije opšte namene.

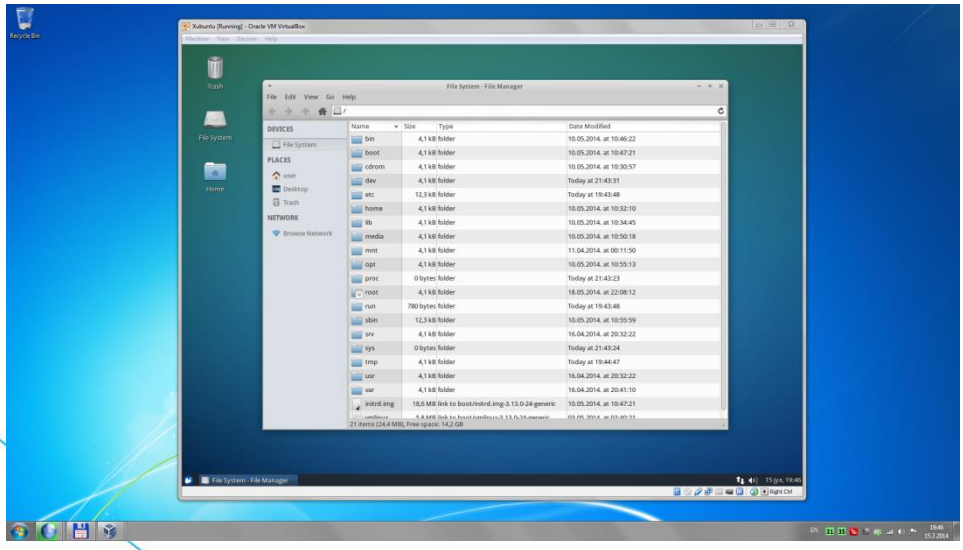
Za razvoj na ovom kursu potrebno je instalirati neku od Linux distribucija. Opcije:

1. Kao jedini operativni sistem na računaru
2. Dual boot sa postojećim OS-om (npr. Windowsom)
3. U okviru postojećeg OS-a unutar virtuelne mašine

Koristićemo treću opciju uz pomoć open-source softvera za virtuelizaciju – Oracle VirtualBox. Razvijaćemo na distribuciji Xubuntu 14.04 LTS (32-bit). Linux kernel 3.13, GCC 4.8.2, grafičko okruženje XFCE 4.10. Cross-kompajler je već instaliran u folderu /home/user/CodeSourcery/ Sourcery_G++_Lite.



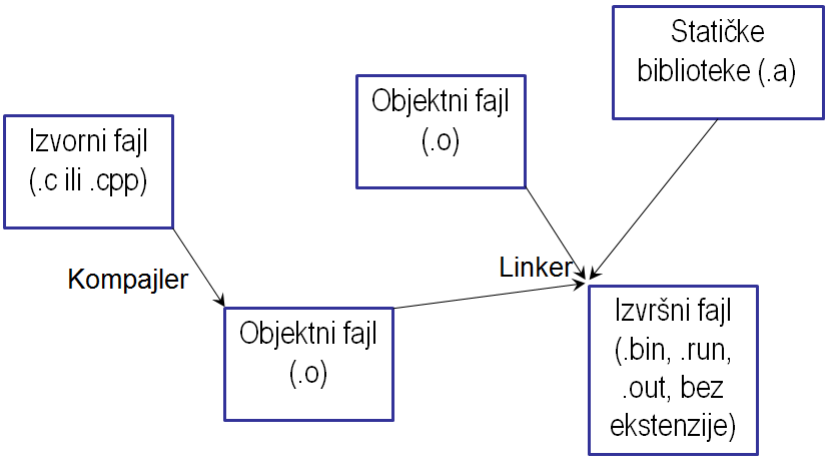
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES



*Ubuntu 14.04 virtuelizovan unutar Windowsa 7*

### 7.6 Prevođenje programa

Sav razvoj u ovom delu kursa moguće je obaviti iz komandne linije i bez razvojnog okruženja. Izvorni kôd Linux kernela je previše velik da bi se uvezao u razvojno okruženje. Verzija 3.0 iz 2011. godine sadrži preko 37 hiljada fajlova i preko 15 miliona linija kôda. Umesto projektnih fajlova, koriste se tzv. Makefiles. Isti princip važi za većinu open-source softvera na Linuxu i sličnim sistemima.




*Prevođenje programskog kôda u izvršni program*

Primeri prevođenja programa direktnim pozivanjem C kompajlera iz komandne linije:

**gcc program.c**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Prevešće „program.c“ u objektni fajl, linkovaće ga sa standardnom C bibliotekom i proizvesti izvršni fajl podrazumevanog imena „a.out“ (jer nismo sami naveli ime).

**gcc program.c -o program**

Slično kao u prethodnom slučaju, osim što smo naveli da hoćemo da se izlazni izvršni fajl (opcija „output“) zove „program“.

**gcc -c program.c**

Prevešće „program.c“ u objektni fajl „program.o“, ali ga neće linkovati ni sa čim. Objektni fajl se može naknadno linkovati u izvršni fajl (sa ili bez drugih objektnih fajlova), ili se od njega može proizvesti statička biblioteka (sa ekstenzijom .a)

**gcc main.c gfx.o utilities.a -o program**

Prevešće izvorni fajl „main.c“ u objektni fajl, linkovati ga sa ranije prevedenim objektnim fajlom „gfx.o“ i dodatnom statičkom bibliotekom „utilities.a“, i od svega toga proizvesti izvršni fajl po imenu „program“. S obzirom da kreiramo izvršni program, jedan od ulaznih fajlova (prevedenih ili ne) mora implementirati funkciju main().

### 7.7 Makefile

„Makefile“ je tekstualni fajl koji sadrži „recept“ za kompajliranje i linkovanje projekta. Na Linux distribucijama se najčešće koristi alat GNU Make koji prolazi kroz Makefile i interpretira uputstva.

Format Makefile-a podržava grananje, simboličke konstante i promenljive, uvoz podataka iz drugih fajlova itd. Veliki projekti (npr. Linux kernel) mogu sadržati mnogo Makefile-ova za svoje komponente, koji se kontrolišu putem glavnog fajla. GNU Make se ne mora koristiti isključivo za razvoj softvera, ali je za to originalno napisan.

### 7.8 Alati za baratanje fajlovima

S obzirom da ne koristimo razvojno okruženje koje bi nam nudilo mogućnost kros-referenciranja identifikatora, od velikog je značaja imati dobre alate za baratanje fajlovima i njihovu pretragu. Kriterijumi za izbor fajl-menadžera:

1. Rad u više panela (više prikazanih foldera istovremeno)
2. Sinhronizacija foldera
3. Tretiranje arhiva (zip, tar...) kao foldera
4. Jednostavno kopiranje kompletnih foldera
5. Efikasna pretraga sadržaja fajlova (case sensitive on/off...)
6. Prikaz rezultata pretrage u panelima

7. Ugrađen pregledač (naročito hex.) i editor

Odlična rešenja:

1. Midnight Commander (tekstualni interfejs, napredni klon nekada popularnog Norton Commandera)
2. Krusader (grafički interfejs, klon Total Commandera za Windows)

**Kompare** je alat za grafički prikaz razlika između tekstualnih fajlova. Oslanja se na alat „diff“ iz komandne linije, odnosno interno ga poziva, preuzima njegov tekstualni izlaz i grafički ga formatira tako da je prikaz pregledniji za korisnika. Ako je već instaliran na Linux distribuciji, integrisaće se u Krusader.

### 7.9 Tekst editori za uređivanje izvornog kôda

Za udobno kodiranje potrebno je imati dobar editor. Kriterijumi:

1. Sve standardne opcije (search & replace, word wrap…)
2. Sintaksno bojenje programskog kôda
3. Obeležavanje parova otvorenih i zatvorenih zagrada
4. Obeležavanje tekuće linije u kojoj se nalazi kursor
5. Obeležavanje desne margine
6. Rad sa više fajlova u istoj instanci programa
7. Promena boje teksta i pozadine (uz predefinisane grafičke teme)
8. (Opciono) Otvaranje i zatvaranje blokova kôda
9. (Opciono) Interni pregledač foldera

Preporuka: **Gedit i Kate**

# 8. Osnove administracije Linux sistema

### 8.1 Osobine Linuxa kao operativnog sistema

Linux je višekorisnički i višeprocesni operativni sistem.

„Višeprocesni“ znači da implementira multitasking, i to u preemptivnom maniru. Svaki proces se pokreće u izolaciji, „misleći“ da se jedini izvršava na računaru i da mu je memorija u celom adresnom opsegu na raspolaganju. Kernel samoinicijativno (koristeći neki od algoritama za raspoređivanje)



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

pauzira jedan proces i aktivira neki drugi koji je bio na čekanju. Procesi „ne znaju“ ništa o tome da su pauzirani.

„Višekorisnički“ znači da se strogo poštuje razdvajanje korisničkih naloga. Korisnici imaju sopstveni folder na disku, fajlovi imaju korisničke dozvole i vlasnika, procesi koje je korisnik pokrenuo ne mogu operisati nad fajlovima i folderima nad kojima korisnik nema dozvole, korisnici mogu definisati lozinke koje će biti tražene pri logovanju, korisnički folderi se mogu šifrovati koristeći te lozinke, više korisnika može biti istovremeno ulogovano na sistem i istovremeno pokretati procese pod odgovarajućim privilegijama, glavni administratorski nalog (*root*) ima apsolutne privilegije.

### 8.2 Komandni interpreter

Linux je nastao po uzoru na Unix koji se koristio na mainframe računarima, pa grafičko okruženje nije bilo prioritet. Grafičko okruženje nije sastavni deo Linuxa i realizuje se kao skup dodatnog korisničkog softvera. Linux nativno podržava tekstualni interfejs (na ekranu, serijskom portu itd).

Korisnički program koji interpretira unete tekstualne komande zovemo komandni interpreter ili „shell“. Postoji više implementacija, a osnovnu funkcionalnost propisuje POSIX standard. Shell je na prvom mestu komandni interpreter, ali takođe predstavlja i programski jezik, jer može da interpretira komandne skripte sa promenljivima, grananjem itd.

Na Unix mainframe računare vezivali su se hardverski terminalni uređaji (praktično monitori s tastaturama) putem serijske veze. Mainframe računar mogao se nalaziti npr. u podrumu zgrade, dok su zaposleni radili na terminalima iz svojih kancelarija.

Program koji na lokalnom računaru softverski imitira terminalni uređaj zovemo „terminal emulator“. Terminal emulator **nije** komandni interpreter (shell). Radi se o dva odvojena programa. Terminal emulator samo obezbeđuje prihvatanje znakova koje korisnik unosi preko tastature (ili šalje sa udaljenog računara serijskom vezom), kao i ispis teksta koji programi izbacuju (uključujući i ono što izbacuje komandni interpreter, jer je i on običan program).

Kada na tipičnoj Linux distribuciji sa grafičkim okruženjem pokrenemo grafički terminal emulator, on će iscrtati svoj prozor i u njemu pokrenuti podrazumevani komandni interpreter (shell). Znakovi koje korisnik unese preko tastature terminal emulator će proslediti shell-u. Tekst koji shell izbacuje tokom rada, terminal emulator će prihvatiti i prikazati u svom grafičkom prozoru.

### 8.3 Primeri često korišćenih shell programa

Verovatno najpopularniji shell je program **bash** (skraćeno od Bourne Again Shell). Originalni Unix shell zvao se Bourne Shell, po imenu autora (Stephen Bourne). Bourne Again Shell je igra reči i simbolizuje da „ponovo rođenje“ ovog shella, sada napisanog za GNU projekat pod GPL licencom. Na embedded sistemima često se koristi jednostavniji **ash** shell, jer je bash prilično glomazan i spor (u kontekstu jako ograničenih hardverskih resursa). Izvršni fajl bash-a je osam do deset puta veći od ash-a (~1 MB naspram ~100 kB).

Shell signalizira da je spreman da prihvati komandu tako što ispisuje prompt. **Ash** i **bash** prikazuju različite promptove zavisno od korisničkog naloga. Znak "**#**" označava da je u trenutnoj sesiji aktivan administratorski root nalog. Znak "**$**" označava da je aktivan običan korisnički nalog ograničenih privilegija.

Pored izvršavanja komande za komandom, shell podržava interpretiranje tzv. shell skripti. To su tekstualni fajlovi sastavljeni od niza komandi koje bismo inače unosili jednu po jednu preko tastature. Na ovaj način, moguće je automatizovati često ponavljane poslove. Pandan shell skriptama na Windowsu su .bat i .cmd fajlovi (od batch, na engleskom gomila ili hrpa, simbolizujući „gomilu" komandi koju je potrebno izvršiti jednu za drugom).

Pored komandi, u shell skriptama je moguće koristiti promenljive, kontrolne strukture (for petlje, if-then-else, while-do, case...)

**Bash** drastično proširuje shell funkcionalnost koju propisuje POSIX standard. Sa jedne strane, to omogućava veću fleksibilnost pri pisanju shell skripti. Sa druge strane, bash skripte se neće izvršavati kako treba (ili neće nikako) na jednostavnijim shellovima kao što je **ash** (koji takođe proširuje POSIX specifikaciju, ali ne toliko drastično).

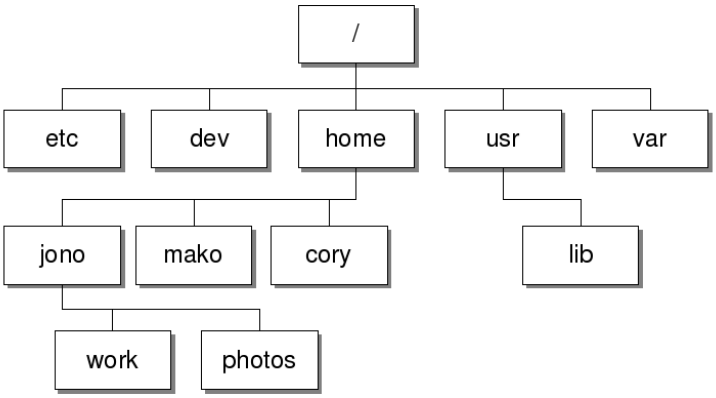
Preporuka: Ne oslanjati se na **bash** ekstenzije, nego pisati skripte tako da budu kompatibilne sa **ash** shellom, naročito ako se pišu za embedded sisteme.

**8.4 Fajl sistemi**

Kao što na Windowsu mora postojati particija na kojoj je instaliran operativni sistem i sa koje se podiže, tako i Linux traži postojanje particije koja će predstavljati root filesystem. Ne mora se raditi o particiji na lokalnom mediju. Root filesystem se može povezati i sa udaljenog računara koristeći NFS (Network File System) protokol. Sadržaj root fajl sistema povezuje se („montira") u koreni folder koji se označava sa „/". Za razliku od DOS-a i Windowsa koji koriste obrnutu kosu crtu za razdvajanje putanja foldera ( \ ), svi Unix klonovi (uključujući Linux i Mac OS X) koriste običnu kosu crtu ( / ). Svi ostali mediji montiraju se u neki podfolder root fajl sistema. Diskovi se ne označavaju slovima i dvotačkama (C:, D: itd) kao na Windowsu.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Jednostavan root fajl sistem*

Primer:

1. Root fajl sistem montiran je sa hard diska tokom podizanja sistema. Particija na kojoj se nalazi root fajl sistem formatirana je kao ext4.
2. Lokalna Windows particija montirana je u /media/windows i koristi fajl sistem NTFS.
3. Ubačen DVD disk montiran je u podfolder /media/dvd i koristi fajl sistem UDF.
4. Ubačen USB fleš drajv montiran je u podfolder /media/stick i koristi fajl sistem FAT32.
5. Network share preko SMB protokola montiran je u podfolder /media/smb.
6. Korisnik sve ovo vidi kao jedinstven fajl sistem i uopšte ne mora biti svestan da se radi o različitim uređajima.

Ovakva mogućnost „priključivanja“ različitih uređaja i fajl sistema u jedinstven koreni fajl sistem moguća je zahvaljujući sloju apstrakcije unutar Linuxa koji se zove **Virtual File System**. VFS predstavlja sloj između konkretnih fajl sistema i ostatka operativnog sistema. OS ne pristupa fajlovima direktno, već preko VFS-a, koji operativnom sistemu servira podatke i informacije o fajlovima u očekivanom obliku (bez obzira što se oni mogu značajno razlikovati u zavisnosti od konkretnog fajl sistema). Zahvaljujući ovome, Linux podržava veliki broj fajl sistema. Podrška za nove fajl sisteme realizuje se kao dodatni interfejs ka VFS-u.

Linux omogućava da deo RAM memorije bude povezan u fajl sistem zarad bržeg pristupa privremenim fajlovima koji ne moraju biti sačuvani pri isključenju sistema (npr. keš web browsera). Moguće je navesti tačnu veličinu RAM diska prilikom montiranja u fajl sistem, a moguće je dozvoliti da operativni sistem dinamički smanjuje i povećava ovakvu „particiju“ (uz mogućnost swapovanja na pravi disk). „Fajl sistem“ koji se specificira pri ovakvom montiranju je **tmpfs**.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Konzorcijum Linux Foundation definiše strukturu sistemskih foldera za Linux, zvanu Filesystem Hierarchy Standard. Najnovija verzija je iz 2004. godine. Distribucije mogu a i ne moraju poštovati FHS.

Linux fajl sistem podržava više tipova fajlova. Tri osnovne grupe:

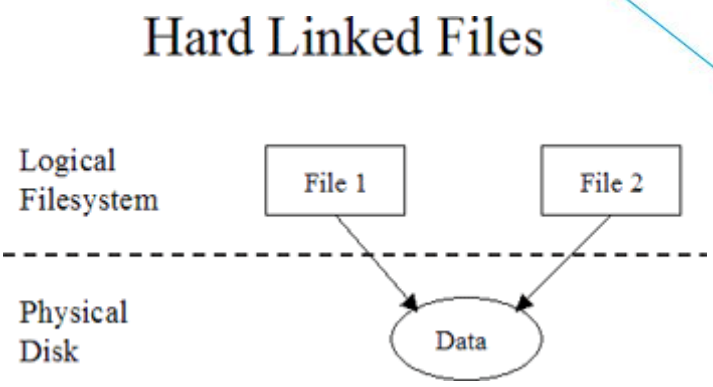
1. Regularni fajlovi (-)
2. Direktorijumi, tj. folderi (d)
3. Specijalni fajlovi

Specijalni fajlovi mogu biti:

1. Simbolički link (l)
2. Characted device (c)
3. Block device (b)
4. Cev, pipe fajl (p)
5. UNIX domain socket (s)

**8.5 Specijalni fajlovi**

Hard link je dodatni unos u fajl sistemu koji pokazuje na isti fizički fajl na disku. Hard linkovi ka fajlu moraju biti u istom fajl sistemu. Tek kada je poslednji hard link na fajl obrisan iz evidencije fajl sistema, prostor koji fajl zauzima na disku biće obeležen kao slobodan za novu upotrebu. Hard linkovi nisu tako očigledni u listinzima foldera.



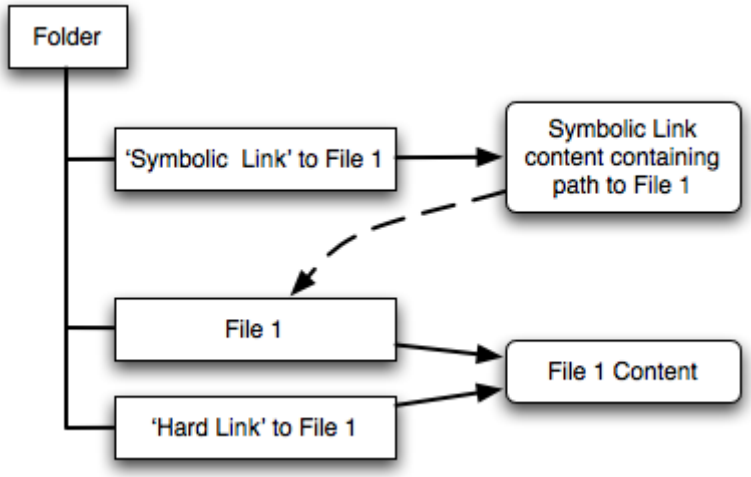

*Hard linkovi*

Simbolički link („soft link“) je dodatni fajl čiji je sadržaj apsolutna ili relativna putanja ka nekom postojećem fajlu (bilo kog tipa). Windows pandan: „shortcut file“ (.lnk). Ako se target fajl obriše,



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

simbolički link neće biti validan („broken link“). Ako se target fajl ponovo uspostavi, simbolički link će opet biti validan. Ako se simbolički link obriše, target fajl neće biti diran.




*Simbolički linkovi naspram hard linkova*

Regularni programi otvaranjem simboličkog linka za čitanje i pisanje automatski otvaraju target fajl. Sistemske komande uglavnom razlikuju simbolički link od ostalih fajlova. Kopiranje simboličkog linka će iskopirati samo link, ne i target fajl. Apsolutne ili relativne putanje u simboličkim linkovima. Preporuka: koristiti relativne kad god je to moguće.

**8.6 Korisnički nalozi**

Glavni administratorski nalog:

1. Apsolutno sve privilegije
2. Korisnički folder u **/root**

Ograničeni nalozi:

1. Privilegije određuje administrator
2. Korisnički folderi u **/home/ime_korisnika**
3. Korisničke grupe

Baza podataka o korisnicima: tekstualni fajl **/etc/passwd**. Sadrži ime, polje za lozinku, brojčani identifikator korisnika, brojčani identifikator grupe, puno ime naloga (npr. ime i prezime), korisnički folder i program koji će biti pokrenut nakon logovanja (npr. **/bin/sh**).

Korisničke lozinke se najčešće više ne čuvaju u **/etc/passwd**. Polje za lozinku sadrži specijalnu oznaku ('x') koja kaže da ona nije tu. Lozinke se sada čuvaju u **/etc/shadow** i to u hešovanom obliku (samo



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

root ima pristup fajlu). Polje za lozinku takođe može sugerisati da li je lozinka uopšte prisutna, kao i da li je nalog trenutno onemogućen. U istom fajlu su i dodatna polja o korisniku. Vreme proteklo od poslednje izmene lozinke, rok trajanja trenutne lozinke, koliko dugo je nalog onemogućen...

Korisnici mogu imati vlasništvo nad fajlovima i pravo pristupa. U fajl sistemima karakterističnim za Unix-kompatibilne operativne sisteme, fajlovi sadrže „masku dozvola“. Devet bita koji određuju read/write/execute prava za vlasnika fajla, grupu kojoj vlasnik pripada i sve ostale, u formatu rwx rwx rwx.

„Execute“ bit na folderu znači pravo izlistavanja sadržaja foldera. Dodatni specijalni bitovi:

1. Sticky bit: Primenjen na folderu, označava da njegove fajlove mogu da menjaju samo root i vlasnici fajlova.
2. Set User ID bit: Proces pokrenut od nekog fajla imaće efektivni user ID vlasnika fajla, a ne korisnika koji je pokrenuo proces (‘x’ u bitmasci biće označeno kao ‘s’).
3. Set Group ID (‘x’ u bitmasci biće označeno kao ‘S’).

Set Group ID za foldere: svi fajlovi kreirani u folderu pripadaće istoj grupi korisnika kao onoj kojoj pripada folder.

### 8.7 Procesi

Proces je instanca programa koja se nalazi u fazi izvršavanja (zajedno sa programskim kôdom u memoriji, stekovima, heapom, stanjem procesorskih registara, spiskom file handle-ova, korisničkim dozvolama...). Kernel u nekom trenutku tokom podizanja sistema pokreće prvi proces iz root fajl sistema (**init**) koji ostaje aktivan do isključenja sistema. Svi procesi imaju jedinstveni brojčani identifikator (process ID). **Init** dobija process ID 1. Postojanje **init**-a je obavezno (a samim tim i postojanje root fajl sistema). Ukoliko ne pronađe **init** u fajl sistemu, kernel će prikazati grešku „kernel panic“.

Drugi procesi u toku rada nastaju tako što **init** klonira sebe u memoriji (sistemski poziv fork), klonirani proces učita drugi binarni imidž iz fajl sistema (sistemski poziv exec) pregazivši sopstveni, i nastavlja sa izvršenjem učitanog imidža. Nakon fork-ovanja, kopirani proces imaće drugačiji process ID. Takođe, rezultat sistemskog poziva fork biće različit u roditelju i kopiji. Na ovaj način možemo proveriti u kom procesu se tačno nalazimo, i izvršiti odgovarajuće akcije (npr. kopija će učitati novi program, roditelj će nastaviti sa svojim poslom).

Zašto su fork i exec razdvojeni?

1. Forkovana kopija zadržava neke resurse roditelja (file handle-ove...)
2. Između ova dva poziva, moguće je izvršiti dodatne režijske poslove (zatvoriti nepotrebne file handle-ove, modifikovati procesne privilegije...)



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

3. Možda nam ne treba učitavanje novog imidža, možda hoćemo samo da dupliramo trenutni proces (fork).

4. Možda nećemo da nastavimo sa izvršavanjem trenutnog procesa, pa nije potrebno kreiranje duplikata već ćemo samo nastaviti sa izvršenjem novog kôda (exec).

5. Drugi procesi takođe mogu pokretati nove procese na isti način, ali je init prvi proces-roditelj.

Originalni i kopirani proces zadržavaju odnos roditelj-dete (numerički identifikator Parent ID). Kada proces-dete završi s radom, proces-roditelj dobija signal o tome. Tek kad roditelj očita izlazni status, proces-dete biva uklonjeno iz memorije. Ako je roditelj završio s radom pre deteta, signal o završetku procesa-deteta neće imati ko da konstatuje. U tom slučaju, proces-dete ostaje u memoriji i ne oslobađa Process ID (broj identifikatora nije beskonačan!)

Ovakav proces se zove zombie process. **Init** je proces menadžer. Osim pokretanja drugih procesa, njegova uloga je da preuzima roditeljstvo nad procesima koji su još u izvršavanju, a roditelji više nisu („siročad“). **Init** povremeno očitava izlazne statuse siročadi koja su završila sa izvršavanjem, kako ne bi postala zombiji.

Proces u toku rada može otvarati fajlove za čitanje, upis i dopunjavanje. Svakom otvorenom fajlu se unutar procesa dodeljuje jedinstveni numerički identifikator („file descriptor“, „file handle“) preko koga se kasnije vrše pozivi read, write, close, itd.

### 8.8 Standardni ulaz i izlaz

Tri fajl identifikatora su dodeljena procesu odmah po pokretanju:

1. Standardni ulaz, 0 (ili makro STDIN_FILENO u <unistd.h>)

2. Standardni izlaz, 1 (STDOUT_FILENO)

3. Standardni izlaz za greške, 2 (STDERR_FILENO)

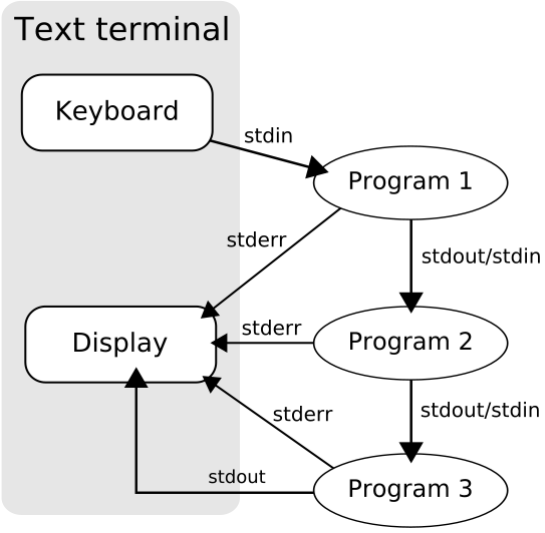
Funkcija printf ispisuje tekst na standardni izlaz. Isto što i **fprintf(stdout, …)** Ispis na standard error stream: **fprintf(stderr, …)**

Funkcija scanf očitava sadržaj standardnog ulaza. Isto što i fscanf(stdin, ...)

Ako je program pokrenut na uobičajen način iz shell-a, shell će prihvatati standardni izlaz i error procesa i ispisivati ih na ekranu, a prosleđivaće sadržaj tastaturnog bafera terminala standardnom ulazu procesa.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Primer ulaznih i izlaznih tokova*

Standardni ulaz, izlaz i error stream moguće je preusmeravati. Ovo se intenzivno koristi naročito u shell skriptama. Ako program pokrenemo sa program > izlaz.txt, sve što program ispiše na svoj standardni izlaz neće se pojaviti na ekranu, nego će biti upisano u tekstualni fajl izlaz.txt. Ako program pokrenemo sa program < ulaz.txt, svo programsko čitanje sa standardnog izlaza neće biti vršeno preko tastature nego iz tekstualnog fajla ulaz.txt. Preusmeravanje sa „>“ obrisaće postojeći sadržaj fajla. Umesto toga, moguće je nadovezati ispis na postojeći sadržaj fajla koristeći redirekciju sa „>>”.

Moguće je preusmeriti standardni izlaz jednog procesa na standardni ulaz drugog. Na ovaj način se ostvaruje međuprocesna komunikacija koristeći FIFO bafer u čijem punjenju i pražnjenju u memoriji se brine kernel. Ovakva komunikacija zove se „pipeline“. Proces koji čita iz pipeline-a biće blokiran sve dok proces koji upisuje u pipeline ne pošalje podatke.

### 8.9 Runleveli

Postoji više implementacija **init**-a.

1. BSD init (iz BSD Unix-a)
2. System V init (iz System V Unix-a)
3. Upstart (iz Ubuntu Linuxa)
4. systemd

Na svom pokretanju, **init** očitava konfiguracione fajlove kako bi znao koje sve procese treba da pokrene.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

1. BSD init implementacija: shell skripta **/etc/rc**
2. System V init implementacija: fajl **/etc/inittab**
3. Upstart koristi sopstvene „job configuration“ fajlove za brže asinhrono pokretanje procesa, ali može da radi u sysvinit kompatibilnom režimu.
4. Systemd je još napredniji, a podržava Upstart i sysvinit kompatibilne režime.

System V init podržava tzv. **runlevel** nivoe. Svaki runlevel nosi numeričku oznaku i sadrži spisak procesa koje treba pokrenuti na ulasku u runlevel, odnosno okončati na izlasku.

Pandan na Windowsu: „Start Windows normally“, „Safe mode“, „Safe mode with command prompt“, „Safe mode with networking“...

U toku rada, moguće je prelaziti iz runlevela u runlevel bez restartovanja računara.

Runleveli koje definiše Linux Standard Base:

0: isključivanje računara

1: single-user režim

2: multi-user režim bez mrežnih funkcija

3: multi-user režim sa mrežnim funkcijama

4: ne koristi se

5: multi-user režim sa grafičkim okruženjem

6: restart računara

Popularne Linux distribucije ovo uglavnom ne poštuju. Podrazumevani runlevel je definisan u **/etc/inittab**. Prelazak između runlevela vrši se komandom **telinit <broj>**. Prikaz trenutnog runlevela vrši se komandom **runlevel**.

# 9. Git, sistem za kontrolu verzija fajlova (VCS)

### 9.1 VCS – Version control sistemi

Od fundamentalnog značaja je koristiti VCS pri razvoju softvera. VCS služi za čuvanje prethodnih verzija fajlova uz opcione komentare pri svakom unosu prethodne verzije u bazu. Omogućava upoređivanje razlika između verzija, „premotavanje“ na starije verzije fajlova, grananje projekta (npr. radi eksperimentisanja) i lakšu kolaboraciju na projektu.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Prvi VCS se zvao RCS („Revision Control System“). Izašao 1982. Godine i operisao samo na pojedinačnim fajlovima. Naslednik: CVS (1990. godine). Originalno, to je bio samo skup skripti koje pozivaju RCS program.

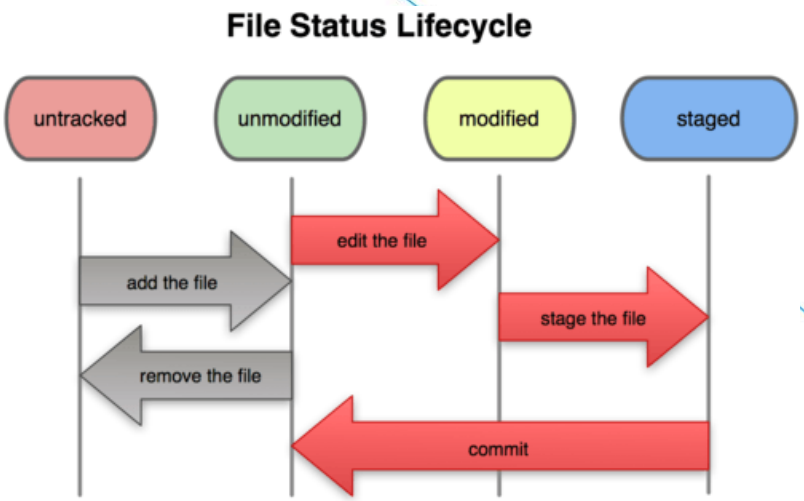
Brojni VCS sistemi su danas u upotrebi. Postoje tri grupe VCS sistema: Lokalni (RCS…), centralizovani (CVS, SVN, Team Foundation Server…) i distribuirani (Git, Bazaar, Mercurial, BitKeeper…).

### 9.2 Git

Git je 2005. godine razvio Linus Torvalds, originalni autor Linux 221oi me. Izvorni kôd Linuxa je do 2002. godine održavan prostim arhiviranjem fajlova i razmenom patcheva između developera. Od 2002. do 2005. godine korišćen distribuirani CVS 221oi menu BitKeeper. Iako vlasnički softver, bio je besplatan za developere koji nisu učestvovali u razvoju konkurentskih CVS-ova (Richard Stallman nije bio zadovoljan time što se i dalje radilo o vlasničkom softveru). 2005. godine BitKeeper prestaje da bude besplatan. Linus potom piše sopstveni distribuirani CVS. Glavni kriterijumi bili su:

1. Brzina pri upravljanju ogromnim projektima
2. Suštinski jednostavan dizajn
3. Odlična podrška za grananje

Rezultat: Git. Prva verzija izašla je već u aprilu 2005.



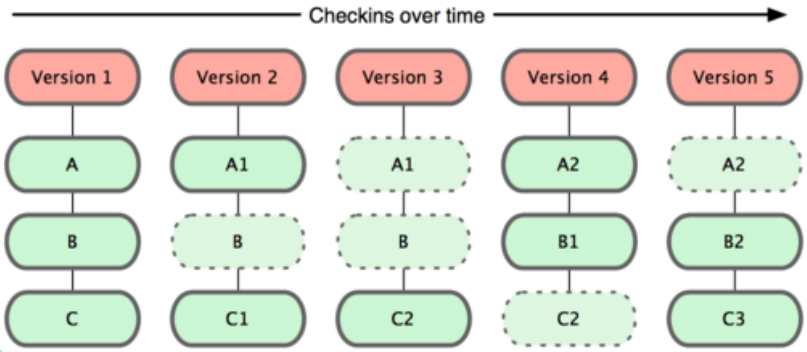

*Moguća stanja fajlova praćenih Git-om*

Git ne čuva razlike između fajlova, već snapshotove celih foldera. Ako fajl nije menjan između verzija, Git baza interno čuva samo link na fajl iz prethodne verzije, radi uštede prostora. Gotovo sve operacije su lokalne, zbog postojanja lokalnog repozitorijuma. Manja zavisnost od mreže znači veću brzinu rada. Git praktikuje strogo hešovanje fajlova i unosa u repozitorijum, pa je praktično



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

nemoguće da se podaci promene (npr. zbog greške u mrežnom prenosu ili fajl sistemu) a da Git to ne primeti. Koristi se 20-bajtni (40 cifara) heš po SHA-1 algoritmu. Svi unosi se čuvaju (makar kao privremeni zapisi u internoj bazi), pa je vrlo teško napraviti slučajne greške brisanjem podataka.



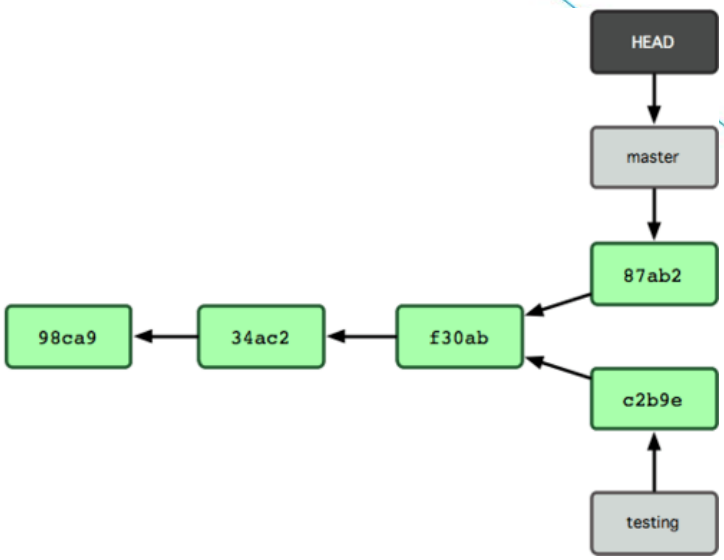

*Skladištenje verzija fajlova kod Git-a*

### 9.3 Običan nasuprot „bare“ repozitorijumu.

Običan repozitorijum sadrži radnu kopiju i Git bazu u skrivenom podfolderu **.git**. Bare repozitorijum sadrži samo Git bazu koja nije u podfolderu. Isti format baze važi u oba slučaja. Bare repozitorijum se najčešće koristi na 222erver projekta i predstavlja tačku kolaboracije učesnika na projektu.

### 9.4 Grananje projekta

„Jeftino“ (lightweight) grananje bilo je jedan od glavnih kriterijuma pri dizajnu Git-a. Cilj je bio omogućiti brzi kreiranje grana, brzo prebacivanje između grana i jednostavno spajanje grana (npr. test grane sa glavnom).




*Grane predstavljaju samo imenovane pokazivače na određene unose*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

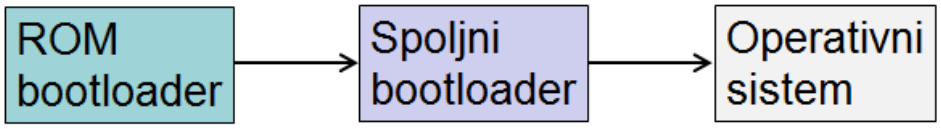
# 10. U-Boot, univerzalni bootloader

### 10.1 Uloga bootloadera pri podizanju Linuxa na embedded sistemima

Pre nego što glavni operativni 223trapp počne sa izvršavanjem, neko mora da „pripremi teren“, odnosno da izvrši minimalnu inicijalizaciju hardvera i podigne operativni 223trapp sa spoljnog medija („boot“). Tačna realizacija zavisi od arhitekture. Kod Intel x86 računara je to standardizovano zarad PC kompatibilnosti (tj. ako računar ne može da pokrene nemodifikovani MS-DOS, nije x86 PC). Kod procesora koji implementiraju ARM arhitekturu to nije standardizovano i ne postoji zvanični boot framework na ARM-u.

Proizvođači procesora ARM arhitekture implementiraju minimalni bootloader programiran u ROM unutar procesora. Uloga ROM bootloadera je da izvrši najosnovniju inicijalizaciju ugrađenih kontrolera i učita pravi bootloader u interni RAM u procesoru. U nekim slučajevima (npr. NOR Flash) moguće je pokrenuti bootloader direktno sa spoljne memorije, bez učitavanja u SRAM. Proces učitavanja i pokretanja spoljnog bootloadera zove se bootstrapping.

Uloga učitanog bootloadera je da pripremi sve potrebne hardverske komponente (uključujući inicijalizaciju SDRAM memorije) i pusti u rad glavni operativni 223trapp. Za razliku od PC BIOS-a, kada glavni OS krene sa izvršavanjem, bootloader će biti pregažen u memoriji i njegove usluge više neće biti dostupne. Glavni operativni 223trapp u ovom slučaju mora samostalno izvršiti svu (re)konfiguraciju hardvera, jer ne postoji mogućnost poziva BIOS rutina kao na PC-u.




*Najčešći način podizanja operativnog sistema na ARM embedded platformama*

ROM bootloader u ARM procesoru je apsolutno minimalistički, ne radi interaktivno i ne izbacuje tekstualne poruke ili GUI na ekran ili serijski port. Svi prohtevi ROM bootloadera i detalji rada opisani su u referentnom uputstvu za konkretan 223trapping. Na procesoru Texas Instruments AM3984/DM8168, interni ROM je veličine 48 kB. ROM bootloader može imati ograničenja po pitanju veličine spoljnog bootloadera koji može biti učitan. Iako AM3894/DM8168 ima interni RAM veličine 512 kB, binarni fajl spoljnog bootloadera može biti najviše 255 kB, osim u slučaju pokretanja sa MMC/SD kartice kada je ograničenje 128 kB.

Ponašanje ROM bootloadera može se konfigurisati na određeni način, zavisno od konkretnog procesora (na primer, putem pin 223trapping).



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Na AM3984/DM8168, postoji pet pinova koji se podešavanjem spoljnih pull-up i pull-down otpornika bira jedan od ponuđenih boot prioriteta (npr. SD, SPI, UART, Ethernet MAC). Pored prioriteta spoljnog medija/interfejsa, jedan od pinova služi za biranje da li će se bootloader pokrenuti direktno sa spoljne memorije (gde je to moguće, npr. NOR) ili će se prethodno učitati u interni RAM.

**10.2 U-Boot**

Često korišćeno rešenje na embedded sistemima je **Das U-Boot**. Projekat je otvorenog kôda i pod GPL licencom: http://www.denx.de/wiki/U-Boot/

Započet 1999. godine. Visoko je portabilan, zvanično podržava nekoliko arhitektura (PowerPC, ARM, MIPS, x86, Motorola 68K, NIOS, Microblaze, Blackfin) i preko 200 platformi. Visoko konfigurabilan. Isključivanjem i uključivanjem podrške za razne uređaje, protokole, fajl-sisteme i komande, veličina rezultujućeg binarnog fajla kreće se od nekoliko desetina do nekoliko stotina kilobajta. Konfigurisanje se vrši pre kompajliranja, popunjavajući odgovarajuće C zaglavlje koje će glavni Makefile uključiti u prevođenje.

U-Boot podržava tekstualni interfejs i obradu komandi putem komandne linije. Postoje jednostavni komandni interpreter i napredniji "Hush" shell interpreter. Podržava čuvanje i učitavanje konfiguracionih parametara sa spoljne memorije (npr. EEPROM, Flash, MMC/SD). Podizanje operativnog sistema može biti potpuno automatizovano, „hardkodovanjem" niza komandi u bootloaderu ili učitavanjem komandne skripte sa spoljnog medija.

Svi imidži koje U-Boot može da učita i pokrene moraju imati odgovarajuće zaglavlje. Zaglavlje je veličine 64 bajta, definisano u **include/image.h**. Sadrži arhitekturu, operativni sistem, tip imidža, tip kompresije, load adresu, entry point adresu, ime imidža, da li je „execute in place" ili ne... Odgovarajući imidž kreira se alatom **mkimage** na host računaru.

U-Boot prepoznaje više tipova binarnih imidža:

1. Samostalni programi iz kojih je moguć povratak u U-Boot
2. Imidži kernela operativnih sistema (nema povratka)
3. RAM disk imidži (lokacija i veličina se prosleđuje učitanom OS-u)
4. „Multi-file" imidži, arhive sa više fajlova, npr. jezgro OS-a plus RAM disk. Korisno pri butovanju preko mreže.
5. Firmware imidž
6. Komandne skripte

Makefile automatski poziva **mkimage** za pripremanje odgovarajućeg imidža, zavisno od build targeta.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 11. Arhitektura Linuxa

### 11.1 Tipovi jezgra (kernela)

Jezgro (kernel) operativnog sistema čini skup podsistema („modula“) koji obezbeđuju osnovnu funkcionalnost operativnog sistema. Kerneli se po svojoj arhitekturi mogu razvrstati u tri grupe: 225ernel225rnel, monolitni, hibridni.

Primeri mikrokernela: QNX, Minix, GNU Hurd.

Primeri hibridnog: Windows NT (XP, Vista, 7, 8…), Apple XNU (Mac OS X, iOS).

Primer monolitnog: Unix System V, BSD, Linux.

Mikrokernel sadrži apsolutni minimum funkcionalnost operativnog sistema (mehanizme međuprocesne komunikacije, memorijsko i procesno upravljanje i raspoređivanje, kao i I/O najnižeg nivoa).

Monolitni sadrži mnogo više (npr. drajvere za periferne uređaje, TCP/IP stek, grafički podsistem, podršku za razne fajl sisteme…)

Hibridni pristup je rešenje kao „najbolje iz oba sveta“. Grafički podsistem je do Windowsa XP bio deo 225ernel, a od Viste je izdvojen zarad veće stabilnosti 225ernel.

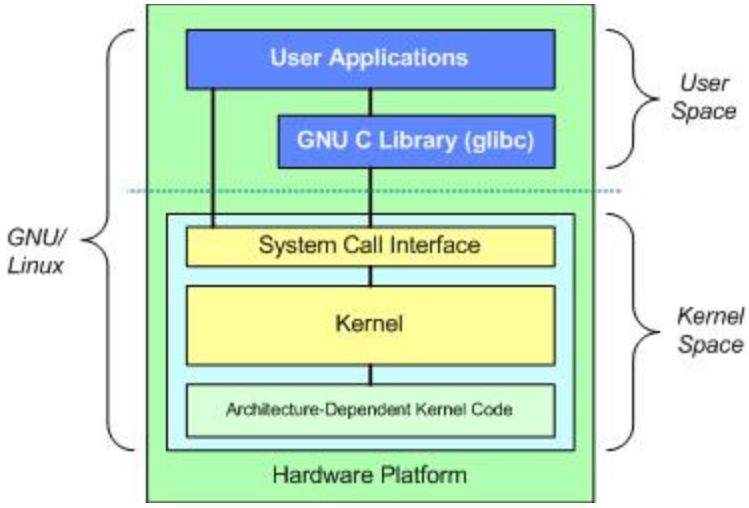
Linux podržava eksterne module koji se mogu učitati i izbaciti iz kernelskog memorijskog prostora po potrebi. Na primer, drajveri koji su ugrađeni u kernel imidž će stalno biti u memoriji iako za tim nema potrebe (npr. u sistemu ne postoji uređaj za koji je taj drajver pisan). Ako se izvedu kao kernelski moduli, biće učitani tek ako se detektuje prisustvo odgovarajućeg hardvera, ili će učitavanje biti svejedno pokušano, a drajver će proveriti prisustvo hardvera i okončati izvršavanje ako nema. Kernel moduli se čuvaju u fajlovima sa ekstenzijom **.ko**.

### 11.2 User space i kernel space

Kernel space je memorijski prostor u kome se izvršavaju kernel i njegovi moduli. Kôd koji se izvršava u kernelskom prostoru ima pune privilegije, mogućnost izvršavanja svih procesorskih instrukcija i ima pristup svoj memoriji i I/O opsegu.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Arhitektura GNU/Linux distribucije*

User space je ograničeno okruženje u kome se izvršavaju aplikacije. Implementacija se vrši koristeći hardversku zaštitu:

1. Ring levels 0-4 na x86. Linux koristi Ring 0 za kernel i Ring 4 za user space aplikacije.
2. Osam operativnih režima na ARM-u, najznačajniji „user mode“ i „sypervisor mode“. Dodatni režimi: IRQ mode, monitor mode, undefined mode, system mode…

**11.3 Komponente kernela**

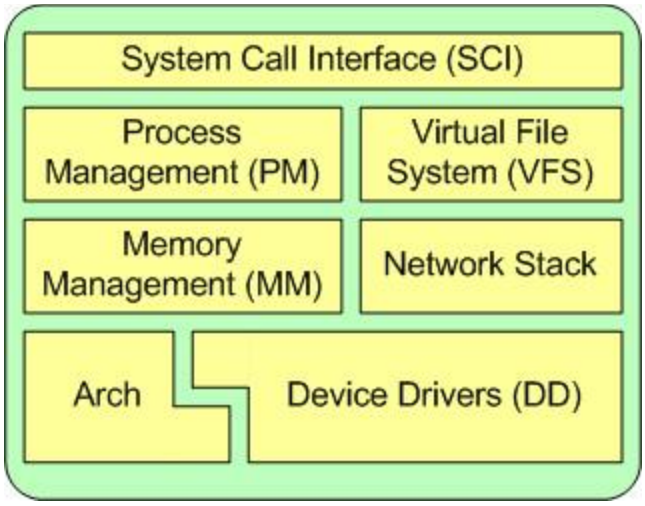
„System call interface“ je tanak sloj koji obezbeđuje mehanizam za pozivanje kernelskih funkcija iz userspace aplikacija. Zavisi od arhitekture i može se razlikovati između procesora iste arhitekture. Implementacija u vidu izvornog kôda nalazi se u folderima „kernel“ (nezavisna od arhitekture) i „arch“ (zavisna od arhitekture).

Upravljanje procesima brine se o pokretanju, izvršavanju, zaustavljanju i prebacivanju između procesa. Procesi u kernelu su „lightweight“ procesi (threads) opisani kôdom, podacima, stekom i stanjem CPU registara. U userspace-u, procesi imaju svoj memorijski prostor, a threadovi unutar procesa dele isti prostor i određene resurse. Implementacija u folderima „kernel“ (nezavisna od arhitekture) i „arch“ (zavisna od arhitekture).

Upravljanje memorijom vrši se u vidu stranica od 4 kB. Implementacija u folderu „mm“.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES



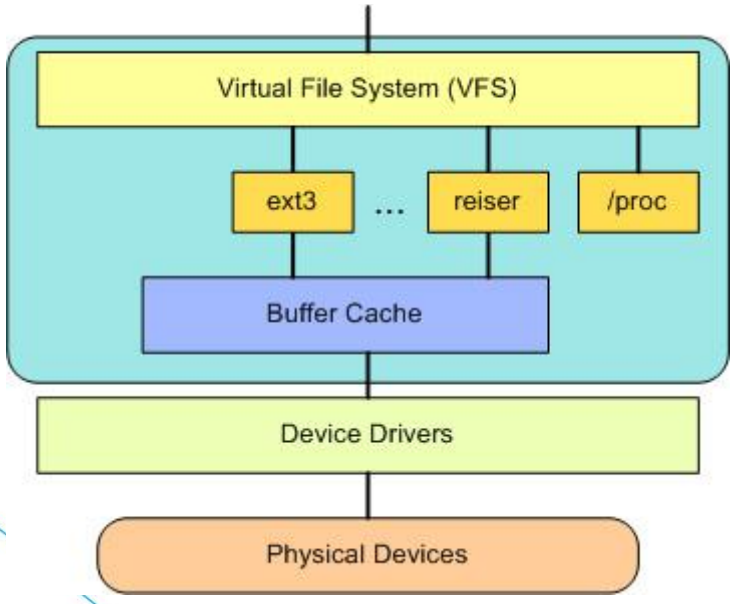

*Grubi prikaz komponenti Linux kernela*

Linux podržava virtuelnu memoriju i straničenje na spoljnu memoriju. Umesto fajla u root fajl sistemu koji služi kao „swap file“, rezerviše se odvojena particija koja se ne formatira („swap partition“). Procesi operišu isključivo nad virtuelnim adresama. Za kernel module postoji sistemski poziv koji od fizičke adrese daje odgovarajuću virtuelnu, pošto se na mnogim arhitekturama (uključujući ARM) pristup periferijama vrši isključivo koristeći „memory mapped input/output”. Intel x86 sadrži odvojenu I/O magistralu i podržava pristup periferijama preko dodatnog opsega adresa I/O portova, mada se danas i tu primarno koristi memory mapped I/O.

Mrežni stek obezbeđuje implementaciju IP protokola, transportnog protokola (TCP) i socket sloja, kome se pristupa preko sistemskih poziva. Socketi služe kao interfejs ka mreži iz userspace aplikacija. Postoji standardizovani API koji obezbeđuje pristup brojnim protokolima, od sirovih frejmova, preko IP protocol data unitsa, do TCP-a i UDP-a. Implementacija u folderu „net“.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Virtual File System*

Drajveri, upravljački programi za periferne uređaje. Moguće više slojeva apstrakcije (npr. drajver za SPI Flash memoriju, koji se oslanja na generičke interfejse ka SPI magistrali a koje implementira konkretan SPI bus drajver). Uređaji se userspace-u predstavljaju virtuelnim fajlovima, a „hendlere“ za operacije nad virtuelnim fajlovima obezbeđuju drajveri. Implementacija je u folderu „drivers“.

Sav kôd koji nije zajednički i zavisi od konkretne arhitekture nalazi se u folderu „arch“ i njegovim podfolderima. Treba izbegavati pisanje architecture-dependant kôda u ostalim podsistemima.

# 12. Unutrašnjost Linuxa

### 12.1 Procesi

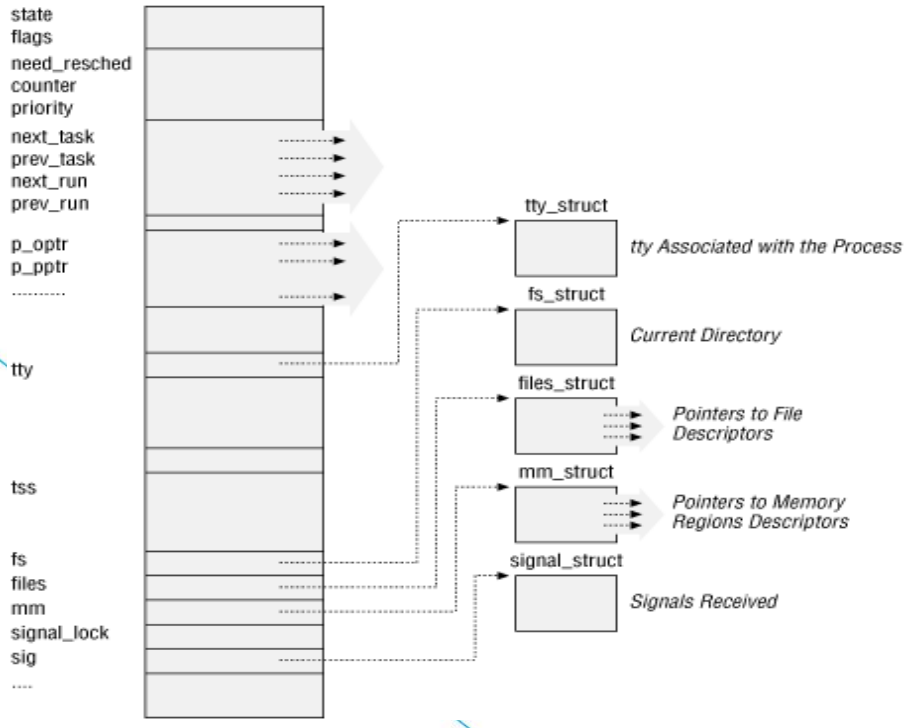
Proces je instanca programa u fazi izvršavanja. Proces ima vreme nastanka, određeno trajanje, može proizvesti jedno ili više dece. Iz perspektive kernela, proces je entitet koji rezerviše i troši sistemske resurse (memoriju, procesorsko vreme...). Proces se kreira od programskog kôda i promenljivih, učitanih iz izvršnog fajla koji se nalazi u fajl-sistemu. Proces manipuliše dodeljenim resursima koristeći sistemske pozive. „Process descriptor“ je struktura podataka u kernelu koja sadrži sve informacije o nekom procesu, uključujući dodeljene resurse.

Proces je virtuelizovana apstrakcija. Linux svakom procesu obezbeđuje linearan memorijski prostor (jer podržava koncept virtuelne memorije) i virtuelizovani procesor (jer podržava preemptivni multi-tasking). Svaki proces „misli“ da se izvršava kao jedini u sistemu i da je kompletan računar samo na njegovom raspolaganju. Program može biti pisan da sarađuje sa drugim programima, ali u toku rada proces svejedno „ne zna“ kada je pauzirani i nastavljen. Ovakva apstrakcija ostvaruje se koristeći



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

hardverske mehanizme savremenih procesora. „Savremeni procesori“ su svi od Intela 80386 naovamo.




*Proces descriptor*

### 12.2 „Init“ proces

Svaki proces identifikuje se pozitivnim celim brojem zvanim „Process ID“ (PID). PID prvog kreiranog procesa je 1. Svaki sledeći dobija PID za broj veći, ili prvi slobodni PID nekog procesa koji je okončao izvršavanje. Prvi proces (init) učitava i kreira sam kernel. Okončani procesi se ne uklanjaju momentalno iz memorije, već šalju povratni rezultat roditelju koji bi trebalo da ga proveri (sistemski poziv „wait“ koji reaguje na „promenu stanja“ deteta). Tek kad roditelj konstatuje da je dete-proces okončalo rad, ono se uništava.

Proces koji je završio s radom, a roditelj to još nije konstatovao, zove se „zombi“. Ako se roditelj okončao pre deteta, dete-proces u teoriji može „večno“ ostati zombi. Init povremeno radi „wait“ poziv i proverava stanje sve svoje dece koja su još u memoriji, čime se sprečava masovna „zombifikacija“. Zbog toga init ima ulogu „proces menadžera“.

### 12.3 Multi-threading

Proces se formalno sastoji od jedne ili više „niti izvršavanja“ (thread of execution). Thread je apstrakcija odgovorna za trenutno izvršavanje kôda. Veliki broj procesa je i dalje jednonitan (single-threaded).



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Thread ima sopstvene atribute: Stek koji čuva lokalne promenljive i povratne adrese, procesorsko stanje (tj. stanje registara), uključujući trenutnu lokaciju u programskom kôdu (registar program counter). Ostatak resursa je deljen između threadova istog procesa. Svi threadovi u procesu izvršavaju se u istom memorijskom prostoru. Tačna implementacija podrške za threadove zavisi od konkretnog operativnog sistema. U davnim verzijama Linuxa nije postojala podrška za threadove unutar procesa na nivou kernela. Kernel nije razlikovao single-threaded i multi-threaded aplikacije. Kreiranje threadova i njihov scheduling vršilo se na userspace nivou, mehanizmima POSIX pthread biblioteke. Kasnije je na nivou kernela ubačena podrška za „lightweight procese".

Multi-threaded aplikacija se implementira koristeći grupe procesa koje dele isti memorijski prostor, otvorene fajlove... Userspace aplikacije ne koriste sistemske pozive za kreiranje threadova. Multi-threaded aplikacije i dalje kreiraju threadove koristeći API funkcije biblioteke pthread (koja je na Linuxu deo standardne GNU C biblioteke). Kako onda kernel zna da je na userspace nivou kreiran novi thread i da sistemski scheduler treba da ga ima u vidu?

Počev od kernela 2.6, koristi se Native POSIX Thread Library (sada takođe sastavni deo GNU C biblioteke) čije usluge koriste funkcije iz pthread biblioteke za potrebe kreiranje threadova.

### 12.4 Sistemski pozivi

Sistemski pozivi su interfejs između userspace aplikacija i kernelskih usluga (npr. pristupa hardveru). Nekoliko prednosti: Aplikacija ne mora znati sve detalje o operativnom sistemu i hardveru; Veća sigurnost sistema: kernel proverava da li je sve po PS-u; Aplikacije su portabilnije ako se oslanjaju na skup sistemskih poziva dostupan na više platformi.

Linux implementira nekoliko stotina sistemskih poziva, naspram više hiljada na Windowsu. Preko 90% sistemskih poziva je implementirano za sve arhitekture koje Linux podržava. Linux garantuje kompatibilnost sistemskih poziva u budućim verzijama.

Sistemski pozivi se najčešće ne pozivaju direktno, već preko funkcija „omotača" (wrapper) iz standardne C biblioteke. Tačan način izazivanje sistemskog poziva zavisi od arhitekture. Čak i na istoj arhitekturi može postojati više načina. Aplikacija pokreće sistemske pozive preko POSIX API funkcija (iz standardne C biblioteke), ali treba praviti razliku između bibliotečne funkcije i sistemskog poziva. Aplikativnog programera u velikom broju slučajeva ne interesuje razlika, već je bitan rezultat funkcije.

Wrapper funkcije za sistemske pozive vraćaju celobrojnu vrednost koja signalizira da li je poziv izvršen uspešno ili ne.

### 12.5 Signali

Signali su mehanizam komunikacije (definisan POSIX standardom) između userspace procesa. Takođe i mehanizam kojim kernel obaveštava procese o bitnim događajima. Osnovne uloge:

1. Obaveštavanje procesa o dešavanju određenog događaja



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

2. Aktiviranje odgovarajućeg signal handlera, ako je implementiran u programu

Informacija koju proces dobije o nekom signalu je najčešće samo broj signala. Broj podržanih signala zavisi od arhitekture.

| Signal | Value | Action | Comment |
|---|---|---|---|
| SIGHUP | 1 | Term | Hangup detected on controlling terminal or death of controlling process |
| SIGINT | 2 | Term | Interrupt from keyboard |
| SIGQUIT | 3 | Core | Quit from keyboard |
| SIGILL | 4 | Core | Illegal Instruction |
| SIGABRT | 6 | Core | Abort signal from abort(3) |
| SIGFPE | 8 | Core | Floating point exception |
| SIGKILL | 9 | Term | Kill signal |
| SIGSEGV | 11 | Core | Invalid memory reference |
| SIGPIPE | 13 | Term | Broken pipe: write to pipe with no readers |
| SIGALRM | 14 | Term | Timer signal from alarm(2) |
| SIGTERM | 15 | Term | Termination signal |
| SIGUSR1 | 30,10,16 | Term | User-defined signal 1 |
| SIGUSR2 | 31,12,17 | Term | User-defined signal 2 |
| SIGCHLD | 20,17,18 | Ign | Child stopped or terminated |
| SIGCONT | 19,18,25 | Cont | Continue if stopped |
| SIGSTOP | 17,19,23 | Stop | Stop process |
| SIGTSTP | 18,20,24 | Stop | Stop typed at terminal |
| SIGTTIN | 21,21,26 | Stop | Terminal input for background process |
| SIGTTOU | 22,22,27 | Stop | Terminal output for background process |

*Standardni POSIX signali*

Postoje tri načina na koja proces može reagovati:

1. Eksplicitno ignorisanje signala
2. Izvršenje podrazumevane akcije
3. Prijem signala i izvršenje signal handlera

Pored regularnih signala, POSIX definiše i „real-time signale“.

### 12.6 Mehanizmi međuprocesne komunikacije

Linux podržava više mehanizama za međuprocesnu komunikaciju (IPC, inter-process communication):

1. Neimenovane cevi (one-way)
2. Imenovane cevi (FIFO fajlovi)
3. Usermode semaphore
4. Message queue



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

5. Deljena memorija
6. Socket

### 12.7 Upravljanje memorijom

Linux se za upravljanje memorijom oslanja na hardversku MMU jedinicu unutar procesora. Stranica je najmanja količina RAM-a kojom MMU može upravljati. Razne arhitekture podržavaju razne veličine stranica (često u više režima).

Linux podržava virtuelnu memoriju, pa svi procesi dobijaju ceo virtuelni adresni prostor, iako možda nema dovoljno fizičke memorije. Na spoljnoj memoriji (npr. hard disku) moguće je kreirati odvojenu particiju koja će služiti samo za zamenu stranica („page swapping“). Takva particija se ne formatira ni u jedan fajl sistem i pristupa joj samo kernel.

Više stranica virtuelne memorije mogu se mapirati na istu fizičku stranicu, naročito kada su u pitanju stranice iz različitih virtuelnih adresnih prostora koji pripadaju različitim procesima. Na ovaj način više procesa mogu deliti istu fizičku memoriju.

Na primer, gotovo svi procesi koriste usluge standardne C biblioteke (**glibc**). Umesto učitavanja kopije za svaki proces, biblioteka će biti učitana jednom, a svi procesi će mapirati njenu memoriju u svoj adresni prostor. Deljeni podaci mogu biti rezervisani samo za čitanje, samo za pisanje ili oba.

Kod stranica po kojima je moguće pisati, svi procesi će „trenutno“ videti promene nakon upisa. MMU takođe može presresti pokušaj upisa i izazvati interapt. Kernel će potom kreirati kopiju stranice i vršiti upis po njoj umesto po originalnoj. Dinamička alokacija unutar procesa vrši se na njegovom heap-u (**.data** segment). Alternativno, moguće je dinamički zauzimati memoriju na steku procesa.

### 12.8 Obrada interapta

Interapti se mogu obrađivati samo u kernel modu. Izvori interapta zavise od arhitekture. GPIO ulazi mogu biti izvori interapta. Možda nisu podrazumevano tako setovani, već postoji registar GPIO kontrolera gde to treba uključiti. Implementiramo funkciju koja će služiti kao interrupt handler.

### 12.9 Real-time programiranje na Linuksu

Standardni Linux kernel ispunjava samo soft real-time kriterijume za jedan operativni sistem, omogućujući „kontrolu“ vremena iz userspace-a preko implementacije odgovarajućih POSIX implementacija, ali bez garancije o striktnom ispunjenju vremenskih rokova (hard real-time). Takođe, standardni Linux kernel u osnovnoj konfiguraciji omogućava preempting taskova od strane taskova višeg prioriteta samo pri izvršavanju userspace procesa.

Značaj poboljšanja real-time performansi Linuxa ogleda se u smanjenju latencije, tj. slanja zahteva operativnom sistemu i vremena kada taj zahtev počne da se obrađuje (npr. interrupt latency, scheduling latency), kao i u povećanju throughput-a, tj. količine korisnog posla koji računar može obaviti u datom vremenskom periodu.

Počev od generacije 2.6, Linux podržava opciju CONFIG_PREEMPT, sa kojom se kernel može buildovati tako da kernelski kôd postane preemptibilan. Do tada, samo userspace procesi su mogli biti preemptibilni. Osnovna ideja: omogućiti preemptovanje procesa nižeg prioriteta dok rade u kernelskom režimu (tj. dok kernel u njihovo ime izvršava određene usluge, npr. sistemske pozive). Rezultat: smanjenje ukupne latencije sistema u situacijama pod velikim opterećenjem procesora. Ipak, ovo ne omogućava preempting baš svog kernelskog kôda. Određene kritične funkcije se i dalje izvršavaju bez prekida (npr. U blokovima zaštićenim spinlock-om, ili privremenim isključenjem interapta lokalnog procesora).

Sa zvaničnom opcijom CONFIG_PREEMPT, Linux i dalje nije hard real-time sistem. Odzivnost je povećana, ali vreme izvršenja i dalje nije garantovano. Na primer, uvek se može desiti obrada interapta, a interapt hendleri se izvršavaju atomično. Odziv celog sistema je i dalje reda veličine nekoliko milisekundi.

Ideja iza CONFIG_PREEMPT_RT projekta: pretvaranje Linuxa u hard real-time operativni sistem koji može da garantuje izvršavanje poslova u određenom roku (koliki god on bio). Ovo nije jedino rešenje tog tipa. Na primer, postojao je projekat RTLinux (WindRiver), gde se pod okriljem RTOS mikrokernela Linux kernel izvršavao kao još jedan preemptibilan proces, i samim tim je bio pod punom kontrolom mikrokernela.

Projekat CONFIG_PREEMPT_RT (po istoimenoj opciji u Linux kernel konfiguraciji) kao skup patcheva razvijaju kernel developeri još od 2005. godine. Cilj: kompletan kernel postaje preemptibilan (što i dalje nije potpuno moguće, ali je svejedno na višem nivou nego sa CFG_PREEMPT).

Najbitnije osobine:

1. Isključenje interapta i dalje onemogućava preemptovanje
2. Spinlock-ovi postaju mutexi (osim onih najkritičnijih).
3. Interapt hendleri postaju kernel threadovi (što znači da mogu biti schedulovani).
4. Procesi koji se izvršavaju u kernelskom režimu zadržavaju procesne prioritete.

# 13. Uređaji i drajveri pod Linuksom

### 13.1 Podrška za periferijske uređaje

Osnovna uloga operativnog sistema je da obezbedi okruženje u kome će se izvršavati aplikativni softver. Aplikacije komuniciraju s hardverom, implicitno ili eksplicitno. Zbog bezbednosti i stabilnosti sistema, aplikacije su izolovane u user mode i nemaju direktan pristup hardveru, svim procesorskim instrukcijama, obradi interapta... Uloga drajvera je da obezbedi što je moguće veće transparentno rukovanje hardverom. Idealno, tako da aplikacije znaju samo najpotrebniji minimum osobina hardvera.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Ključna stvar: apstrakcija. Aplikacije vide uređaje i upravljaju njima preko virtuelnih fajlova u folderu **/dev** („Unix filozofija“: sve je fajl). Aplikacije mogu dobijati određene informacije o hardveru putem virtuelnih fajlova u folderu **/sys**. Aplikacija može otvoriti virtuelni fajl iz foldera **/dev** i nad njim vršiti uobičajene pozive kao što su read i write. Specijalne funkcije nad uređajem mogu se obaviti koristeći sistemski poziv **ioctl** (drajver za uređaj implementira odgovarajuće handlere za svaki podržani **ioctl** poziv).

### 13.2 Tipovi uređaja

Karakter uređaji su oni čijim podacima se pristupa nebaferisano (bez keširanja). Mogu se tretirati kao FIFO uređaji, ali ne moraju biti. Na primer, drajver može implementirati funkciju „llseek“ kojim će menjati vrednost internog pokazivača i „premotavati“ po podacima (primer iz prošlosti: magnetna traka za bekapovanje podataka).

Blok uređaji su oni kod kojih se podacima pristupa keširano, u blokovima bajtova (npr. NAND flash memorija, diskovi). Pristup nije FIFO, već je slobodan („random access“). Pristup blok uređajima je baferisan.

Mrežni uređaji služe za komunikaciju između računara pa je implementacija drajvera značajno drugačija od drajvera za karakter i blok uređaje. Podaci se razmenjuju u vidu paketa koristeći neki od dostupnih protokola. Osim komunikacije, drajveri se bave podešavanjem adresa uređaja, parametara prenosa. Implementacija I/O funkcija zavisi od konkretnog protokola (TCP, UDP…). Primer: Ethernet MAC drajver.

Pored fizičkih uređaja, uređaji mogu biti i virtuelni, kreirani u određenom memorijskom opsegu. Na primer, terminal (tty uređaj, skraćeno od „tele-typewriter“) nije hardverski terminal povezan na računar već je implementiran kao virtuelni uređaj u memoriji, ali mu ostali softver pristupa isto kao da je u pitanju fizički uređaj.

Pseudo-uređaji su oni koji nisu povezani sa fizičkim uređajima. Primer: **/dev/zero** – upis nema efekta, a čitanje proizvodi nula-bajtove (u beskonačnoj količini).

### 13.3 Obrada interapta

Obrada interapta je od ključnog značaja za implementaciju drajvera. Drajver implementira funkciju koja će služiti kao ISR, a onda je registruje kod kernela i vezuje za određeni IRQ. U ISR rutinama, zabranjeno je korišćenje bilo kakvog kôda koji može izazvati „sleep“ čekanje. Čak ni funkcije koje indirektno pozivaju funkcije i sistemske pozive koji mogu izazvati „sleep“ čekanje. ISR rutine nisu povezane sa userlandom putem procesnog konteksta. Ako se uradi „sleep“ u ISR, sheduler će prebaciti na neki drugi proces, a pošto ISR nema svoj proces kontekst, nema povratka.

Konflikt: brada interapta treba da traje što je kraće moguće, a istovremeno, u obradi interapta treba da se odradi sav posao jer na izlasku nema povratka. Rešenje: top-half i bottom-half pristup.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

ISR funkcija predstavlja „gornju polovinu“, funkciju koja je samo signalizirala kernelu da je slučaj primljen i „obrađen“. U stvarnosti, ISR funkcija je samo prebacila posao na „donju polovinu“, odnosno neku drugu rutinu koja će da završi konkretan posao.

Tri implementacije:

1. SoftIRQ (ne može biti preemptovan)
2. Tasklet (ne može biti preemptovan)
3. Work queue (može biti preemptovan)

SoftIRQ je direktna bottom-half implementacija. Taskleti se oslanjaju na SoftIRQ, ali su jednostavniji za upotrebu. Work queues se izvršavaju u procesnom kontekstu i mogu raditi „sleep“.

### 13.4 Komunikacija drajvera sa userspace-om

Iz drajvera nije moguće otvoriti fajl zarad upisa i čitanja. Komunikaciju treba vršiti preko virtuelnog fajlsistema **/proc**.

Primer: korišćenje GPIO pinova iz programa ili shell skripti. Ako se uključi opcija CONFIG_GPIO_SYSFS pri buildovanju kernela, GPIO drajver kreira virtuelne fajlove preko kojih se može kontrolisati ponašanje i očitavanje stanja GPIO pinova iz userspace-a. Potom se stanje GPIO pinova može menjati i očitavati iz aplikativnog softvera (uključujući sistemske komande):

**$ echo 30 > /sys/class/gpio/export**

**$ echo "out" > /sys/class/gpio/gpio30/direction**

**$ echo 1 > /sys/class/gpio/gpio30/value**

**$ echo 0 > /sys/class/gpio/gpio30/value**

**$ echo "in" > /sys/class/gpio/gpio30/direction**

**$ cat /sys/class/gpio/gpio30/value**

**$ echo 30 > /sys/class/gpio/unexport**

GPIO pinovi koje je neki drajver već rezervisao za sebe (npr. pozivom funkcije **gpio_request**) ne može biti kontrolisan iz „sysfs“ fajl sistema, osim ako u istom drajveru to nije omogućeno pozivom funkcije **gpio_export**.

Pregled GPIO pinova koji su trenutno zauzeti od drajvera:

**$ mount -t debugfs debugfs /sys/kernel/debug**

**$ cat /sys/kernel/debug/gpio**

# 14. Linux na embedded platformama

### 14.1 Specifičnosti Linuxa na embedded platformama

Iako su današnje embedded 236ystem236236 bazirane na ARM Cortex A jezgrima dovoljno moćne za veliki broj poslova nekada rezervisan za stone računare, to su i dalje sistemi ograničenih hardverskih resursa. Zbog svega toga, Linux distribucije za embedded 236ystem236236 mogu se značajno razlikovati od Linux distribucija za stone računare ili servere. To je i dalje operativni 236ystem zasnovan na Linux 236ystem preuzetom sa kernel.org, ali uz :

1. Root fajl 236ystem smanjen na minimum
2. Bez grafičkog 236ystem236
3. Bez paket-menadžera koji razrešavaju međuzavisnosti programa i deljenih biblioteka
4. Podrazumevani korisnički nalog je *root* nalog
5. U produkciji ne mora biti nikakve interakcije s korisnikom
6. Često se koristi Flash memorija kao storage
7. Često se fajl 236ystem podiže preko mreže

### 14.2 Specijalne verzije C biblioteka

Standardna GNU C biblioteka može biti prevelika i 236revise memorijski zahtevna za rad u embedded okruženju. Na primer, standardna implementacija funkcije printf koristi malloc alokaciju, varijabilne liste argumenata… GNU C biblioteka u sebi sadrži razne druge biblioteke koje ne moraju biti potrebne u embedded okruženju (libNSS, libDB…). GNU C biblioteka čuva binarnu kompatibilnost sa prethodnim verzijama. Postoje varijante GNU C biblioteke namenjene embedded sistemima: eglibc, µClibc, Bionic (specifična za Android).

Programi statički linkovani sa ovakvim bibliotekama mogu biti 10 do 100 puta manji nego kad se linkuju sa glibc.

### 14.3 BusyBox

Projekat BusyBox od suštinskog značaja za upotrebu u embedded sistemima. Koristi se kao osnova minijaturnih root fajl sistema za embedded uređaje. Sve je počelo kada je Bruce Perens, koosnivač Open Source Inicijative, 1995. godine hteo da podigne minijaturnu Linux distribuciju sa jedne jedine flopi diskete od 1,44 MB.

Minimalni root fajl 236nterv bi od izvršnih fajlova trebalo da sadrži init, shell, udev i osnovni skup sistemskih komandi iz paketa GNU Coreutils (ls, ln, cp, mv, rm, chmod, chown, cat…)



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Problem: Svaki izvršni fajl sadrži odgovarajuće zaglavlje, a Coreutils komande mogu sadržati redundantan kôd.

Rešenje: Objediniti funkcionalnost inita, shella i coreutils komandi u isti izvršni fajl.

Prednosti:

1. Nema 237nterval zbog pojedinačnih izvršnih fajlova
2. Minimizacija redundantnog kôda
3. Manja potrošnja RAM memorije (svi procesi dele iste stranice koje sadrže .text segment BusyBox-a)

BusyBox se može pokrenuti u različitim režimima:

1. Pokretanje BusyBox izvršnog fajla sa podrazumevanim imenom (busybox) ali bez dodatnih parametara izlistaće sve komande (uključujući init i shell) koje su ugrađene i dostupne u izvršnom fajlu
2. Pokretanje s parametrom (npr. busybox ls) izvršiće navedenu komandu (ls). Naravno, podržano je prosleđivanje parametara (busybox ls /dev)
3. Ako se BusyBox izvršni fajl zove isto kao komanda, on će to prepoznati i automatski funkcionisati u režimu te komande)

Funkcionalnost koju BusyBox nudi je ograničenija u poređenju sa GNU alatima (ali je ključna funkcionalnost sačuvana). Na primer, BusyBox sadrži "ash" shell koji je u potpunosti POSIX kompatibilan, ali manje fleksibilan u odnosu na "bash". Ostale komande nemaju sve opcije (npr. **ping** nema podešavanje 237nterval između pingovanja, **ls** ne podržava sortiranje ispisa...)

Komponente BusyBox-a mogu se uključivati i isključivati po želji pre buildovanja, koristeći konfiguracioni fajl istog formata kao što koristi Linux kernel. Umesto editovanja konfiguracionog fajla, moguće podešavanje preko „grafičkog" interfejsa pozivanjem **make menuconfig**, isto kao što se može konfigurisati i Linux kernel.

### 14.4 Flash memorije

Kada se flash memorija koristi kao storage, bitno je minimizovati upise u root fajl 237revis na flashu (zbog brzine i ograničenog broja upisa), a naročito ne treba koristiti swap particiju na flashu.

Često korišćeni tipovi flash memorija: NOR i NAND.

NOR flash je word-adresibilan („random access"), slično RAM-u. Obrisan bit je bit koji ima vrednost 1. Programiranje (upis nule) može se vršiti na nivou bita. Ostali biti neće biti menjani. Brisanje (upis



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

jedinice) iziskuje brisanje cele stranice (npr. 64 ili 128 kB). Mikrokontroleri i procesori za embedded primene često mogu izvršavati kôd direktno iz NOR-a, bez prethodnog kopiranja instrukcija u RAM.

NAND flash se može posmatrati kao blokovski uređaj, podeljen na erase blokove (npr. 128, 256 ili 512 kB). Upis i čitanje vrše se isključivo na nivou stranica (npr 1, 2 ili 4 kB). Zbog toga se NAND ne može mapirati u adresni prostor RAM memorije. Prazna (obris ana) stranica je ona koja sadrži samo jedinice. Upis nula u praznu stranicu je moguć bez diranja drugih stranica u erase bloku. Brisanje (vraćanje nula u jedinice) u okviru stranice iziskuje prethodno brisanje kompletnog erase bloka. NAND stranice sadrže dodatan prostor koji se može iskoristiti za ECC.

NOR flash je 238revise skup i spor da bi se koristio za root fajl 238revis (npr 32 MB NOR flash-a košta kao 512 MB NAND flash-a). NOR se često koristi samo za podizanje bootloadera i Linux 238revis. Root fajl 238revis se montira iz NAND flash-a.

Iako je NAND blokovski uređaj, ne možemo ga tretirati identično kao hard disk, zbog wear levelinga. Zašto SSD možemo, kad je SSD takođe sastavljen od NAND čipova? Zato što SSD pored flash čipova sadrži i odgovarajući kontroler koji se o tome brine umesto nas, a „spolja“ se predstavlja kao hard disk.

Drugi problem: 238revise čipova različitih karakteristika („geometrije“). Kada neki čip uđe u end-of-life fazu, često ga u dizajnu ploče nije lako zameniti drugim. Zbog toga se danas sve češće prelazi na druga rešenja: eMMC čipove povezane na MMC/SD interfejs, SSD drajvove povezane na SATA interfejs…

Ako ipak moramo da radimo direktno sa NAND čipova, moramo poznavati specifičnosti rada.

### 14.5 Linux MTD

Linux kernel sadrži softversku komponentu zvanu MTD („Memory Device Technology“) koji obezbeđuje sloj apstrakcije ka uobičajenim tipovima karakter i blok uređaja koje implementiraju drajveri.

MTD sloj „eksportuje“ flash particije u virtuelni fajl 238eveli u oba oblika: karakter uređaje (**/dev/mtd1**, **/dev/mtd2**…) i blok uređaje (**/dev/mtdblock1**, **/dev/mtdblock2**)

Potrebno je koristiti specijalne fajl sisteme:

1. JFFS2 (Journaling Flash File System 2)
2. UBIFS (Unsorted Block Image File System)

Oni su prilagođeni karakteristikama „sirovog“ flash-a i vode računa o wear 238eveling. Proverom ECC-a mogu obeležiti problematične blokove kao neispravne, zarad preskakanja budućeg upisa.

JFFS2 koristi kompresiju, ali sa druge strane:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

1. Prilikom montiranja, sve stranice moraju biti skenirane, pa montiranje velikih particija može trajati i po nekoliko minuta
2. Upis mnogo malih blokova ne sarađuje dobro sa kompresijom
3. Nije moguće jasno reći koliko ima slobodnog prostora

Dodatni problem: JFFS2 radi wear-leveling samo na nivou particija.

Preporuka je da se koristi napredniji UBIFS koji uvodi dodatni sloj, UBI (Unsorted Block Image), oslanjajući se na mehanizme koje obezbeđuje MTD sloj. UBI predstavlja „erase block management“ sloj jer vodi računa o wear-levelingu i obeležavanju loših blokova.

Putem UBI-ja, ceo čip se apstrahuje u jedan veliki „disk“. Potom se particije formatiraju u UBIFS i montiraju na takvom disku, a UBI sloj se brine da se ceo kapacitet koristi za wear-leveling. UBIFS takođe podržava kompresiju, ali se ona može isključiti. UBI ne skenira blokove tako da montiranje particija traje daleko kraće.

### 14.6 NFS fajl sistem

Montiranje root fajl sistema moguće je i preko mreže. Koristi se Network File System (NFS). Otvoreni protokol koji je 1984. godine razvio Sun Microsystems, u cilju omogućavanja pristupanju fajlovima preko mreže kao da se radi o lokalnom skladištu (disku i fajl sistemu).

## 15. Android u embedded okruženju

### 15.1 Specifičnosti Androida u odnosu na Linux

Android je operativni sistem zasnovan na Linux kernelu, prvenstveno namenjen prenosnim uređajima kao što su mobilni telefoni i tablet računari. U najširem smislu, Android se može smatrati još jednom Linux distribucijom, ali **vrlo** specifičnom Linux distribucijom.

Android je u načelu POSIX-kompatibilan, ali se autori nisu trudili da po svaku cenu ispoštuju POSIX kompatibilnost. Prioritet: izolacija aplikacija.

Tek od verzije kernela 3.3, izmene koje su pravili autori Androida postale deo mainline kernela. Specifičnosti:

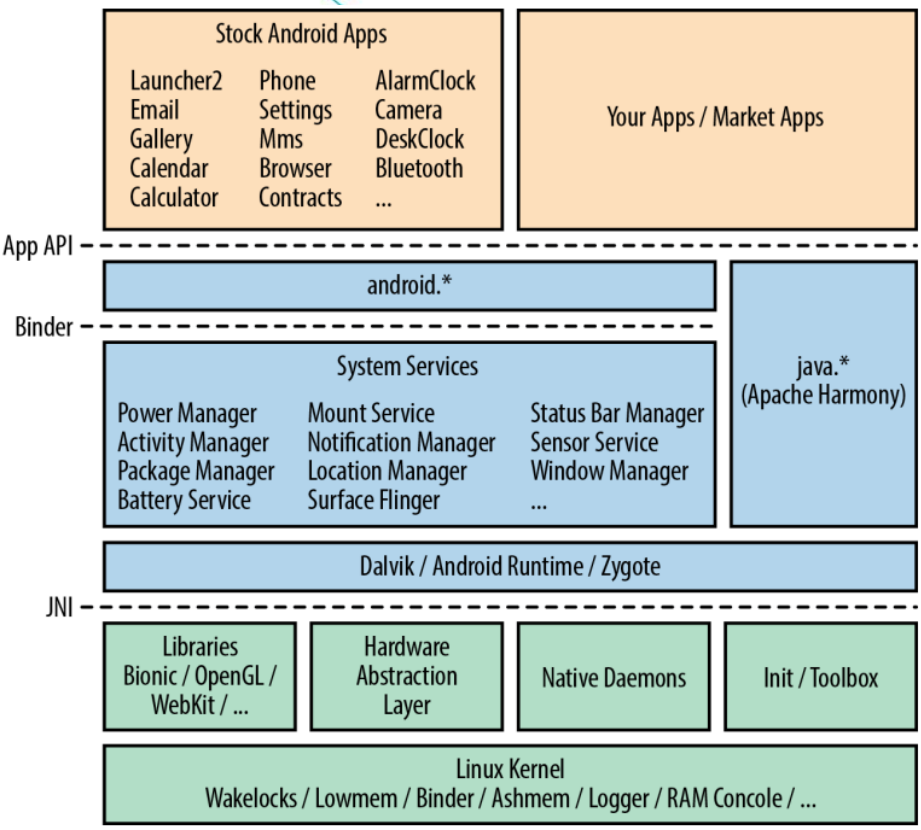
1. Binder, novi mehanizam međuprocesne komunikacije
2. „ashmem“, nova implementacija deljene memorije (/dev/ashmem)
3. PMEM, „process memory allocator“, smišljen s ciljem upravljanja velikih fizički kontinualnih memorijskih regiona koji treba da budu deljeni između userspace-a i drajvera. Za razliku od



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

„ashmem“-a koji upravlja virtuelnom memorijom, „pmem“ upravlja fizičkom memorijom (npr. za potrebe GPU-a, DSP-a itd)

4. Logger, novi mehanizam sistemskog logovanja poruka
5. Wakelocks, novi mehanizam power managementa koji omogućava držanje uređaja „budnim“ dokle god je wakelock „zaključan“
6. OOM („out of memory“) mehanizam prati stanje memorije i terminiše procese kada nivo slobodne memorije postane kritičan
7. Alarm tajmeri, koji omogućavaju userspace aplikacijama da putem servisa AlarmManager kažu kernelu kada bi htele da budu „probuđene“
8. Paranoid network security, onemogućavanje pojedine mrežne funkcionalnosti zavisno od grupe kojoj proces pripada (npr. nemogućnost kreiranja Bluetooth, Ipv4 ili Ipv6 socketa)
9. Timed GPIO, mehanizam koji omogućava manipulaciju GPIO pinovima iz userspace-a (uključujući automatsko vraćanje na prethodni nivo posle određenog vremena)




*Arhitektura Androida*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Umesto GNU C biblioteke, Android koristi biblioteku „Bionic“ (BSD licenca). Android ToolBox, pandan BusyBox-u na Linuxu, ne sadrži init i shell. Ugrađene komande imaju manje funkcionalnosti od GNU Coreutils komandi. Androidov init čita iz fajla /init.rc, odakle se vrši kompletna inicijalizacija (promenljive, mountovanje...).

Linux aplikacije komuniciraju sa hardverom koristeći virtuelne fajlove u folderu /dev. Android implementira kompletan “hardware abstraction layer” u vidu C++ biblioteka, na koje se naslanja Java API. Od aplikacija se očekuje da koriste funkcionalost hardvera putem sistemskih servisa koji se mogu koristiti u objektno orijentisanom maniru API-ja. Postoji oko sedamdeset sistemskih servisa.

### 15.2 Dalvik virtuelna mašina

Dalvik je Java virtuelna mašina napisana specijalno za Android. Za razliku od Sun/Oracle Java virtuelne mašine koja implementira stek-procesor, Dalvik je bliži uobičajenim procesorima sa registrima. Sun/Oracle Java virtuelna mašina očekuje .jar fajl koji sadrži .class fajlove (Java programe „prevedene“ u „mašinski“ kôd Java virtuelne mašine, takozvani „byte-kôd“). Dalvik očekuje .dex fajl sastavljen od .class fajlova obrađenih alatom **dx** i prilagođenih Dalvik virtuelnoj mašini. Do verzije Androida 2.2, Dalvik je izvršavao Java aplikacije čistom interpretacijom. Od verzije 2.2 implementiran je „just-in-time compilation“, odnosno prevođenje (u RAM) onih blokova byte-kôda u nativni mašinski kôd za koje se očekuje da će uskoro biti pozvani i izvršavani.

### 15.3 ART virtuelna mašina

ART je Java virtuelna mašina za Android sledeće generacije, koju je Google razvijao dve godine u tajnosti. Debitovala u Androidu 4.4 kao opciona Java virtuelna mašina koju korisnici mogu eksplicitno uključiti. Biće podrazumevana Java virtuelna mašina u sledećoj verziji kodnog imena „Android L“. ART koristi „ahead-of-time“ kompilaciju, odnosno prevođenje u mašinski kôd prilikom instalacije aplikacije, umesto prilikom pokretanja. Ovo rezultuje boljim performansama, nauštrb veće potrošnje skladištenog prostora, pošto Android čuva kako instalacioni APK paket, tako i verziju programa prevedenu u nativni kôd konkretne platforme na kojoj se izvršava (Intel, ARM ili MIPS).

### 15.4 JNI, Java Native Interface

JNI (Java Native Interface) omogućuje pisanje delova aplikacije (ili celih aplikacija) u C ili C++ umesto u Javi. Koristi se za delove kôda koji iziskuju kritične performanse, a koji se odvajaju u kompajlirane **.so** biblioteke („shared object“, pandan DLL fajlovima na Windowsu) koje Java kôd učitava i poziva po potrebi. Za razvoj JNI koda potrebno je koristiti NDK (Native Development Kit).

# 16. Literatura


[1] MSP430x5xx and MSP430x6xx Family User's Guide, **SLAU208M**, Texas Instruments, Februar2013.

[2] MSP430F5438A, MSP430F5437A, MSP430F5436A, MSP430F5435A, MSP430F5419A, MSP430F5418A Mixed Signal Processor (Device Datasheet), **SLAS655D**, Texas Instruments, Avgust 2013.

[3] MSP430 Architecture (Chapter 4), Texas Instruments University Program for MSP430, 2009.

[4] Device Systems and Operating Modes (Chapter 5), Texas Instruments University Program for MSP430, 2009.

[5] General purpose Input/Output (Chapter 6) , Texas Instruments University Program for MSP430, 2009.

[6] Timers (Chapter 7), Texas Instruments University Program for MSP430, 2009.

[7] Data Acquisition (Chapter 9), Texas Instruments University Program for MSP430, 2009.

[8] Communications (Chapter 14), Texas Instruments University Program for MSP430, 2009.

[9] MSP430 Hardware Tools, User's Guide, Texas Instruments, 2013.

[10]IAR Embedded Workbench ®, IDE Project Management and Building Guide, IAR Systems, 2013.

[11]NASA C Style Guide, Software Engineering Laboratories Series, Avgust 1994.

[12]MISRA-C: 2004 – Guidelines for the use of the C language in critical systems

[13]David E. Simon, An Embedded Software Primer, Pearson Education, 1999.

[14]Richard Barry, The FreeRTOS™Reference Manual

[15]Stevens / Rago, Advanced Programming in the UNIX Environment (3rd Edition), Addison-Wesley, 2013.

[16]M. Tim Jones, Anatomy of the Linux kernel, ibm.com, 2007.

[17]Matthew / Stones, Beginning Linux Programming (4th Edition), Wrox, 2008.

[18]Yaghmour / Masters / Ben-Yossef / Gerum, Building Embedded Linux Systems (2nd Edition), O'Reilly, 2008.

[19]Ivan Bowman, Conceptual Architecture of the Linux Kernel, oss.org.cn, 1998.

[20]Christopher Hallinan, Embedded Linux Primer (2nd Edition), Prentice Hall, 2010.

[21]Sreekrishnan Venkateswaran, Essential Linux Device Drivers, Prentice Hall, 2008.

[22]Ariane Keller, Kernel Space User Space Interfaces, wiki.tldp.org, 2008.

[23]Blum / Bresnahan, Linux Command Line and Shell Scripting Bible (2nd Edition), Wiley, 2011.

[24]Doug Abbott, Linux for Embedded and Real-time Applications (3rd Edition), Newnes, 2013.

[25]Siever / Figgins / Love / Robbins, Linux in a Nutshell (6th Edition), O'Reilly, 2009.

[26]Robert Love, Linux Kernel Development (3rd Edition), Addison-Wesley, 2010.

[27]Greg Kroah-Hartman, Linux Kernel in a Nutshell, O'Reilly, 2006.

[28]Robert Love, Linux System Programming (2nd Edition), O'Reilly, 2013.

[29]Bovet / Cesati, Understanding the Linux Kernel (3rd Edition), O'Reilly, 2005.

[30]Understanding u-boot-min startup for DM814x, processors.wiki.ti.com

# ARM 32

**Predavači:** Marko Pavlović, Milan Milanović, Milan Oklobdžija

**Ciljevi kursa:**

Sticanje osnovnih znanja o embeded sistemima zasnovanih na ARM arhitekturi.

**Ishod kursa:**

Sticanje znanja i smernica kako projektovati uspešan sistem i postići zadovoljstvo poslodavca.

**Teorijsla nastava:**

1. Uvod u embedded sisteme. Specifikacija zahteva, arhitektura i dizajn električne šeme.
2. Predleg najčešće korišćenih interfejsa u embedded sistemima.
3. Osnove high speed dizajna.
4. Razmatranje zahteva integriteta signala.
5. Saveti i smernice za dizajn elektronskih štampanih ploča.
6. PCI Express arhitektura sistema.
7. Verifikacija dizajna i automatsko testiranje.

**Praktične vežbe:**

20% to 50% trajanja kursa. Tokom praktičnih vežbi polaznici će savladati osnove projektovanja embedded sistema. Praktične vežbe sačinjene su iz sledećih tema: Upoznavanje sa softverskim alatom Altium Designer, crtanje električne šeme, hijerarhijsko projektovanje, dizajn footprint-ova komponenata, postavljanje komponenata na električnu štampanu ploču, povezivanje signala procesora, izrada plana dizajna elektronskih štampanih ploča, proračun parametara elektronskih štampanih ploča, dizajn arhitekture PCIe, alati za konfiguraciju i analizu PCIe na Linux i Windows operativnim sistemima.

**Sadržaj kursa:**

Prikaz embedded sistemima zasnovanih na ARM arhitekturi tj. upoznavanje sa dizajnom hardvera i analizom softverskih zahteva ovakvih sistema. Upoznavanje polaznika sa procesom projektovanja sistema počevši od blok dijagrama, dizajna elektronskih štampanih ploča i razmatranja zahteva integriteta signala.

**Trajanje kursa:**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Kurs će biti realizovan kroz 30 školskih časova (25 za teoretska predavanja i prektične vežbe i 5 za testiranje). Predavanja će biti realizovana tokom jednog kalendarskog meseca (na nedeljnom nivou polovina tokom petka, polovina tokom subote).

**Kurikulum**

Kurs će biti podeljen u tri dela:

- Specifikacija sistema, arhitektura i dizajn električnih šema sa pregledom komunikacionih interfejsa koji se često koriste u embedded sistemima.
- Uputstva i preporuke za dizajn elektronskih štampanih ploča, high speed dizajn.
- PCI Express, verifikacija i automatsko testiranje.

Svaki deo kursa predavaće novi predavač. Prva nedelja posvećena je upoznavanju sa osnovnim elementima i karakteristikama embedded sistema. Predavanja će obuhvatiti specifikaciju sistema, ahitekturu i dizajn električnih šema sa naglaskom na često korišćene komunikacione interfejse u embedded sistemima. Na kraju prve nedelje polaznici će utvrditi stečena teoretska znanja u softverskom alatu Altium Designer. Druga nedelja predavanja obuhvata predavanja o osnovama high speed dizajna i specifikaciji zahteva integriteta signala električne štampane ploče. Praktični primeri sadržaće proračun parametara elektronskih štampanih ploča, planiranje povezivanja, kao i savete i preporuke za povezivanje signala. Treća nedelja je posvećena dizajnu sistema baziranih na PCI Express arhitekturi. Takođe biće pokriveni i verifikacija dizajna i automatsko testiranje u proizvodnji. Praktični primeri će uključiti dizajn jednostavnog sistema baziranog na PCI Express arhitekturi i analizu PCI Express arhiteture računara polaznika. Četvrta nedelja će biti rezervisana za praktične primere embedded sistema, pitanja, diskusiju i finalni test.

**Ocenjivanje**

**1.** Dizajn šeme uređaja - Polaznici će imati zadatak da naprave novi dokument za šemu uređaja u alatu Altium Designer na osnovi šeme koja će biti data na testu. Test će biti ocenjen sa ocenom od 1 do 3. Polaznici će položiti test samo ako dobiju ocenu 2 ili 3.

2. Dizajn štampane ploče uređaja - Test će biti ocenjen sa ocenom od 1 do 3. Polaznici će položiti test samo ako dobiju ocenu 2 ili 3.

3. Test iz oblasti PCIe sistema i automatskog testiranja i verifikacije uređaja - Test će biti ocenjen sa ocenom od 1 do 3. Polaznici će položiti test samo ako dobiju ocenu 2 ili 3.

4. Finalni ABCD test. Polaznici će položiti test samo ako odgovore tačno na više od 50% pitanja. Polaznici će uspešno položiti kurs ako polože sva 4 testa.

**Testiranje**

Testiranje za svaki deo kursa će biti vršeno nedelju dana posle završetka predavanja. Na kraju četvrte nedelje kandidati će raditi završni test koji će obuhvatiti sve oblasti koje su predavane ranije.

Pitanja na testovima i predmet zadataka na projektu neće biti objavljeni zbog poverljivosti



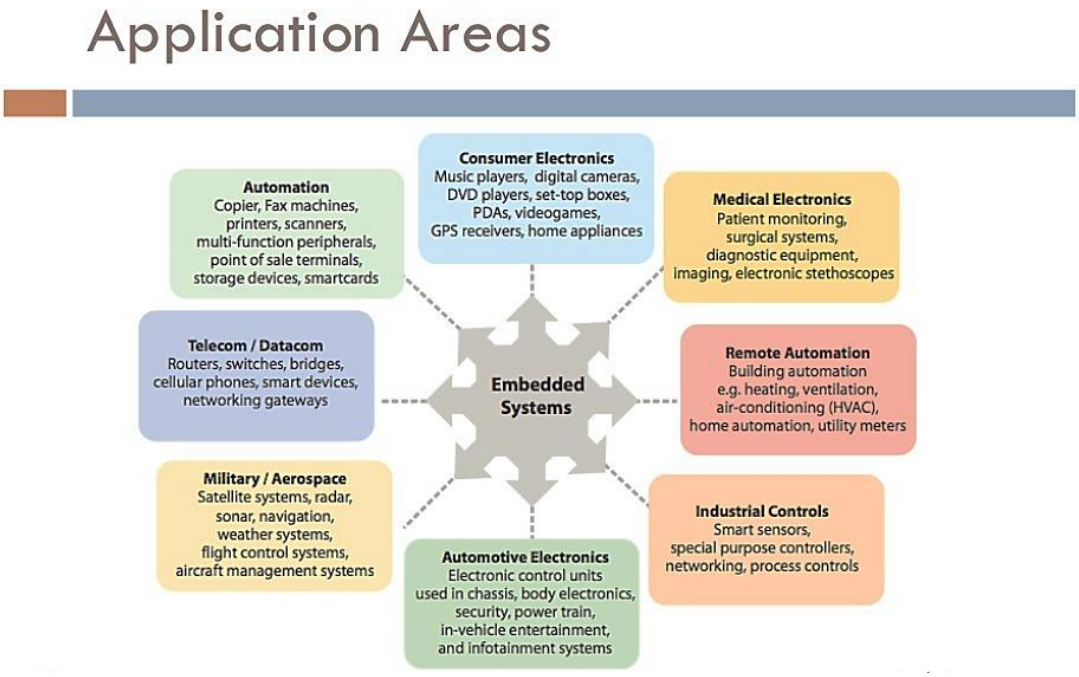
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 1. Namenski sistemi

Namenski (*Embedded*) sistem je s246istem sa specijalizovanom namenom. Kombinacija hardvera i softvera namenjenih određenoj aplikaciji ili proizvodu.

Namenski s246istem je deo većeg s246istema koji ne mora biti kompjuterski s246istem. Za razliku od standardnih kompjutera (PC), namenski s246istem izvršava predefinisane zadatke. Zbog toga inženjeri koji projektuju takav 246sistem mogu da izvrše optimizaciju smanjujući dimenzije i cenu proizvoda.

Jezgro namenskog s246istema je mirkoprocesor ili mikrokontroler koji izvršava određeni broj taskova.



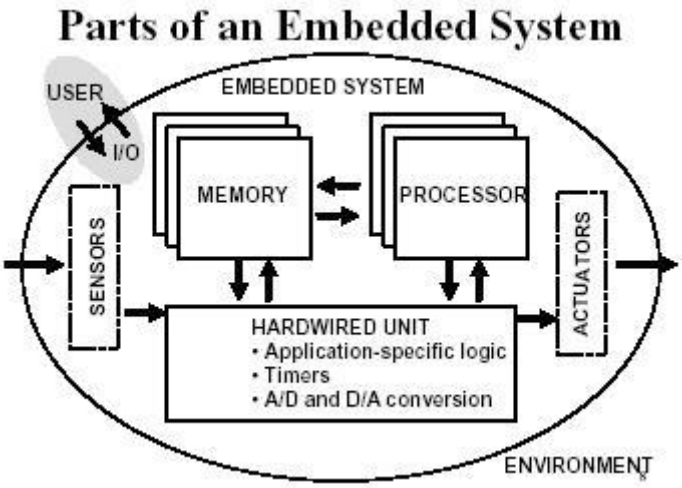

**Slika 2. Oblasti primene namenskih sistema.**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**1.1 Zahtevi od namenskih sistema**

- Niska cena.
- Male fizičke dimenzije.
- Niska potrošnja. Često baterijski napajani.
- Otporan na smetnje, ali ne sme ni da emituje smetnje.
- SoC (System on chip) arhitektura.
- Odziv u realnom vremenu (Real Time system).
- Koriste embedded Linux i RTOS operativne sisteme.




**Slika 3. Opšti blok diagram namenskog sistema**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

## 1.2 Interesantni primeri namenskih Sistema

**Product: Sonicare Elite toothbrush.**

**Microprocessor:8-bit**

**Has a Programmable Speed Control ,Timer &charge gauge.**

**Product: vendo vue 40vending machine**

**Microprocessor: Two** 16-bir Hitachi H8/300H Processors

A robot hand dispenses items

Product: Dresser Wayne Ovation iX Gas Pump.

Microprocessor: Marvel Xscale (ARM)

Os: Windows CE

Display video ads & is networked to gas station's back office computer systemes.also has remote maintenance features.

nan

Product: NASA's Twin Mars Rovers.

Microprocessor: Radiation Hardened 20Mhz Power Pc.

Commercial Real-Time OS

Software and OS was developed during Flight To mars and Downloaded using a Radio link.

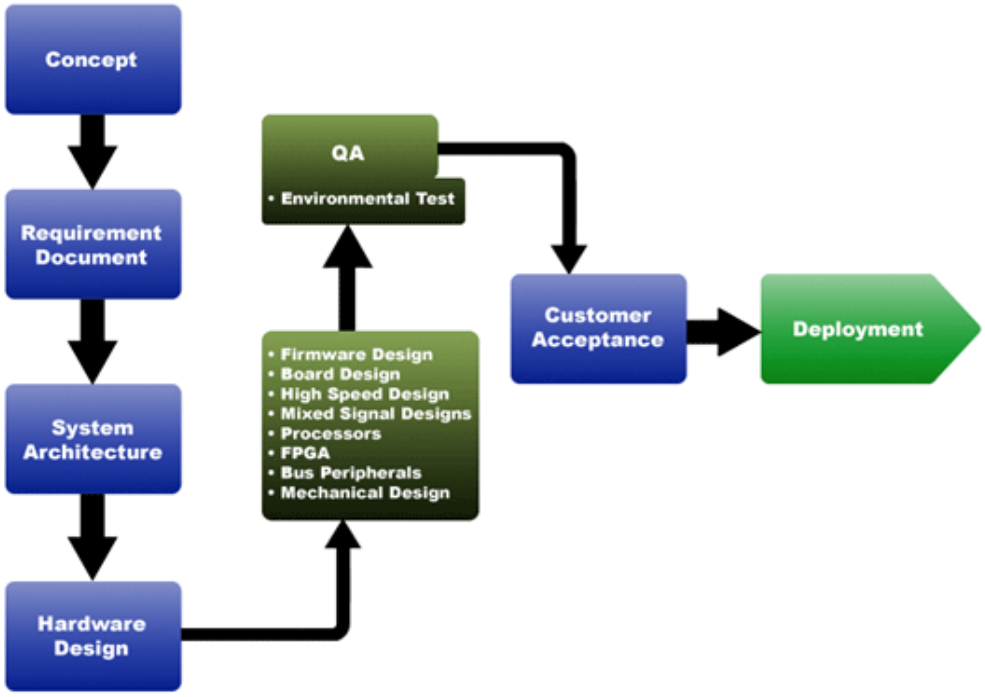
## 1.3 Razvoj namenskih Sistema

Razvoj nameneskih sistema se odvija u nekoliko faza:

1. Koncept
2. Specifikacija
3. Arhitektura
4. Hardver dizajn
5. Softver dizajn
6. Verifikacija



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 5. Diagram razvoja namesnkih sistema.**

# 2 Specifikacija Sistema

Funkcionalna specifikacija je dokument koji jasno i precizno opisuje krucijalne tehničke zahteve, kao i procedure po kojima se može utvrditi da li su zahtevi ispunjeni u procesu razvoja uredjaja. Specifikacija pomaže da se izbegne dupliranje i nekonzistencija i dozvoljava da se precizno proceni potrebni rad i resursi.

Koristi se i kao pregovaracka osnova i dokument pomocu koga se prate promene tokom razvoja. Pruža precizan opis problema koji je potrebno rešiti, na osnovu čega se može efikasno dizajnirati sistem.

Specifikacija pruža vodilje za testiranje i verifikaciju svih tehničkih zahteva. Funkcionalna specifikacija ne definiše sam način rada na razvoju uređaja, niti specificira načina na koji će se zahtevane funkcije implementirati. Umesto toga, specifikacija se fokusira na ono sto će biti vidljivo sa strane korisnika.

Tipična rečenica iz funkcionalne specifikacije glasi:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

"Kada korisnik pritisne dugme za pokretanje uređaja, on mora da da postane potpuno operativan za 30 sekundi."

Postoji puno razloga zbog kojih postoji funkcionalna specifikacija. Jedan od primarnih je da se u timu postigne neka vrsta koncenzusa oko toga sta sve uredjaj treba da radi pre nego sto se pristupi vremenski zahtevnijim fazama razvoja uređaja. Takođe, potrebno je potvrditi da li uređaj zadovoljava sve zahteve koje postavlja klijent.

Dakle svrha specifikacije je:

- Da razvojni tim zna šta treba da razvije.
- Da tim za testirenje zna koje testove treba da izvrši.
- Da klijent zna šta će dobiti.

# 3 Arhitektura sistema

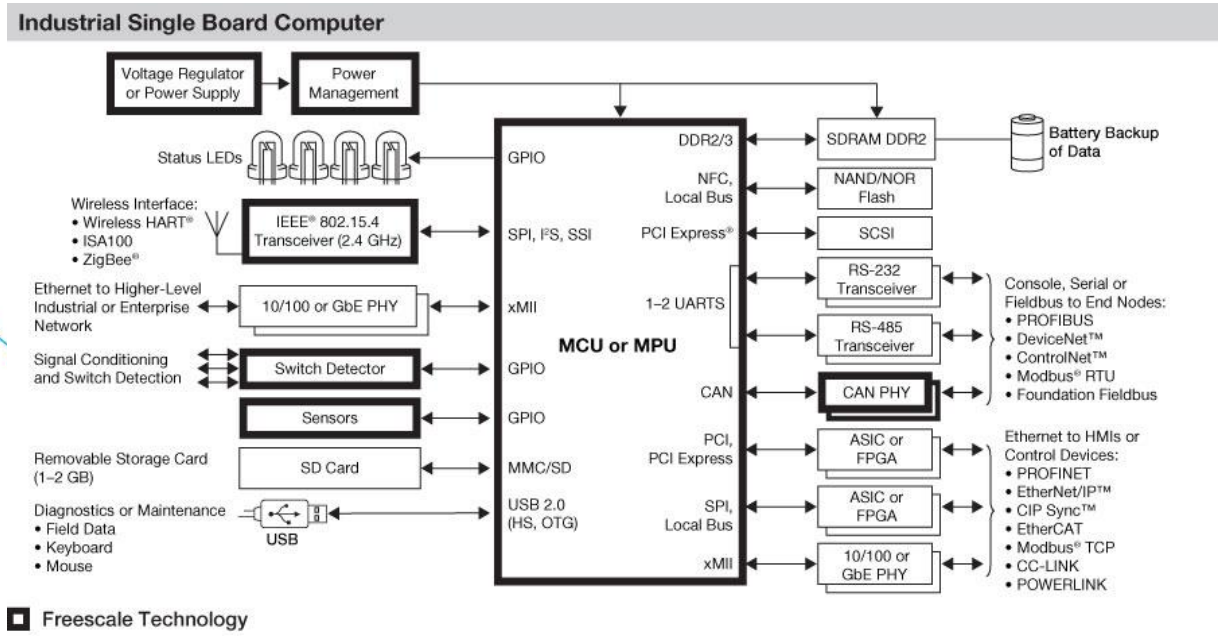
Arhitektura sistema se odnosi na identifikaciju fizičkih komponenti u sistemu i na njihovu međusobnu povezanost. Omogućava hardverskim inženjerma da razumeju kako se komponente uklapaju u sistem ali i da obezbedi softverskim inženjerima važne informacije potrebne za razvoj i integraciju softvera.

Jasna definicija hardverske arhitekture omogućava da više timova različitih specijalnosti (inženjerima elektronike, mehanike, softvera) efikasno rade zajedno na razvoju i proizvodnji novih mašina, uredjaja i komonenti.

Hardverska arhitektura se primarno odnosi na interne električne, a ponekad i mehaničke, interfejse između sistemskih komponenti ili podsistema, i interfejsa između sistema i spoljašnje sredine.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 6. Arhitektura sistema - blok diagram.**

### 3.1 Izbor komponenti

Najznačajnije komponentu u sistemu je potrebno odabrati vec prilikom definisanja arhitekture sistema. Komponente se najpre biraju po svojoj funkciji ali pored funcije postoje i mnogi drugi faktori koji određuju koja će se komponenta zapravo koristiti.

- Funkcija komponente
- Pretraživanje
- Sajtovi za pretrazivanje i nabavku (Distributeri)
    - Digikey, Mouser, Arrow
- Raspoloživost.
- Velicina ploce, koliko sitne komponente treba birati?
- Prilagoditi komponente potrebama dizajna.
- Cena!



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 4 Kumunikacijoni ineterfejsi – osnove

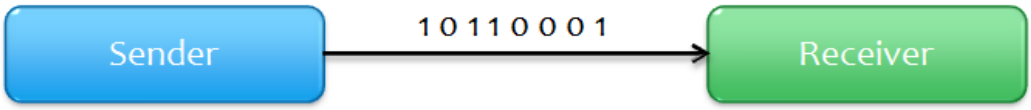
Komunikacija je proces razmene informacija preko dogovorenog sistema znakova. Kumunikacija izmedju dva čipa (Mikrokontrolera, temp. senzora...) je proces razmene bitova po unapred definisanom protokolu.

Osnovna podela:

- Serijska komunikacija
- Paralelna komunikacija

## 4.1 Serijska komunikacija

Proces slanja/primanja jedog bita podatka u jednom vremenskom slotu.



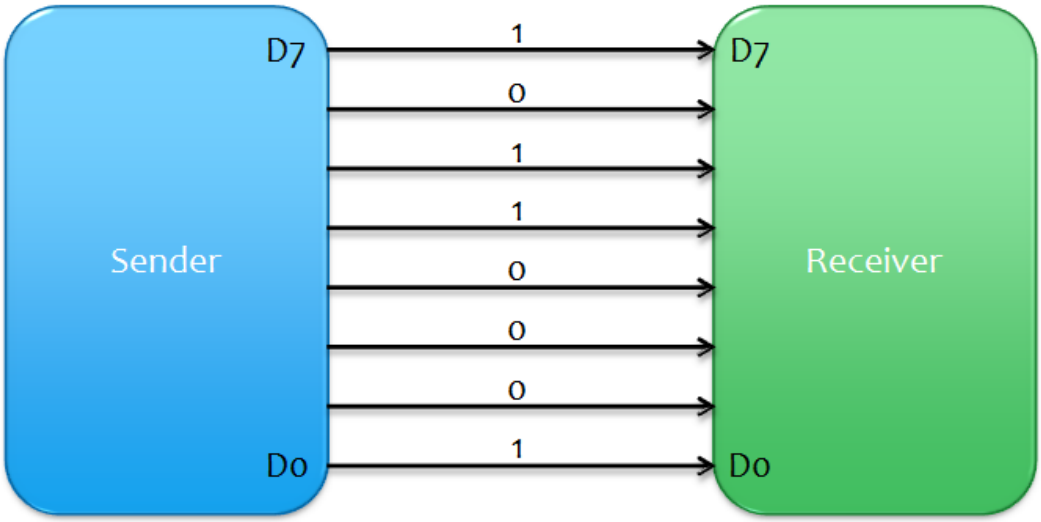

**Slika 7. Serijska komunikacija.**

## 4.2 Paralelna komunikacija

Proces slanja/primanja više bitova podataka u jednom vremenskom slotu preko više paralelnih kanala.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 8. Paralelna komunikacija.**

### 4.3 Poredjenje serijske i pralelene komunikacije

| **Serial Communication** | **Parallel Communication** |
|---|---|
| 1. Jedan bit podatka je poslat/primljen u jedinici vremena | 1. Više bitova podatka je poslato/primljeno u jedinici vremena |
| 2. Sporija | 2. Brža |
| 3. Manji broji provodnika je potreban za prenos podataka. | 3 Veći broji provodnika je potreban za prenos podataka. |

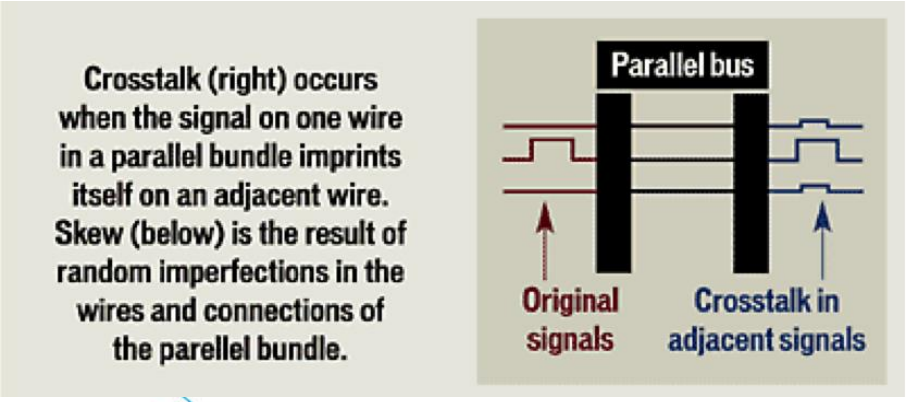
Paralelna komunikacija izgleda bolja od serijske? Da bismo mogli da odgovorimo moramo se upoznati sa osnovnom treminologijom:

- **Bit Rate:** Broj bitova koji se pošalje/primi po jedinici vremena.
- **Clock Skew:** Vremenaka razlika dolaska signala do dva prijemnika. U paralalnim magistralama to je neizbežno zbog postojanja različitih fizičkih uslova kao sto su temperatura, otpornost, dužina putanje signala...

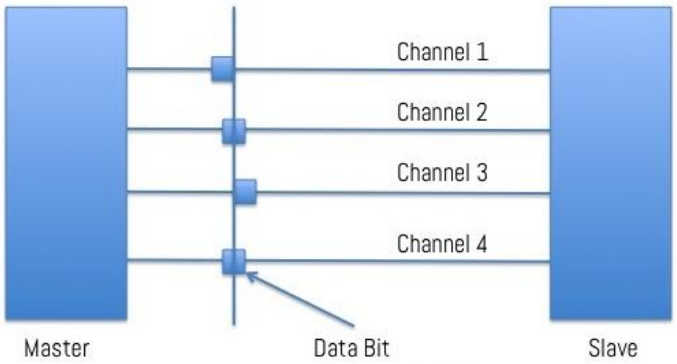


Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- **Crosstalk:** Fenomen u kome signal koji se šalje jednim kanalom pravi neželjene efekte na drugom kanalu. Preko neželjenih kapacitivnosti i induktivnosti između fizički bliskih signala dolazi do prenosa dela energije sa jednog signala na drugi.




**Slika 9. Crosstalk efekat.**




**Slika 10. Clock skew efekat.**

Glavni razlozi koji ograničavaju upotrebu paralelne komunikacije:

- **Brzina**: Teoretski, brzina paralelne veze je jednaka *bit rate*broja kanala.* Praktično, *clock skew* ekefat smanjuje brzinu svih kanala na brzinu najsporijeg.

- **Dužina kabla**: *Crosstalk* efekat stvara smetnje između paralelenih kanala koje postaju sve veće sa povećanjem dužine kabla. To ograničava moguću dužinu kabla.

Prednosti serijske u poređenju sa pralelnom komunikacijom:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Iako se serijaka komunikacija čini lošijom od pralelne, obzirom da može da šalje manje podataka po taktu, često je slučaj da serijska komunikacija može imati mnogo brži takt od paralelne i time postići veću brzinu prenosa.

Razlozi:

- Clock skew: Lakše se postiže izjednačavanje dužina jer nema puno signala, samim tim nema problema uzrokovanih clock skew efektom.
- Crosstalk: Obzirom da u seriskoj vezi ima samo nekoliko signala i samim tim mnogo više mesta za izolaciju, blizina provodnika više nije problem.

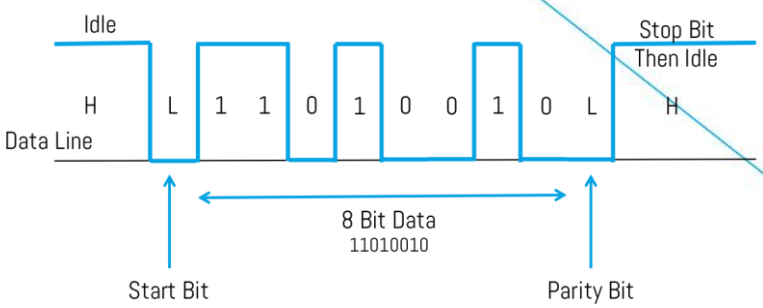
Serijska magistrala je jeftinija od paralelene jer zahteva manje pinova na čipovima (jeftinije pakovanje), manje kablova potrebno (cena bakra?).

Zbog svih ovih faktora seriska komunikacija se danas preferira u odnosu na paralelnu.

### 4.4 Modovi serijske komunikacije

**Asinhroni**

Prenos podataka se naziva asinhronim kada bitovi podataka nisu sinhronizovani sa taktom, odnosno signal takta uopste i ne postoji.



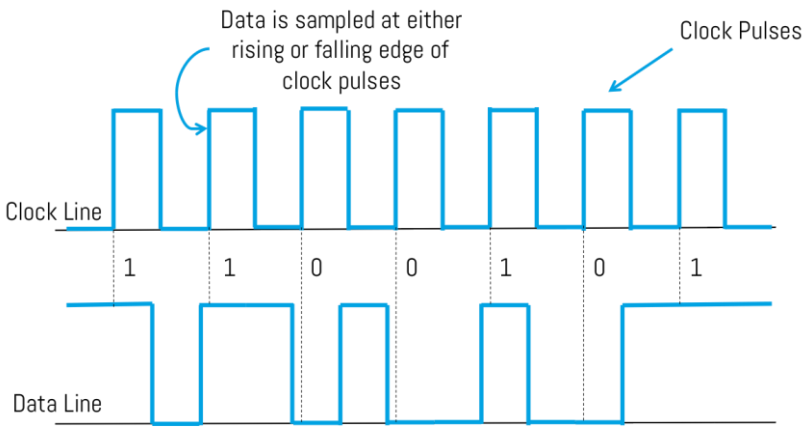

**Slika 11. Asinhroni prenos podataka.**

**Sinhroni**

Prenos podata se naziva sinhronim kada su bitovi podataka sinhronizovani sa signalom takta.



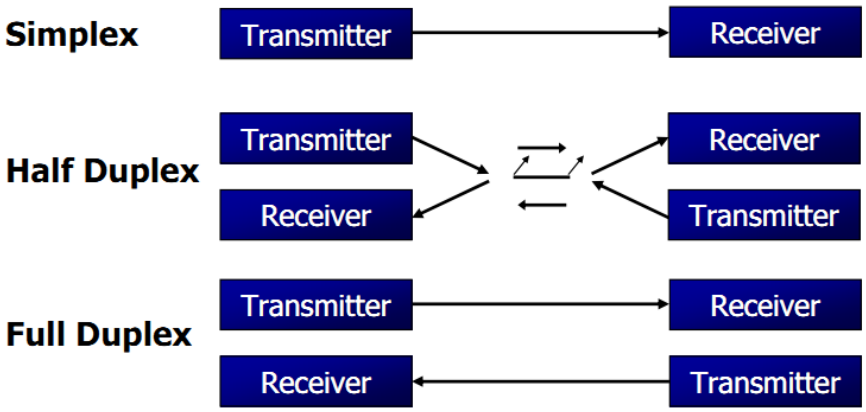
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 12. Sinhroni prenos podataka.**

### 4.4.1 Podela po smeru komunijkacije

- **Simplex:** Podaci mogu da se šalju samo jednosmero od predajnika do prijemnika, obrnuto nije moguće.
- **Half Duplex:** Znači da podaci mogu da se šalju u oba smera ali ne istovremeno.
- **Full Duplex:** Komunikacija je moguća u oba smera istovremeno.



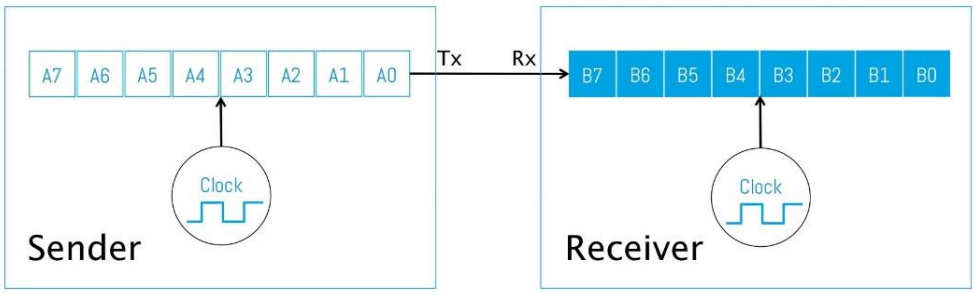

**Slika 13. Podela po smeru komunikacije.**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

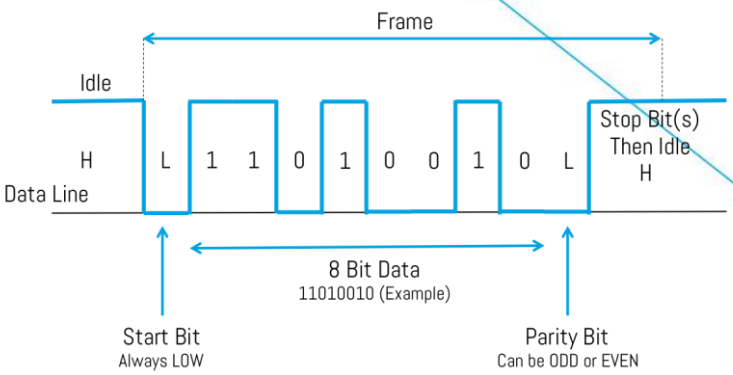
# 5 UART (Universal Asynchronous Receiver Transmitter)

UART je jednostavan, full-duplex, asinhroni serijski protokol. Obezbeđuje jednostavnu komunikaciju između dva identična uređaja. Bilo koji od uređaja može da inicira komunikaciju. Svaki uređaj sadrži prijemnik i predajnik u sebi.




**Slika 14. UART predajnik i prijemnik.**

Tipični format podataka za UART/RS232 je uglavnom 10 bita dugačak: 1 Start bit, 8 bitova podataka i Stop bit. Može imati i Parity bit.




**Slika 15. UART protokol.**

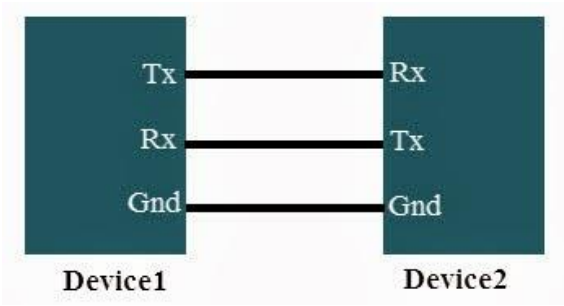
Kod UART komunikacionog interfejsa uređaji su povezani preko dve signalne žice.

- **RX - Recieved data**: Ulaz koji prima podatke, odnosno povezan je na ulaz prijemnog shift registra.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- **TX – Transmitted data:** Izlaz koji šalje podatke, odnosno povezan je na izlaz predajnog shift registra.



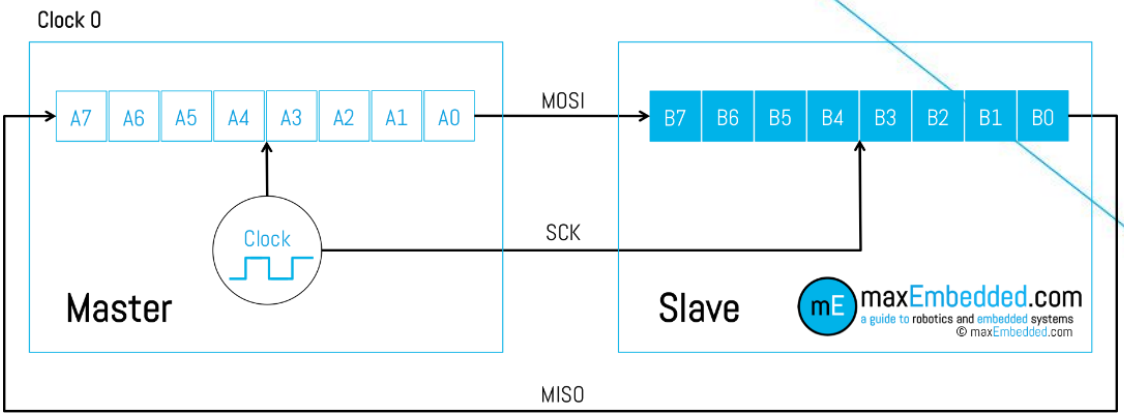

**Slika 16. Povezivanje UART uređaja.**

# 6 SPI (Serial Peripheral Interface)

Serial Peripheral Interface ili skraćeno SPI, je sinhroni seriski protokol za prenos podataka nastao u kompaniji Motorola. Služi za povezivanje dva ili više seriskih uređaja u full-duplex modu. Ovi uređaji mogu biti glavni uređaj (Master) ili sporedni uređaj (Slave).

U SPI vezi može biti više glavnih uređaja i više sporednih uređaja ali je veoma retko da se u sistemu nađe više od jednog glavnog uređaja. Glavni uređaj inicira i kontroliše konekciju. Kada je konekcija inicirana, glavni uređaj i jedan ili više sporednih uređaja mogu da razmenjuju podatke.

Rad SPI interfejsa se bazira na pomeračkim registrima (shift registers). Svaki uređaj, bilo da je Master ili Slave, ima 8-bitni (10-bitni, 12-bitni) pomerački registar u sebi.




**Slika 17. SPI Master i Slave.**



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Izlaz Master shift registra je povezan na ulaz Slave shift registra, i izlaz Slave shift registra je povezan na ulaz Master shift registra. Ova konfiguracija čini circular/ring buffer.

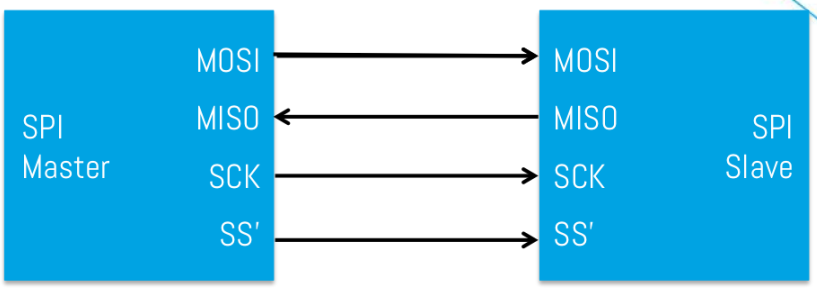
Kao što je već pomenuto, SPI je sinhrona serijska magistrala, što znači da mora postojati takt za prenos podataka. Takođe je rečeno da je Master zadužen za iniciranje i upravljanje vezom, što znači da se koristi Master-ov izvor takta za sinhronizaciju prenosa podataka. Zato se izvor kloka nalazi u Master uređaju koji kontroliše rad oba shift registra.

Da bi se komunikacija odvila potreno je da se uradi sledeće:

- Maste i Slave moraju da postave podatke koje žele da pošalju u svoje shift registre pre nego što komunikacija počne.
- Maste generiše 8 taktnih impulsa. Posle svakog impulsa, jedan bit podataka je prenešen iz Mastera u Slave i obrnuto.
- Posle 8 taktnih impulsa Master je primo sve podatke od Slave-a i Slave je primio sve podatke Master-a. Zbog toga je to full-duplex komunikacija.

Maste i Slave uređaji su povezani sa četiri žice. Svaka od ovih žica nosi određeni signal definisan SPI protokolom:

- **MOSI – Master Out Slave In:** Ovo je žica/signal koji ide od izlaza Master shift regista do ulaza Slave shift registra.
- **MISO – Master In Slave Out:** Ovo je žica/signala koji ide od izlaza Slave shift registra do ulaza Master shift registra.
- **SCK/SCLK – Serial Clock:** Ovo je izlaz iz generatora takta Master-a i ulaz za Slave uređaj.
- **SS – Slave Select:** Signal kojim se vrši selekcija odgovarajućeg Slave uređaja.



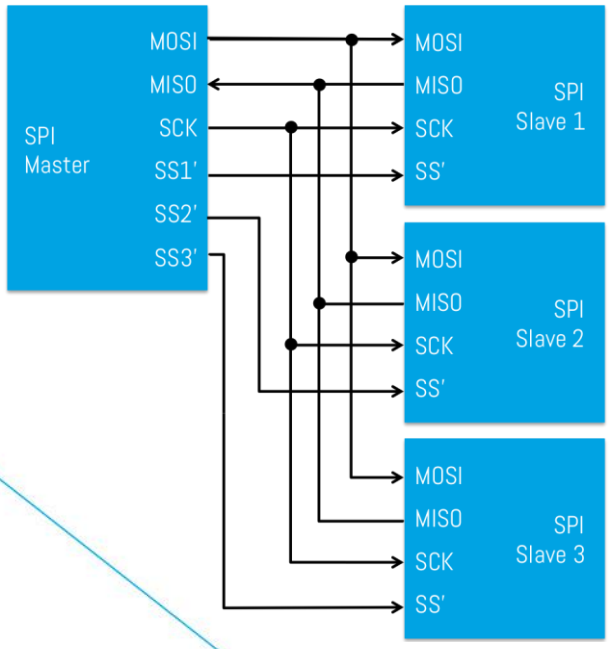

**Slika 18. Povezivanje dva SPI uređaja.**

Kao što je ranije pomenuto, SPI se može koristiti u konfiguraciju u kojoj je povezan jedan Master uređaj i više Slave uređaja na istoj magistrali. U tom slučaju se koristi SS (Slave Select) signal.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

SS signal je aktivan na niskom logičkom nivou odnosno da bi se selektovao određeni Slave uređaj potrebno je da se na njegov SS ulaz dovede niski naponski nivo (0V).




**Slika 19. Povezivanje više SPI uređaja.**

U konfiguraciji sa više Slave uređaja svi Slave uređaji dele iste MOSI, MISO i SCK signala. SS signal je zaduzen za izbor odgovarajućeg Slave uređaja.

Postoji mogućnost i povezivanja više Slave uređaja u lanac (dasy chain) kada je izlaz Matera vezan na ulaz prvog Slave-a, njegov izlaz za ulaz drugog Slave-a i tako dalje. Izlaz poslednjeg Slave-a u lancu vezan je za ulaz Master-a. Mada se ova konfiguracija retko koristi.

Imajući sinhronizaciu u vidu, uloga Master-a nije samo da generiše takt na određenoj frekvenciji (koja je obično u okviru od 10 kHz do 100 MHz). Zapravo Master i Slave takođe moraju da se slože oko određenog protokola ili će komunikacija teći pogresno i svi podaci će biti izgubljeni. Ovde nastupa koncept polariteta takta (CPOL - Clock Polarity) i faze takta (CPHA – Clock Phase).

CPOL – Clock Polarity: Određuje osnovnu vrednost taktnog signala, odnosno nivo taktnog signala kada je SPI u neaktivnom stanju:

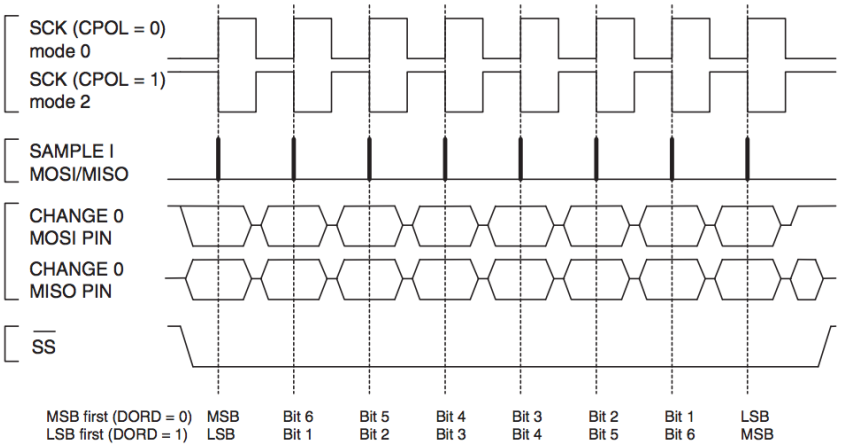
- Kada je CPOL = 0, vrednost takta je “0” odnosno SCK je na niskom logičkom nivou u nekativnom stanju.
- Kada je CPOL = 1, vrednost takta je “1” odnosno SCK je na visokom logičkom nivou u nekativnom stanju.



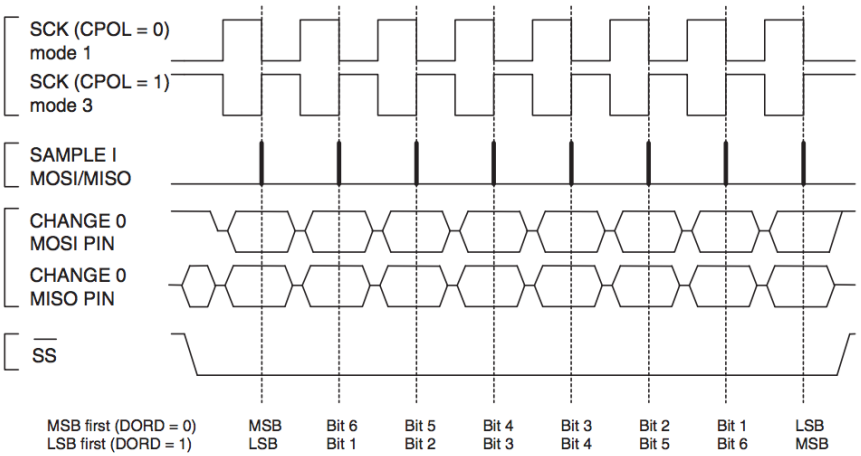
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

CPHA – Clock Phase: Određuje ivicu takta pri kojoj će se izvršiti odabir podatka.

- Kada je CPHA = 0, podatak se odabira na rastuću/vodecu (rising/leading) ivicu signala.
- Kada je CPHA = 1, podatak se odabira na opadajuću/prateću (falling/trailing) ivicu signala.




**Slika 20. SPI format prenosa podataka kada je CHPA = 0.**




**Slika 21. SPI format prenosa podataka kada je CHPA = 1.**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

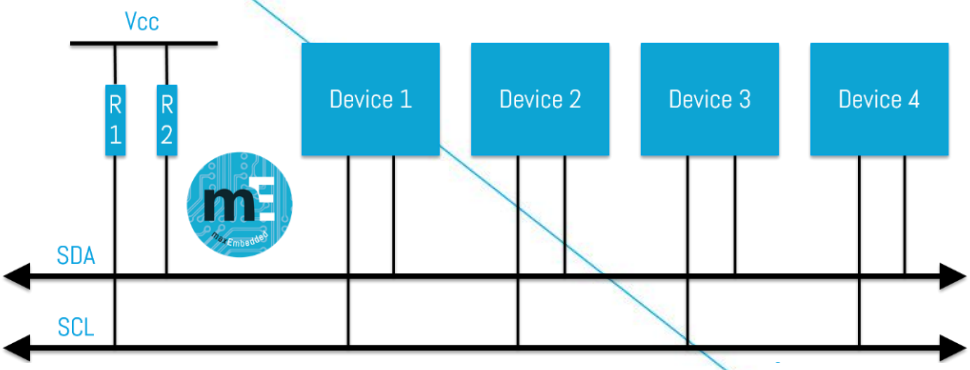
# 7 I²C (Inter-Integrated Circuit)

Kao što samo ime kaže, Inter-IC (ili Inter-Integrated Circuit) , skraćeno I2C, je bio razvijen kao komunikacioni protokol za interakciju različitih IC na matičnim pločama, kao jednostavna interna magistrala.

To je bila revulocionarna tehnologija razvijena od strane kompanije Philips Semiconductor 1982. godine, i koristila se za povezivanje sporih preriferija (kao što su tastatura, miš, memorije, IO/serijski/paralelni portovi, ...) sa matičnom pločnom na kojoj se nalazi procesor koji radi na mnogo većoj brzini.

Danas se može naći mnogo uređaja koji su kompatibilni sa I2C protokolom koje proizvode razne kompanije (Intel, TI, Freescale, STMicroelectronics, ...).

Jedna od najprivlačnijih stvari kod I2C-a je što se uređaji povezuju samo sa dva pina. Naravno potrebno je jos dva za Vcc i Gnd ali se to podrazumeva.




**Slika 22. I2C interfejs.**

Svi uređaji su prikačeni na istu I2C magistralu samo sa dva pina. Ti uređaji mogu da budu CPU, IO uređaji, ADC, ili bilo koji drugi uređaj koji podržava I2C protokol. Uređaji prikačeni na magistralu su ili Master ili Slave uređaji (isto kao kod SPI). I2C magistrala se satoji od dva bidirekciona "open-drain" signala – SDA i SCL – povezana na VCC preko otpornika (pull up).

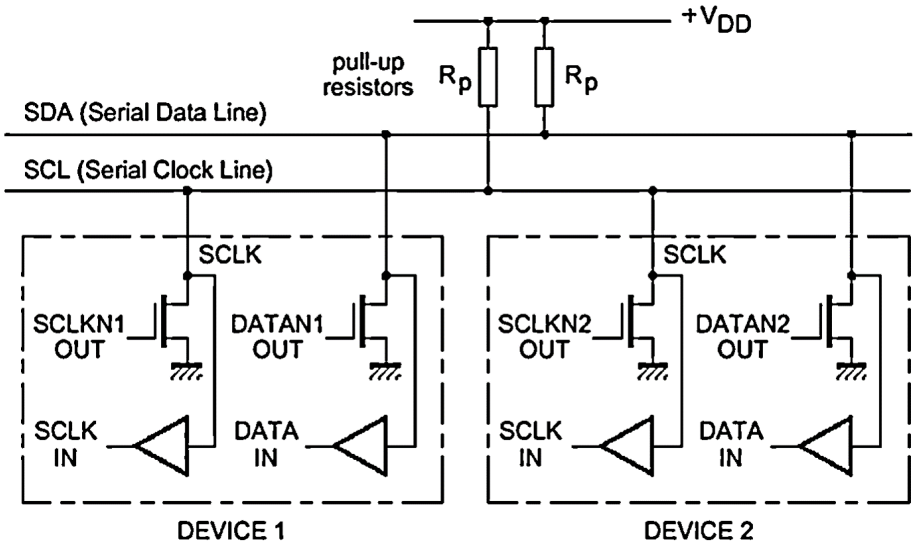
- **Serial Data Line (SDA)** je linija preko koje se šalju podaci (kao što ime kaže). Svi podaci koji se razmenjuju idu preko ove linije.
- **Serial Clock Line (SCL)** je linija takta. I2C je sinhroni protokol, stoga, SCL služi za sinhronizaciju.

Šta znači da su linije open drain? To znači da uređaji prikačeni na I2C magistralu mogu da obore bilo koju od ove dve linije na nizak naponski nivo, ali ne mogu da ih podignu na viskok naponski nivo. Ako uređaj želi da liniju podigne na visok naponski nivo potrebno je da jednostavno otpusti tu



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

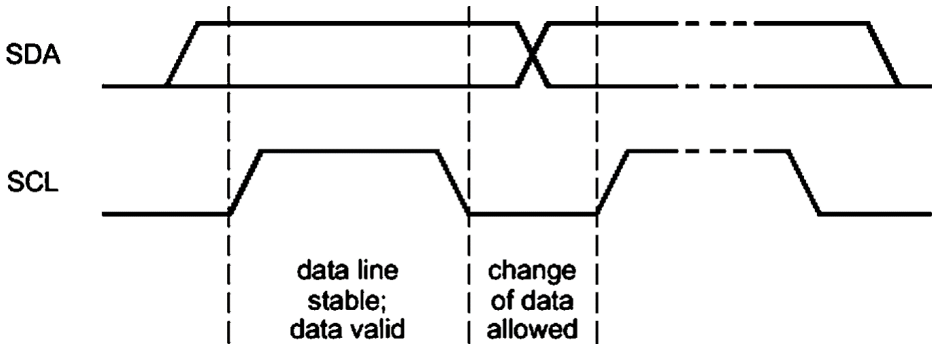
liniju, i pull up otpornik će je podići na visok naponski nivu. Da bi uređaj mogao da obori naponski nivo potrebno je da na Gate primeni visok naponski nivo. Ako je napon na Gate-u nizak linija će biti na visokom naponskom nivou.




**Slika 23. I2C Principska šema.**

Validacija podataka:

- Da bi podatak bio validan na SDA liniji, ona mora ostati nepromenjena dok je SCL na visokom naponskom nivou. Podaci na SDA liniji smeju da se menjaju samo dok je SCL linija na niskom naponskom nivou.




**Slika 24. Validacija podataka.**

**Naponski nivoi i vrednosti otpornika:**

I2C podržava širok opseg naponskih nivoa, tako da se lako mogu primeniti naponi od 5 volti, ili 3.3 volti, kao i drugi viši ili niži naponi.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Vrednosti otpornika, zavise od napona koji se primeni i od broja uređaja koji se nalaze na magistrali, ali se kreću u rasponu od 1k do 47k. Vrednosti manje od 10k Ohm-a se uglavnom preferiraju.

**I$^2$C Master i Slave:**

Koncept Mastera i Slava je sličan kao i kod SPI protokola. Svi uređaji na magistrali su ili Master ili Slave uređaji. Master uređaji su oni koji iniciraju transfer i upravljaju taktom SCL.

Na jednoj I2C magistrali tipično ima više Slave uređaja i jedan Master. Ali takođe, kao i kod SPI, može biti više Mastera zakačenih na istoj I2C magistrali. Ovakvi slučajevi se uglavnom izbegavaju, ali I2C podržava detekciju sudara i protokol za arbitraciju kada ima više Master uređaja.

**I$^2$C Brzina**

I2C podržava seriski 8-bitni dvosmerni prenost podataka brzinom do 100 kbps, koja je ujedno i standardna brzina takta SCL.

I2C može da radi i na većim brzinama – Fast Mode (400 kbps) i High Speed Mode (3.4 Mbps). Ipak, većina uređaja je napravljena da radi na brzinama od 100 kbps (ovo je ipak magistrala napravljena za spore uređaje).

**I$^2$C Smer komunikacije**

Jednosmerni prenos podataka:

- Master-transmitter → Slave-receiver
- Slave-transmitter → Master-receiver

Dvosmerni prenos podataka

- Master → Slave i Slave → Master

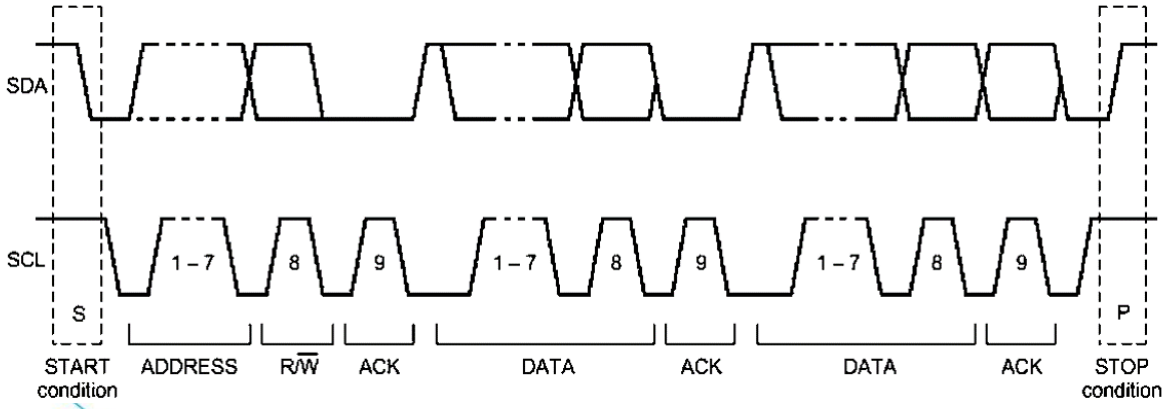
**I$^2$C Start/Stop sekvenca – protokol:**

Da bi Master otpočeo komunikaciju sa Salve, on mora da obavesti Slave o tome. Ove se postiže specijalnom Start sekvencom. Ranije je bilo pomenuto da SDA ne sme da menja stanje dok je SCL na visokm naponskom nivou. To ne važi za Start i Stop sekvencu, zato one i jesu specijalne.

- Kada je SCL na visokom naponskom nivou SDA prelazi iz visokog naponskog nivoa u niski, i time označava početak komunikacije Mastera sa Slave-om/ovima.
- Kada SDA prelazi sa niskog naponskog nivoa na visoki dok je SCL na visokom naponskom nivou, to označava kraj komunikacije Mastera sa Slave-om/ovima (Stop).



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 25. I2C Start i Stop sekvenca.**

Komunikacija se odvija na sledeći način:

- Komunikacija počinje Start sekvencom.
- Zatim Master šalje adresu Slave sa kojim želi da razgovara. To je 7-bitna adresa praćena R/W bitom koji određuje da li želimo da pišemo u ili čitamo iz uređaja.
- Slave se javlja sa ACK (acknowledge) bitom potvrđujući adresu.
- Kada je Slave adresiran, to znači da je spreman za slanje/primanje podataka od Mastera. Tada počinje prenos. Podaci su uvek 8-bitni, i prijemnik mora da pošalje ACK signal posle svakog bajta podataka.
- Kada je prenos potataka završen Master završava komunikaciju STOP sekvencom. Alternativno, Master može da započne novu komunikaciju ponavljanjem START sekvence.

**Zašto I2C?**

Zahteva samo dva pina za serijsku komunikaciju. Prijemnik uvek šalje povratnu informaciju predajniku (ACK) o uspešnosti prenosa, što vodi visokoj otpornosti na smetnje. Iako ima malu brzinu od 100 kHz, moderna I2C specifikacija podržava brzine i do 3.4 MHz.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

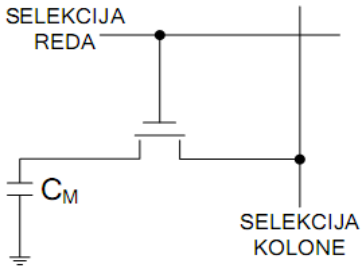
# 8 DDR SDRAM memorije

## 8.1 Trendovi u razvoju DRAM memorija

U savremenim računarskim sistemima, postoji stalni zahtev za povećanjem brzine i kapaciteta računarskih memorija, kao i za smanjenjem fizičkih dimenzija i snage koju memorije troše. Ovi zahtevi i potrebe predstavljaju vodilje u razvoju DRAM tehnologije. Većina DRAM memorija su se razvijale godinama i pratile razne tehnološke napretke, počev od

- SDRAM (Sinhroni DRAM),
- DDR (dvostruka brzina podataka) SDRAM,
- DDR2 SDRAM,
- DDR3 SDRAM…

Ćelija u SRAM memoriji se sastoji od najmanje 4 do 6 tranzistora. Sa DRAM memorijama je moguće postići veću gustinu obzirom da je to memorija sa samo jednim tranzistorom i jednim kondenzatorom po memorijskoj ćeliji.




**Slika 26. Ćelija DRAM memorije.**

Ovakva memorija bazira pamćenje informacije na električnom punjenju kondenzatora.

**Računarske memorije**

- Zahtevaju veće, brže i fizički manje memorije sa manjom potrošnjom.
- Zahtevaju viši nivo kompatibilnosti sa trenutnim i narednim stepenom razvoja računara.
- Memorija mora biti kompatibilna sa velikim brojem memorijskih kontrolera korišćenim od strane raznih proizvođača.
- Memorija mora biti u stanju da radi i kada se koristi zajedno sa memorijama različitih proizvođača u jednom memorijskom sistemu računara.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**Memorije u namenskim (Embedded) sistemima**

- Obično koristie fiksnu memorijsku konfiguraciju, što znači da korisnik ne može menjati memorijski sistem nakon kupovine proizvoda.
- Proizvođač ovakvih namenskih sistema, dakle, poseduje potpunu kontrolu nad korišćejnem memorija određenih proizvođača u namenskom sistemu.
- Uobičajena je optimizacija performansi i cene namenskih sistema korišćenjem jedne specifične vrste memorija jednog proizvođača.
- Kao rezultat imamo da je u namenskim sistemima, u poređenju sa računaskim, manje bitno da se poseduje visok nivo kompatibilnosti memorija različitih proizvođača.

**Standard**

JEDEC (zajednički odbor inženjera elektronskih uređaja) je pomogao memorijskoj industriji stvaranjem memorijskih specifikacija u obliku JEDEC standarda. JEDEC je neprofitabilna organizacija čiji su članovi ustvari proizvođači memorija, računara, oprema za testiranje, itd.

Otvoreni JEDEC standardi definišu zahtevanu specifikaciju koju proizvođači moraju primenjivati pri proizvodnji svojih memorija kako bi one bile u stanju da rade sa memorijama drugih proizvođača, kao i drugim memorijskim kontrolerima.

## 8.2 DRAM

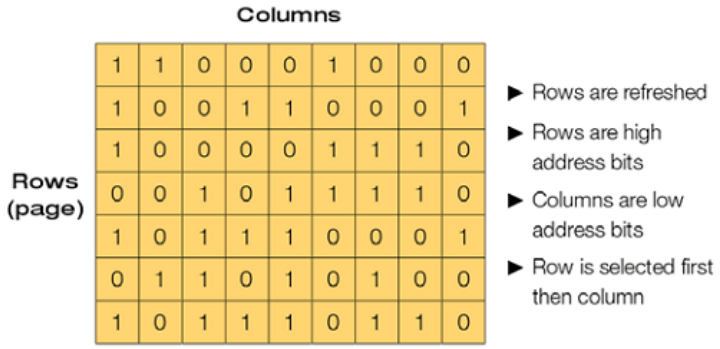
Prednost DRAM-a nad ostalim tipovima memorija je njena mogućnost da se primenjuje sa manjim brojem tranzistora po memorijskoj ćeliji na IC (integrisanom kolu). Memorijske ćelije DRAM-a su zasnovane na smeštanju naelektrisanja na kondenzatoru. Tipična DRAM ćelija je izgrađena uz pomoć jednog kondenzatora, i jednog, ili tri, MOS tranzistora.

Tipična SRAM (statička memorija sa direktnim pristupom) memorijska ćelija koristi šest MOS-ova. SRAM su jednostavnije za korišćenje, lakše za povezivanje, i poseduju kraće vreme pristupa podacima od DRAM memorija.

Arhitektura DRAM jezgra se sastoji od memorijskih ćelija organizovanih u dvodimenzionalne nizove redova i kolona.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 27. Arhitektura DRAM memorije.**

Pristup određenoj memorijskoj ćeliji zahteva dva koraka. Prvo se mora adresirati željeni red, a zatim i željena kolona u tom redu. Drugačije rečeno, prvo se vrši interno čitanje čitavog reda u DRAM integrisanom kolu, a zatim adresa kolone bira koja će se kolona u kolu DRAM pročitati, ili upisati, preko ulazno/izlaznih pinova DRAM-a.

Očitanja DRAM-a su destruktivna, što podrazumeva da se podaci u nizu memorijskih ćelija uništavaju prilikom operacije čitanja.

Neophodno ponovo zapisati isti red podataka nakon okončanja operacije upisivanja, ili čitanja, iz tog reda. Ova operacija se zove *Precharge*, i to je poslednja operacija u jednom redu. Ona se mora obaviti pre pristupanja novom redu i zbog toga se naziva i "zatvaranje otvorenog reda".

DRAM memorijske ćelije se moraju osvežavati kako bi se izbeglo gubljenje određenih podataka. To podrazumeva osvežavanje kondenzatora pre nego što on izgubi svoje naelektrisanje.

Prvi DRAM-ovi su posedovali kontrolne signale poput RAS# (selektovanje adrese reda - nizak aktivan nivo) i CAS# (selektovanje adrese kolone - nizak aktivan nivo) za odabir reda i kolone prilikom izvršavanja adresne operacije.

Dodatni DRAM kontrolni signali uključuju WE# (nizak aktivni nivo za omogućavanje upisa) za odabir operacija upisivanja ili čitanja, CS# (nizak aktivni nivo za biranje čipa) za odabir DRAM, i OE# (nizak aktivni nivo za omogućavanje izlaza).

Prvi tipovi DRAM su imali asinhrone kontrolne signale i različite specifikacije tajminga koje su pokrivale razne sekvence i vremenske trenutke za određivanje načina rada DRAM-a.

### 8.3 SDRAM

SDRAM (sinhroni DRAM) je dizajniran da sinhronizuje DRAM operacije sa ostatkom računarskog sistema. SDRAM je dodao signal takta i koncept memorijskih komandi. Tip memorijske komande je određen stanjem CE#, RAS#, CAS# i WE# signala na rastućoj ivici SDRAM taktnog signala.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

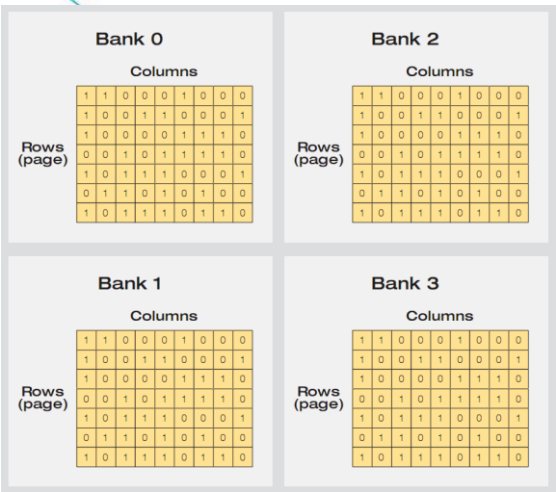
### 8.4 DDR SDRAM

DDR (dvostruka brzina podataka) SDRAM je povećao brzinu i performanse povećanjem taktne brzine, u režimu sa paketnim prenosom i prenosom dva bita podataka u jednom taktnom ciklusu. DDR SDRAM prenosi u paketu (eng. *Burst*) više memorijskih lokacija u jednoj komandi čitanja, ili upisivanja.

| DDR SDRAM | Data Rate | Memory Clock |
|---|---|---|
| DDR-266 | 266 Mb/s/pin | 133 MHz |
| DDR-333 | 333 Mb/s/pin | 166 MHz |
| DDR-400 | 400 Mb/s/pin | 200 MHz |

**Slika 28. DDR SDRAM brzine podataka i taktne brzine.**

DDR SDRAM poseduju više grupa (Bank) kako bi omogućili višestruki pristup memoriji, što povećava ukupni propusni opseg.




**Slika 29. Memorijske banke.**

Jedna grupa predstavlja jedan niz u memoriji, dve grupe su dva niza, itd. Četiri grupe zahtevaju dva bita za adrese grupa (BA0 i BA1).

Snaga koja je potrebna za DDR SDRAM povezana je sa brojem grupa sa otvorenim redovima. Više otvorenih redova zahtevaju više snage, a i veći redovi takođe.

### 8.5 DDR2 SDRAM

Nekoliko unapređenja u odnosu na DDR SDRAM.

- Taktne brzine



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- DDR2 SDRAM poboljšava integritet signala podataka omogućavanjem ODT (eng. *On Die Termination*) opcije.
- Niži radni napon. Potrebna manja snaga. DDR2 radi na 1.8V, što je 72% od napona za DDR SDRAM (2.5V).
- Poboljšava propusni opseg za module od 1Gb i 2Gb korišćenjem osam grupa.

| DDR2 SDRAM | Data Rate | Memory Clock |
|---|---|---|
| DDR2-400 | 400 Mb/s/pin | 200 MHz |
| DDR2-533 | 533 Mb/s/pin | 266 MHz |
| DDR2-667 | 667 Mb/s/pin | 333 MHz |
| DDR2-800 | 800 Mb/s/pin | 400 MHz |
| DDR2-1066 | 1066 Mb/s/pin | 533 MHz |

**Slika 30. DDR2 SDRAM brzine podataka i taktne brzine.**

### 8.6 DDR3 SDRAM

Taktne brzine počinju od 800 Mb/s, što je najveća brzina podržana kod DDR2 SDRAM memorija. Napon napajanja DDR3 SDRAM memorije 1.5V, što je 83% od 1.8V koje koristi DDR2 SDRAM memorija.

Drajveri podataka DDR3 SDRAM memorija imaju višu impedansu od 34 Ohm, dok drajveri DDR2 SDRAM memorija imaju nižu impedansu od 18 Ohm.

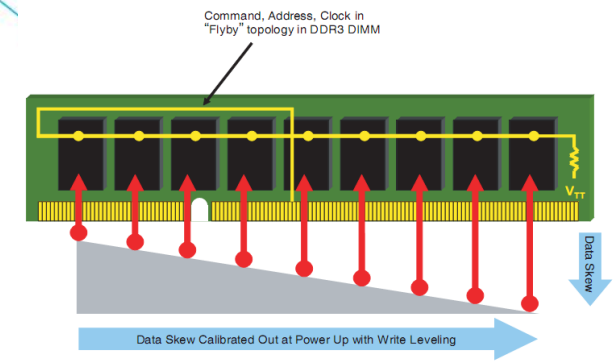
Kod DDR3 SDRAM su DQS (eng. *Date strobe*) linije diferencijalne, dok kod DDR2 mogu da budu i diferencijalne i single-ended.

Kod DDR3 se pojavljuje i novi asinhroni pin, RESET#, aktivan na niskom logičkom nivou, koji dodatno poboljšava stabilnost sistema tako što omogućava dovođenje SDRAM memorije u unapred definisano stanje, nezavisno od trenutnog stanja.

| DDR3 SDRAM | Data Rate | Memory Clock |
|---|---|---|
| DDR3-800 | 800 Mb/s/pin | 400 MHz |
| DDR3-1066 | 1066Mb/s/pin | 533 MHz |
| DDR3-1333 | 1333Mb/s/pin | 667 MHz |
| DDR3-1600 | 1600 Mb/s/pin | 800 MHz |
| DDR3-1866 | 1866Mb/s/pin | 933 MHz |
| DDR3-2133 | 2133Mb/s/pin | 1066 MHz |

**Slika 31. DDR3 SDRAM brzine podataka i taktne brzine.**

Nova *fly-by* topologija donosi novu karakteristiku *Write leveling* DDR3 SDRAM memorija.




**Slika 32. Fly-by topologija.**

### 8.7 Primer uptrebe DDR3 SDRAM memorije

**Procesor**

- DaVinci DM8168
- Podržava DDR2 i DDR3 SDRAM memorije kompatibilne sa JEDEC standardom koje imaju sledeće karakteristike:
- 16-bitnu ili 32-bitnu magistralu podatka.
- Memorije kapaciteta: 64MB, 128MB, 256MB, 512MB, 1GB i 2GB.
- Dve nezavisne linije za selekciju, zajedno sa odgovarajućim setovima registara, kao i nezavisno praćene stranica.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

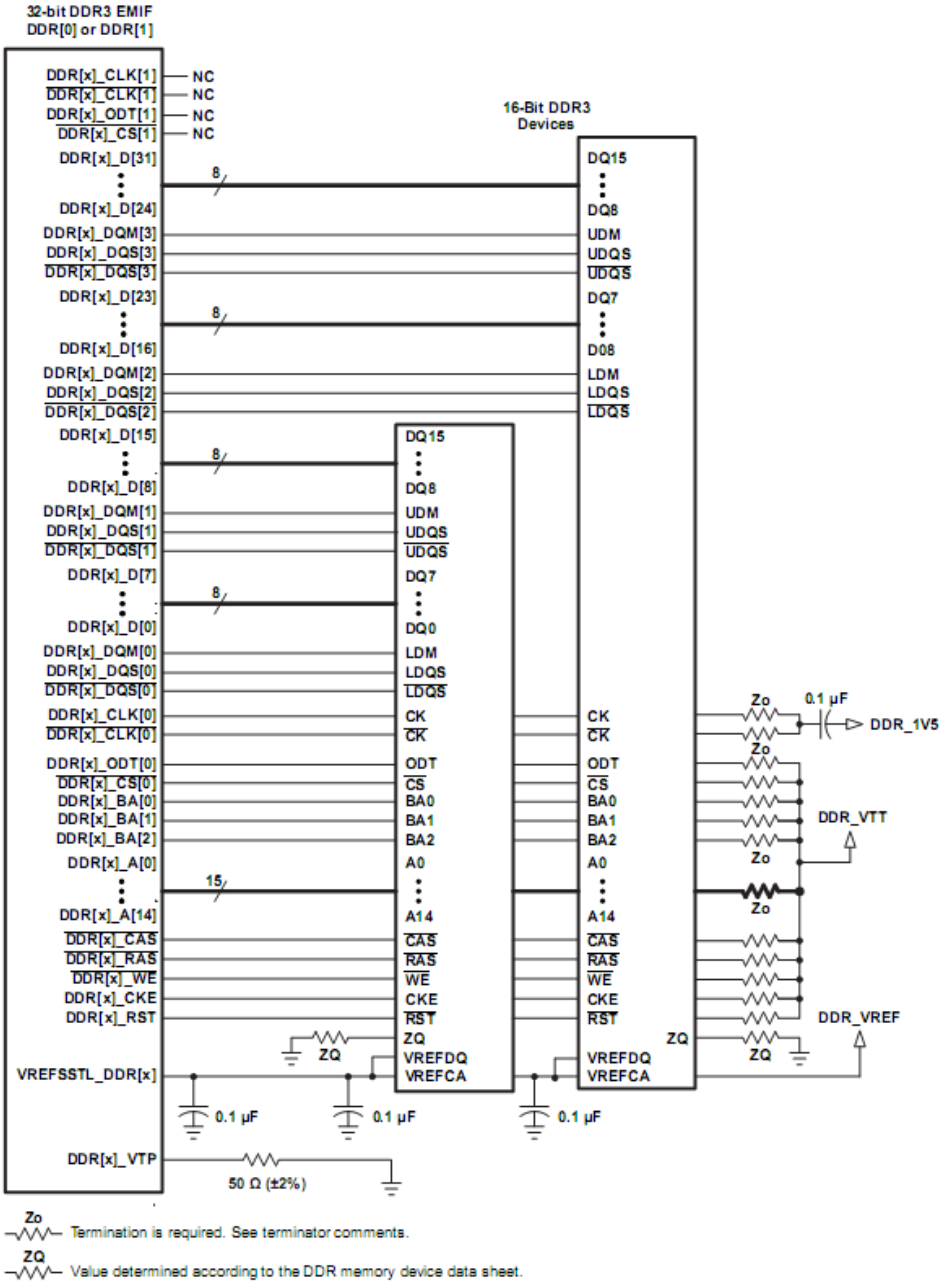
- Dva interfejsa sa pridruženim DDR2/3 fižičkim slojem.
- Omogućeno je i preklapanje (eng. *Interleaving*) podataka između dva memorijska interfejsa.

**Memorija**

- Micron MT41J128M16
- DDR3 SDRAM memorija kapaciteta 2Gb
- Širina magistrale podataka 16 bita.
- Napon napajanja memorije je 1.5V.
- Konfiguracija memorije je 16MB x 16 x 8 banki.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




Terminacija linija je potrebna za klase CK i ADDR_CTRL. Linije podataka su terminisane interno preko ODT, pa nema potrebe da se terminišu spolja. Razlika u dužini linija ADDR_CTRL i CK klase direktno smanjuje *setup i hold* margine, i zbog toga je jako bitno kontrolisati tu razliku kako bi kašnjenje među njima bilo minimalno.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 9 DVI Interfejs

Video interfejs DVI (Digital Visual Interface), povezivanje video izvora i displeja.

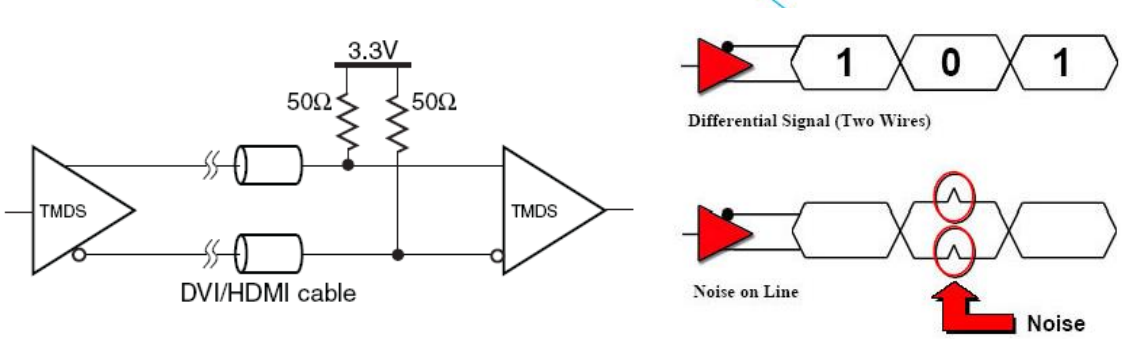
Prenosi nekompresovani digitalni video i podržava više modova:

- DVI-D (samo digitalni)
- DVI-A (samo analogni)
- DVI-I (digitalni i analogni)

DVI specifikacija je kompatibilana sa VGA interfejsom kad se koristi analogna konekcija.

Komunikacioni protokoli:

- Prenos videa je baziran na TMDS (transition minimized differential signaling).
- DDC (Display Data Channel)
  - Protokol za komunikaciju između displeja i grafičke jedinice.
- EDID (extended display identification data ).
  - Opisuje mogućnosti displeja grafičkoj jedinici.




**Slika 33. Diferencijalna komunikacija.**

**DVI Konektori**

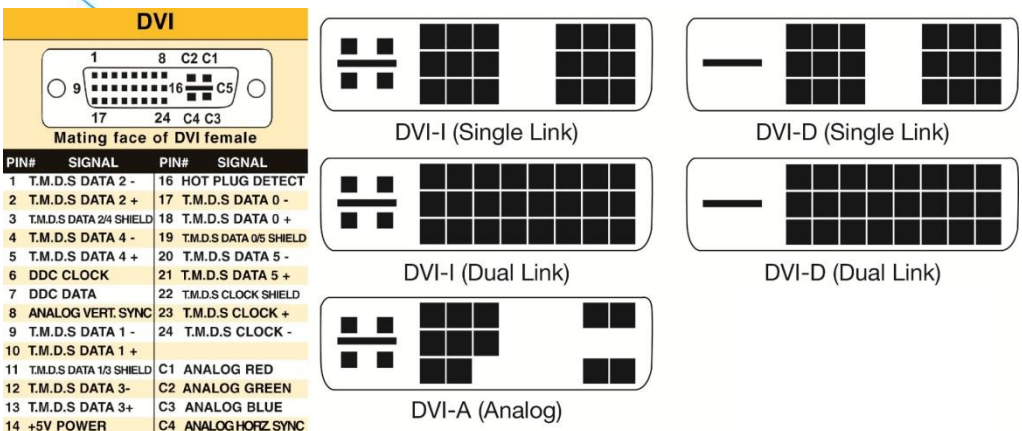
- *DVI-D* (samo digitalni, single link ili dual link)



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- *DVI-A* (samo analogni)
- *DVI-I* (intgegrisani, kombinuje digitalni i analogni u isti konektor; digitalni može biti single ili dual link)
- Single-link DVI ima jedan predajnik na 165 MHz koji podržava rezoluciju do 1920 × 1200 pri 60 Hz. Dual-link DVI dodaje 6 pinova za drugi predajnik podižući opseg i podržanu rezoluciju na 2560 × 1600 pri 60 Hz.

Single link se sastoji od četiri TMDS linka. Svaki prenosi podatke preko uvrnutih parica. Tri linka prenose RGB komponente. Ukupno 24 bita po pikselu.Četvrti link prenosi piksel takt.




| PIN# | SIGNAL | PIN# | SIGNAL |
|---|---|---|---|
| 1 | T.M.D.S DATA 2 - | 16 | HOT PLUG DETECT |
| 2 | T.M.D.S DATA 2 + | 17 | T.M.D.S DATA 0 - |
| 3 | T.M.D.S DATA 2/4 SHIELD | 18 | T.M.D.S DATA 0 + |
| 4 | T.M.D.S DATA 4 - | 19 | T.M.D.S DATA 0/5 SHIELD |
| 5 | T.M.D.S DATA 4 + | 20 | T.M.D.S DATA 5 - |
| 6 | DDC CLOCK | 21 | T.M.D.S DATA 5 + |
| 7 | DDC DATA | 22 | T.M.D.S CLOCK SHIELD |
| 8 | ANALOG VERT. SYNC | 23 | T.M.D.S CLOCK + |
| 9 | T.M.D.S DATA 1 - | 24 | T.M.D.S CLOCK - |
| 10 | T.M.D.S DATA 1 + | | |
| 11 | T.M.D.S DATA 1/3 SHIELD | C1 | ANALOG RED |
| 12 | T.M.D.S DATA 3- | C2 | ANALOG GREEN |
| 13 | T.M.D.S DATA 3+ | C3 | ANALOG BLUE |
| 14 | +5V POWER | C4 | ANALOG HORZ SYNC |
| 15 | GND | C5 | ANALOG GROUND |

**Slika 34. DVI pinout.**

# 10 HDMI

HDMI - High-Definition Multimedia Interface je kompaktni audio/video interfejs. Prenosi nekompresovani video sadržaj i kompresovani ili nekompresovani audio sadržaj. Digitalni interfejs definisan oko DVI 1.0 interfejsa. Kompatibilan sa DVI inerfejsom.

Zaštita sadržaja preko HDCP (High Definition Content Protection). Brzine prenosa zavise od moda displeja. Opseg 25MHz to 340MHz (600 MHz najnovija specifikacija). Licenca se plaća.

Zbog kompatibilnosti sa DVI svaki HDMI mora da podržava RGB reprezentaciju boja sa 8 bita po komponenti. Opciono HDMI podrzava YCbCr reprezentaciju boja. HDMi ima tri fizički odvojena komunikaciona protokola:

- E-DDC (Enhanced Display Data Channel)



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- TMDS (Transition Minimized Differential Signaling)
- CEC (Consumer Electronics Control)

Specifikacija 1.4 uljučuje dva dodatna protokola:

- ARC (Audio Return Channel)
- HEC (HDMI Ethernet Channel)

E-DDC (Enhanced Display Data Channel)

- Komunikacioni kanal baziran na I2C magistrali. Služi za čitanje E-EDID (Enhanced EDID) podataka iz displeja.
- Koristi se i za HDCP.

TMDS (Transition Minimized Differential Signaling) je komunikacion kanal za prenos video, audio i dodatnih podataka.

Tri različita tipa paketa:

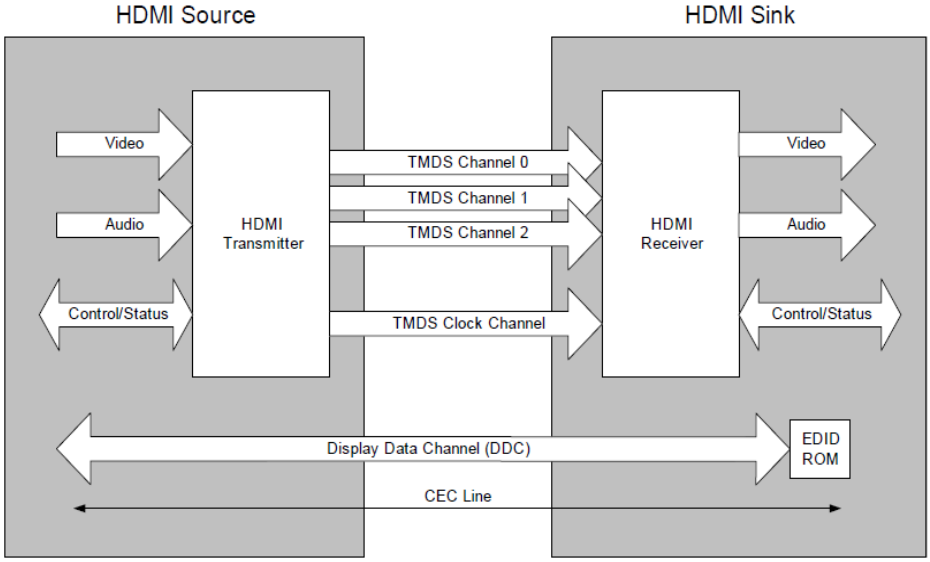
- Video Data Period – Prenos aktivnog video signala
- Data Island Period - Prenos audio i dodatnih podatka. Prenosi se u intervalima horizontalne i vertikalne sinhronizacije.
- Control Period – Period između Video i Data Island perioda.

CEC (Consumer Electronics Control)

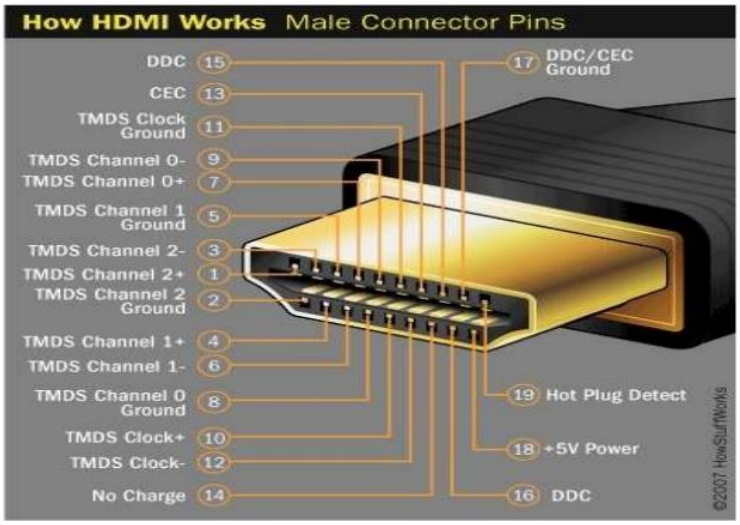
- Protokol koji omogućava korisniku da kontroliše do 15 urđaja, koji poseduju CEC kompatibilnost, koristeći samo jedan daljinski upravljač.
- Jednožična bidirekciona serijska magistrala.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 35. HDMI blok diagram.**




**Slika 36. HDMI raspored pinova.**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

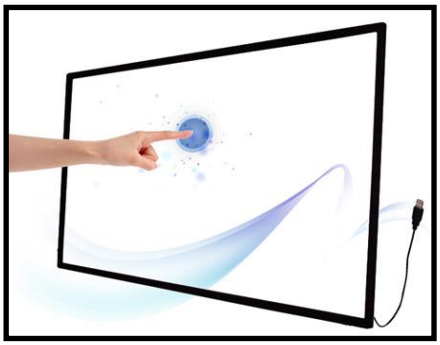
| HDMI version | 1.0 | 1.1 | 1.2 | 1.3 | 1.4 | 2.0 |
|---|---|---|---|---|---|---|
| Date initially released | December 9, 2002 | May 20, 2004 | August 8, 2005 | June 22, 2006 | May 28, 2009 | September 4, 2013 |
| Maximum clock rate (MHz) | 165 | 165 | 165 | 340 | 340 | 600 |
| Maximum TMDS throughput per channel (Gbit/s) including 8b/10b overhead | 1.65 | 1.65 | 1.65 | 3.40 | 3.40 | 6 |
| Maximum total TMDS throughput (Gbit/s) including 8b/10b overhead | 4.95 | 4.95 | 4.95 | 10.2 | 10.2 | 18 |
| Maximum throughput (Gbit/s) with 8b/10b overhead removed | 3.96 | 3.96 | 3.96 | 8.16 | 8.16 | 14.4 |
| Maximum audio throughput (Mbit/s) | 36.86 | 36.86 | 36.86 | 36.86 | 36.86 | 49.152 (IEC61937 and DST audio) |
| Maximum color depth (bit/px) | 24 | 24 | 24 | 48 | 48 | 48 |
| Maximum consumer resolution over single link at 24-bit/px | 1920×1200p60 | 1920×1200p60 | 1920×1200p60 | 2560×1600p75 | 4096×2160p30 | 4096×2160p60 |

**Tabela 22. HDMI verzije.**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 11 Touch Screen tehnologije



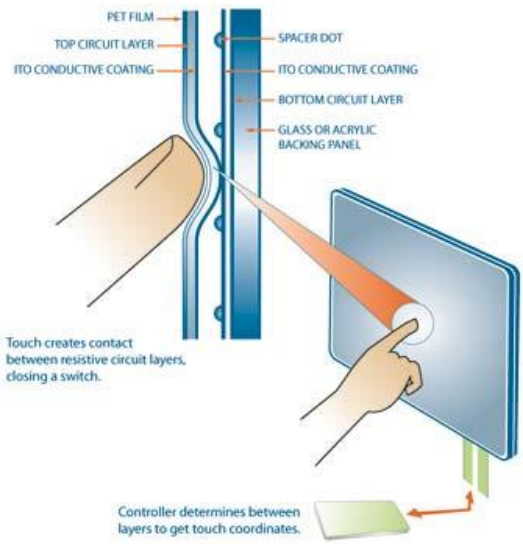
Ulazno/izlazni uređaj koji prihvata ulaze direktno sa monitora, korisnik dodiruje reči, grafičke ikonice, ili simbole prikazane na ekranu da bi aktivirao komande. Nalaze se u pametnim telefonima, tabletima, navigacionim sistemima, ali i u raznim namenskim sistemima.

Danas se koriste četiri različita tipa:

- Resitivni
- Kapacitivni
- SAW (Surface Acoustic Wave)
- Infrared LED ili Optički

## 11.1 Rezistivni Touch Screen

Dva sloja providnog rezistivnog materijala (indium tin oxide -ITO). Dodir pravi spoj između rezistivnih slojeva zatvarajući kolo. Napon se uniformno menja u zavisnosti od pozicije. Kontroler određuje poziciju na osnovu napona. Senzor se može aktivirati bilo kojim objektom.




Postoji nekoliko tipova:

- sa 4 žice
- sa 5 žica
- sa 8 žica

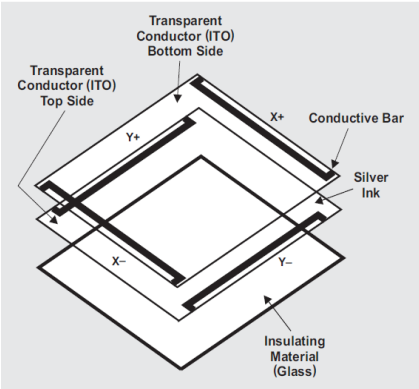
Svaki od tipova ima svoje prednosti.

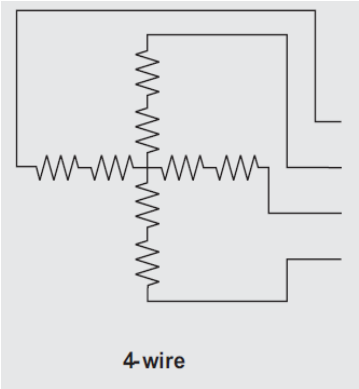
**Rezistivni Touch Screen sa 4 žice**

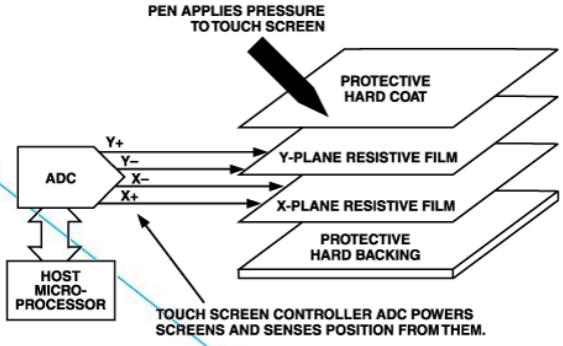
Jedna površina za X osu. Jedna površina za Y osu. Na obe se stvara gradijent napona. Koristi razdelnik napona za određivanje pozicije.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 37. Rezistivni Touch Screen sa 4 žice.**

Prednosti:

- Jeftini i zahvevaju malo snage
- Mogu se aktivirati bilo kojim objektom
- Relativno precizni

Mane:

- Površina od poliestera se može oštetiti
- Propuštaju samo 75% svetlosti
- Slaba izdržljivost (~35 miliona dodira)



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

## 11.2 Kapacitivni Touch Screen

Sastoji se od:

- Izolatora
- Provodnog sloja (ITO)

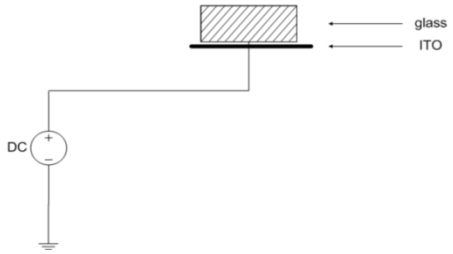
Dva tipa:

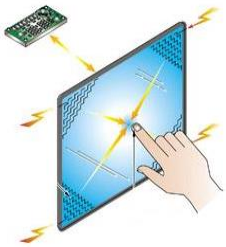
- Površinski (Surface)
- Projektovani (Projected)

**Slika 42. Kapacitivni Touch screen na mobilnom telefonu.**

**Površinski kapacitivni Touch Screen**

Samo jedna strana je obložena provodnim slojem. Na ivicama se nalaze electrode.



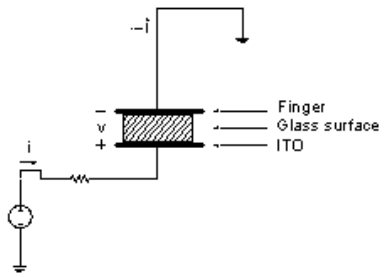



**Slika 39. Površinski kapacitivni Touch Screen.**

Prilikom pritiska prstom formira se kondenzator. Kontroler indirektno određuje lokaciju mereći kapacitivnost na četiri kraja.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES



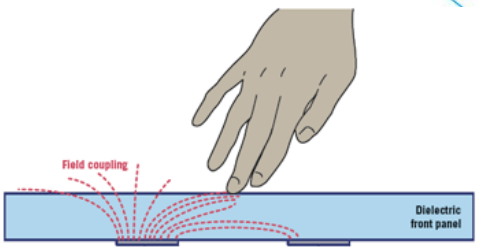

**Slika 40. Prinici rada površinskog kapacitinog Touch Screen-a.**

**Projektovani kapacitivni Touch Screen**

Dva paralelna ITO sloja na kojima se nalaze paralelni vodovi. Dva soja stakla.

**Slika 41. Projektovani kapacitivni Touch screen.**

Električno polje se stvara kroz staklo. Prst preuzima deo polja kada dodirne površinu kapacitivnost se menja.




**Slika 42. Projektovani kapacitivni Touch Screen - princip rada.**

**Poređenje kapacitivnih Touch Screen tehnologija**

Površinska

- Ograničena rezolucija
- Samo jedan dodir
- Radi sa direktnim kontaktom



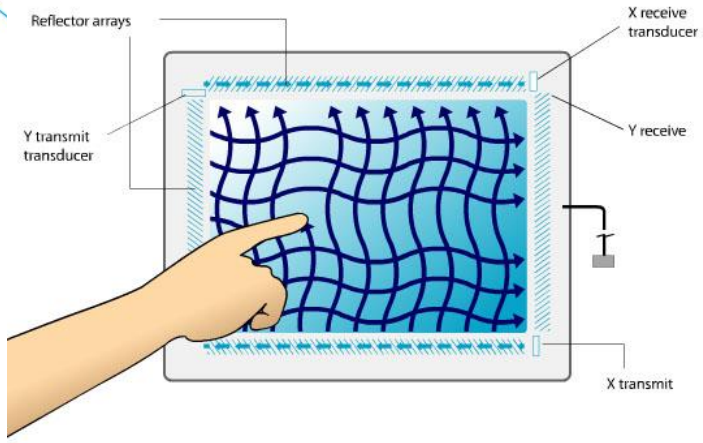
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Projektovana

- Visoka rezolucija
- Više dodira
- Radi sa indireknim kontaktom

**SAW Touch Screen**

Kod SAW tehnologije, piezoelektrični pretvarači raspoređeni po ekranu se koriste da pretvore mehaničke talase u električni signal i obrnuto. Talasi se šire po ekranu odbijajući se od niza reflektujućih elemenata. Detektuju se pomoću dva prijemnika. Akustičko polje slabi kada se površina dodirne. Kordinate se zatim određuju merenjem vremena pri kome amplituda oslabi.




**Slika 43. SAW Touch Screen.**

Prednosti:

- Nema slojeva koji se mogu habati.
- Nema gubitka u kvalitetu slike.
- Moguće je upravljati prstima, u rukavicama, posebnim olovkama.

Mane:

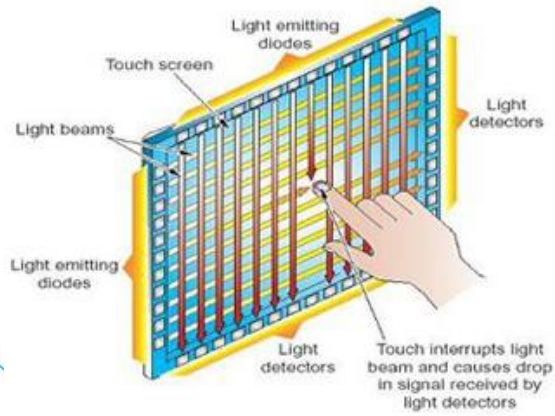
- Visoka cena.
- Prašina i prljavština mogu da stvore lažne signale.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**Infrared LED Touch Screen**

Infared tehnologija radi po principu prekidanja IR svetlosi. Sastoji se od niza IR LE dioda i foto tranzistora koji se nalaze na suprotnim stranama. Dodir prekida svetlost. Foto tranzistor detektuje prekid i kontroler određuje kordinate.




**Slika 44. IR Touch Screen.**

Prednosti:

- Propušta 100% svetlosti sa ekrana. Zastitno staklo je opciono.
- Skoro sve vrste olovki mogu da se koriste.
- Pouzdana tehnologija, dug vek trajanja.

Mane:

- Niska rezolucija.
- Može se koristiti samo na ravnim površinama.
- Može doci do detekcije iako nije izvršen pritisak već samo prelazak iznad površine ekrana.
- Prljavština može da stvori lažna očitavanja.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 12 JTAG

Standard koji definiše Test Access Port i Boundary-Scan arhitekture za digitalna integrisana kola i za digitalni deo analogno/digitalnih kola.

Standard obezbeđuje rešenje za problem testiranja uređaja na štamanim pločama baziranim na kompleksnim digitalnim integrisanim kolima i pločama sa velikom gustinom komponenti.

JTAG standard definiše test logiku koja se može uključiti u integrisano kolo da bi pružila standardizovan pristup:

- testiranju interkonekcija između integrisanih kola koja su uložena na štampanu ploču;
- testiranju samih integrisanih kola;
- posmatranje ili modifikaciju aktivnosti tokom normalnog rada komponente.
- Test logika se sastoji od Boundary-scan registra i drugih blokova kojima se pristupa preko Test Access Port-a (TAP).

**Boundary Scan**

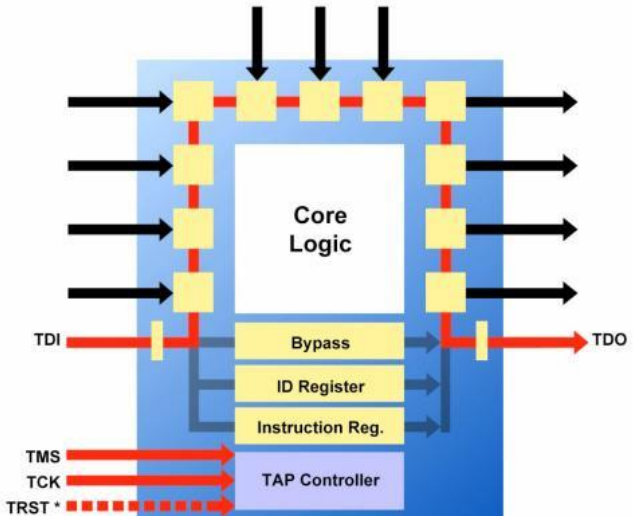
Boundary scan je metodologija koja omogućava kompletnu kontrolu i posmatranje pinova JTAG kompatibilnih uređaja preko softvera.

Test logika definisana ovim standardom uključije tri ključna bloka:

1. TAP Controller
2. Insturction Register
3. Data Register

TAP Controller reaguje na kontrolnu sekvencu koju dobija preko TAP (Test access port) i generiše takt i kontrolne signale potrebne za ispravno funkcionisanje drugih blokova.

Insturction Register je shift registar koji se serijski puni instrukcijama koje određuju koja će se operacija izvršiti.




**Slika 49. JTAG kompatibilni uređaj.**

Data Register predstavlja skup shift registara. Pobuda koja se zahteva za svaku operaciju se serijski puni u Data registar koji je selektovan



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

tekućom instrukcijom. Na pinovima uređaja se mogu izbaciti vrednosti koje su zadate instrukcijama ili pročitati vrednosti definisane spoljnim pobudama.

**Test Access Port**

JTAG Test Access Port (TAP) sadrži četiri pina koja upravljaju blokovima u kolu. Preko njega se serijski učitavaju i očitavaju instrukcije i podaci.

Pinovi Test access porta su:

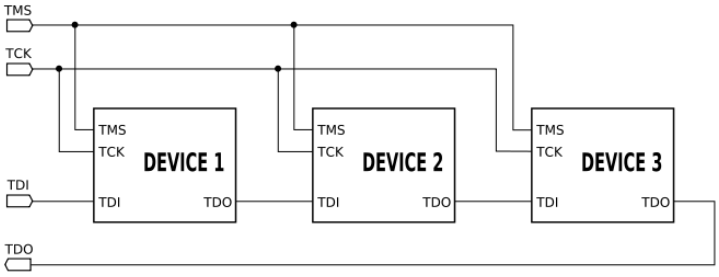
- TMS – Test Mode Select
    - Služi za kontrolu test operacija.
    - Odabira se na uzlaznu ivicu TCK signala.
- TCK – Test Clock
    - Takt za test logiku.
- TDI - Test Data Input
    - Ulaz za instrukcije i podatke.
    - Odabira se na uzlaznu ivicu TCK signala.
- TDO – Test Data Output
    - Test Data Out. Izlaz za instrukcije i podatke.
    - Menja stanje na silaznu ivicu TCK signal i aktivan je samo za vreme pomeranja podataka kroz uređaj.

Opciono može postojati i peti pin:

- TRST* - Test reset input
    - Sluzi za reset test logike.



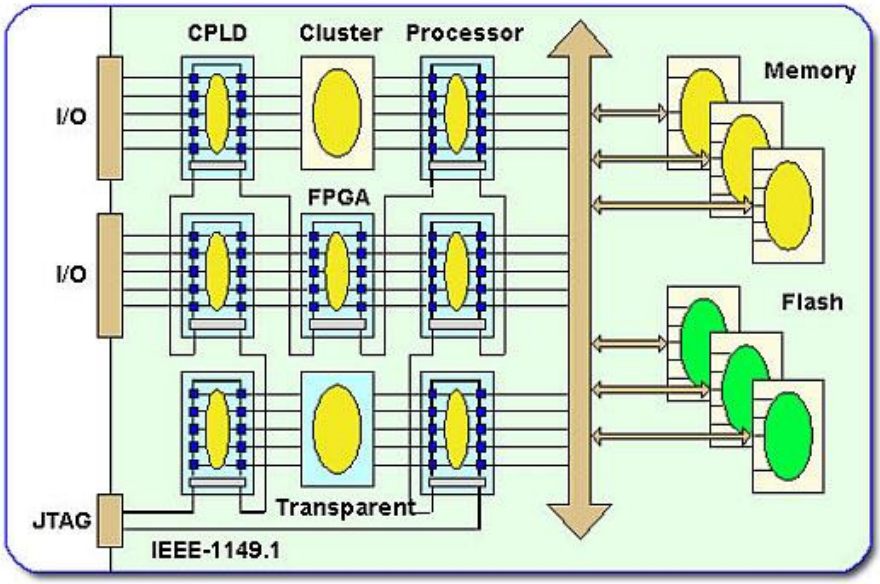
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 46. Način povezivanja JTAG kompatibilnih uređaja.**

**Test procedura**

- Tester primenjuje test ili dijagnostičke podatke na ulazne pinove uređaja.
- Boundary scan ćelije odabiraju podatke sa ulaznih pinova.
- Podaci se izbacuju preko TDO pina i vrši se verifikacija.
- Podaci mogu i da se učitaju u uređaj preko TDI pina.
- Tester tada može da verifikuje podatke na izlaznim pinovima uređaja.




**Slika 47. Primer povezivanja vise JTAG uređaja.**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 13 SD Card

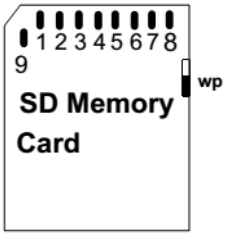
SD Memory Card – Memorijska kartica specijalno dizajnirana da obezbedi sigurnost, kapacitet i performanse. Kotisti obostranu proveru i nove algoritme za zastitu od ilegalnih korišćenja sadržaja kartice.

Komunikacija sa SD memorijskom karticom je bazirana na naprednom 9 pinskom interfejsu.

- Clock, Command, 4x Data i 3x Power.
- Interfejs dizajniran da radi na maksimalnoj frekvenciji od 50 MHz i niskom naponskom opsegu (3.3V, 1,8V).

Komunikacija sa SD karticom može da se odvija na dva načina:

- SD Mode
- SPI Mode



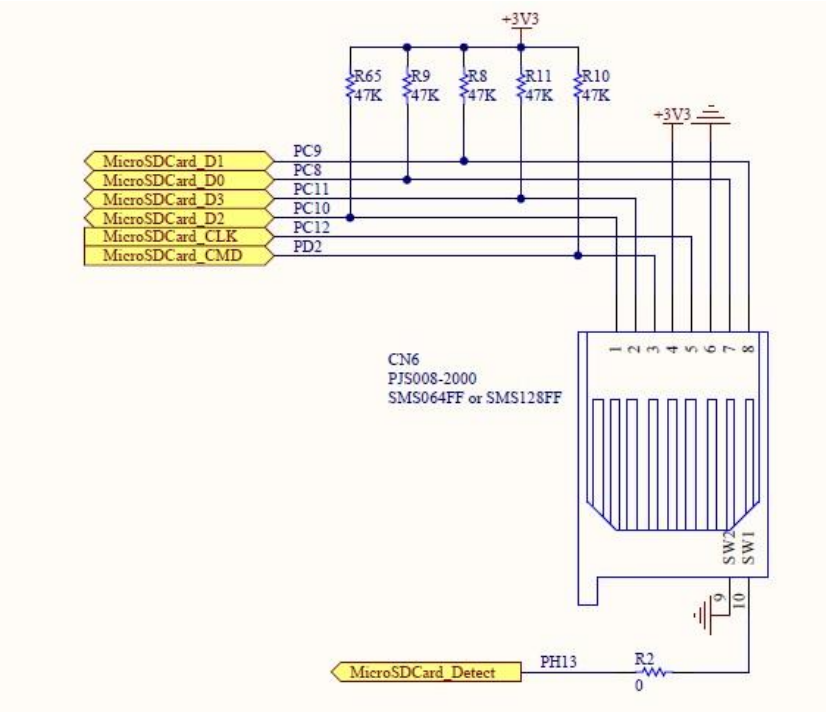

| Pin # | SD Mode | | | SPI Mode | | |
|---|---|---|---|---|---|---|
| | Name | Type[1] | Description | Name | Type[1] | Description |
| 1 | CD/DAT3[2] | I/O/PP[3] | Card Detect/ Data Line [Bit 3] | CS | I[3] | Chip Select (neg true) |
| 2 | CMD | I/O/PP | Command/Response | DI | I | Data In |
| 3 | VSS1 | S | Supply voltage ground | VSS | S | Supply voltage ground |
| 4 | VDD | S | Supply voltage | VDD | S | Supply voltage |
| 5 | CLK | I | Clock | SCLK | I | Clock |
| 6 | VSS2 | S | Supply voltage ground | VSS2 | S | Supply voltage ground |
| 7 | DAT0 | I/O/PP | Data Line [Bit 0] | DO | O/PP | Data Out |
| 8 | DAT1[4] | I/O/PP | Data Line [Bit 1] | RSV | | |
| 9 | DAT2[5] | I/O/PP | Data Line [Bit 2] | RSV | | |

**Slika 48. SD card rapored pinova u SD i SPI modu.**

SD Mode je standardni komunikacioni protokol po kome SD kartice rade. SPI Mode je Sekundarni protokol, podskup SD protokola. Dizajniran da komunicira sa SPI uređajima. Interfejs se bira prvom reset komandom posle podizanja i ne može se menjati nakon prvog podizanja kartice. SPI je jednostavniji interfejs, ali ima gubitak u preformansama u odnosu na SD mod.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 49. SD Mode**

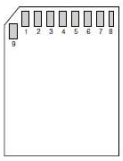


Figure 1: SD Card Diagram [2].

| Pin | Name | Function (SD Mode) | Function (SPI Mode) |
|---|---|---|---|
| 1 | DAT3/CS | Data Line 3 | Chip Select/Slave Select ($\overline{SS}$) |
| 2 | CMD/DI | Command Line | Master Out Slave In (MOSI) |
| 3 | VSS1 | Ground | Ground |
| 4 | VDD | Supply Voltage | Supply Voltage |
| 5 | CLK | Clock | Clock (SCK) |
| 6 | VSS2 | Ground | Ground |
| 7 | DAT0/DO | Data Line 0 | Master In Slave Out (MISO) |
| 8 | DAT1/IRQ | Data Line 1 | Unused or IRQ |
| 9 | DAT2/NC | Data Line 2 | Unused |

Table 1: SD Card Pin Assignments [2].

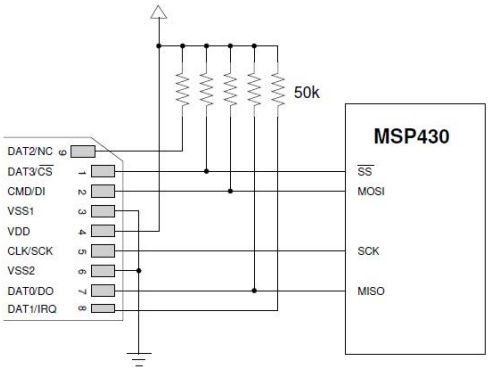



Figure 2: SD Card Schematic – SPI Mode [2].

**Slika 50. SPI Mode**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Kapacitet memorije:

- SDSC (Standard Capacity SD Memory Card): Do 2 GB.
- SDHC (High Capacity SD Memory Card): Više od 2GB do 32GB.
- SDXC (Extended Capacity SD Memory Card): Više od 32GB do 2TB.

Napon napajanja:

- High Voltage SD Memory Card – Opseg napona napajanja: 2.7-3.6 V.
- UHS-II SD Memory Card – Opseg napona napajanja VDD1: 2.7-3.6 V, VDD2: 1.70-1.95V.

Brzine magistrale:

- (1) Default Speed mode: 3.3V naponski nivo, Učestanost do 25 MHz, do 12.5 MB/sec.
- (2) High Speed mode: 3.3V naponski nivo, Učestanost do 50 MHz, do 25 MB/sec.
- (3) SDR12: UHS-I 1.8V naponski nivo, Učestanost do 25 MHz, do 12.5MB/sec.
- (4) SDR25: UHS-I 1.8V naponski nivo, Učestanost do 50 MHz, do 25MB/sec.
- (5) SDR50: UHS-I 1.8V naponski nivo, Učestanost do 100 MHz, do 50MB/sec.
- (6) SDR104: UHS-I 1.8V naponski nivo, Učestanost do 208 MHz, do 104MB/sec.
- (7) DDR50: UHS-I 1.8V naponski nivo, Učestanost do 50 MHz, odabir se vrši na obe ivice takta, do 50MB/sec.
- (8) UHS156: UHS-II RCLK Opseg učestanosti 26MHz - 52MHz, do 1.56Gbps po *Lane*-u.

Ostale karakteristike

- Read-only i Read-Write kartice.
- Zabrana pisanje putem mehaničkog prekidača.
- Zaštita šifrom.
- Vađenje kartice tokom operacije čitanja nece oštetiti karticu.
- Detekcija kartice.

Pakovanja:

- Standardna veličina SD kartice, 24 mm x 32 mm x

- 2.1 mm (normalna debljina).
- 1.4 mm (tanka SD memorijska kartica).

- miniSD, 21.5 mm x 20.0 mm x 1.4 mm.
- microSD, 15.0 mm x 11.0 mm x 1.0 mm.

**USB**

USB (Universal Serial Bus interfejs) je po svojim karakteristikama jednostavan za korišćenje. Obezbeđuje brz i jednostavan prenos podataka. Pristupačna cena interfejsa. Interfejs podržavaju različiti tipovi periferijskih uređaja. Periferije se same identifikuju. Podrška za brzine komunikacije od kb/s do više stotina Mb/s.

Omogućava paralelno funkcionisanje uređaja na magistrali. Podržava povezivanje do 127 fizičkih uređaja. Nizak overhead protokola daje visoku iskorišćenost magistrale. Mogućnost prenosa sa garantovanim protokom.

Podržava različite veličine paketa. U sam protocol je ugrađena kontrola protoka i grešaka u prenosu. Prednost je i dostupnost komponenata (kablova, konektora, kontrolera).

USB interfejs uključuje i linije za napajanje koja obezbeđuje napon od +5V, tako da ovo napajanje mogu koristiti periferijski uređaji čija je potrošnja ispod 500mA.

USB podržava režim smanjene potrošnje. Ova funkcionalnost podrazumeva automatsko isključenje napajanja USB uređaja koji se ne koriste, dok se oni ipak održavaju u spremnom stanju za slučaj da je zahtevana komunikacija sa njima.

Pouzdanost USB interfejsa je rezultat kako hardverske realizacije tako i protokola za prenos podataka. Iako je USB interfejs kompleksniji od postojećih starih interfejsa, cena komponenti i kablova koji se koristi u njegovoj realizaciji je ista ili čak manja od ekvivalenata kod starijih intefejsa.

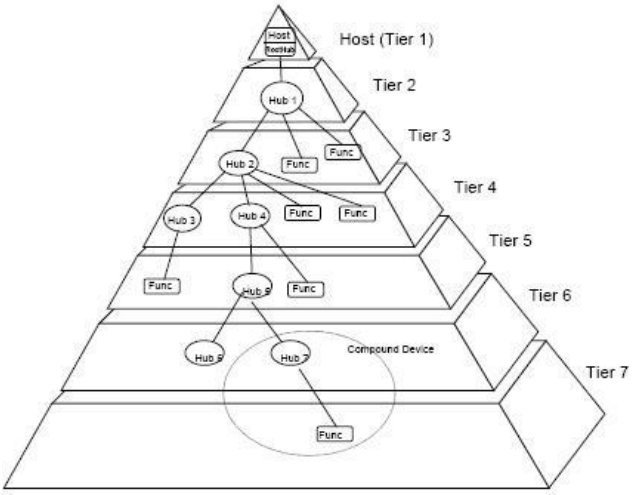
USB podržava četiri različite brzine na magistrali

- Super brzinu (SuperSpeed) od 5 Gbps
- Visoku brzinu (high speed) od 480 Mbps
- Punu brzinu (full speed) od 12 Mbps
- Nisku brzinu (low speed) od 1,5 Mbps.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Svaki PC računar sa USB interfejsom podržava nisku i punu brzinu, dok je visoka brzina dodata u verziji 2.0 USB specifikacije, i podrazumeva postojanje USB 2.0 kompatibilnog hardvera. Super brzinu podržavaju samo USB 3.0 kompatibilni uređaji.




USB sistem se sastoji iz tri različita dela:

- USB veze ili interkonekcije
- USB uređaja
- USB host-a

Sa strane periferijskih uređaja, svaki USB uređaj mora imati implementiranu hardversku podršku za USB interfejs u formi kontrolera koji vodi računa o svim detaljima tokom USB komunikacije.

Informacije o USB interfejsu i njegovoj mplementaciji se mogu naći na forumu:

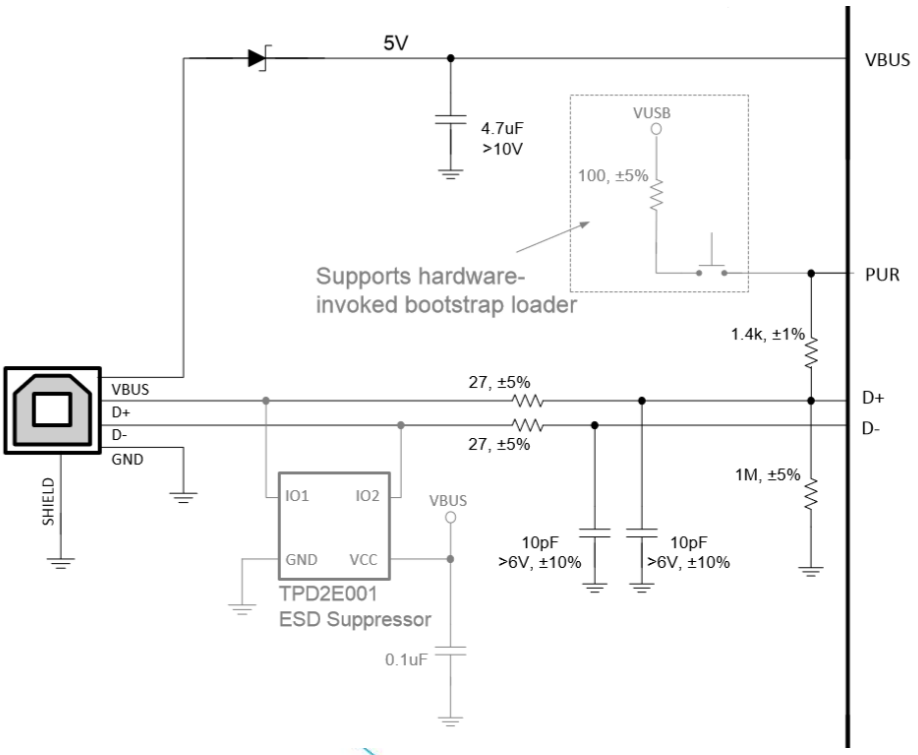
- USB Implementers Forum, Inc. (USB-IF): www.usb.org.

**Provodnici i konektori:**

- USB kabl sadrži četiri provodnika: VBUS, GND, D+ i D-.
- Naponski nivo na VBUS liniji je +5V.
- GND je referentni naponski nivo za VBUS, D+ u D-.
- D+ i D- su diferencijalni par signala.
- USB ikona na konektoru identifikuje USB kabl i gornju površinu konektora kada je konektor priključen u USB port.

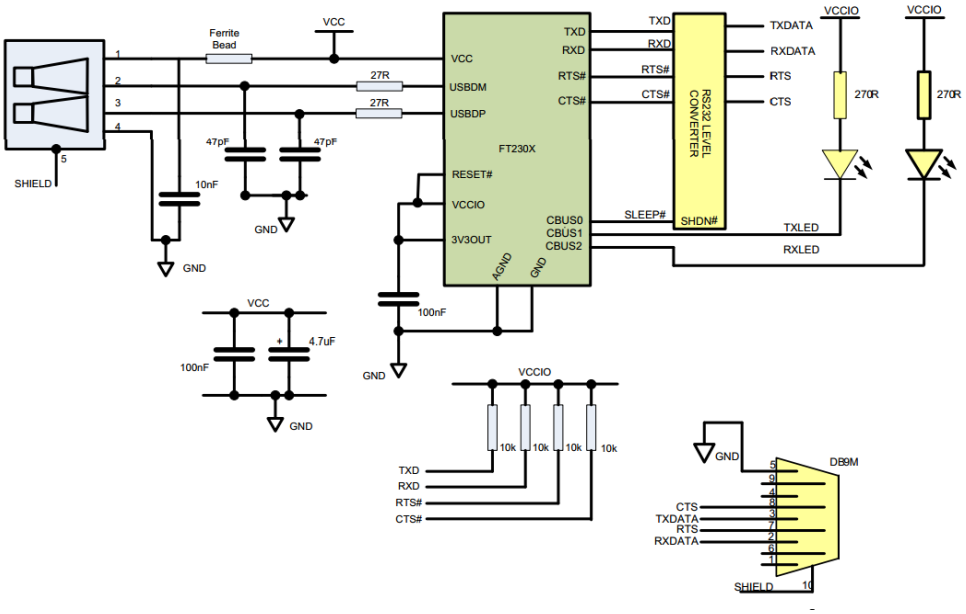


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 51. Primer povezivanja USB sa mikrokontrolerom MSP430.**

**USB to RS232 Converter**




**Slika 52. Primer kovertora USB u RS232.**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**SATA**

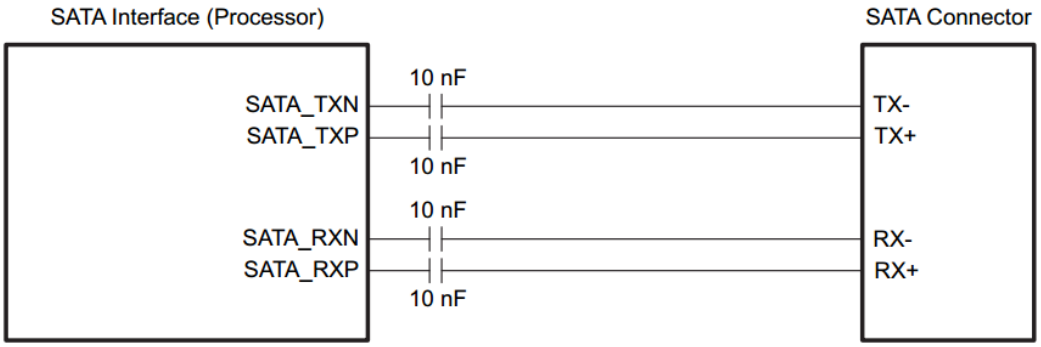
- SATA (*Serial ATA* )
- ATA (*Advanced Technology Attachment*)
- Protokol baziran na frejmovima
    - Mali jeftini konektor i jeftini kabolovi
- Podrzava nasleđeni ATA stek
- Lako se integriše
- Point to point konekcija
- Pobiljšani frekvencijski opseg
- Niža cena

| | Ultra ATA | Serial ATA (SATA) |
|---|---|---|
| Duplex | Half-Duplex | Half-Duplex |
| Data Rate | 133 MB/s | 300 MB/s |
| Connectivity | Parallel<br>80 wire ribbon cable | Serial<br>4 wire cable |
| Availability | Single port<br>drive | Single port<br>drive |
| Bus Connection | Master/Slave | Point to Point<br>Port Multiplier |
| Hot Plug | No | Yes |

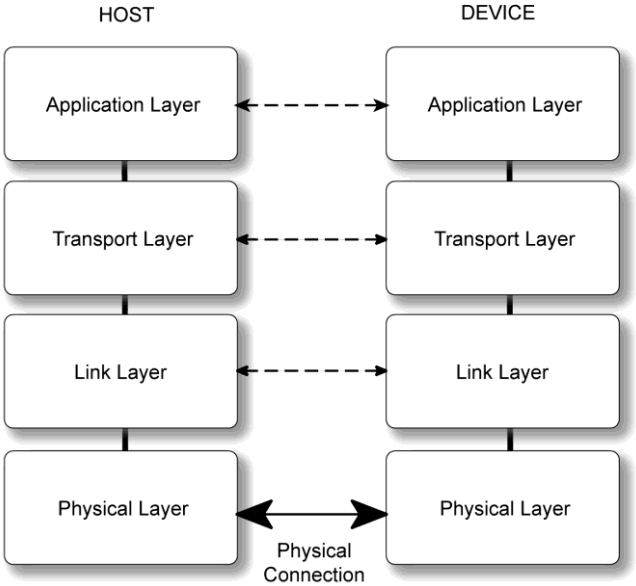
**Slika 53. Poređenje SATA interfejsa sa starijim Ultra ATA interfejsom.**

SATA je Poin-to-point komunikacija, host može da podrži više uređaja ali zahteva više linkova. Obezbeđuje 100% raspoloživost na jednom linku. Otkaz jednog uređaja ili linka ne utiče na ostale linkove.

SATA koristi Half-duplex vezu. Protokol dozvoljava prenos podataka u jednom smeru u jednom trenutku. Svaki link se sastoji od predajnog i prijemnog para. SATA koristi diferencijalne veze sa niskim naponskim nivoima LVDS (Low voltage differential signaling) čiji je nominalni napon +/- 250mV diferencijalno.




**Slika 54. Veza procesora i SATA konektora.**



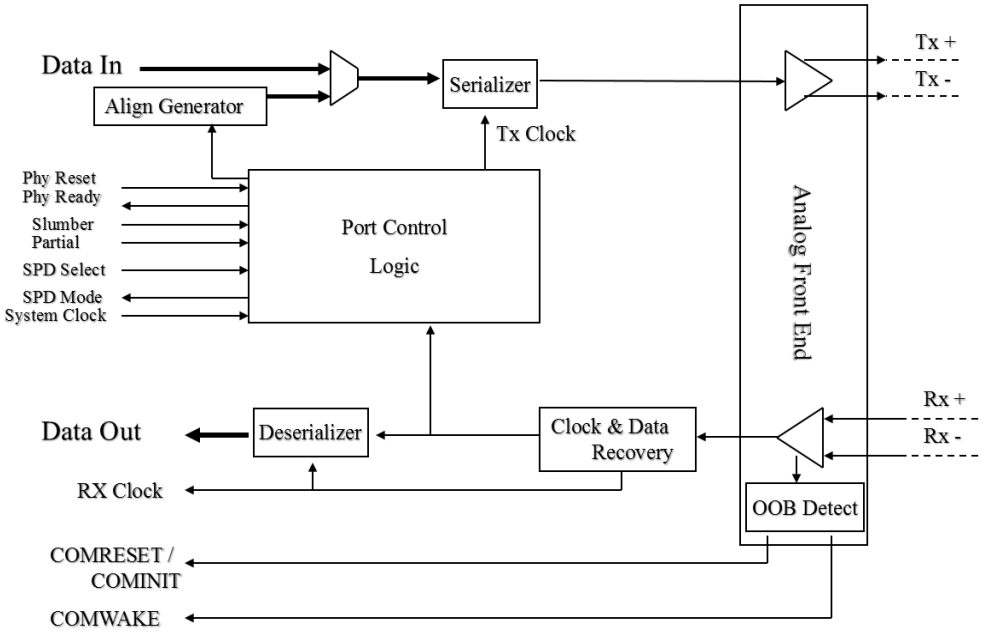

**Slika 55. Slojevi u SATA protokolu.**

- Application Layer – Komunicira sa aplikacijom i prihvata komande koje zatim prosleđuje transportnom sloju.
- Transoprt Layer - Šalje/prima frejmove u odgovarajućoj sekvenci.
- Link Layer – Sadrži informacije o strukturi frejmova (Frame Information Structures - FISs).

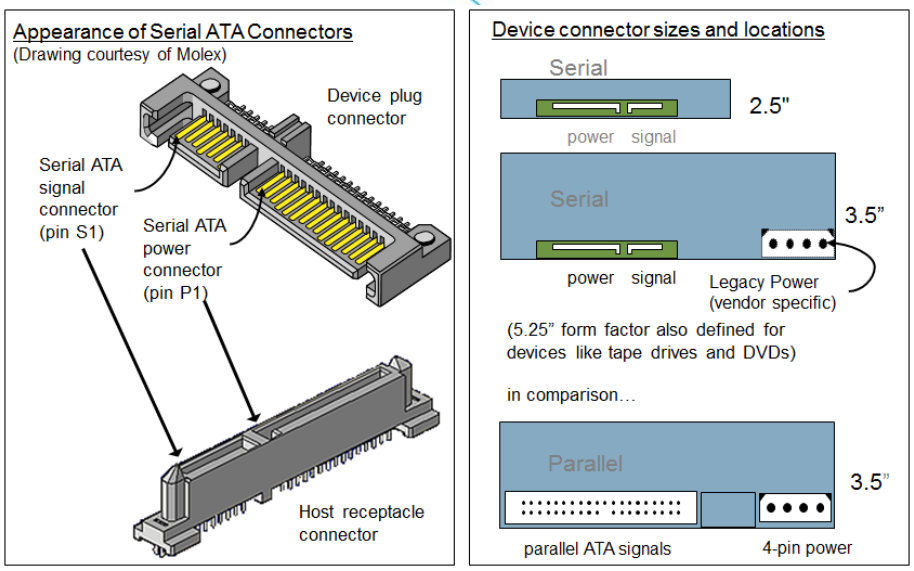


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Physical Layer – Fizički sloj, zadužen za slanje i primanje podataka na najnižem, fizičkom, nivou.




**Slika 56. Primer SATA porta.**




**Slika 57. SATA konektori.**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 14. Osnove *high-speed* dizajna

Svaki elektronski sistem sačinjen je od:

- Izvora signala tj. generatora naponskog signala. Izvor signala definisan je parametrima kao što su izlazna impedansa, vreme usponske ivice generisanog signala, vreme silazne ivice generisanog signala i dinamičkim opsegom.
- Transmisionih linija. Uloga transmisionih linija, provodnika je prosleđivanje električnog signala od izvora signala do potrošača. Mogu biti realizovane na više načina, kao koaksijalni kablovi, upredene parice, provodnici na štampanoj ploči i drugi.
- Potrošača koji primaju električni signal koji generišu izvori signala. Po pojavi električnog signala na svojim ulazima potrošači preduzimaju određene akcije, na primer obavljanje neke logičke operacije ili slično. Obično, potrošači se ponašaju kao male parazitne kapacitivnosti koje po dolasku elektromagnetnog talasa mogu apsorbovati određeni deo njegove energije.
- Izvora napajanja koji obezbeđuju energiju neophodnu za funkcionisanje kola izvora signala i potrošača. Njihova uloga jeste da obezbede stabilnu referencu napona svim elementima elektronskog sistema, kao što su izvori signala i kola potrošača, nezavisno od udaljenosti elemenata.

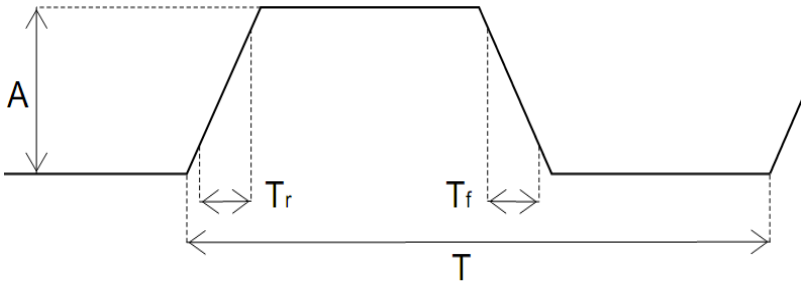
Pri dizajnu elektronskih sistema koji imaju karakteristike *high-speed* dizajna potrebno je biti obazriv. Integrisana kola nisu idealna, ona unose određeno kašnjenje signala od ulaza do izlaza. Takođe, integrisana kola na svojim kontaktima poseduju neželjene parazitne kapacitivnosti kao i induktivnosti koja mogu imati uticaja na integritet signala. Izvori signala nisu idealni, generisani signal zavistan je od vremena usponske i silazne ivice kao i impedanse izvora. Ulazi potrošača takođe nisu idealni, oni poseduju određenu parazitnu kapacitivnost. Konektori mogu imati različitu impedansu od željene, između kontakata moguća je pojava preslušavanja signala. Talasni oblici električnog signala nisu istovetni na početku i na kraju transmisione linije. Potrebno je voditi računa o svim frekvencijama, počevši od DC pa do frekvencija reda GHz. Rad integrisanih kola pri promeni stanja na svojim izlazima dovodi do povećanja potrošnje u kratkim vremenskim intervalima. Važno je napomenuti da su digitalni signali tolerantniji na distorziju od analognih.

Digitalni signal je električni signal čija amplituda može uzimati vrednosti samo iz unapred definisanog skupa vrednosti. Digitalni signal opisan je sledećim parametrima:

- Amplituda - $A$
- Trajanje usponske ivice - $T_r$
- Trajanje silazne ivice - $T_f$
- Periodom - $T$



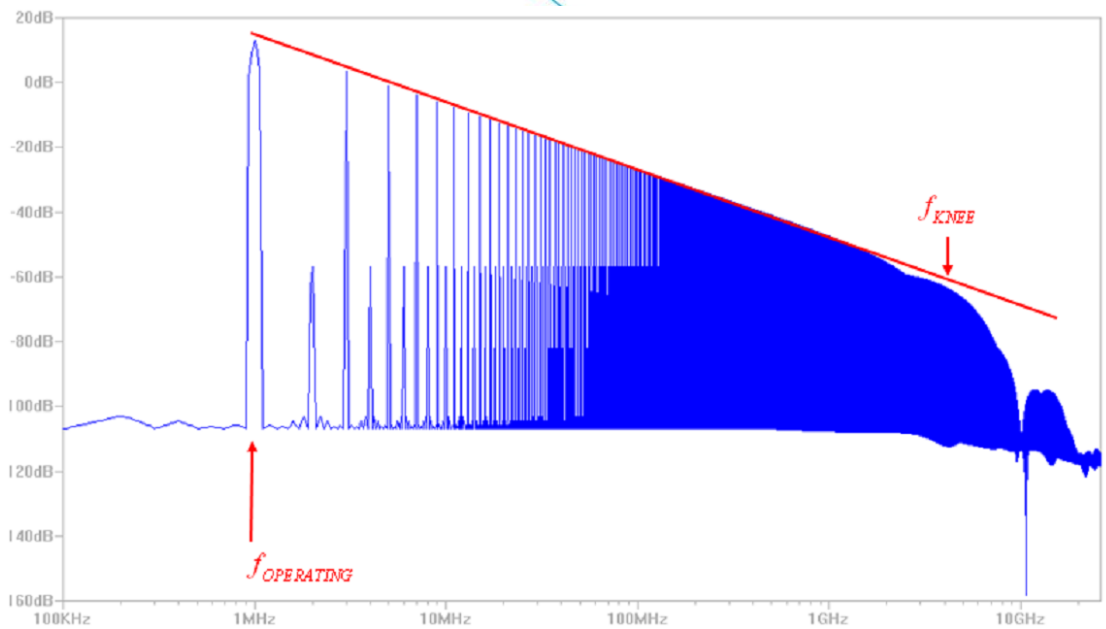
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*High-speed* signali ne moraju biti isključivo signali visoke frekvencije, češće su to digitalni signali sa kratkim vremenima trajanja usponske i silazne ivice signala. *Fknee* – frekvencija kolena predstavlja frekvenciju ispod koje je koncentrisan najveći deo energije digitalnog signala. *Fknee* jednaka je :

$$F_{KNEE} = \frac{0.5}{T_r}$$

Integrisana kola koja imaju ravan frekventni odziv za frekvencije manje od frekvencije kolena, propustiće digitalni signal bez ili sa zanemarljivim izobličenjima. Ponašanje digitalnih kola pri frekvencijama koje su više od frekvencije kolena ima neznatan uticaj na integritet signala. Da bi se izbegla distorzija digitalnog signala, električna veza signala na štampanoj ploči treba biti u mogućnosti da prenese ceo spektar signala. Na slici ispod prikazan je spektar digitalnog signala vremena usponske ivice signala $T_r$ = 100 *ps*, kao i obeležena vrednost *Fknee*.




Vreme za koje će se generisani signal na izvoru pojaviti na strani potrošača zavisno je od udaljenosti izvora signala i potrošača, prijemnika signala. Kašnjenje signala proporcionalno je kvadratnom korenu



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

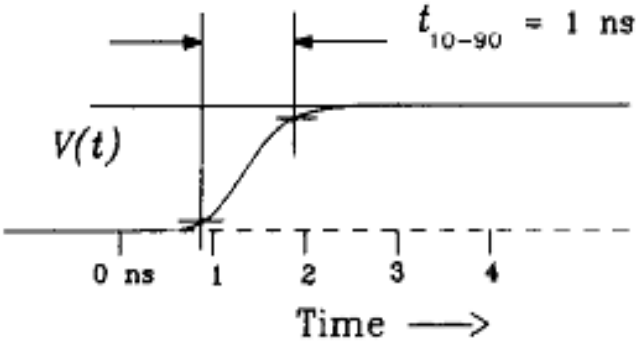
dielektrične konstante mediuma kroz koji se vrši komunikacija. U tabeli ispod data su kašnjena signala zavisno od dužine električne veze i medijuma kroz koji se električna veza prostire.

| Medijum | Kašnjenje (ps/inch) | Dielektrična konstanta |
|---|---|---|
| Vazduh | 85 | 1.0 |
| FR4 štampana ploča na spoljnim slojevima | 140-180 | 2.8 - 4.5 |
| FR4 štampana ploča na unutrašnjim slojevima | 180 | 4.5 |
| Aluminiumska štampana ploča | 240-270 | 8 - 10 |

Odgovor tj. odziv nekog sistema provodnika na dolazeći električni signal u velikoj meri zavisi od toga da li je dužina provodnika manja od efektivne dužine najbrže električne osobine signala. Usponska ivica signala predstavlja električnu osobinu signala. Njeno trajanje zavisi ne samo od karakteristika izvora nego i od kašnjenja na transmisionoj liniji. Pod kolima sa koncentrisanim parametrima smatramo ona kod kojih je sistem provodnika dovoljno mali tj. kratak tako da svaki element veze reaguje istovremeno jednakim potencijalom. Ako se usponska ivica električnog signala prostire kroz sistem provodnika na takav način da potencijal svakog elementa veze u sistemu nije jednak, tada govorimo o kolima sa distribuiranim parametrima. Električna dužina zavisna je od vremena usponske ivice signala, $T_r$, kao i od kašnjenja signala kroz prenosni medijum, $D$, kroz koji se vrši komunikacija. Opisana je jednačinom:

$$l = \frac{T_r}{D}$$

**Primer:** Neka je transmisiona linija realizovana na unutrašnjem sloju elektronske štampane ploče izrađene od FR4 materijala. Ako je vreme usponske ivice signala predajnika 1 ns, izračunati električnu dužinu transmisione linije.






Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

$$l = \frac{1ns}{180\ \frac{ps}{inch}} = \frac{1 * 10^{-9}\ s}{180 * 10^{-12}\ \frac{s}{inch}} = 5.55\ [inch]$$



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 15 Transmisione linije

Na visokim frekvencijama prednosti transmisionih linija, vodova u odnosu na regularno povezivanje tačka-tačka su manja izobljičenja signala, manje zračenje kao i manje preslušavanje među signalima. Impedansa transmisione linije sastoji se od podužne provodnosti $G_0$ , podužne otpornosti $R_0$ , podužne induktivnosti $L_0$ i podužne kapacitivnosti $C_0$. Data je jednačinom:

$$Z_0 = \sqrt{\frac{R_0 + j\Omega L_0}{G_0 + j\Omega C_0}}$$

Kapacitivnost postoji u transmisionim linijama, kablovima kao i na samoj štampanoj ploči. Ukoliko izvor električnog signala promeni stanje svog izlaza sa logičke nule na logičku jedinicu svaka od kapacitivnosti veze sa prijemnikom treba biti "napunjena" da bi se novonastalo stanje logičke jedinice prenelo do potrošača. Parazitne kapacitivnosti obično su reda nekoliko pF. Na niskim frekvencijama parazitne kapacitivnosti u električnim vezama imaju neznatan uticaj na integritet signala i mogu se izostaviti pri analizi kola. Tabela ispod prikazuje tipične vrednosti parazitnih kapacitivnosti određenih struktura na elektronskoj štampanoj ploči.

| Struktura | Kapacitivnost |
|---|---|
| Ulaz integrisanog kola | 5.0 pF |
| 1 inč transmisione linije impedanse 50 Ω | 3.5 pF |
| 1 cm transmisione linije impedanse 50 Ω | 1.4 pF |
| Via prečnika 13mil i dužine 100mil | 0.3 pF |

Induktivnost je svojstvo provodnika da se odupre protoku elektromagnetne energije duž njega. Parazitne iduktivnosti obično su reda nekoliko nH. Na niskim frekvencijama parazitne induktivnosti u električnim vezama imaju neznatan uticaj na integritet signala i mogu se izostaviti pri analizi kola. Pri niskim frekvencijama induktivna reaktansa je mala. Tabela ispod prikazuje tipične vrednosti parazitnih induktivnosti određenih struktura na elektronskoj štampanoj ploči.

| Struktura | Induktivnost |
|---|---|
| 1 inč transmisione linije impedanse 50 Ω | 10 nH |
| 1 cm transmisione linije impedanse 50 Ω | 4 nH |
| Kondenzator u 1206 pakovanju | 1.5 nH |
| Kondenzator u 0805 pakovanju | 0.85 nH |
| Kondenzator u 0603 pakovanju | 0.63 nH |



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

| | |
|---|---|
| Kondenzator u 0402 pakovanju | 0.35 nH |
| Via dužine 100mil | 1 nH |

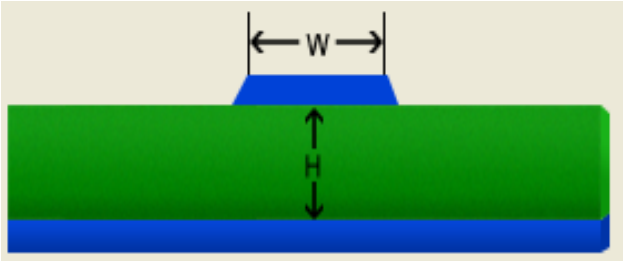
Upotreba transmisionih linija značajno umanjuje *EMI* zračenja tako što ograničava protok povratne struje signala. *Single-ended* signalizacija odnosi se na put električnog signala koji je sastavljen od izvora signala, transmisione linije koja se prostire kroz štampanu ploču i jednog ili više potrošača tj. prijemnika signala. Predstavlja najjednostavniji i najčešće korišćen metod prosleđivanja električnih signala od predajnika, izvora signala koji ih generišu, do prijemnika signala. Električnom vezom se prosleđuje naponsko promenljiva vrednost signala koji je najčešće referenciran na referentnu masu električnog sistema. Ovakva izvedba transmisionih linija sklona je smetnjama koje mogu nastati kao posledica razlike potencijala referentnih masa između kola predajnika i prijemnika signala. Kao i *single-ended* signalizacija, diferencijalna signalizacija predstavlja metod za prenos električnog signala od izvora signala do potrošača. Glavna prednost diferencijalne signalizacije je njihova povećana imunost na smetnje iz okoline. Razlika između potencijala masa izvora signala i prijemnika signala neće imati uticaj na ispravnu detekciju signala. Tačnost dužine uparivanja, odnosno razlika između dužina u paru diferencijalnih transmisionih linija, određuje se na osnovu vremena najbrže usponske ili silazne ivice signala. Transmisione linije koje čine diferencijalni par nalaze se u kontrafazi. To znači da će prilikom diferencijalne signalizacije izvor diferencijalnog signala preko jedne linije davati struju dok će na drugoj primati. Diferencijalnom signalizacijom se obezbeđuje da se izvoru napajanja sistema "vrati" onoliko struje koliko je iz njega "uzeto". Na ovaj način smanjuje se količina smetnji koje izvor diferencijalnog signala unosi u električno kolo u kojem se nalazi. Prednosti upotrebe diferencijalnih transmisionih linija su:

- Manji opseg napona između logičkih nivoa.
- Robusnost.
- Povećana imunost na smetnje.
- Smanjeno elektromagnetno zračenje na ostatak kola.
- Omogućavaju velike brzine razmene podataka.

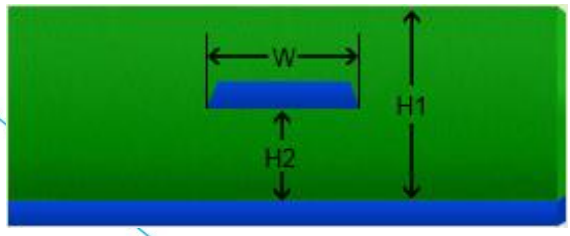
Pri realizaciji transmisionih linija na elektronskoj štampanoj ploči razlikujemo *microstrip* i *stripline* izvedbu. *Microstrip* predstavlja tip transmisione linije koja se formira na spoljašnjim slojevima elektronske štampane ploče. Parametri koji definišu impedansu transmisione linije u *microstrip* izvedbi su njena debljina, visina od provodne površi (*H*) i širina voda (*W*).



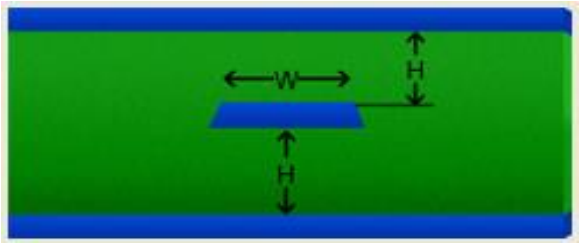
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




Jedna od mogućih izvedbi *microstrip* transmisione linije je tzv. *embedded microstrip*. Parametri koji definišu impedansu transmisione linije u *embedded microstrip* izvedbi su njena debljina, visina od provodne površi (*H2*), visina dielektrika (*H1*) i širina voda (*W*).



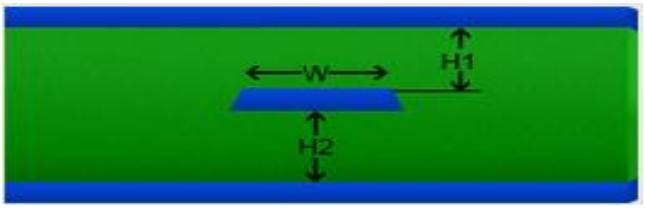

*Stripline* predstavlja tip transmisione linije koja se formira na unutrašnjim slojevima elektronske štampane ploče. Transmisiona linija može biti centrirana između dve provodne površi ili primaknuta jednoj od njih. Tako razlikujemo simetričnu i asimetričnu izvedbu *stripline* transmisione linije. Parametri koji definišu impedansu transmisione linije u simetričnoj *stripline* izvedbi su njena debljina, jednaka visina od provodnih površi (*H*) i širina voda (*W*).




Parametri koji definišu impedansu transmisione linije u asimetričnoj *stripline* izvedbi su njena debljina, visina od prve provodne površi (*H1*), visina od druge provodne površi (*H2*) i širina voda (*W*).

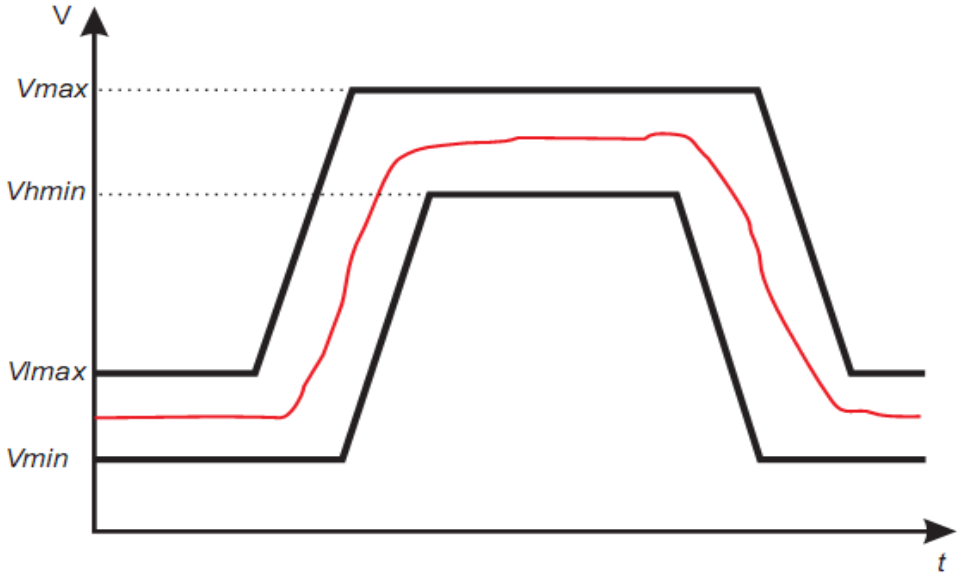


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES


W
H1
H2

# 16 Integritet signala

Inženjering integriteta signala koristi se u svrhu opisa svih koraka koje preduzimamo kako bi smo bili sigurni da će svi električni signali stići do prijemnika i pritom zadržati zadovoljavajući kvalitet. Koji je to zadovoljavajući kvalitet zavisi od prirode samog dizajna za koji vršimo proveru integriteta signala. Stvaranje skupa pravila na osnovu kojeg ćemo modelovati integritet signala počinje proračunavanjem prihvatljivih vrednosti za usponske i silazne ivice svake od komponenata u dizajnu. Na osnovu ovog skupa pravila određuju se smernice za povezivanje komponenata i dizajn sistema napajanja koje će obezbediti pouzdan rad uređaja. Na slici ispod, crvenom bojom prikazan je realan oblik signala, dok su maksimalne i minimalne dozvoljene granice naponske vrednosti signala označene crnom.




- *Vmax* – Maksimalna vrednost napona koju integrisano kolo može da prihvati na svom ulazu.
- *Vhmin* – Minimalna vrednost napona logičke jedinice, donji prag, neophodna za ispravnu detekciju signala logičke jedinice na ulazu integrisanog kola.
- *Vlmax* – Maksimalna vrednost napona logičke nule, gornji prag, neophodna za ispravnu detekciju signala logičke nule na ulazu integrisanog kola.
- *Vmin* – Minimalna vrednost napona koju integrisano kolo može da prihvati na svom ulazu.

Na signale velike brzine utiču refleksije i preslušavanja svake usponske i silazne ivice signala što za posledicu ima određeno vreme “smirivanja” signala. Vreme “smirivanja” signala određeno je:

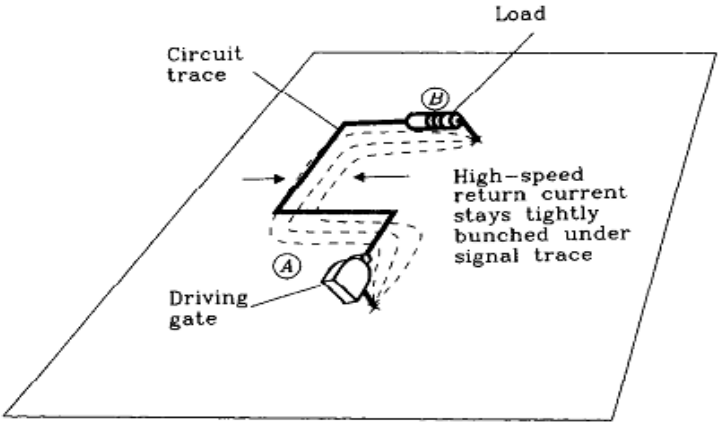
- Usponskom ivicom signala.
- Izlaznom impedansom izvora signala.
- Kapacitivnošću veze na elektronskoj štampanoj ploči.
- Kapacitvnošću veze sa prijemnikom signala.



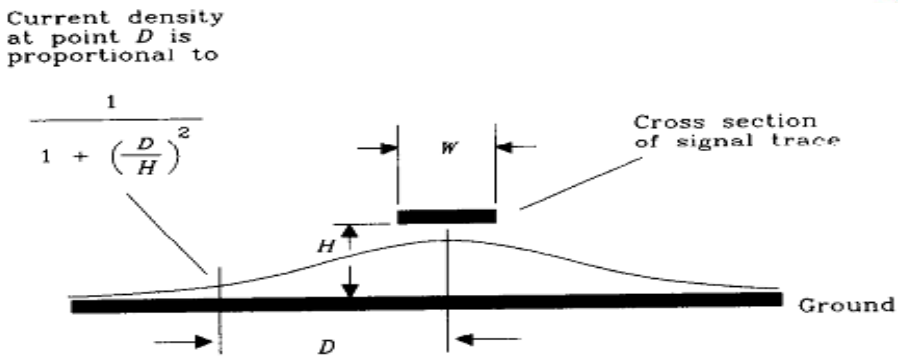
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Ulaznom kapacitivnošću prijemnika.
- Kašnjenjem koje unosi dužina veze na elektronskoj štampanoj ploči.

Provodne površi predstavljaju jedan od slojeva elektronske štampane ploče predviđen za distribuciju električnih veza napajanja ili mase sistema. Provodne površi pružaju stabilne naponske reference za digitalne signale, distribuiraju napajanje do svih elemenata sistema i kontrolišu preslušavanje između bliskih signala. Pri visokim brzinama razmene podataka, induktivnost povratnog puta struje značajno je bitnija nego njegova otpornost. Put povratne struje koji ima najmanju moguću induktivnost nalazi se direktno ispod provodnika i tako minimizuje petlju između odlazeće i povratne struje u kolu. Pri visokim frekvencijama povratne struje prate put najmanje induktivnosti. Povratni put struje u *high-speed* dizajnu prikazan je na slici ispod.




Pri visokim frekvencijama povratne struje teže da prate put direktno ispod (iznad) provodnika. Na slici ispod prikazana je transmisiona linija širine *W*, koja se nalazi na rastojanju *H* od provodne površi mase. Oznakom *D* prikazana je gustina povratne struje, čiji intenzitet opada sa povećanjem udaljenosti od transmisione linije.






Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Provodne površi trebaju omogućiti povratni put struje koji je što bliži odgovarajućujoj transmisionoj liniji. Tokom dizajna štampanih ploča tj. tokom particionisanja sloja provodne površi na više nezavisnih segmenata treba biti obazriv da se particionisanje ne vrši u blizini transmisionih linija koje se mogu naći na susednim slojevima štampane ploče. Za potrebe povezivanja signala velike brzine, pored provodnih površi na koje je povezana masa, provodne površi na koje su povezani naponi napajanja takođe se mogu koristiti. Signali ne moraju uvek biti referencirani na provodnu površ mase. Za ovakav pristup povezivanju signala velike brzine potrebno je obezbediti pravilnu analizu sistema napajanja kola.

Ako iz nekog razloga duž transmisione linije dođe do promene karakteristične impedanse, nebitno iz kog razloga, deo energije elektromagnetnog talasa reflektovaće se nazad, u smeru izvora. Ostatak energije elektromagnetnog talasa koji nastavlja ka prijemniku umanjen je za vrednost reflektovane energije. Koeficient refleksije se može izračunati na osnovu jednačine:

$$\% = \frac{Z_u - Z_i}{Z_u + Z_i}$$

gde je $Z_u$ ulazna impedansa prijemnika signala, a $Z_i$ izlazna impedansa izvora signala. Ako je koeficient refleksije pozitivan, reflektovani napon se dodaje, dok ako je koeficient refleksije negativan, reflektovani napon se oduzima. U slučaju da je impedansa $Z_u$ beskonačna ili u slučaju da se transmisiona linija završava otvorenim krajem, sva energija elektromagnetnog talasa biće reflektovana nazad. Efekat ovakve refleksije duplira napon na kraju otvorene transmisione linije. U slučaju da je impedansa $Z_u$ nula ili u slučaju da se transmisiona linija završava kratkim spojem, sva energija elektromagnetnog talasa biće reflektovana nazad ali sa negativnim koeficientom. Reflektovana energija se oduzima i rezultantni napon iznosi nula volti.

Preslušavanje predstavlja kaplovanje energije iz jedne transmisione linije na drugu. Preslušavanje se javlja kad god su elektromagnetna polja potekla iz različitih struktura u interakciji. U slučaju digitalnog dizajna pojava preslušavanja je učestala. Preslušavanje se može pojaviti:

- Na vezama pakovanja integrisanog kola.
- Unutar štampane ploče.
- Na konektorima.
- Na kablovima.

Kapacitivno preslušavanje nastaje u slučaju blizine dva provodnika koja se nalaze jedan iznad drugog unutar štampane ploče. U ovom slučaju dva provodnika se prostiru paralelno jedan iznad drugog i na taj način formiraju kondenzator određene kapacitivnosti. Induktivno preslušavanje je praktično neizbežno. Znatno je učestalije nego kapacitivno. Razlikujemo preslušavanje na kraju prijemnika i na kraju predajnika.

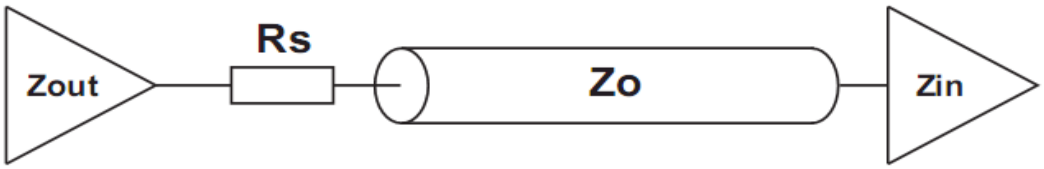
Kada signali velikih brzina dostignu deo transmisione linije koja poseduje diskontinuitet, deo energije signala biva reflektovan nazad ka izvoru signala. Da bi se izbegle refleksije potrebno je projektovati transmisionu liniju na takav način da njena karakteristična impedansa za signale velikih brzina bude



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

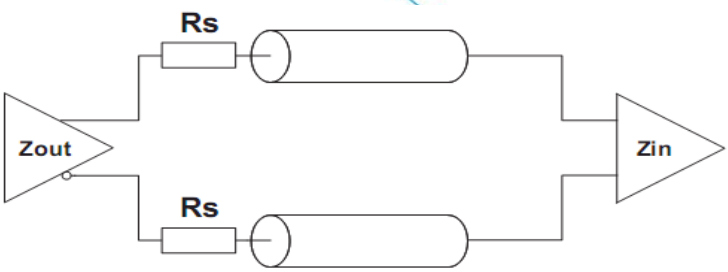
konstantna celom dužinom od izvora signala pa do prijemnika signala. Ovo podrazumeva, pored modelovanja transmisione linije na štampanoj ploči, da karakteristična impedansa veze na štampanoj ploči bude prilagođena izlaznoj impedansi izvora signala. Uglavnom kola izvora signala imaju nisku izlaznu impedansu dok kola prijemnika obično poseduju imedansu visoke vrednosti. Zato nam trebaju terminacije. Karakteristike redne, serijske terminacije su:

- Smanjuje izobličenja signala na strani prijemnika.
- Ne eliminiše prvu refleksiju na prijemniku.
- Nema uticaja na struju i naponske nivoe izvora signala.




$R_s = Z_o - Z_{OUT}$ , redna terminacija *single-ended* transmisionih linija.

Redne terminacije zahtevaju što bliže postavljanje prema izvoru signala, dok paralelne terminacije zahtevaju da otpornici ili kondenzatori budu postavljeni što je moguće bliže prijemniku signala. Dužinu veze između terminacija i izvora/prijemnika signala treba minimizovati.




$R_s = \frac{Z_{odiff} - Z_{diff}}{2}$ , redna terminacija diferencijalnih transmisionih linija.

Signali velikih brzina često zahtevaju upotrebu terminacije bilo na kraju prijemnika ili na kraju izvora signala tj. predajnika. Terminaciju treba postavljati ili na kraj sa prijemnikom ili na kraj sa predajnikom, nikako istovremeno na oba kraja. Upotreba istovetne terminacije i na kraju prijemnika i na kraju predajnika rezultovaće smanjenjem napona na polovinu njegove vrednosti. Kada se izvor signala velike brzine koristi kao predajnik signala za više prijemnika, tada treba razmišljati o upotrebi terminacije na strani prijemnika. Treba imati u vidu da svako grananje voda unosi poremećaj u njegovu karakterističnu impedansu. Tada treba voditi računa o dužini transmisione linije do svakog prijemnika kao i o vrednosti terminacionog elementa.

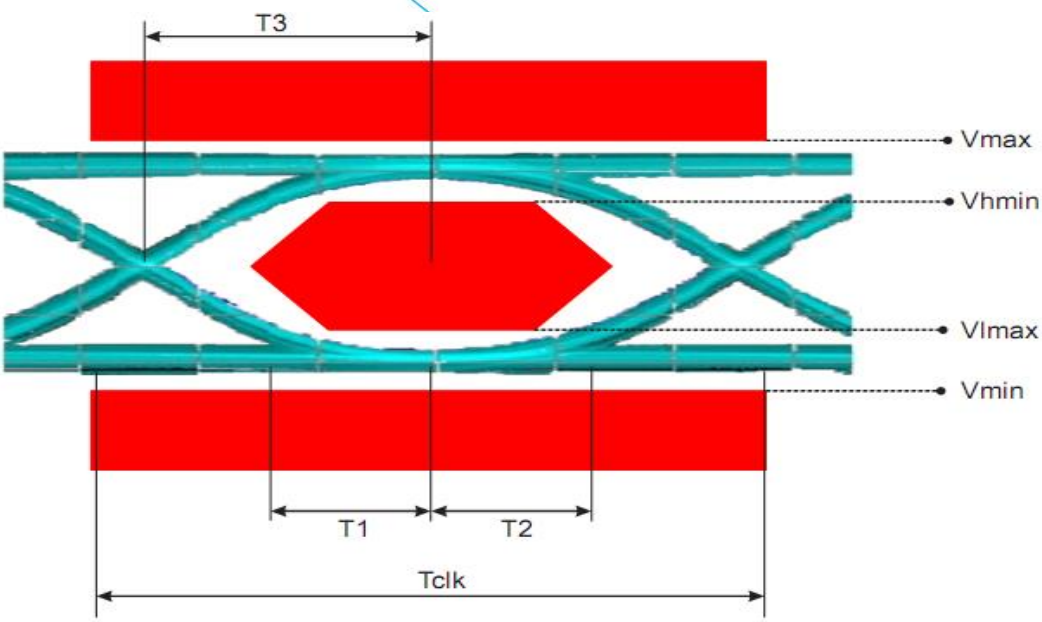


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Signal takta predstavlja jedan od najbitnijih signala u svakom digitalnom dizajnu. Rad celog sistema zavistan je od pravline izvedbe distribucije signala takta. Signal takta obično predstavlja i najbrži signal u sistemu. Za jednu periodu signala podataka potrebno je da postoje dve periode signala takta. Upravo ovaj signal, jedan je od najčešćih uzroka smetnji u *embedded* sistemima jer:

- Uglavnom je to signal najveće frekvencije.
- Snabdeva signalom takta sva integrisana kola u elektronskom sistemu, prostire se većim delom elektronske štampane ploče.
- Pravougaonog je oblika.

Dijagrami oka predstavljaju najefikasniji oblik analize integriteta signala. Prezentuju na jednom mestu sve neophodne informacije o tehničkim parametrima signala koji se posmatra. Obično se upotrebljavaju pri merenju usponske i silazne ivice signala, preskoka signala. Dijagrami oka predstavljaju skup impulsa različitih vrednosti prikazanih na jedinstvenom dijagramu, jedni preko drugih. Oblik svakog narednog impulsa zavistan je od prethodnog stanja tj. da li je prethodni impuls imao vrednost logičke 1 ili 0. Pri ovakvoj analizi koristi se pseudo slučajni niz bitova koji ima ulogu simulacije stvarnog protoka podataka. Na slici ispod prikazan je primer dijagrama oka sa objašnjenjem parametara koji ga opisuju.




$$T_1 = \frac{T_{set}}{T_j/_2} \quad ; \quad T_2 = \frac{T_{hold}}{T_j/_2} \quad ; \quad T_3 = \frac{T_{clk}}{T_S/_2}$$

- $T_{clk}$ = Perioda signala takta.
- $T_s$ = Razlika između kašnjenja signala takta i kašnjenja signala podataka.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- $T_j$ = Džiter podataka sabran sa vremenom signala takta.
- $T_s$ , $T_{hold}$ = *setup* i *hold* vremena integrisanog kola prijemnika.
- $V_{min}$ , $V_{max}$ = Minimalna i maksimalna dozvoljena vrednost napona na ulazu integrisanog kola.
- $V_{lmax}$ = Maksimalna vrednost napona na ulazu prijemnika za napon logičke nule.
- $V_{hmin}$ = Minimalna vrednost napona na ulazu prijemnika za napon logičke jedinice.



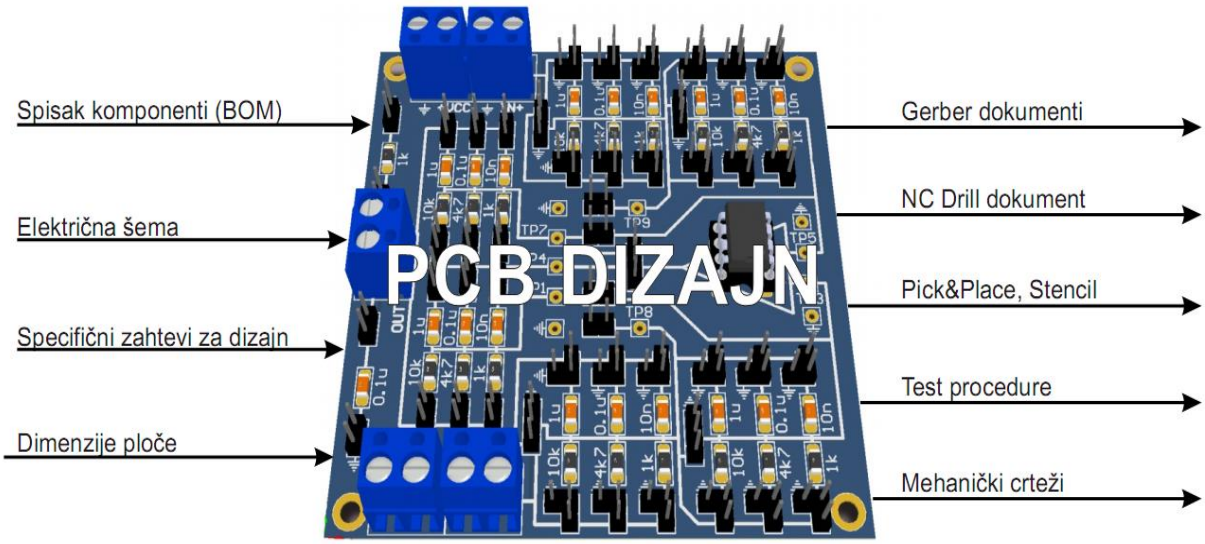
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 17 Dizajn štampanih ploča

Štampana ploča (*PCB, Printed Circuit Board*) predstavlja sistem električnih veza realizovanih na jednoj ploči na koju se mogu montirati elektronske komponente. Mogu biti različitih oblika, veličina i debljina i mogu biti realizovane u više slojeva. Prema načinu izrade elektronske štampane ploče možemo podeliti na tri tipa:

- Rigid.
- Flex.
- Rigid-Flex.

Pri procesu dizajniranja štampanih ploča potrebno je napraviti određene kompromise zavisno od toga kakav kvalitet finalne štampane ploče želimo i kakve ćemo tehnologije i resurse primeniti pri izradi. Da bi smo uopšte započeli dizajn elektronske štampane ploče neophodni su nam određeni ulazni dokumenti. Pre svega potreban je dokument električne šeme elektronskog sistema. Ovaj dokument opisuje međusobnu povezanost komponenata u elektronskom sistemu za koji radimo dizajn elektronske štampane ploče. Spisak komponenti (*BOM, Bill Of Materials*) predstavlja spisak svih komponenata koje će činiti finalni uređaj, te je poželjno da pre početka dizajna ploče ovaj dokument bude u potpunosti definisan. Specifični zahtevi za dizajn takođe trebaju biti definisani. U ovakve zahteve spadaju posebna pravila za *high-speed* dizajn.



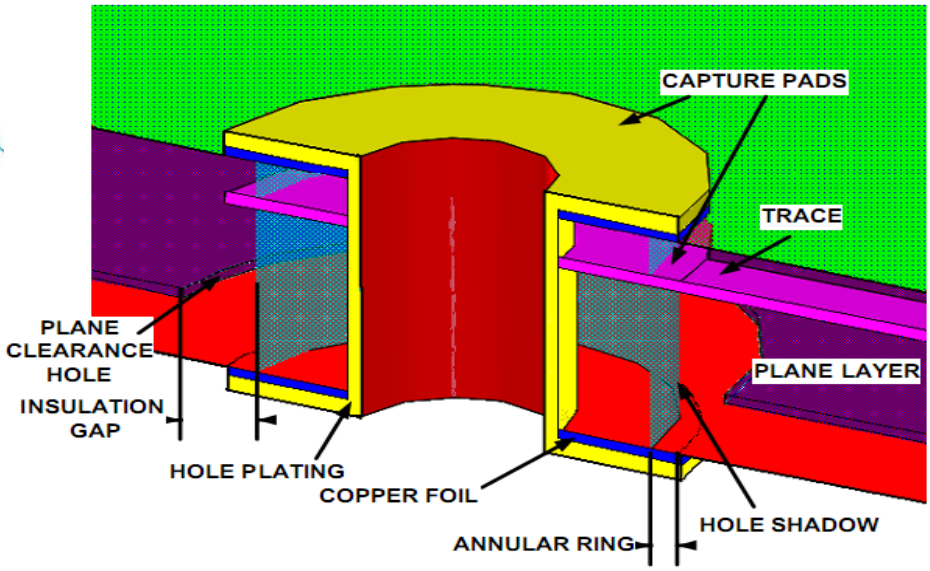

Gerber dokumenti predstavljaju izlaznu dokumentaciju na osnovu koje proizvođač ploča može napraviti elektronsku štampanu ploču. Oni čine skup dokumenata koji prikazuju svaki provodni sloj štampane ploče ponaosob, a mogu se koristiti i za prikaz parametara elektronske štampane ploče koji su od važnosti za proizvođača i/ili samog dizajnera. *NC Drill* dokument, takođe je neophodan proizvođaču. Ovaj dokument sadrži informacije o svim bušenim rupama kao što su pozicija, prečnik i postupak metalizacije koje koristimo na ploči. *Pick&Place* dokument i *Stencil* maska koriste se



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

prilikom automatskog asembliranja komponenata na ploču. *Pick&Place* dokument sadrži informacije o svakoj komponenti u našem dizajnu kao na primer poziciju svake komponente na ploči i njenu rotaciju.

Uloga via jeste povezivanje električnih signala koji se nalaze na različitim slojevima štampane ploče. Na slici ispod prikazana je via sa objašnjenjem parametara koji je opisuju.



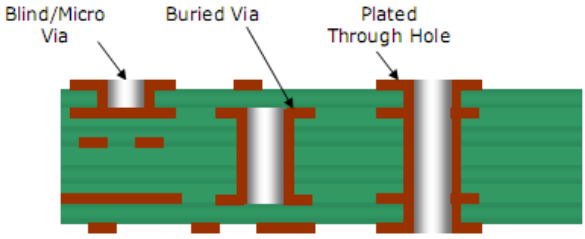

- *Capture pad* – predstavlja veličinu bakarnog prstena na spoljnim slojevima štampane ploče.
- *Hole shadow* – predstavlja zonu definisanu krugom unutar kojeg ne sme biti provodnih, bakarnih površina na svim slojevima štampane ploče koje nisu spojene za sam prsten vie. To je izolaciona zona unutar koje će biti izbušena rupa za viu.
- *Hole plating* – predstavlja sloj metalizacije koji je nanet da bi se ostvario kontakt između bakarnih površina koje se nalaze na različitim slojevima štampane ploče.
- *Copper foil* – bakarna folija koja obezbeđuje električne veze na spoljnim slojevima štampane ploče.
- *Insulation gap* – izolacioni materijal koji se nalazi između zone unutar koje ne sme biti provodnih, bakarnih površina na svim slojevima štampane ploče koje nisu spojene za sam prsten vie i ostalih bakarnih površina.
- *Annular ring* – predstavlja bakarni prsten čiji je prečnik veći od prečnika izolacione zone vie. Većeg je prečnika od izolacione zone kako bi uvek ostao deo bakra koji je povezan sa rubom bušene rupe.
- *Plane* – predstavlja bakarnu provodnu površ koja se nalazi na unutrašnjem sloju štampane ploče.

Postoji nekoliko vrsta via:

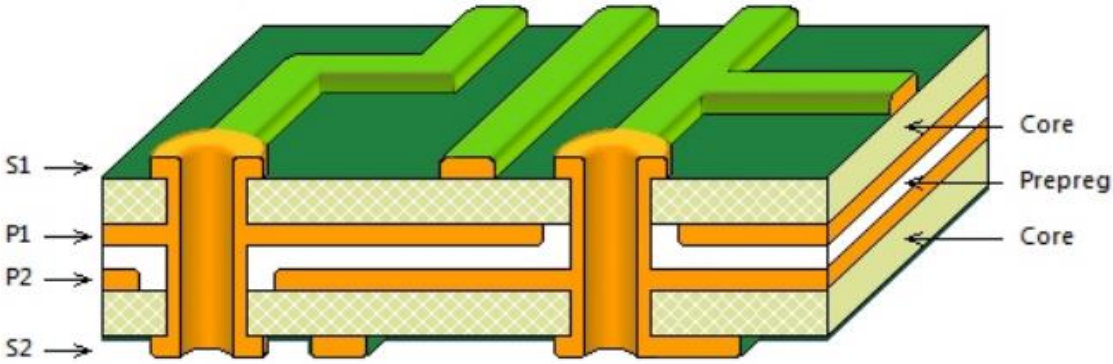


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Vie koje prolaze kroz celu ploču (*PTH, Plated Through Hole via*)
- Vie koje spajaju jedan spoljnji i jedan unutrašnji sloj (*Blind via*)
- Vie koje se nalaze unutar štampane ploče (*Burried via*)



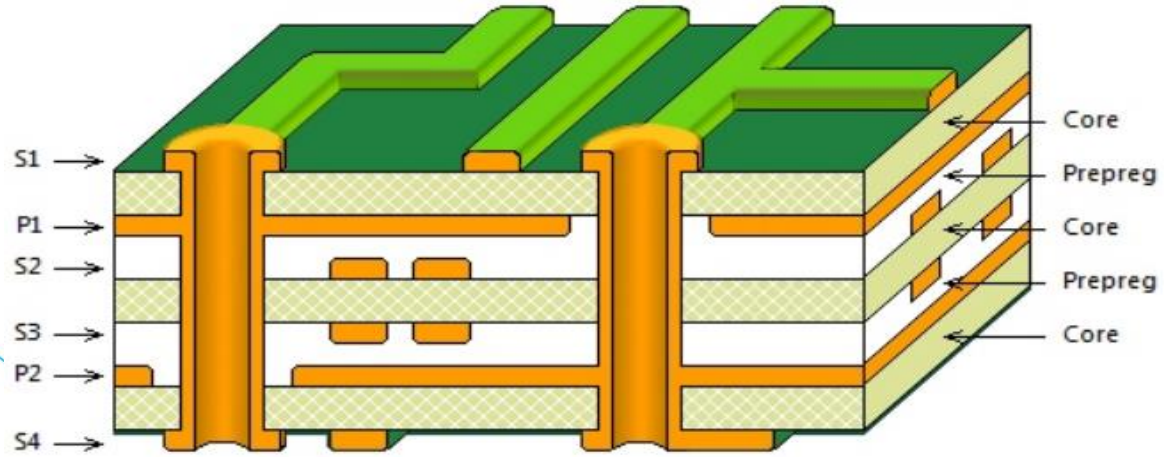

Izuzev jednoslojnih, štampane ploče se uvek izrađuju tako da sadrže paran broj provodnih slojeva. Između provodnih slojeva nalaze se neprovodni slojevi koji služe kao izolator. Pažljivim odabirom redosleda, debljina i uloga slojeva možemo uticati na integritet signala. Najjednostavnije štampane ploče izrađene su kao jednoslojne ili dvoslojne. Samim tim i cena njihove izrade je manja. Ukoliko želimo voditi računa o integritetu signala preporučljivo je odabrati najmanje četvoroslojnu štampanu ploču. Sa četvoroslojnom elektronskom štampanom pločom, električne signale i komponente možemo povezivati po spoljašnjim slojevima *S1* i *S2*, dok su za provodne površi rezervisani unutrašnji slojevi *P1* i *P2*, kao što je to prikazano na slici ispod.



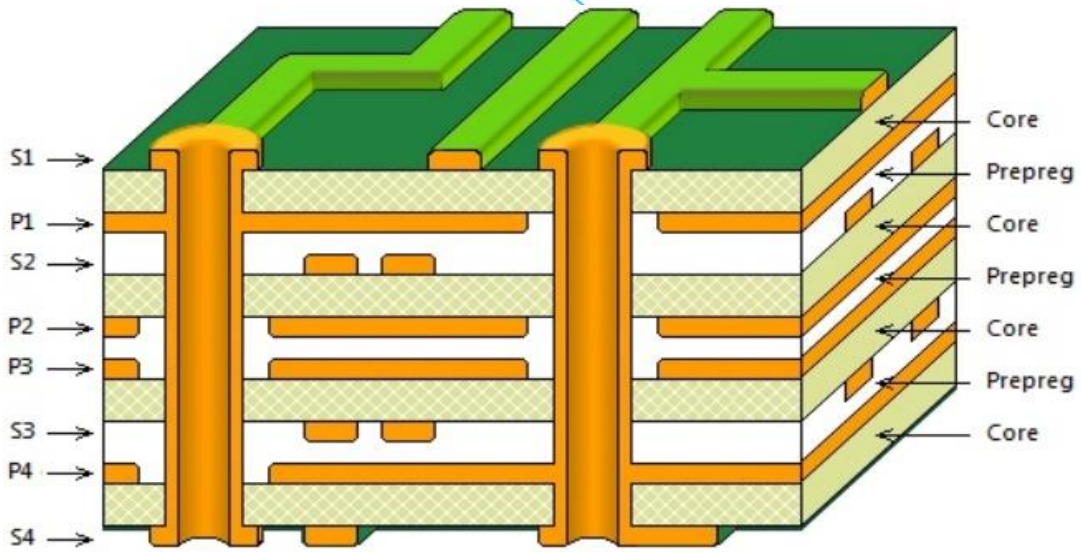

U slučaju kompleksnijeg dizajna, gde postoji potreba za povećanom interkonekcijom električnih signala, kao nadogradnja četvoroslojne štampane ploče koristi se ploča sa šest slojeva. Sa šestoslojnom elektronskom štampanom pločom, električne signale i komponente možemo povezivati po spoljašnjim slojevima *S1* i *S4*, električne signale po unutrašnjim slojevima *S2* i *S3*, dok su za provodne površi rezervisani unutrašnji slojevi *P1* i *P2*, kao što je to prikazano na slici ispod.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




Pored potrebe za povećanom gustinom interkonekcija električnih signala, dizajn određenih *embedded* sistema zahteva i veći broj različitih napona napajanja. Tip elektronske štampane ploče koji pored povećane gustine interkonekcija signala može obezbediti i više slojeva rezervisanih za provodne površi jeste osmoslojna štampana ploča. Sa osmoslojnom elektronskom štampanom pločom, električne signale i komponente možemo povezivati po spoljašnjim slojevima *S1* i *S4*, električne signale po unutrašnjim slojevima *S2* i *S3*, dok su za provodne površi rezervisani unutrašnji slojevi *P1, P2, P3* i *P4*, kao što je to prikazano na slici ispod.

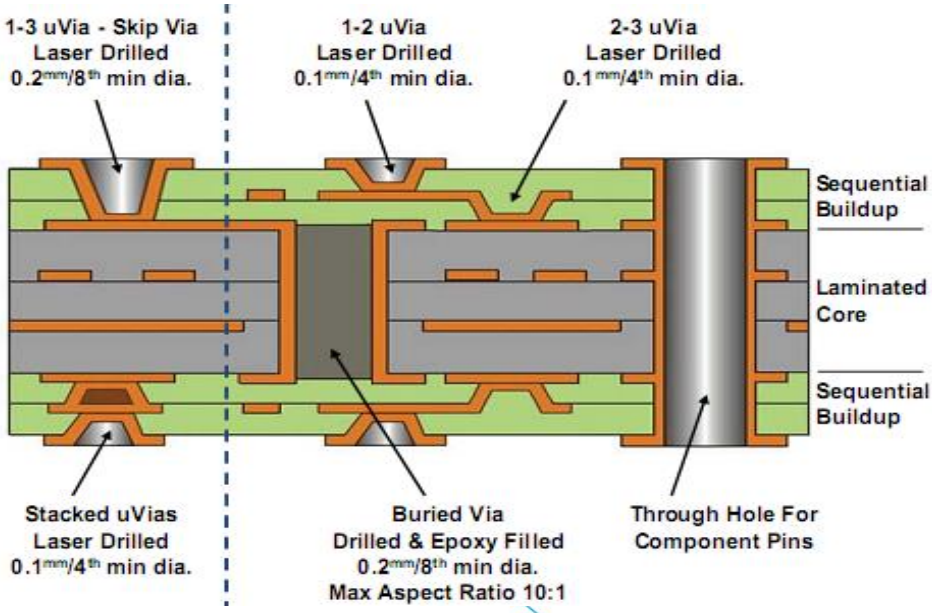



Razvoj tehnologije doveo je do toga da kućišta komponenata postaju sve manja dok se broj električnih veza povećava. Primer za takve elektronske komponente su one dostupne u *BGA* (*BGA, Ball Grid Array*) pakovanjima kod kojih nije redak slučaj da poseduju lemne kontakte razmaknute za



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

samo 0.4 milimetra. Takve komponente mogu posedovati i više od hiljadu lemnih izvoda. Da bi se omogućila implementacija ovakvih elektronskih kola na štampane ploče razvijena je *HDI* (*HDI, High Density Interconnect*) tehnologija izrade štampanih ploča. Ovakvom tehnologijom omogućeno je korišćenje via (*uVia*) čiji je prečnik bušene rupe 0.1 milimetar koje mogu povezivati susedne slojeve na štampanoj ploči i tako omogućiti znatno veću gustinu električnih veza. Na slici ispod prikazane su karakteristične strukture na elektronskoj štampanoj ploči koje omogućava upotreba *HDI* tehnologije.




Proces razvoja elektonskih štampanih ploča možemo podeliti na sledeće faze:

- Određivanje dimenzija elektronske štampane ploče.
- Postavljanje komponenti.
- Planiranje distribucije napajanja.
- Planiranje povezivanja signala.
- Odabir tipa i parametara štampane ploče.
- Povezivanje kritičnih signala.
- Povezivanje preostalih signala.
- Provera dizajna.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 18 Literatura


[31] H. Johnson, M. Graham, "High-speed digital design – A handbook of black magic", Prentice-Hall, 1993.

[32] Lee W. Ritchey, "Right the first time – A practical handbook on high-speed PCB and system design", Speeding Edge, 2003.

[33] Stephen C. Thierauf, "High-speed circuit board signal integrity", Artech house, 2004.

[34] Stephen H. Hall, Garret W. Hall, James A. McCall, "High-speed digital system design – A handbook of interconnect theory and design practices", John Wiley & sons, 2010.

[35] Tektronix, "Application note – Anatomy of an Eye Diagram", Tektronix, 2010.

[36] Alan Herr, "Getting EMC design right first time", EMC FastPass, 2014.

[37] James Colotti, "Printed Wiring Board (PWB) Design", Telephonics – Command systems division, 2003-2005.

[38] James Colotti, "EMC Design Fundamentals", Telephonics – Command systems division, 2003-2005.

[39] Eur Ing Keith Armstrong, "Advanced PCB design and layout for EMC", Cherry Clough Consultants.

[40] C. Pfeil, H. Holden, "HDI layer stackups for large dense PCBs", Mentor Graphics, Technical publication, 2007.

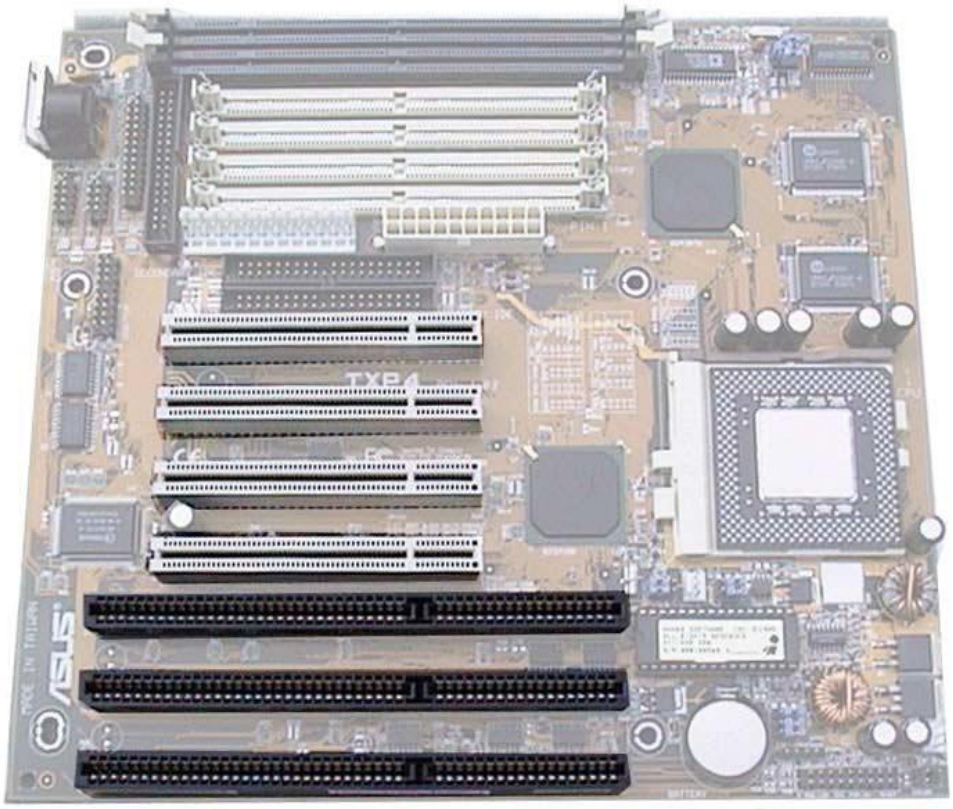
[41] Joseph Fjelstad, "Flexible Circuit Technology", BR Publishing Inc, 2006.

[42] Happy Holden, "The HDI Handbook", BR Publishing Inc, 2009.

# 19. **PCI Express Magistrala podataka**

### 19.1 PCI Express Arhitektura

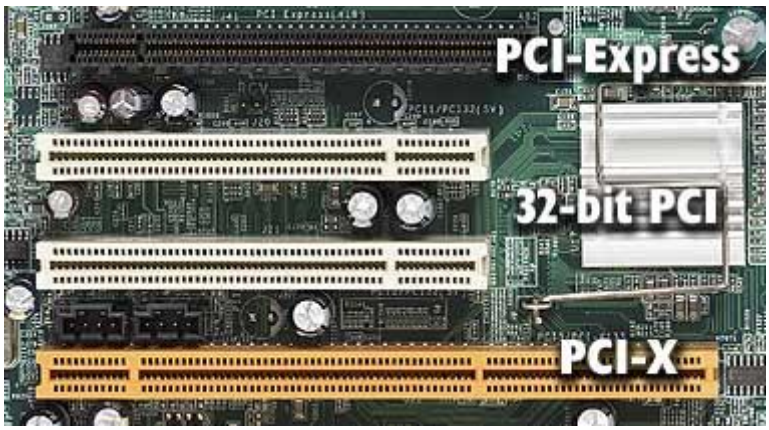
PCI Express (PCIe) je treća generacija komunikacione arhitekture (ISA -> PCI -> PCI Express). PCIe standard je definisan od strane konzorcijuma PCI-SIG https://www.pcisig.com/home. PCI-SIG čine velike kompanije u industriji poluprovodnika kao što su: Agilent, AMD, Dell, HP, Intel, Synopsys, NVIDIA i Qualcomm. PCIe zadržava mnoge karakteristike prethodne arhitekture (PCI) kao što su memorijski pristup, ulazno-izlazni pristup, kao i pristup konfiguraciji. Naovaj način omogućena je softverska kompatibilnost sa PCI arhitekturom. Operativni sistem vidi PCIe kao vrstu PCI magistrale. Implementacija PCIe magistrale je podigla brzinu prenosa, smanjila latence, povećala pouzdanost i omogućila podršku za različite prioritete prenosa podataka za različite uređaje. Na slikama 1,2,3 i 4 prikazani su različite vrste PC magistrala a u tabeli 1 uporedni pregled performansi.



**Slika 58 Slotovi PCI (beli) i ISA magistrale (crni)**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 59 PCI-Express, PCI i PCI-X slotovi**

**Slika 60 PCI-Express x16/x4/x1 slotovi (žuti) i PCI slot (zeleni)**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**Slika 61 PCI-Express Card i slot za PCI-Express Card modul**

| Tip Magistrale | Klok | Propusni opseg | Broj periferija |
|---|---|---|---|
| PCI 32 bit | 33 MHz | 133 MBajta/sec | 4-5 |
| PCI-X 32 bit | 66 MHz | 266 Mbajta/sec | 4 |
| PCI-X 32 bit | 266 MHz | 1066 Mbajta/sec | 1 |
| PCIe x1 | 2.5 GHz | 500 Mbajta/sec | 1 |
| PCIe x4 | 2.5 GHz | 2000 Mbajta/sec | 1 |
| PCIe x16 | 2.5 GHz | 8000 Mbajta/sec | 1 |

**Tabela 3 Poređenje performansi različitih magistrala**

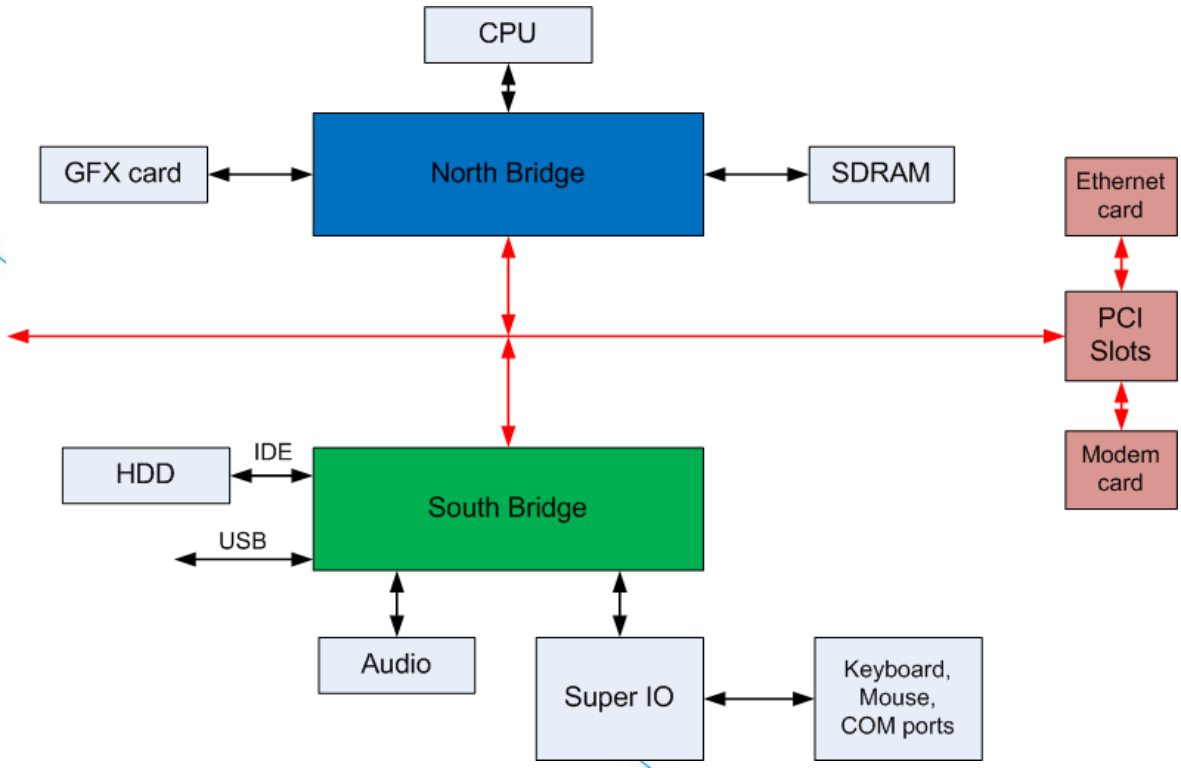
### 19.1.1 PCI Magistrala

PCI magistrala predstavlja 32 bitnu ili 64 bitnu magistralu sa paralelnim linijama (32 odnosno 64 linije za adresu i podatke). PCI magistrala sadrži i kontrolne signale kao što su FRAME#, DEVSEL#, TRDY#, IRDY#, STOP#, takođe postoje posebne linije za prekide i reset.Na PC računarima PCI slotovi imaju i posebne linije za napajanje. Na slici 5 je prikazana arhitektura sistema sa PCI magistralom.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Postoje tri metode razmene podataka na PCI magistrali: Programirani ulaz/izlaz, Direct Memory Access i Peer to Peer mod. Na slici 6 su prikazane ove vrste transfera na PCI magistrali.




**Slika 62 PCI arhitektura PC računara**

**Programirani ulaz/izlaz (Programmed I/O)**

Programirani ulaz/izlaz (Programmed IO) predstavlja način pristupa periferiji gde se procesor direktno obraća periferiji preko North Bridge čipa.

Procesor prvo šalje adresu uređaja sa kojim želi da komunicira na PCI magistralu i zatim čita podatke sa memorije uređaja i smešta ih u sopstvene registre da bi ih na kraju smestio u operativnu memoriju. Informisanje procesora da postoje podaci za prenos na kartici se obavlja putem prekida. Ovaj način komunikacije je dosta zahtevan u pogledu resursa procesora.

**Direct Memory Access (DMA)**

Drugi princip komunikacije je korišćenje DMA kontrolera na samoj PCI kartici (*bus master* princip). Sada DMA kontroler direktno prenosi informacije preko *North Brigde*-a u memoriju dok se procesor inforimiše o završetku transakcije putem prekida. Kada se podaci prenose u suprotnom smeru



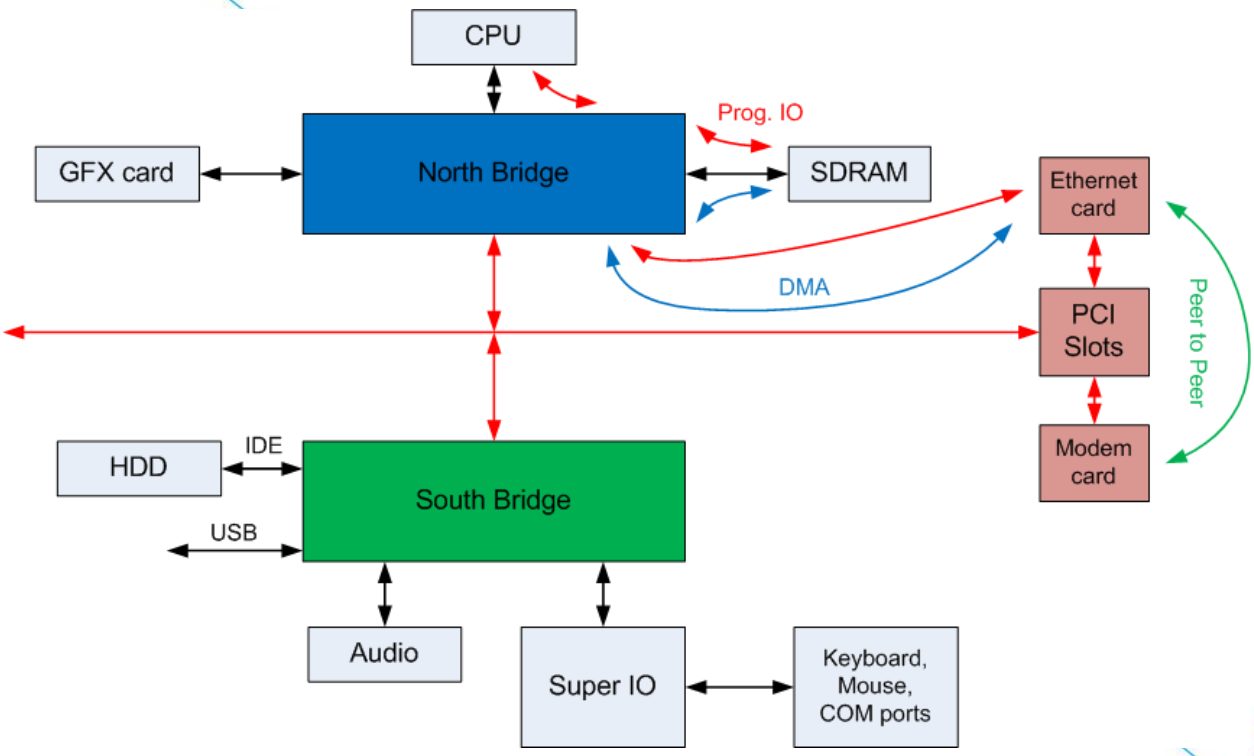
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

procesor upućuje DMA kontroler kartice na adresu u memoriji sa spremnim podacima dok DMA kontroler vrši sam prenos podataka iz memorije na samu karticu.

Ovaj metod komunikacije je dosta efikasniji u pogledu opterećenja procesora.

**Peer to Peer mod**

PCI magistrala omogućava i direktnu komunikaciju dva uređaja npr. Imeđu dve kartice na PCI magistrali. I u ovom slučaju master uređaj inicira komunikaciju slanjem adrese ka target uređaju. Target uređaj zauzima magistralu i započinje transakciju.




**Slika 63 PCI arhitektura – vidovi transakcija**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**Arbitracija na PCI magistarli**

Regulisanje pristupa PCI magistrali razrešava arbitar koji se nalazi na North Brigde-u. Arbitar daje dozvolu PCI uređaju da pristupi magistrali. PCI protokol ne pruža mogućnost odabira načina arbitracije.

Da bi se povećala raspoloživost magistrale u komunikaciji se primenjuje princip odloženih transakcija – *Retry* i *Disconnect* protokoli.

Ako se u momentu pristupa uređaju na PCI magistrali u njemu još ne nalaze podaci za slanje transfer se mora odložiti. Ovo se postiže na dva načina: ubacivanjem wait ciklusa ili odlaganjem transfera za sledeći ciklus arbitracije – *Retry*. *Retry* protokol omogućava efikasnije korišćenje magistrale kod dugih pauza.

*Disconnect* protkol je principski sličan *Retry* protokolu sa tom razlikom da se deo podataka prenosi u prvom obraćanju uređaju ali se za ostatak prenosa čeka drugi ciklus arbitarcije.

**Prekidi**

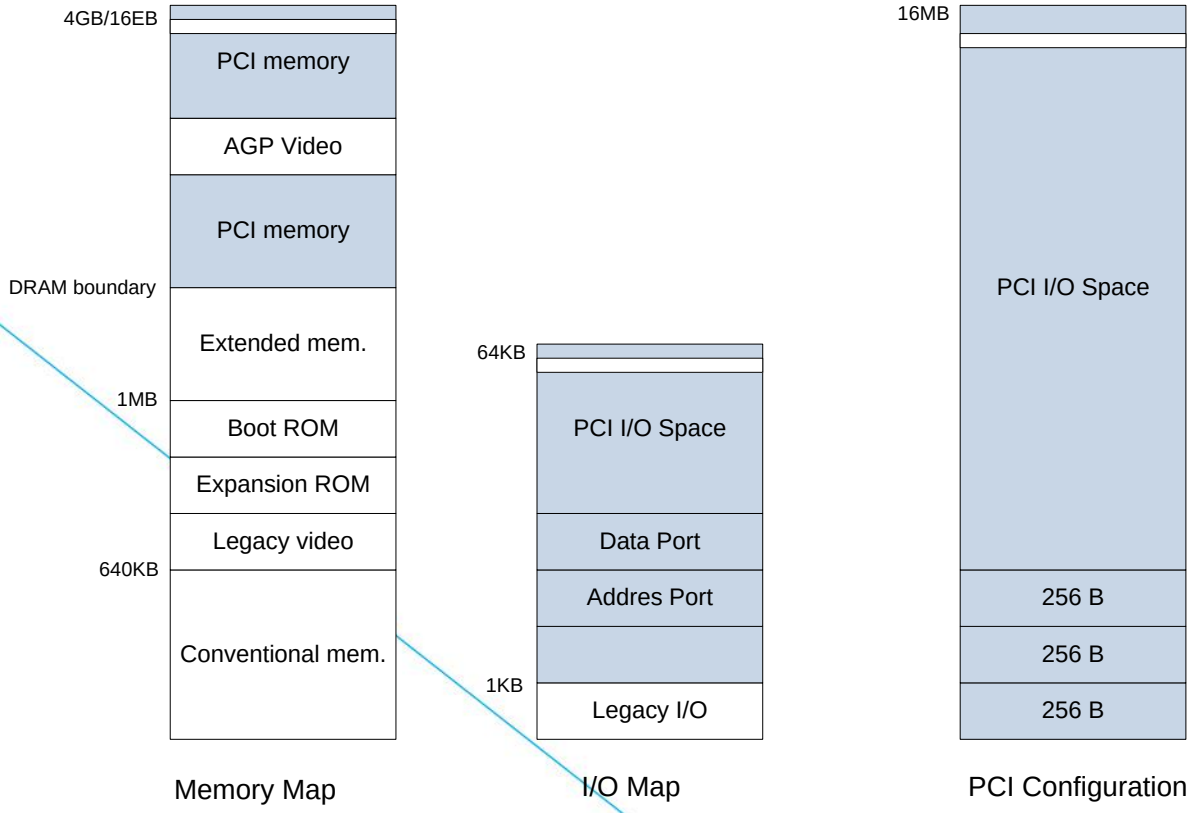
PCI uređaji šalju signale prekida preko četiri dodeljene linije (INTA#, INTB#, INTC#, INTD#) . Ove linije su deljene tako da operativni sistem mora pristupom odgovarajućem registru utvrditi koji je uređaj poslao prekid. Procesor se o prekidu informiše preko INT linije ili slanjem APIC (Advanced Programmable Interrupt Controller) poruke.

**Obrada grešaka**

PCI uređaji detektuju greške parnosti pri prijemu adrese ili podataka i aktivira odgovarajuće linije greške SERR# (adresa) tj. PERR# (podaci). Greška najčešće izaziva zaustavljanje sistema bez dalje obrade.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 64 PCI adresni proctor**

**Memorija x86 računara**

Memorijska mapa predstavlja celokupnu memoriju PC računara (Slika 7).

Najniži segment je I/O Space (do 64kB) nakon čega se nastavlja konvencionalna memorija do 640kB. Iznad ovoga je memorijski prostor rezervisan za stari opseg video memorije na video kartici (*Legacy video*).

Procesor ovoj memoriji pristupa kao da je to standardna memorija računara – Memory Mapped I/O (MMIO). Iznad stare video memorje nalaze se opsezi za pristup drugim periferijama do veličine od 1MB. Iznad ovog opsega se nalazi se nalazi RAM memorija – *Extended Memory* . Prostor rezervisan za PCI kartice se nastavlja iznad granice ove memorije – *PCI Memory*. Jedan segment memorije odvojen je za AGP video pa se opet nastavlja prostor za PCI uređaje.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**Pristup konfiguracionom prostoru**

Procesor pristupa PCI konfiguraciji preko severnog mosta. Upisom adrese u adresni port (Address Port) u I/O prostoru memorije se pristupa odgovarajućem registru u PCI konfiguraciji koja se može pročitati u Data port-u.

Kompleksniji PCI uređaji mogu direktno pristupati drugim PCI uređajima kao i memoriji preko severnog mosta.

**Konfiguracioni prostor**

PCI konfiguracioni prostor se može podeliti na dva dela:

- Donja 64 bajta standardnih registara koji imaju istu funkciju na svim uređajima
- Viših 192 bajta specifičnog prostora kojima pristupaju isključivo drajveri koji poznaju funkciju ovih registara

**Ograničenja PCI magistrale**

- Takt na PCI magistrali je ograničen na 66 MHz zbog složenih uslova propagacije signala
- Efikasnost PCI magistrale je oko 50-60%
- Obrada prekida je neefikasna usled deljenja linija
- Obrada grešaka je previše gruba. Svaka greška izaziva zaustavljanje procesora (CPU Halt)

### 19.1.2 PCI Express Magistrala

PCI Express magistrala povezuje uređaje po principu tačka-tačka. Ovde se koristi se diferencijalno signaliziranje po paru linija za slanje i prijem. Par linja za slanje i prijem čini jedan Lane dok jedan ili više Lane-ova čine Link. Broj Lane-ova u Link-u je ograničen na x1, x2, x4, x8, x12, x16 ili x32 (Slika 8).

**Diferencijalno signaliziranje**

Osnovna karakteristika diferencijalnog signaliziranja je da se napon na jednom paru linija (D+ i D-) menja simetrično. Pozitivna razlika napona između D+ i D - znači logičku 1 dok negativna razlika predstavlja logičku 0. Peak-to-peak razlika napona je u granicama od 800-1200mV na predajniku.

DC komponenta se eliminiše postavljanjem kondenzatora na predajniku. Impedansa prijemnika se usaglašava sa impedansom diferencijalnog voda na ploči tako da nema refleksija. Ovime se postiže mnogo veća robustnost signala u odnosu na *single ended* signalne linije.

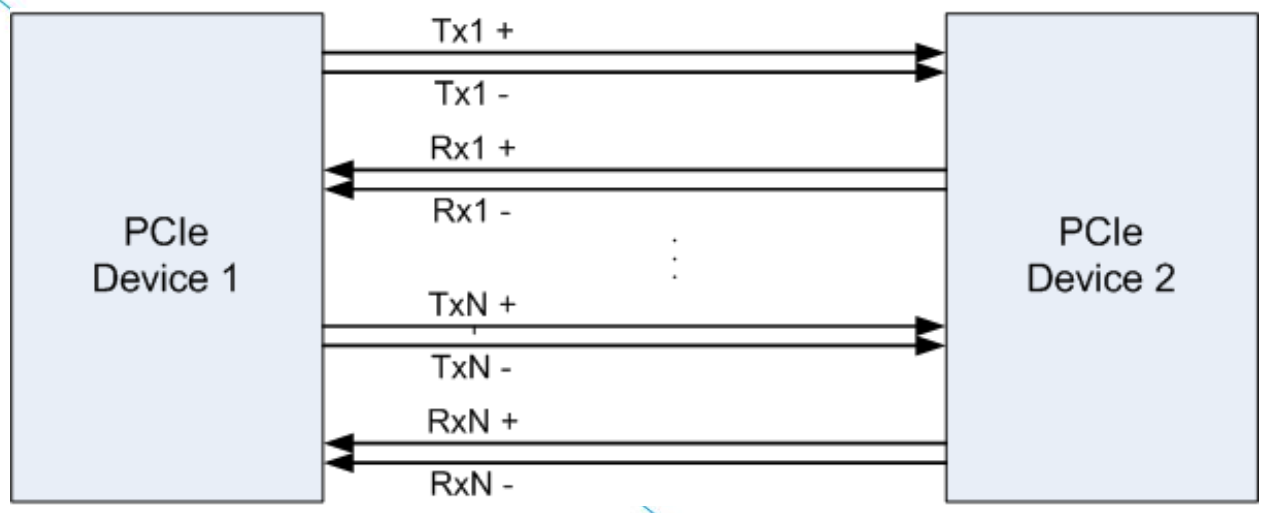
**Karakteristike**

Osnovne karakteristike PCI Express (PCIe) magistrale su:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Paketski protokol
- Brzina signaliziranja po Lane-u 2.5Gbit/sec za PCIe v1.0 standard, 5Gbit/sec za PCIe v2.0 i 8Gbit/sec za PCIe v3.0
- PCIe podržava isti adresni prostor kao PCI sa tim da je sa 256B povećan na 4KB



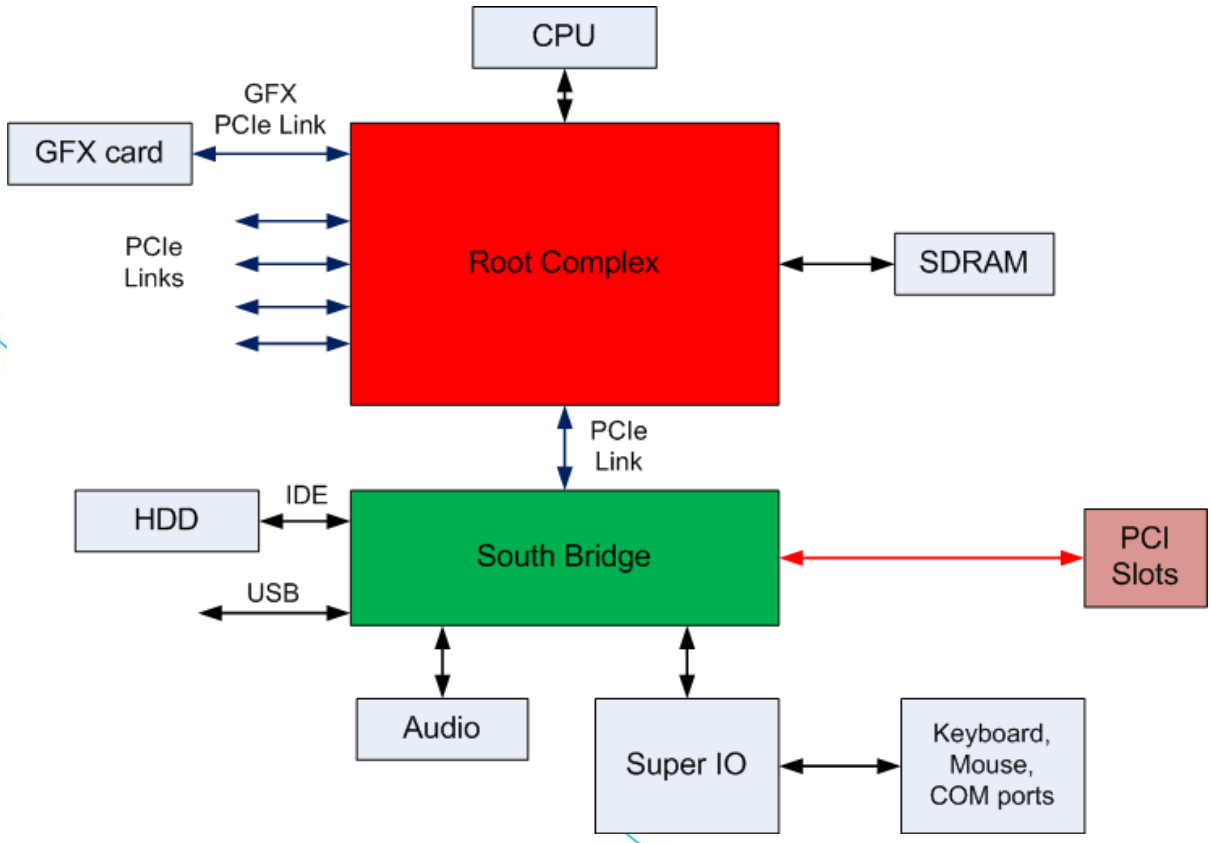

**Slika 65 Primer PCIe linka**

**PCIe Topologija**

PCI Express hijerarhija predstavlja ceo skup uređaja povezanih na root complex sa njihovim odnosima. Hijerarhijski domen predstavlja skup uređaja povezanih na jedan port root complex-a.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 66 Arhitektura jednog PCI Express sistema**

PCI arhitektura se sastoji od nekoliko tipova uređaja na PCI experss magistrali od kojih su najvažniji *Root Complex*, *Endpoint*, *PCIe Switch* (Slika 9).

Root Complex je blok koji spaja procesor i memoriju sa PCIe magistralom. Može imati jedan ili više PCIe portova. Jedan PCIe port sa krajnjim uređajem ili svičem predstavlja sub hijerarhiju. Root complex pokriva funkcije detekcije grešaka, obradu prekida i hot-plug kao i power management funkcije.

PCIe Endpoint je krajnji uređaj u PCIe lancu – periferijske kartice Ethernet, video, audio itd. Dele se na Legacy i PCI express endpoint-e. Svaki Endpoint ima ID koji se sastoji iz broja magistrale, broja uređaja i oznake funkcije (bus number, device number, function number). Mogu imati do 8 funkcija - Multi-Function Endpoints



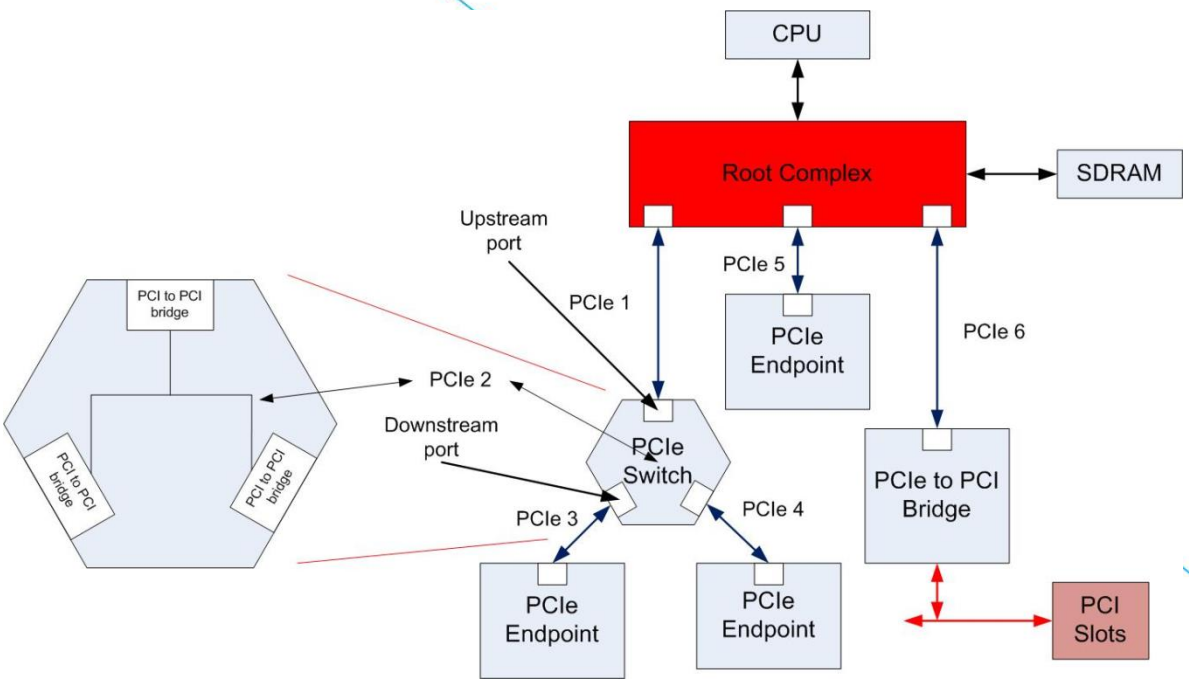
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

PCI Express-to-PCI(-X) Bridge povezuje PCIe i PCI magistralu. Requester uređaj koji pokreće PCIe transfer. Completer uređaj kome se obraća requester uređaj.

PCIe Port se nalazi između PCIe Linka i PCIe uređaja. Čine ga diferencijalni prijemnik i predajnik. Postoji nekoliko vrsta PCIe portova:

- Upstream port – port prema root complex-u
- Downstream port – port od root complex-a
- Ingress port – port koji prima paket
- Egress port – port koji šalje paket

PCIe svič je uređaj koji omogućava grananje PCIe magistrale tj. povezivanje jednog ili više endpoint uređaja ili svičeva na jedan PCIe port (Slika 10). Može se posmatrati kao dva ili više PCI-PCI mosta od kojih je svaki pridružen jednom PCIe portu. Svičevi moraju da obezbede tri tipa rutiranja paketa: rutiranje po adresi, ID-u ili implicitno.




**Slika 67 Arhitektura i pozicija PCI Express sviča u okviru PCI Express sistema**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**PCIe Transakcije**

Na PCIe sistemu je podržan PCI model transakcija (memory read/write, I/O read/write, configuration read/write) sa time da je dodat i novi vid transakcije preko poruka (Message transaction). Transakcije se dele na non-posted (očekuju potvrdu) i posted (ne očekuju porvrdu).

**Obrada grešaka**

Detekcija grešaka na PCIe sistemu se ostvaruje kontrolom CRC polja u svakom paketu. Kontrola se obavlja na link sloju (između predajnika i prijemnika paketa) bez intervencije u softveru. Postoji i opcija za end-to-end detekciju grešaka. Obrada grešaka može biti prosta kao kod PCI ali i složenija (RAS - Reliable, Available, Serviceable).

**Kvalitet servisa - Quality of Service (QoS)**

Funkcije za obezbeđivanje kvaliteta servisa na PCIe sistemu omogućavaju rangiranje paketa po prioritetu. Moguće je zahtevati određeni protok ili latencu u prenosu podataka kod real-time prenosa videa npr.

**Obrada prekida**

Prekidi se u PCIe arhitekturi prenose preko posebnih paketa po MSI (Message Signaled Interrupt) principu. PCI interapti (INTA#, INTB#, INTC# and INTD#) su podržani preko posebnih paketa i njihovo korišćenje se u principu izbegava kada je to moguće.

**PCI kompatibilan softverski model**

PCIe je u pogledu softvera kompatibilan sa PCI sistemima. Isti konfiguracioni prostor od 256B omogućava korišćenje starijih operativnih sistema koji ne poznaju PCIe. Omogućeno je proširenje ovog prostora na 4KB preko PCI Express enhanced configuration mehanizma.

## 19.2 PCIe Transakcije

Komunikacija na PCIe magistrali se obavlja razmenom TLP-ova (Transaction Layer packets). Transakcija predstavlja razmenu jednog ili više paketa da bi se obavila komunikacija između requester i completer uređaja. Transakcije se mogu grupisati u četiri grupe: memorijske, I/O, konfiguracione i poruke. Transakcije se mogu podeliti u non-posted i posted tranakcije.

Posted transakcije predstavljaju transakcije kod kojih se ne očekuje potvrda prijema odnosno podaci.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Non-posted transakcije predstavljaju razdeljene transakcije kod kojih se na paket sa zahtevom odgovara odgovarajućim paketom potvrde koji može sadržati i podatke ako je u putanju read transakcija. U tabeli 2 su prikazane sve vrste paketa na PCI Express magistrali.

| **TLP Packet Types** | **Abbreviated Name** |
|---|---|
| Memory Read Request | MRd |
| Memory Read Request - Locked access | MRdLk |
| Memory Write Request | MWr |
| IO Read | IORd |
| IO Write | IOWr |
| Configuration Read (Type 0 and Type 1) | CfgRd0, CfgRd1 |
| Configuration Write (Type 0 and Type 1) | CfgWr0, CfgWr1 |
| Message Request without Data | Msg |
| Message Request with Data | MsgD |
| Completion without Data | Cpl |
| Completion with Data | CplD |
| Completion without Data - associated with Locked Memory Read Requests | CplLk |
| Completion with Data - associated with Locked Memory Read Requests | CplDLk |

**Tabela 4 Tipovi PCIe transakcija**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 19.2.1 Non-posted read transakcija

Non-posted read transakcije započinje slanjem read request TLP paketa prema completer uređaju. Ovo mogu biti paketi tipa MRd, IORd, CfgRd0, CfgRd1 koje inicira root complex ili endpoint (izuzev CfgRd0, CfgRd1 paketa za endpoint). Paket se rutira kroz PCIe svičeve korišćenjem informacija u zaglavlju TLP paketa. Kada primi paket, completer uređaj dekoduje informacije iz njega i prikuplja odgovarajuće podatke. TLP paket (CplD) sa podacima (ili više njih) se rutira istom putanjom do request uređaja. Requester koristi tag polje u primljenom paketu kako bi utvrdio na koju transakciju i completer se taj paket odnosi. U slučaju da completer ne može da prikupi podatke šalje se CplD paket sa porukom o grešci. Requester uređaj odlučuje kako da obradi ovu grešku.

### 19.2.2 Locked non – posted read transakcija

Ovaj vid transakcije (MRdLk paket) je sličan prethodnom sa tom razlikom da ga može inicirati samo root complex i to prema legacy endpoint uređaju. Tokom trajanja transakcije cela putanja između dva uređaja se zaključava. U slučaju greške putanja se oslobađa a requester uređaj obrađuje grešku.

### 19.2.3 Non – posted write transakcija

Ovaj vid transakcije je sličan non – posted read transakciji i obuhvata transakcije (IOWr, CfgWr0, CfgWr1). Mogu ga inicirati i root complex i endpoint. Razlika je u tome da se sada šalju podaci od requester uređaja ka completer uređaju.

### 19.2.4 Posted write transakcija

Ovaj vid transakcije je sličan non – posted write transakciji sa time da se ne očekuje potvrda upisa podataka. Na ovaj način skraćuje se vreme za prenos podataka jer nema povratnih paketa.

### 19.2.5 Message transakcija

Poruke su vid transakcija koje se mogu slati od requester do completer uređaja ali mogu biti i broadcast (od root complex-a do svih endpoint-a). Poruke su paketi posted tipa tj. ne očekuje se potvrda prijema. Postojanje poruka eliminiše posebne linije koje bi se koristile za ovu namenu (prekide npr.)

## 19.3 Slojevi na PCI Express magistrali i paketski prenos

PCI Express magistrala funkcioniše na principu razmene paketa. Sloj predajnika kreira TLP pakete u sloju transakcije prosleđuje ih nižim slojevima koji na paket dodaju svoja zaglavlja. Sloj prijemnika

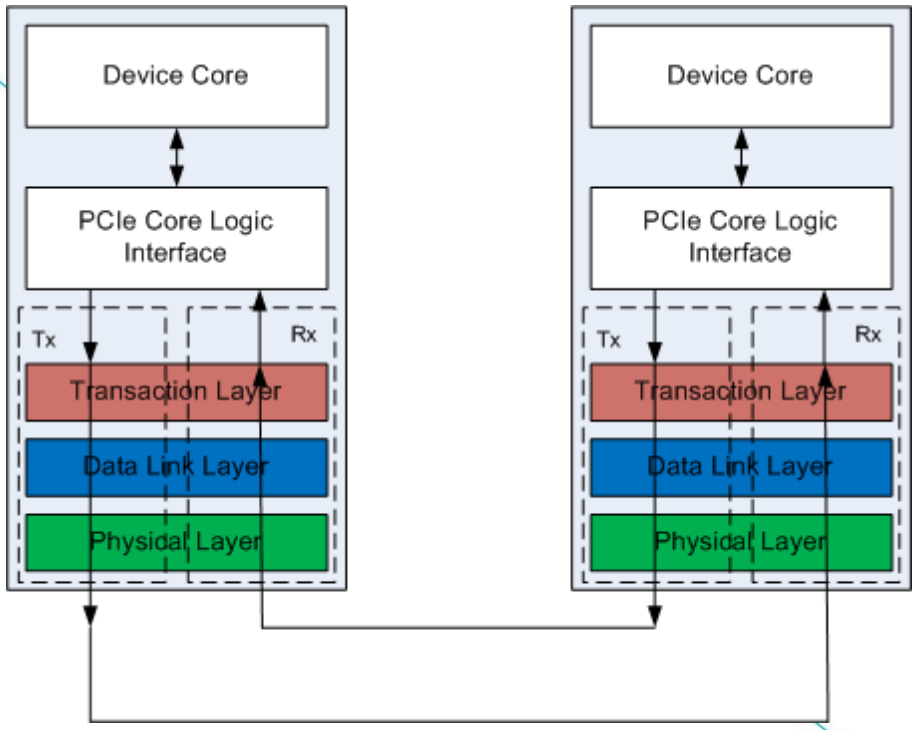


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

funkcioniše na sličan način gde niži slojevi obrađuju i skidaju svoja zaglavlja sa primljenog paketa i prosleđuju paket višem sloju. Na slici 11 su prikazani slojevi na PCI Express magistrali.

### 19.3.1 Transaction Layer (TL) Packet

TLP paket se sastoji iz zaglavlja, podataka i opcionog polja za proveru integriteta podataka sa kraja na kraj – ECRC. ECRC omogućava dodatnu proveru od strane uređaja kome je paket upućen. Na takav paket niži slojevi dodaju svoja polja: Sequence number, LCRC, Start i End.




**Slika 68 Prikaz slojeva na PCIe magistrali**

### 19.3.2 Data Link Layer (DLL) Packet

DLL paketi se razmenjuju između dva susedna uređaja i obavljaju funkciju upravljanja linkom. Postoje paketi za potvrdu prijema, kontrolu potrošnje i kontrolu protoka podataka.

Paketi se sastoje iz informacionog polja koje definiše tip paketa i polja za proveru na greške (LCRC).



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 19.3.3 Physical Layer (PhL) Packet

Paketi na fizičkom sloju se razmenjuju između dva susedna PCIe uređaja.

Paketi služe za trening linka kao i kompenzaciju razlike u taktovima.

### 19.3.4 Funkcije slojeva

Iznad sloja transakcije se nalaze logika samog uređaja (device core). Device Core je zadužen za generisanje samog sadržaja paketa, tip paketa, adresu primaoca kao i klase transakcije ili tip poruke ako se ona šalje.

U sloju transakcije se nalaze blokovi za generisanje TPL paketa kao i blokovi za kontrolu protoka, virtuelne kanale kao i redosled slanja/prijema paketa (ordering).

Virtuelni kanali (VCs) se koriste za određivanje prioriteta prosleđivanja paketa i imaju svoje bafere u okviru ovog sloja gde se smeštaju paketi za slanje tj. primljeni paketi.

### 19.3.5 Polja TLP paketa

TLP paket se sastoji od zaglavlja veličine 3-4 DW (double word) koje sadrži: adresu, TLP tip, veličinu podataka, requester ID/completer ID, tag, klasu transfera i atribute.

Veličina polja za podatke je od 1 do 1024 DW i oni se mogu preneti u jednom ili više paketa.

### 19.3.6 Kontrola protoka

Kontrola protoka osigurava da ne dođe do prepunjavanja VC bafera prijemnika. Prijemnik periodično šalje DLLP pakete koji signaliziraju stanje bafera na strani prijemnika. Ovaj proces je potpuno transparentan sa strane softvera u kome samo postoje dodatni baferi za skladištenje primljenih paketa.

### 19.3.7 Kvalitet servisa (Qos)

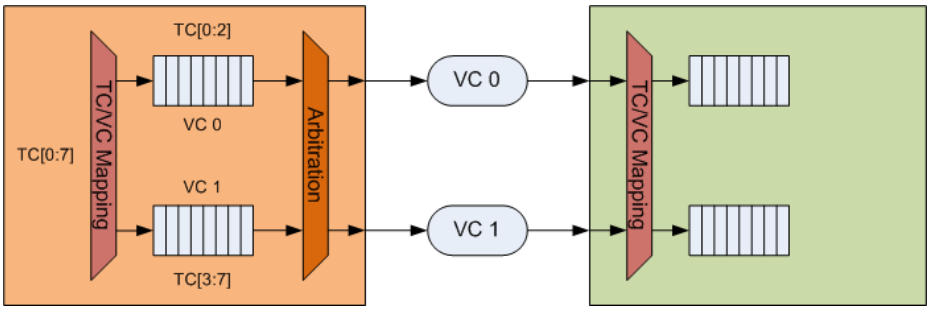
Paketi se po željenom prioritetu prenosa svrstavaju u klase prenosa (Traffic classes) od 0 do 7. TC se mapiraju u do osam virtuelnih kanala – virtual channels (VC) sa time da se više TCs-a može mapirati u jedan VC ali se jedan TC ne može mapirati u više VC (Slika 12).

Virtuelni kanali predstavljaju grupe klasa prenosa. Postoji maksimalno osam virtuelnih kanala u PCIe uređaju. Prioritet i rutiranje paketa se vrše prema pripadnosti određenom VC-u.

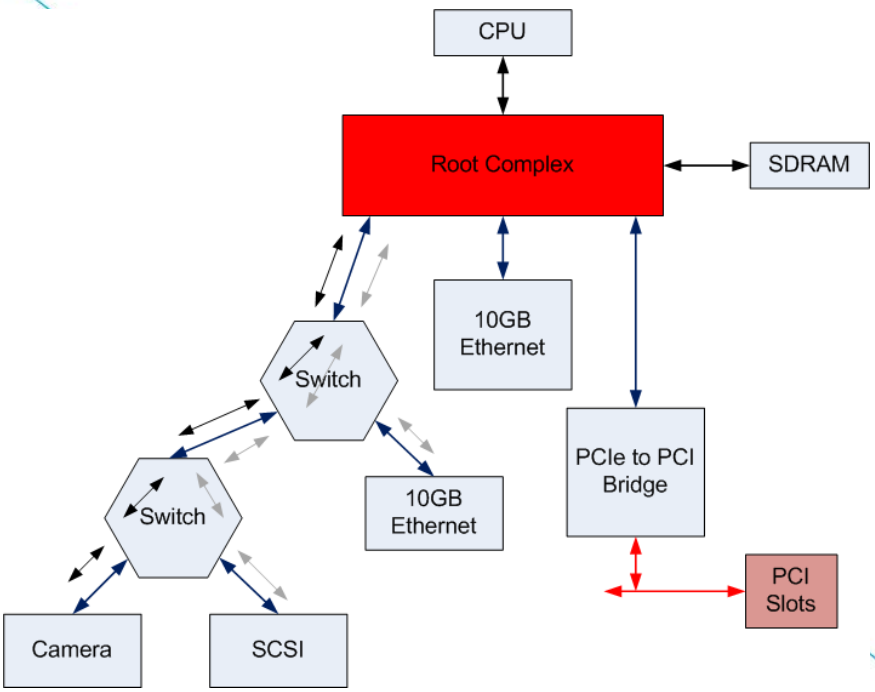
Kvalitet servisa se implemetira preko klasa prenosa (TC) i virtuelnih kanala (VC). Na uređaju može postojati do osam (ili manje) VC u koje se mapiraju TC. Viši VC imaju prednost pri rutiranju od nižih VC. Postojanje ovog mehanizma QoS je jedna od prednosti PCIe magistrale u odnosu na PCI magistralu. Na slici 13 je prikazan način rutiranja paketa sa prioritetom.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 69 Prikaz obezbeđivanja kvaliteta servisa na PCI magistrali**




**Slika 70 Prikaz rutiranja paketa sa prioritetom na PCI magistrali**

### 19.3.8 Arbitracija u PCI Express svičevima

PCIe svičevi moraju vršiti ulogu arbitracije paketa odnosno odlučivanja koji paket će biti prosledjen sa ulaznog na izlazni port. Zahtevi koji se postavljaju pred arbitraciju jeste da podrži različite vidove



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

prenosa podataka preko PCIe magistrale kao i da omogući odgovarajući propusni opseg i garantovano nisku latencu.

PCIe svičevi poseduju dva mehanizma arbitracije: po portu (sa kog porta se uzima paket) i VC kanalu kome pripada paket. Na taj način se određuje prioritet prosleđivanja paketa od ulaznih ka izlaznim portovima.

Arbitracija po portu se odvija između dva paketa koji dolaze po dva različita porta i prenose se u VC istog ranga. Šeme arbitracije koje se ovde primenjuju su round-robin (kružno uzimanje paketa sa svih portva), weighted round robin (neki portovi se ponekad preskoče čime drugi imaju prioritet) ili programabilni round robin baziran na vremenskim slotovima.

Arbitracija po VC se obavlja posle arbitracije po portovima. U ovoj arbitaciji se određuju koji paketi će se preneti zavisno od VC u kom pripadaju. Ovde se koristi striktno princip prednosti, round robin ili weighted round-robin arbitracija. Pre konkurisanja VC moraju da zadovolje uslove kontrole protoka i redosled isporuke paketa.

### 19.3.9 Kontrola grešaka

Prenos paketa po DLL sloju se bazira na potvrdi prijema preko ACK/NAK paketa. Paketu se dodaje sequence ID u zaglavlje a on sam se čuva u baferu do potvrde isporuke. Ako prijemnik po CRC proveri utvrdi ispravnost podataka on šalje ACK paket sa sequence ID paketa čiji prijem potvrđuje.

Po prijemu ACK paketa predajni uređaj briše isti paket iz bafera. U slučaju greške na prijemu šalje se NAK paket a predajni uređaj radi retransmisiju paketa. Ukoliko predajni uređaj primi četiri uzastopna NAK odgovora prijavljuje se greška i radi se ponovni trening linka.

### 19.3.10 Fizički sloj (Physical Layer)

Fizički sloj se sastoji iz logičkog i električnog sloja. Paketi primljeni od DLL sloja se obrađuju u logičkom sloju dodavanjem start i end bajta. Ovi bajtovi su neophodni za određivanje početka i kraja paketa. Podaci se zatim skrembluju da bi se smanjila ponavljanja određenih šema bita radi smanjivanja zračenja.

Nakon skremblovanja paketi se koduju putem 8b/10b koda čime se eliminišu DC komponente signala i radi lakše sinhronizacije. Paket se zatim premešta u šift registar i šalje na liniju preko diferencijalnog predajnika brzinom od 2.5GBit/sec (PCIe ver 1.0).

Na prijemnoj strani ponavlja se isti proces u obrnutom smeru.

### 19.3.11 Trening linka na fizičkom sloju

U toku treninga linka vrši se konfiguracija i inicijalizacija linka.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Dizajnom PCIe magistrale omogućava se arbitrarno povezivanje uređaja gde se neke karakteristike automatski konfigurišu i to:

- Link width – moguće je povezati uređaje sa različitim brojem putanja u linku.
- Lane reversal – moguće je povezati dva uređaja gde se putanje ukrštaju što će biti automatski prepoznato. Ovako se postiže lakše rutiranje ploče.
- Polarity inversion – ukrštanje D+ i D- para će biti prepoznato.
- Link data rate – prepoznavanje maksimalne brzine linka 2.5 Gbit/sec ili 5 Gbit/sec ili više.
- Lane-to-Lane De-skew – omogućene su različite dužine putanja. U tom slučaju prijemnik mora biti u stanju da uvede odgovarajuće kašnjenje između različitih putanja. Omogućene su dužine linka od čak 20 do 30 inča.

### 19.4 Rutiranje na PCI Express magistrali

Kod PCIe magistrale veze između uređaja su point to point tipa i rutiranje je neophodno da bi se paketi preneli preko magistrale. Svaki PCIe uređaj je povezan sa jednim ili više okolnih uređaja. Pri prihvatanju paketa na ulaznom portu vrši se kontrola paketa na greške i vrši se odlučivanje kako se obrađuje pristigli paket.

Paketi se mogu obraditi na tri načina:

- Paket se prihvata i obrađuje interno u PCIe uređaju
- Paket se prosleđuje ka odgovarajućem izlaznom portu
- Paket se odbacuje jer ne pripada ni prijemnom uređaju niti nekom uređaju koji je povezan na portove uređaja (postoje i drugi razlozi za odbacivanje)

Paketi se prema kontrolnim simbolima koje nose dele u tri klase Ordered sets, DLLP i TLP. Paketi tipa Ordered sets i DLLP se prihvataju i obrađuju interno jer ne sadrže nikakve informacije za rutiranje i služe samo za održavanje linka. TLP paketi poseduju informacije bitne za rutiranje i mogu se kretati duž PCIe magistrale.

Rutiranje preuzimaju na sebe uređaji sa više portova tj. PCIe svičevi ili opciono Root complex. Endpoint uređaji imaju samo jedan port tako da ne mogu rutirati pakete nego ih prihvataju ili odbacuju. Da bi se rutiranje moglo izvršiti potrebno je ustanoviti strategiju rutiranja. Rutiranje se bazira na tipu TLP paketa, sistemskoj memoriji i IO adresnom prostoru i to tako što se prilikom enumeracije svakom uređaju dodeljuju odgovarajućiu resursi.

Ordered sets pakete kreira fizički sloj i služe za inicijalizaciju linka, kompenzaciju takta i prevođenje linka iz idle stanja. Paketi se sastoje iz simbola od 10 bita i to 4 ili 16 karaktera. Postoji pet vrsta paketa Fast Training, Training seq 1, Training seq 2, Idle i Skip.

DLL pakete generiše DLL sloj i imaju funkcije kontrole prijema paketa (ACK/NAK), kontrole potrošnje i kontrole protoka podataka. Paketi sadrže 16 bitni CRC koji služi za proveru ispravnosti paketa a takođe postoje i timeout mehanizam u slučaju da se paket ne detektuje uopšte.

TLP paketi se kreću od TLP sloja jednog PCIe uređaja do TLP sloja drugog PCIe uređaja. Prema tipu adresnog opsega koji paket koristi dele se na: memorijske, IO, konfiguracione i poruke. U komunikaciji između uređaja koristi se split transaction (non-posted) metod gde se odvajaju paketi sa zahtevom (requester) i paketi sa odgovorom (completer).

Primenom split transaction metoda link se oslobađa za druge transfere što je prednost u odnosu na PCI magistralu. Neki zahtevi se mogu tretirati kao posted transakcije gde se ne traži povratni paket kao što su: memorijski upis i poruke.

Postoje tri principa za rutiranje TLP paketa: Adresno rutiranje, ID rutiranje i Implicitno rutiranje (Tabela 3). PCIe zadržava kompatibilnost sa PCI u slučaju rutiranja jer su adresni i ID princip isti kao kod PCI magistrale

PCIe dodaje jedan novi tip rutiranja a to je implicitno rutiranje koje se koristi kada je pozicija primaoca fiksna kao u slučaju root complex-a ili susednog uređaja.

| **TipTLP paketa** | **Metod rutiranja** |
|---|---|
| Memory Read (MRd), Memory Read Lock (MRdLk), Memory Write (MWr) | Address Routing |
| IO Read (IORd), IO Write (IOWr) | Address Routing |
| Configuration Read Type 0 (CfgRd0), Configuration Read Type 1 (CfgRd1) Configuration Write Type 0 (CfgWr0), Configuration Write Type 1(CfgWr1) | ID Routing |
| Message (Msg), Message With Data (MsgD) | Address Routing, ID Routing, |

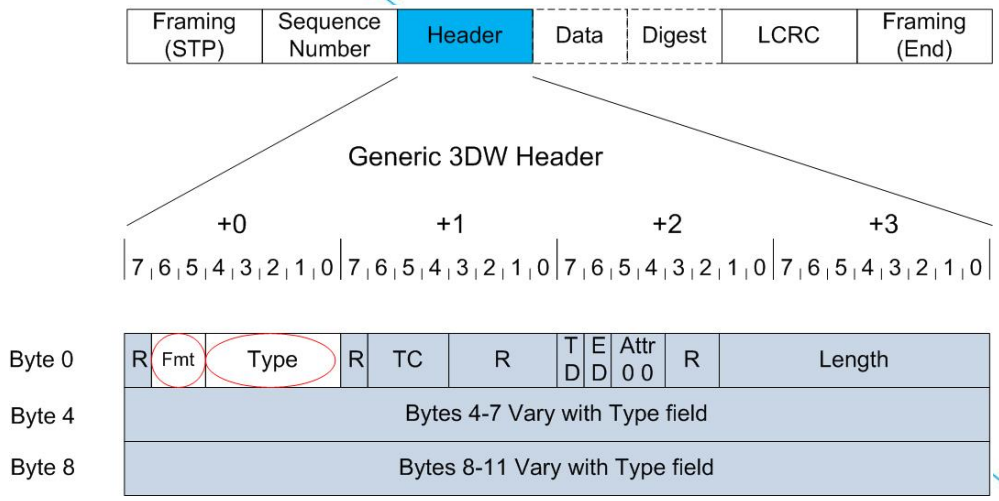
| | |
|---|---|
| | or Implicit routing |
| Completion (Cpl), Completion With Data (CplD) | ID Routing |

**Tabela 5 Prikaz tipova rutiranja različitih paketa na PCI Express magistrali**

PCIe koristi poseban tip paketa koji nije postojao na PCI magistrali – poruke.

Poruke služe da zamene određene linije signala koje su postojale na PCI magistrali ali ne i na PCIe kao što su: prekidi, greške, kontrola potrošnje. Poruke podržavaju implicitno rutiranje kojim se izbegava korišćenje posebnih memorijskih adresa za ove pakete.

TLP paketi se po pristizanju proveravaju na greške i zatim se obrađuje Type i Format u Header polju paketa čime se utvrđuje koji tip paketa je u pitanju (Slika 14). Paket se zatim obrađuje interno, prosleđuje dalje ili se odbacuje.




**Slika 71 Zaglavlje TLP paketa**

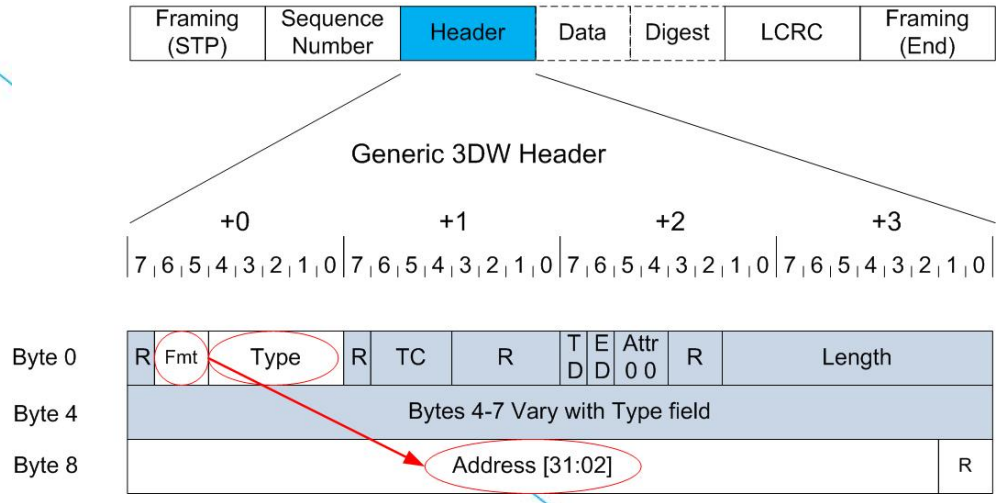
Iz pristiglog TLP-a se izvlače informacije neophodne za rutiranje koje se koriste za određivanje tipa rutiranja. Registri u konfiguraciji PCIe uređaja (Base Address Registers, Base/Limit registers i Bus Number registers) se zatim porede sa ovim vrednostima i vrši se rutiranje paketa.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 19.4.1 Adresno rutiranje TLP paketa

Po prijemu TLP-a koji traži adresno rutiranje pristupa se adresi koja se u Header delu paketa nalazi u bajtovima 8-11 ako je u pitanju 32-bitna adresa ili u bajtovima 8-11 (high DW) i 12-15 (low DW) u slučaju 64-bitne adrese (Slika 15). U slučaju da je prijemni uređaj endpoint ova adresa se poredi sa vrednošću Base address registara (od 0 do 5). Ukoliko adresa odgovara paket se prihvata i obrađuje a u suprotnom se odbacuje (Slika 16).




**Slika 72 Pozicija 32 bitne adrese u zaglavlju TLP-a**

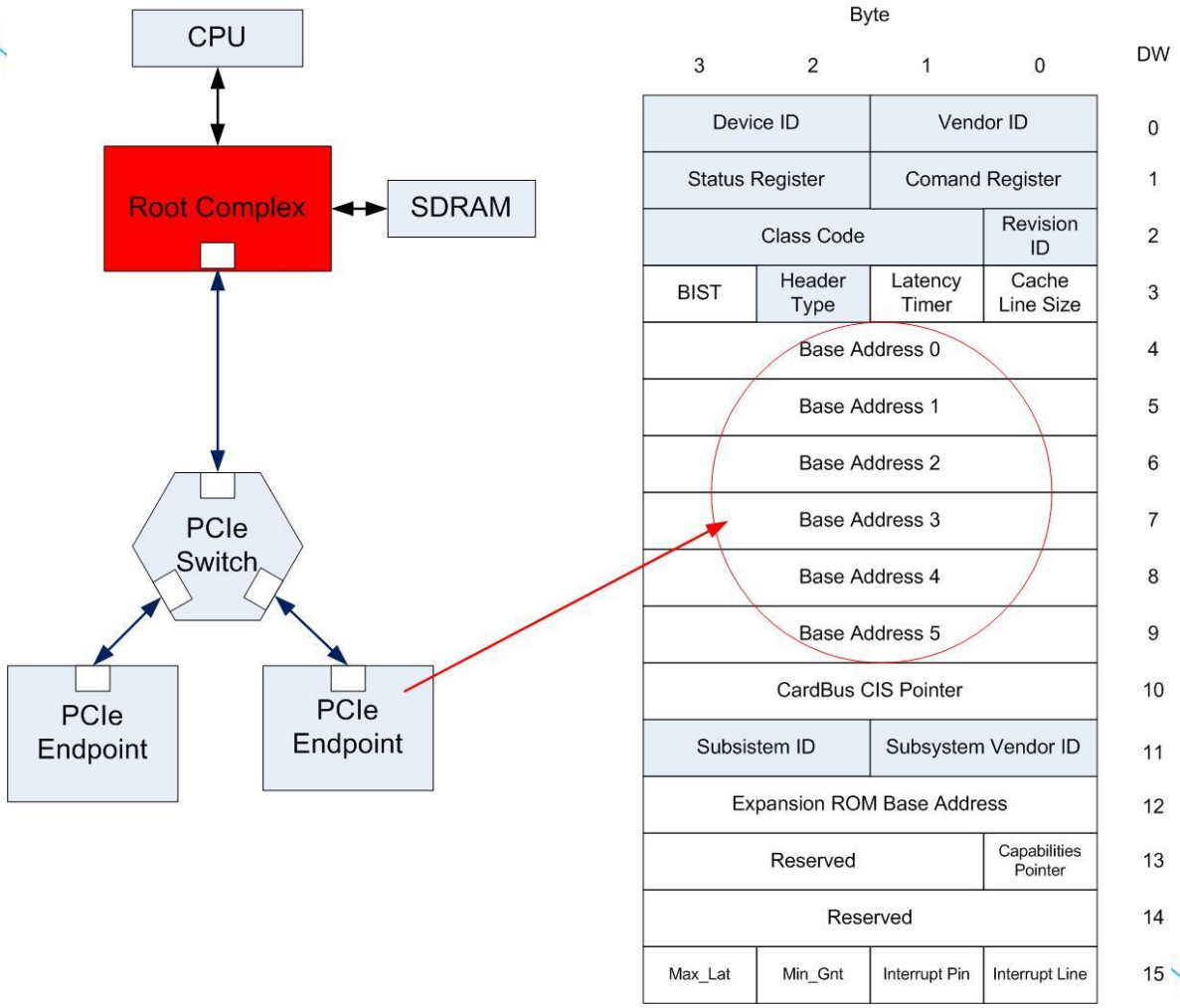
U slučaju kada je prijemni uređaj PCIe svič moguće je i prosleđivanje paketa.

Adrese se prvo porede sa Base Address 0/1 registrima i ako se poklope paket se obrađuje u sviču. Ako nema poklapanja porede se registri konfiguracionog prostora: non-prefetchable memory (MMIO), prefetchable Base/Limit registers ako transakcija cilja memoriju ili I/O Base and Limt registers ako transakcija cilja I/O proctor. Ako nema poklapanja adresa paket će biti odbačen.

Ako paket cilja opseg adresa uređaja iznad sviča biće prosleđen gore u svakom slučaju izuzev ako nisu usmereni na link na dole (Slika 17).



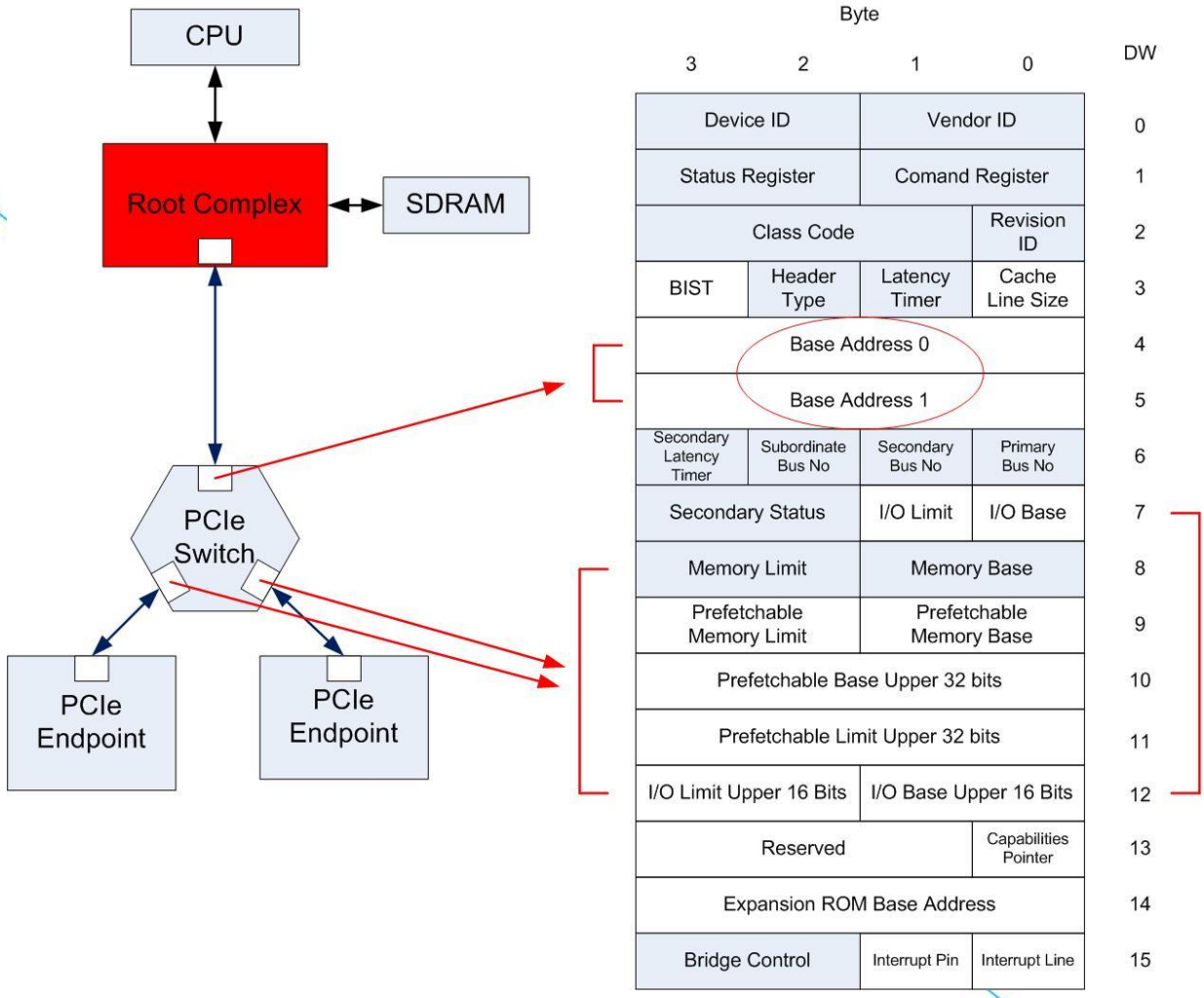
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 73 Adresno rutiranje u slučaju PCI Express Endpoint uređaja**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 74 Adresno rutiranje u slučaju PCI Express svič uređaja**

### 19.4.2 ID rutiranje TLP paketa

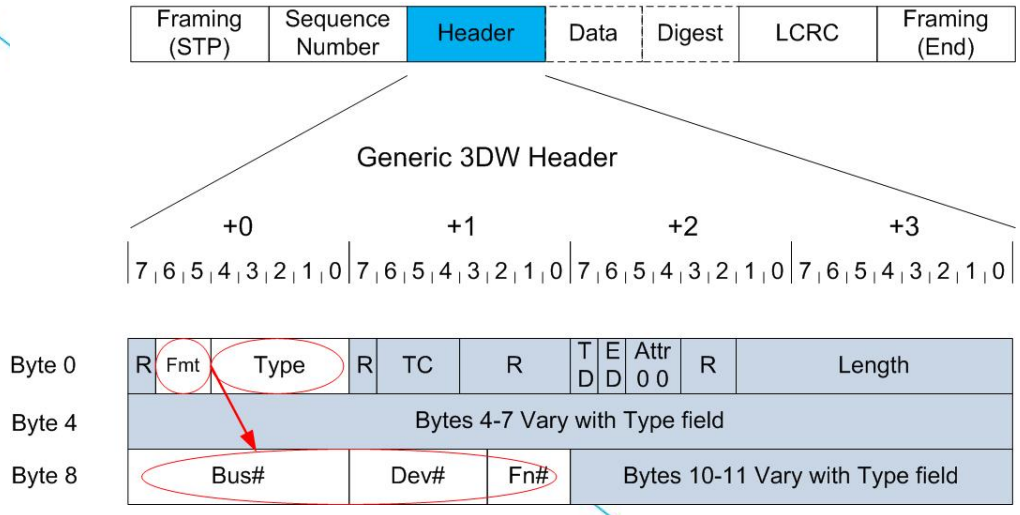
ID rutiranje se bazira na logičkoj poziciji uređaja (Bus Number, Device Number, Function Number) u okviru PCI topologije. Ovaj vid rutiranja je kompatibilan sa rutiranjem na PCI magistrali. ID rutiranje se koristi za transakcije konfiguracije, pakete za kompletiranje transakcije i poruke.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

PCIe ima ista ograničenja u topologiji kao i PCI – 256 magistrala/linkova u sistemu, 32 uređaja po magistrali/linku i 8 funkcija po uređaju. PCIe nudi veći konfiguracioni prostor od PCI magistrale 4kB prema 256B.

Po pristizanju paketa koji traži ID rutiranje informacije se izvlače iz odgovarajućih polja u hederu TLP. Hederi paketa mogu biti veličine 3DW i 4DW sa tim da se informacije o tipu rutiranja nalaze u Fmt Type poljima. U bajtovima 8 i 9 nalaze se ID informacije - Bus Number, Device Number, Function Number (Slika 18).




**Slika 75 Pozicija ID polja u zaglavlju TLP-a**

Kada Endpoint uređaj primi paket sa ID rutiranjem on poredi svoja polja Bus Number, Device Number, Function Number sa onima dobijenim u paketu. Ako se polja poklapaju paket se prihvata a u suprotnom odbacuje. Endpoit uređaj po resetu nema upisana ID polja već se ona upisuju u postupku enumeracije. Pozicije ovih polja nisu definisane standardom.

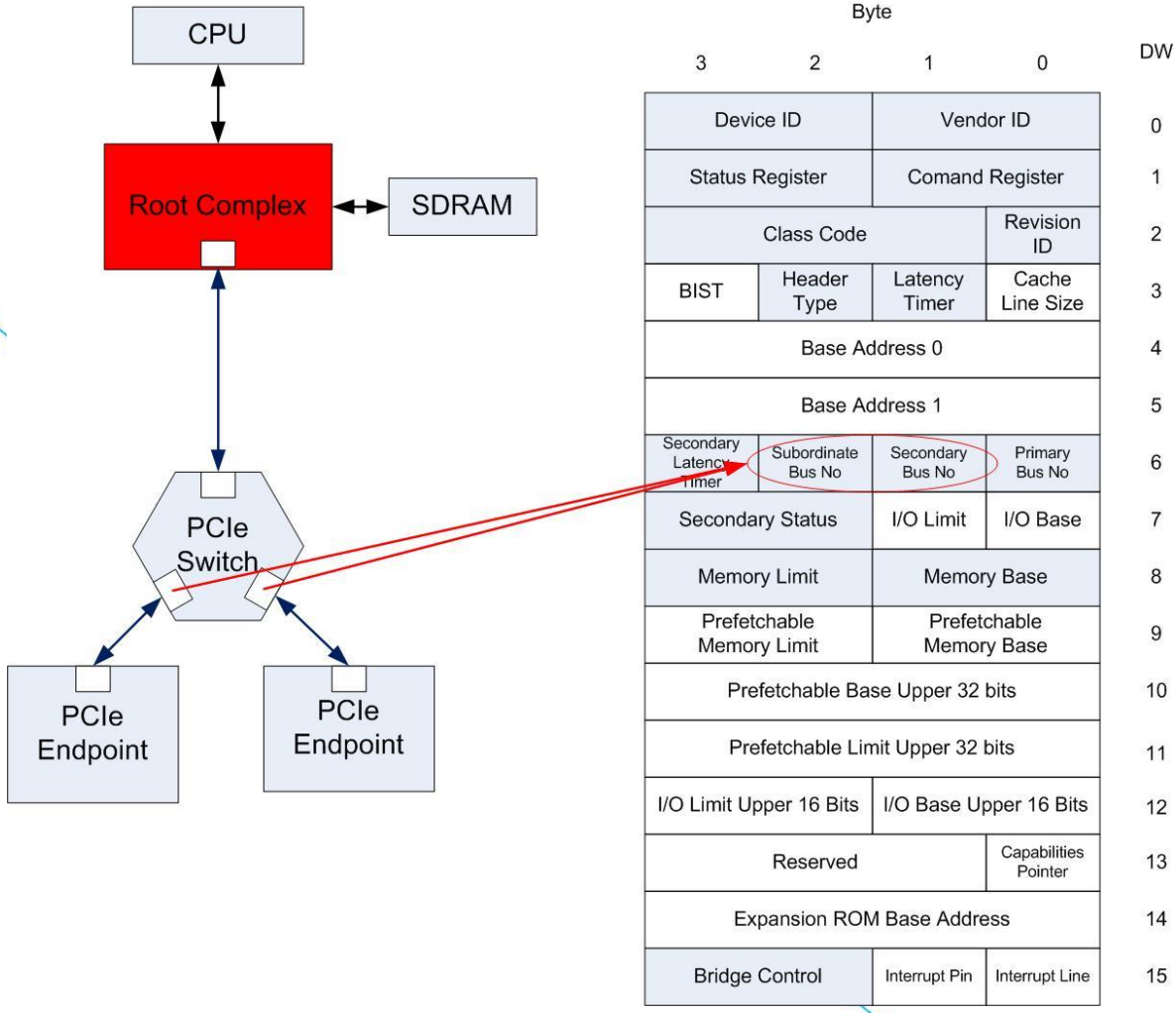
Kada Svič uređaj primi paket koji traži ID rutiranje vrši se nekoliko provera. Prvo se proverava da li ID polje odgovara ID polju sviča i ako to jeste slučaj svič koristi paket. Ukoliko to nije slučaj proverava se da li Bus number u ID polju odgovara opsegu Secondary-Subordinate Bus Number registara jednog od bridge interfejsa i ako je to slučaj paket se prosleđuje na dole (Slika 19).

Ukoliko Bus number u ID polju ne odgovara opsegu Secondary-Subordinate Bus Number registara nijednog od bridge interfejsa paket se odbacuje.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Paketi koji se kreću nagore se uvek prosleđuju na upstream link ukoliko se ne poklapaju sa internom lokacijom ili nekim drugim downstream linkom .




**Slika 76 ID rutiranje u slučaju PCI Express svič uređaja**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 19.4.3 Implicitno rutiranje TLP paketa

Implicitno rutiranje se bazira na tome da uređaji na PCIe magistrali inheretno znaju poziciju Root complex uređaja. Root complex je odredište poruka koje se tiču prekida, kontrole potrošnje, grešaka, hot-plug mehanizama itd. Obrada poruka sa razlikuje zavisno od toga da li je uređaj koji je prima svič ili endpoint.

Obrada poruka sa implicitnim rutiranjem se u slučaju endopointa svodi na proveru da li je poruka odgovarajućeg tipa (broadcast npr.). PCIe Svič vrši obradu implicitno rutiranih poruka zavisno od tipa paketa. Broadcast paket koji stiže od Root complex-a će biti prosleđen svim downstream portovima a u slučaju da dolazi od downstream porta biće odbačen.

Paketi sa implicitnim rutiranjem koji su adresirani na Root complex a stižu sa downstream linkova će biti prosleđeni na upstream link a ako ovakav paket stigne na upstream link smatra se za grešku. Ako paket sa implicitnim rutiranjem stigne na bilo koji port a adresiran je na prijemnik svič će obraditi paket.

## 19.5 ACK/NAK Protokol

DLL sloj na PCI Express magistrali je zadužen za pouzdanu isporuku paketa i mora da ispuni sledeće funkcije:

- Obezbeđivanje prijema paketa bez grešaka
- Ponovno slanje paketa koji su primljeni sa greškama
- Prijem paketa u odgovarajućem redosledu uz ponovna slanja paketa ako se ovaj redosled naruši
- Potvrda prijema paketa se izvodi preko ACK/NAK paketa koje šalje DLL sloj

DLL sloj zadužen za ACK/NAK protokol se može podeliti na predajnu i prijemnu stranu.

### 19.5.1 Predajna strana DLL sloja

Elementi na predajnoj strani su: Replay bufer, NEXT_TRANSMIT_SEQ brojač, LCRC Generator, REPLAY_NUM brojač, REPLAY_TIMER brojač, ACKD_SEQ brojač i blok za DLLP CRC proveru (Slika 20).

Replay bufer služi za smeštanje paketa dobijenih od sloja transakcije zajedno sa poljima DLL sloja. Paketi se smeštaju po redu pristizanja i brišu tek po dobijanju potvrde o prijemu.

NEXT_TRANSMIT_SEQ brojač služi za generisanje broja paketa u DLL sloju. Ovo je 12 bitni brojač koji se po prekoračenju vraća na nulu.

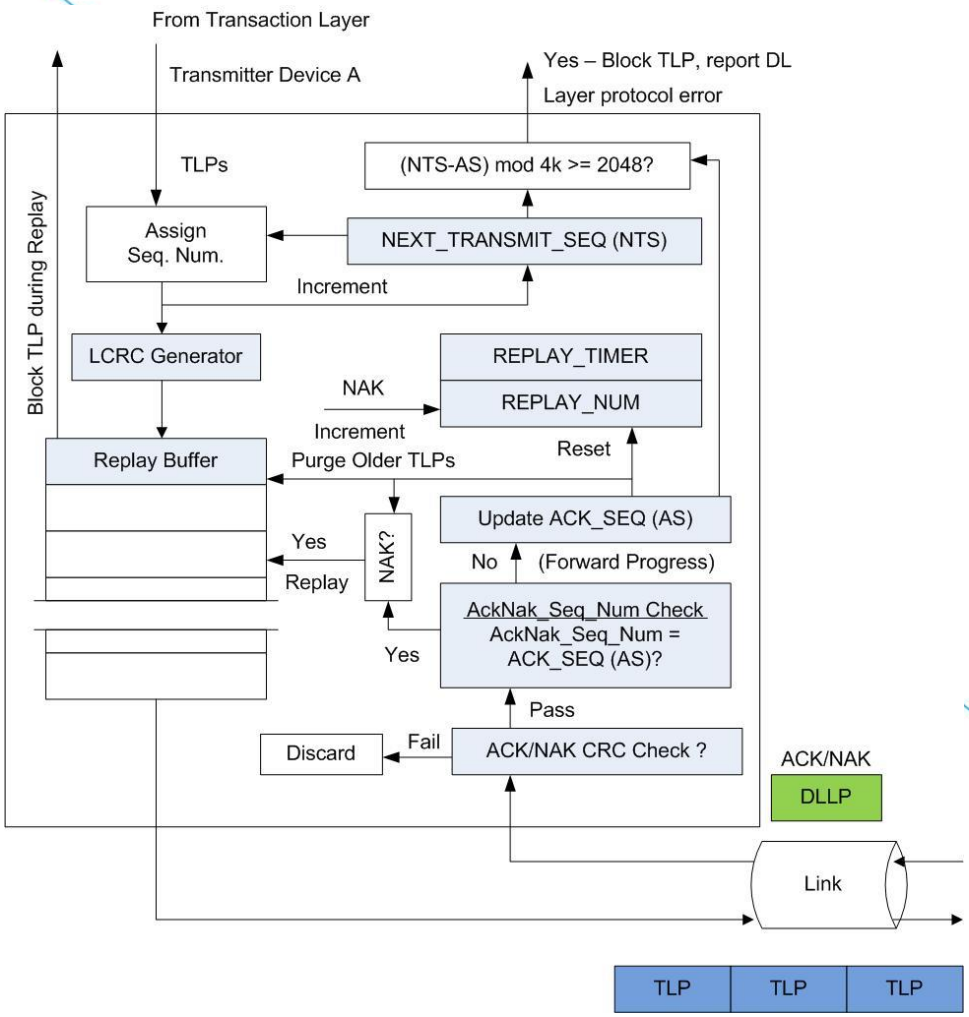
LCRC Generator se koristi za generisanje 32 bitniog CRC polja od svih polja TLP paketa kao i rednog broja.

REPLAY_NUM je brojač od dva bita koji se inkrementira pri svakom ponovnom slanju paketa. U slučaju prekoračenja inicira se ponovni trening linka na fizičkom sloju.

REPLAY_TIMER meri vreme od slanja poslednjeg bajta paketa pa do prijema ACK/NAK paketa odgovra ili do prekoračenja tajmera.

ACKD_SEQ brojač smešta broj paketa iz primljenog ACK/NAK paketa i služi za proveru sekvence pristizanja paketa. Ukoliko razlika ACKD_SEQ i NEXT_TRANSMIT_SEQ po modulu 4096 premaši 2048 signalizira se fatalna greška.

Blok za DLL CRC proveru služi za proveru CRC polja pristiglih ACK/NAK paketa.




**Slika 77 Blok šema predajnika na DLL sloju**

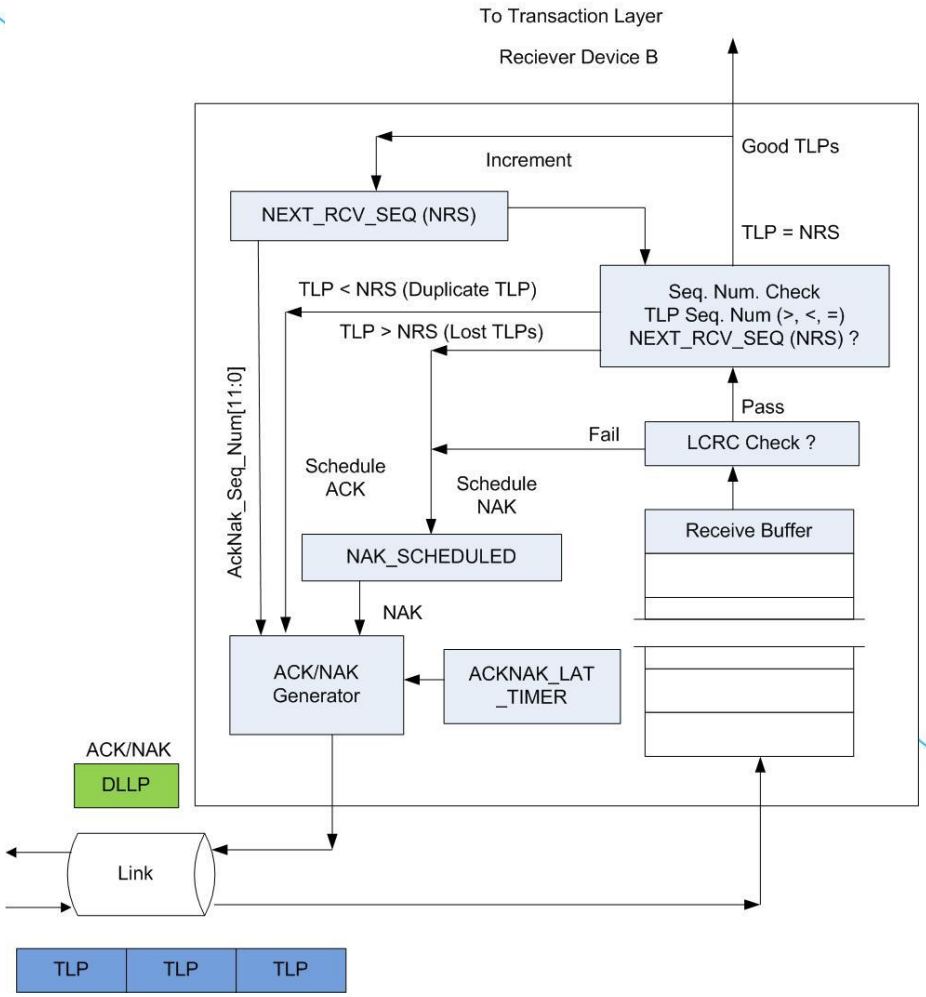


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 19.5.2 Prijemna strana DLL sloja

Elementi DLL bloka na prijemnoj strani su: Receive Bafer, LCRC Error Check, NEXT_RCV_SEQ brojač, NAK_SCHEDULED Flag, ACKNAK_LATENCY tajmer, Generator ACK/NAK paketa (Slika 21).

Receive bafer služi za privremeno smeštanje TLP paketa dok se vrši CRC provera. Ako je paket uspešno primljen on se upućuje na TLP sloj. Ako se javi greška paket se odbacuje i podiže se NAK fleg Ako je paket dupliran odbacuje se i generiše ACK.




**Slika 78 Blok šema prijemnika na DLL sloju**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Blok za DLL CRC proveru se koristi za proveru ispravno primljenog paketa.

NEXT_RCV_SEQ brojač se inkrementira pri svakom uspešno primljenom paketu. Ako je paket sa lošim CRC-om, ili van redosleda brojač se ne menja.

NAK_SCHEDULED Flag polje se postavlja kada se primi paket sa greškom.

ACKNAK_LATENCY tajmer se koristi za merenje vremena od kada se indikuje da treba poslati ACK ili NAK paket radi obezbeđenja blagovremene potvrde paketa.

Generator ACK/NAK DLLP generiše ACK ili NAK paket sa brojem paketa koji se potvrđe. Ovaj broj je NEXT_RCV_SEQ – 1.

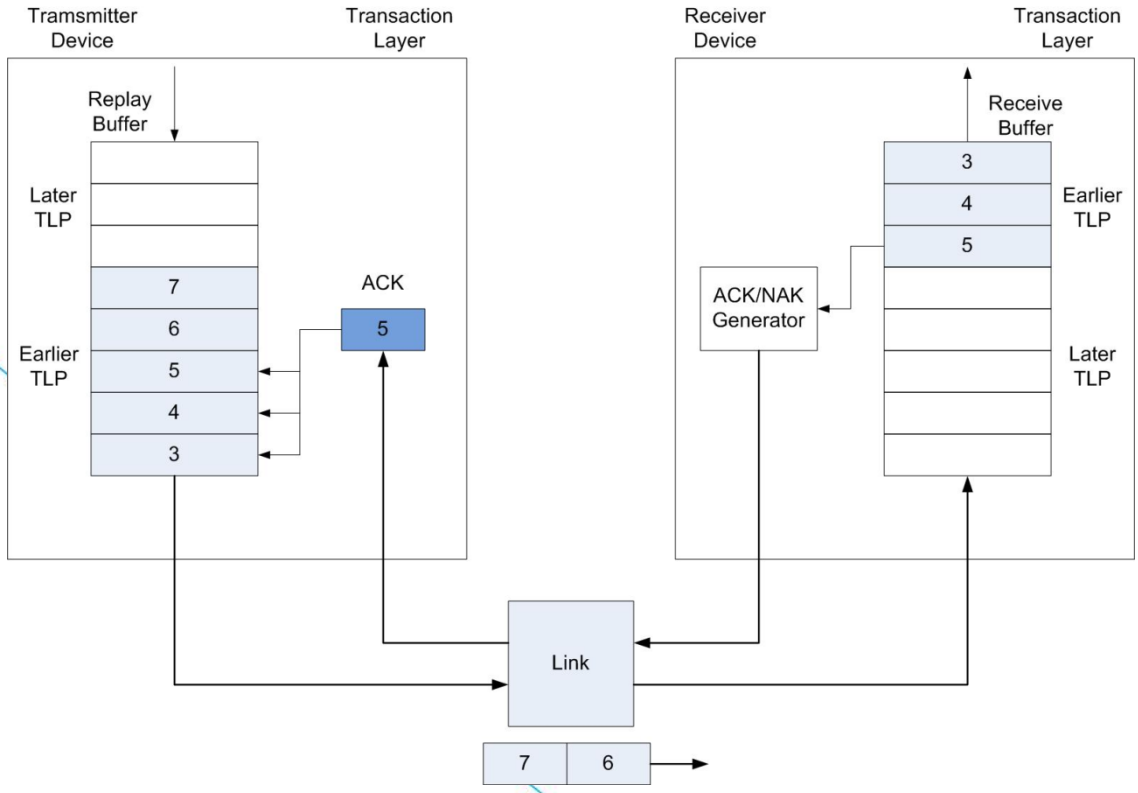
### 19.5.3 Principi rada ACK/NAK protokola - Predajnik

Predajnik obrađuje TLP pakete za slanje tako što im dodaje broj paketa i CRC polje i zatim ih smešta u Replay bafer. Broj paketa se inkrementira do vrednosti 4095 preko koje se postavlja na nulu. Veličina Replay bafera se dimenzioniše tako da ne bude zastoja uzrokovanih latencom pristizanja ACK paketa, podizanja linka iz režima štednje energije i kašnjenja zbog fizičke implementacije linka.

Replay bafer čuva TLP pakete do dobijanja potvrde o prijemu preko ACK paketa. Svi paketi ispod broja paketa u ACK paketu se brišu iz bafera. Po prijemu ACK paketa se takođe resetuju REPLAY_NUM brojač a u ACKD_SEQ se smešta broj paketa dobijen u ACK paketu. Prikaz ispravnog prijema paketa je prikazan na slikama 22 i 23.



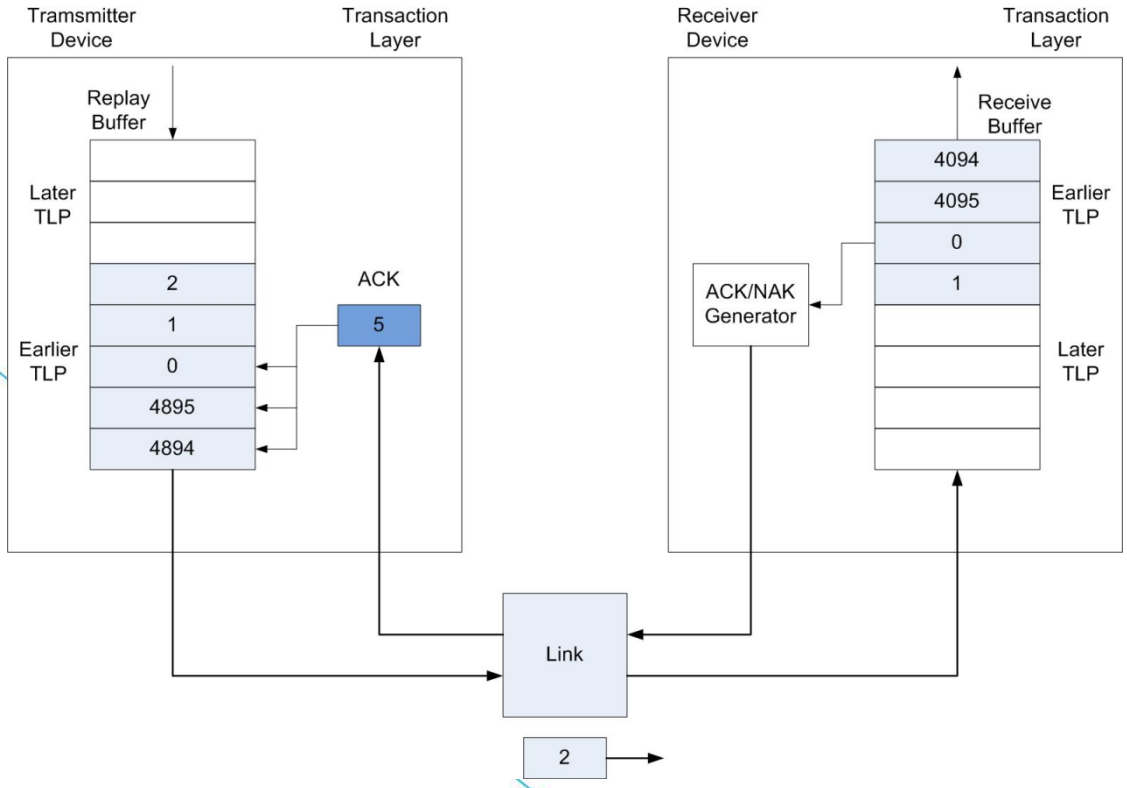
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 79 Primer potvrde prijema paketa**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 80 Primer potvrde prijema paketa sa prekoračenjem brojača**

Prijem NAK paketa se posmatra kao ACK za sve pakete sa manjim brojem a paket za koji je dobijen NAK se ponavlja. U slučaju prijema NAK paketa predajnik prvo briše sve pakete iz Replay bafera koji imaju manji i isti broj paketa kao u NAK paketu. Blokira se prijem paketa iz transakcionog sloja dok se ne pošalju ponovo preostali paketi u replay baferu.

Paketi se čuvaju do potvrde o prijemu. REPLAY_NUM i REPLAY_TIMER se resetuju samo ako je potvrđen prijem paketa broja većeg od onog u ACKD_SEQ. U ACKD_SEQ se smešta vrednost iz NAK paketa (Slika 24).

Za vreme ponovnog slanja paketa predajnik mora barem čuvati ili obrađivati ACK/NAK pakete. Potvrđeni paketi se mogu brisati iz replay bafera tokom ponovnog slanja ako stigne ACK/NAK paket. Brojač REPLAY_NUM se koristi za merenje progresa u slanju paketa i inkrementira se po prijemu NAK paketa sa istim brojem kao u ACKD_SEQ. U suprotnom se resetuje pošto postoji progres u slanju.

Ako brojač pređe vrednost 3 signalizira se problem na fizičkom sloju i fizički sloj ulazi u Recovery State.

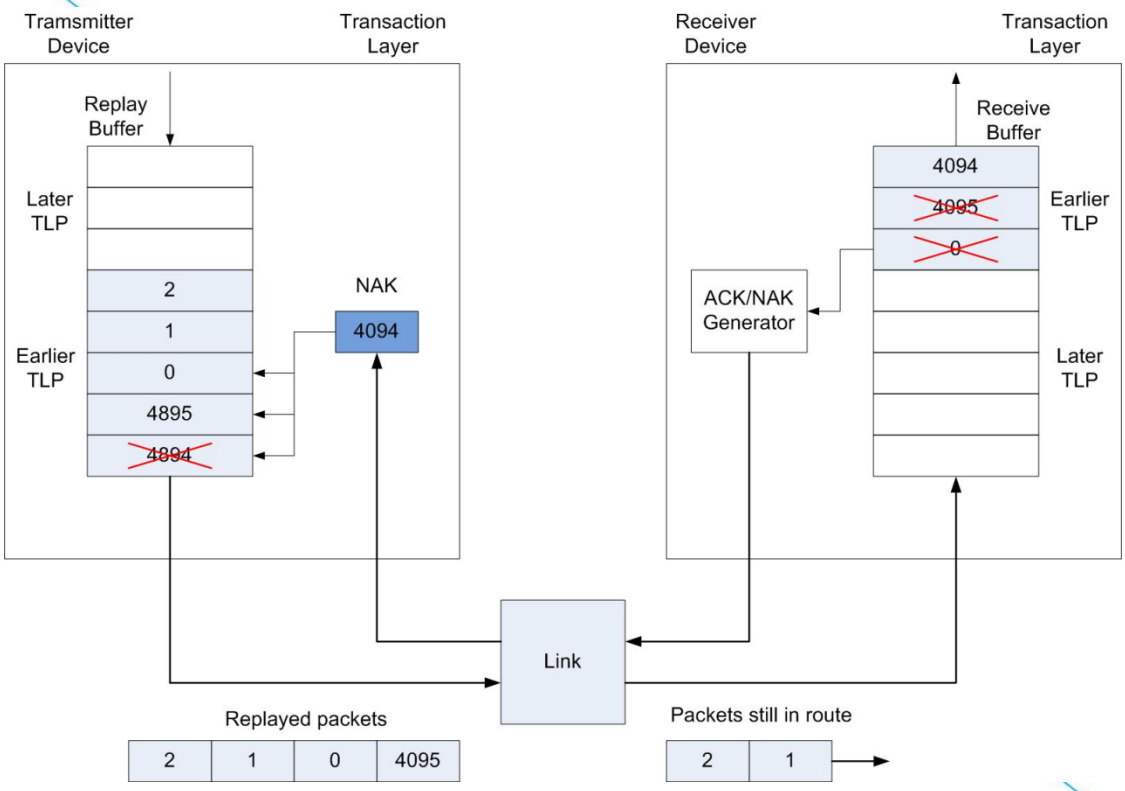


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Posle ponovnog treninga linka nastavlja se sa slanjem paketa. Ukoliko iz nekoliko sukcesivnih ponavljanja progres u slanju nije uspostavljen DLL treba da signalizira Uncorrectable Fatal Error.

Transmiter poseduje REPLAY_TIMER koji indikuje da odgovor prijemne strane nije stigao na vreme što inicira ponovno slanje paketa. REPLAY_TIMER startuje po slanju poslednjeg simbola paketa. Tajmer se resetuje po prijemu ACK paketa ili slanju poslednjeg simbola paketa tokom ponavljanja paketa.

Tajmer se postavlja na nulu i drži tako: kada nema više paketa za slanje, po prijemu NAK paketa do završetka ponovnog slanja prvog paketa, kada tajmer istekne ili ako DLL nije aktivan.




**Slika 81 Primer ponavljanja paketa**

### 19.5.4 Principi rada ACK/NAK protokola – Prijemnik

Pristigli paketi se prvo obrađuju na fizičkom sloju i ako su uspešno primljeni prosleđuju se DLL sloju i smeštaju u prijemni bafer. Ukoliko to nije slučaj DLL sloj se obaveštava o tome. Primljeni paket na vrhu prijemnog bafera se testira proverom CRC-a i ako prođe ovu proveru upoređuje se broj paketa



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

sa NEXT_RCV_SEQ i ako su jednaki šalje se sloju Transakcije a paralelno se inkrementira brojač NEXT_RCV_SEQ. Ukoliko potoji greška (CRC ili broj paketa) paket se odbacuje i inicira se slanje NAK paketa sa brojem poslednjeg primljenog paketa. NEXT_RCV_SEQ se ne inkrementira.

Slanje ACK paketa se može se inicirati po prijemu ispravnog paketa ili na nekoliko primljenih paketa sa tim da ACK sadrži broj poslednjeg ispravno primljenog paketa. ACK paket se mora poslati ako ACKNAK_LATENCY tajmer istekne. U momentu kada primi ispravan paket i treba da pošalje ACK paket prijemnik može u svom predajnom baferu već imati paket koji se trenutno šalje pa se slanje ACK paketa može odložiti. Čim se bafer isprazni prijemnik šalje ACK paket sa brojem poslednjeg ispravno primljenog paketa.

Greška na prijemu može biti uzrokovana greškom pri CRC proveri ili neslaganjem broja primljenog paketa sa brojem u NEXT_RCV_SEQ. Paketi mogu imati manji ili veći broj od broja u NEXT_RCV_SEQ.

Pri detekciji greške prijemnik indikuje slanje NAK paketa koji sadrži broj poslednjeg uspešno primljenog paketa.

U momentu kada se indikuje slanje NAK paketa predajni bafer u prijemniku može biti popunjen paketom koji se trenutno šalje pa se slanje NAK paketa odlaže dok se bafer ne oslobodi. Za to vreme prijemnik nastavlja prijem sledećih paketa ali ih odbacuje zbog očuvanja redosleda.Kada se pojavi ponovljeni paket on se potvrđuje i prenos se nastavlja.

Postoje tri slučaja pri poređenju broja paketa i brojača NEXT_RCV_SEQ:

- Brojevi se poklapaju – Šalje se ACK paket sa brojem paketa a NEXT_RCV_SEQ se inkrementira.
- Broj paketa je manji od broja u NEXT_RCV_SEQ – paket se odbacuje ali se šalje potvrda sa brojem ponovljenog paketa.
- Broj paketa je veći od broja u NEXT_RCV_SEQ – paket se odbacuje i šalje se broj paketa za jedan manji od NEXT_RCV_SEQ.

Na ovaj način DLL garantuje prijem svih paketa i to u ispravnom redosledu.

ACKNAK_LATENCY tajmer meri vreme od kada je indikovano slanje ACK/NAK paketa.

Kada tajmer istekne ACK/NAK paket se sprema za slanje sa poslednjim brojem uspešno primljenog paketa. Maksimalna vrednost tajmera je podešena na 1/3 od vrednosti Replay tajmera čime se garantuje da se ACK/NAK paketi šalju na vreme. Tajmer se resetuje na nulu kada je ACK/NAK paket upućen na slanje. Tajmer se resetuje i drži na nuli kada više nema novih pristiglih paketa i kada je DLL neaktivan.

### 19.5.5 Prioriteti slanja paketa

Paketi se šalju po prioritetu definisanom u sledećoj listi:

- Završetak slanja trenutnog paketa



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

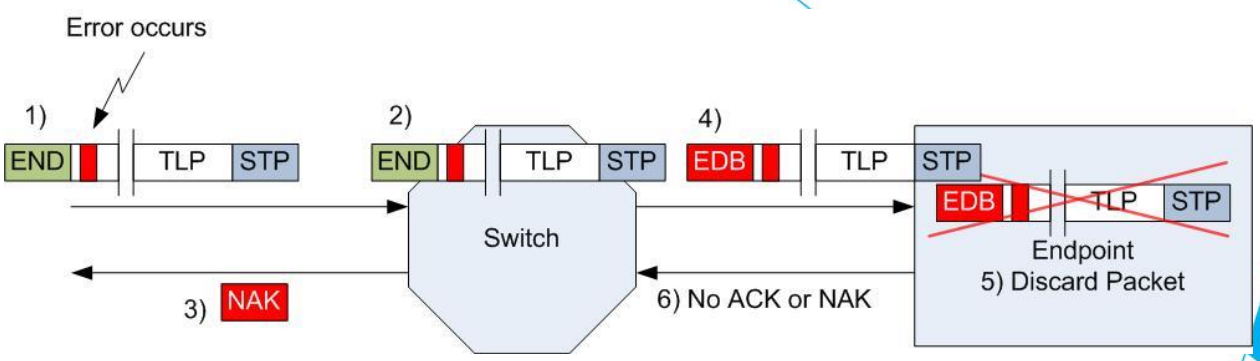
- Paketi fizičkog sloja
- NAK paket
- ACK Paket
- Flow Control (kontrola protoka) paket
- Retransmisija iz replay bafera
- TLP paketi koji čekaju u sloju Transakcjie
- Svi drugi DLL paketi

### 19.5.6 Switch Cut-Throug mod

PCIe svič može imati drugačiji princip slanja paketa radi smanjenja latence.

Paket se odmah po prijemu prosleđuje na izlazni port bez čekanja tj. provere CRC-a. U momentu prijema kompletnog paketa vrši se CRC provera i ako se ustanovi da postoji greška CRC u paketu koji se prosleđuje se invertuje a paket se završava EDB umesto END poljem. Na ovaj način prijemnik zna da je to paket od sviča koji je loše primljen i da ne treba da šalje NAK paket koji svič nema mogućnost da obradi (nije primio ispravan paket). Kada svič primi ponovljen paket on ga prosleđuje ka izlaznom portu i prijemniku.

Na ovaj način se značajno smanjuje latenca sviča a greška na paketu ne uzrokuje problem u komunikaciji.




**Slika 82 Primer Switch Cut-Throug moda**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 19.6 Kvalitet servisa i arbitracija

Kvalitet servisa (QoS) predstavlja sposobnost magistrale da garantuje određeno kašnjenje i propusni opseg. Praktični primeri aplikacija sa ovakvim zahtevima su prenos video i audio podataka preko magistrale. PCIe definiše izohroni princip prenosa podataka specijalno prilagođen ovim zahtevima ali se pravila QoS mogu primeniti na bilo koje transakcije. Uređaji na PCIe magistrali moraju koristiti ove mogućnosti.

QoS može obuhvatiti više karakteristika koje utiču na performanse i to: period komunikacije, efektivni protok, latenca, frekvencija grešaka i druge. PCIe obezbeđuje QoS korišćenjem nekoliko mehanizama: Klase saobraćaja (TC), Virtuelni kanali (VC), Arbitracija na portu, Arbitracija na VC i kontrola protoka. PCIe podržava dve vrste transakcija: izohrone i asinhrone.

Izohrone transakcije su vid transakcija na PCIe magistrali koje traže konstantni protok u regularnim vremenskim intervalima i to sa garantovanom latencom. Primer bi bio prenos video signala gde kamera u regularnim intervalima isporučuje jedan frejm slike koji se mora brzo preneti radi smeštanja u memoriju i obrade istog. Izohrone transakcije smanjuju latencu i potrebu za velikim baferima.

Asinhrone aplikacije obuhvataju široku lepozu aplikacija sa različitim zahtevima za propusnim opsegom i latencom. Da bi se omogućila podrška za aplikacije sa većim zahtevima uvodi se sistem prioriteta gde se paketima zahtevnijih aplikacija daje prednost u prenosu. Drugim rečima uspostavlja se hijerarhija transakcija – diferenciranje servisa.

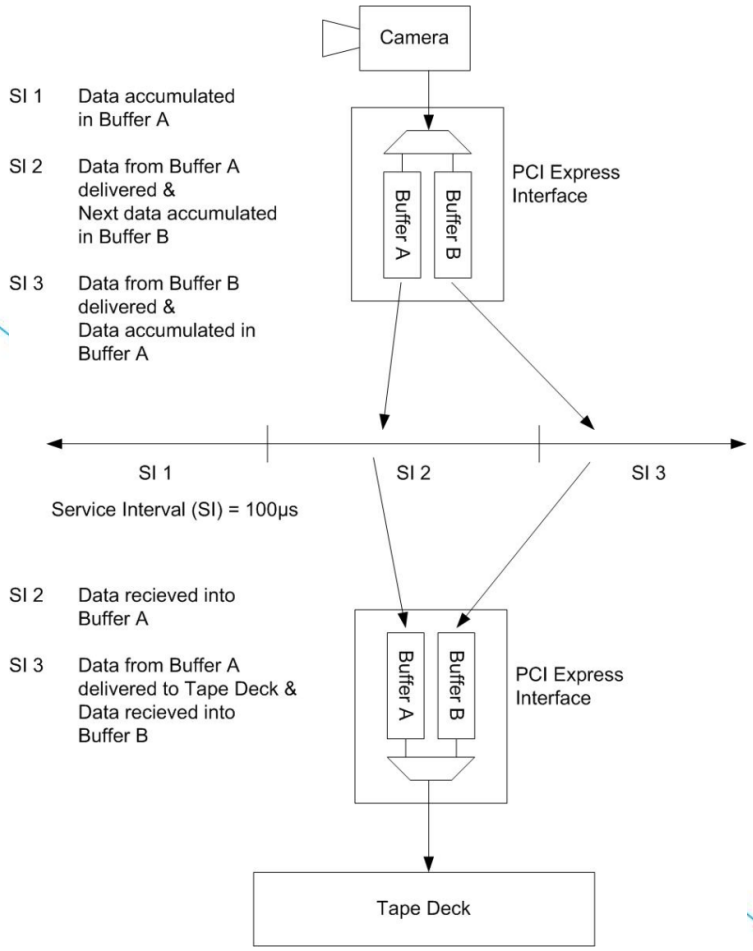
#### 19.6.1 Podrška za izohrone transakcije

Aplikacije koje inače zahtevaju sinhronu komunikaciju dva uređaja mogu se povezati putem izohrnoih transakcija i korišćenjem bafera. Tipičan primer je prenos video podataka sa kamere do uređaja za snimanje. Kada su ovi uređaji direktno povezani kamera generiše referentni signal takta na koji se sinhroniše uređaj za snimanje. Kašnjenje je uslovljeno samo kašnjenjem signala i manjim prenosnim baferima.

Kod izohronih transakcija toga nema već se podaci iz npr jednog frejma ili manjeg segmenta slike moraju postaviti u bafer A koji se u regularnim vremenskim intervalima prenosi preko PCIe magistrale

Za vreme prenosa novi podaci na kameri se smeštaju u bafer B. Preneti blok podataka se smešta u bafer A na uređaju za snimanje i po smeštanju isti se iščitava i podaci čuvaju na odgovarajućem medijumu. Kada se bafer B na kameri napuni šalje se u vremenskom slotu koji odgovara sledećem intervalu izohrone transmisije. Bafer A je za to vreme prenesen preko PCIe magistrale i spreman je za novi blok podataka. Primljeni podaci na uređaju se snimaju u bafer B dok je bafer A sada spreman za prijem podataka i ceo proces se ciklično ponavalja (Slika 26).

Podrška za ovakav tip transakcija se bazira na korišćenju odgovarajućeg TC visokog prioriteta dok prenosni uređaji na putanju smeštaju TC u VC visokog prioriteta i konfigurisani su da prihvataju ovakve transakcije u regularnim vremenskim intervalima.




**Slika 83 Primer izohronih transakcija**

### 19.6.2 Virtuelni kanali (VC) i klase saobraćaja (TC)

Različiti servisi koji potpadaju pod asinhrone transakcije traže različite prioritete. Npr. eternet paketi traže viši prioritet od uređaja za skladištenje podataka. Prioriteti se ostvaruju primenom TC i VC ali PCIe specifikacija ne definiše metode određivanja prioriteta.

Tokom inicijalizacije PCIe uređaja drajver od operativnog sistema traži određen kvalitet servisa od operativnog sistema koji vraća TC nivo. Tokom konfiguracije sistema se vrši povezivanje TC na



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

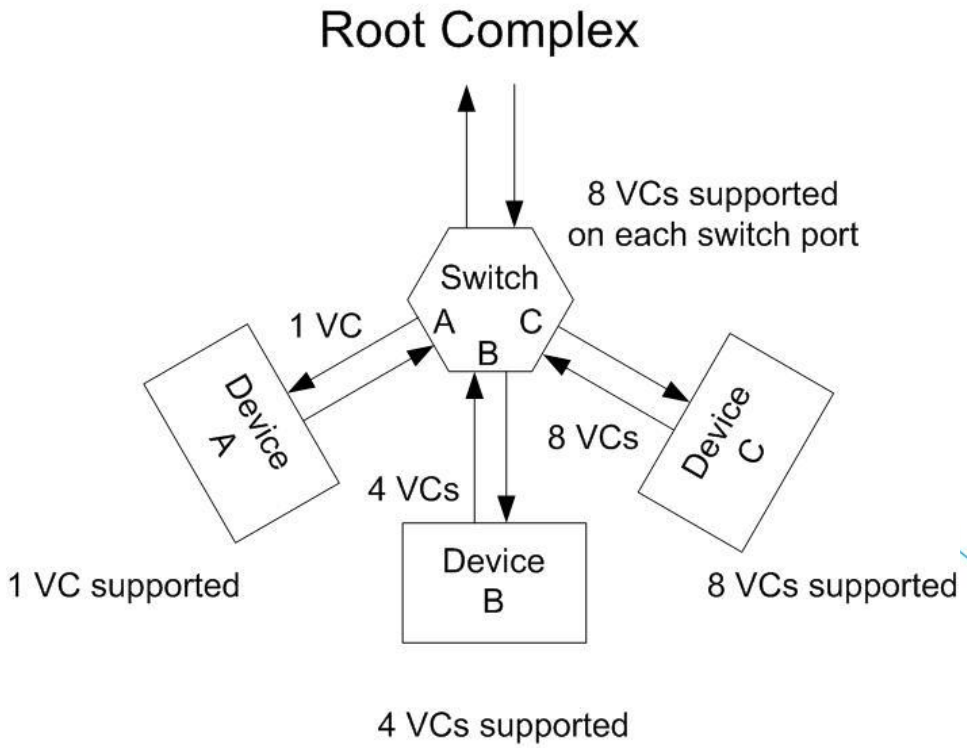
raspoložive VC u uređajima. TC vrednost se nalazi u TLP i može imati osam vrednosti TC0-TC7 sa time da je podrška za TC0 obavezna.

Uređaji mogu podržati maksimalno osam virtuelnih kanala VC0-VC7 a obavezno je da podrže VC0

TC0 se uvek mapira na VC0. Mapiranje TC na VC se vrši u registrima u dodatnom PCIe konfiguracionom prostoru. PCI specifikacija zahteva da neke vrste paketa koriste isključivo TC0/VC0.

Tokom konfiguracije uređaja i njihovog povezivanja broj VC kanala se određuje u komunikaciji sa susednim uređajem. Npr. PCIe switch sa tri izlazna porta A,B,C podešava maksimalni broj VC koji uređaj ispod podržava. Tako port A ima samo VC0, port B VC0-3 a port C svih osam VC (Slika 27).

Tokom konfiguracije uređaja vrši se određivanje broja raspoloživih VC kao i mapiranje VC brojeva – ID. Ovi brojevi ne moraju biti dodeljeni redom ali moraju biti jedinstveni (npr. VC0 i VC6). U konfiguracionom prostoru se takođe vrši mapiranje TC na VC. Ovo mapiranje se vrši tako da grupa TC može da se mapira na jedan VC ali nikad isti TC na dva VC.




**Slika 84 Primer određivanja broja VC kanala po uređaju**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 19.6.3 Vrste Arbitracije

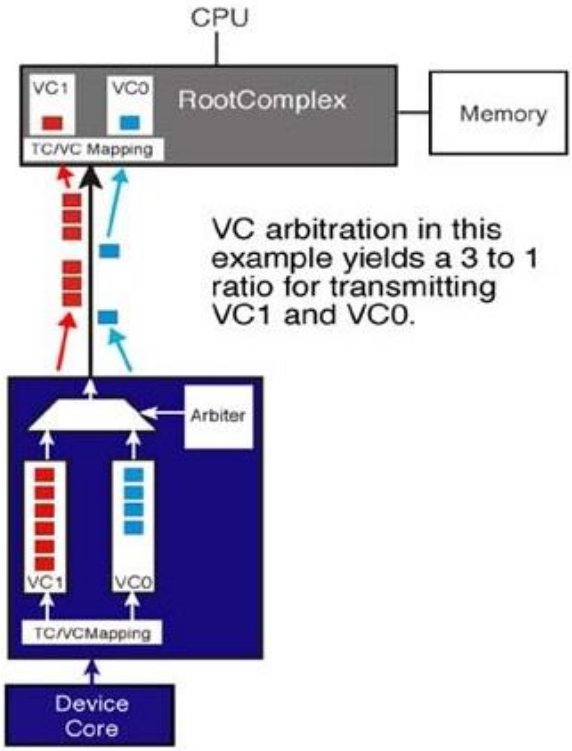
Postoje dve vrste metoda kojima se postižu izohrone transakcije i diferencirani servisi:

- Arbitracija virtuelnih kanala – određuje prioritet transakcija koje se šalju sa istog porta
- Arbitracija porta – predstavlja arbitraciju paketa na izlaznom portu prema prioritetima ulaznih portova

Arbitracija porta se ostvaruje u okviru jednog VC-a na izlaznom portu.

### 19.6.4 Arbitracija virtuelnog kanala

Arbitracija VC-a pored prioriteta mora da omogući i progres na svim VC da ne bi došlo do prekoračenja vremenskog limita na split transakcije. Svaki uređaj koji inicira transakcije i ima više od jednog VC mora da omogući arbitraciju kao i drugi uređaji koji imaju više VC-a (PCIe svič). Arbitracija VC omogućava uređaju da odredi prioritet transakcija (Slika 28).




**Slika 85 Primer arbitracije po virtuelnom kanalu**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Pored ovoga arbitracija VC mora da ispuni sledeće kriterijume:

- svaki VC koji je u upotrebi mora imati svoje bafere i kontrolu protoka
- transakcije paketa se odvijaju uvek po istom redosledu
- nema nikakve zavisnosti u redosledu slanja paketa na različitim VC

Postoji nekoliko vrsta arbitracije VC koji se mogu primeniti:

- Arbitracija sa striktnim prioritetom
- Arbitracija sa podeljenim prioritetom gde se VC dele na dve grupe jedna sa striktnim prioritetom i druga sa nekom vrstom round robin šeme
- Round robin šema (obična ili težinska) za sve kanale

Kod arbitracije sa striktnim prioritetom paketi sa višim VC ID uvek imaju prednost nad onima sa nižim. Ovo dovodi do toga da paketi sa višim VC ID mogu da spreče prenos paketa sa nižim VC ID.

Sa druge strane paketi sa višim VC ID dobijaju celokupni opseg i minimalne latence (Slika 29). PCIe specifikacija zahteva da se saobraćaj visokog prioriteta reguliše kako ne bi ugušio saobraćaj nižeg prioriteta.

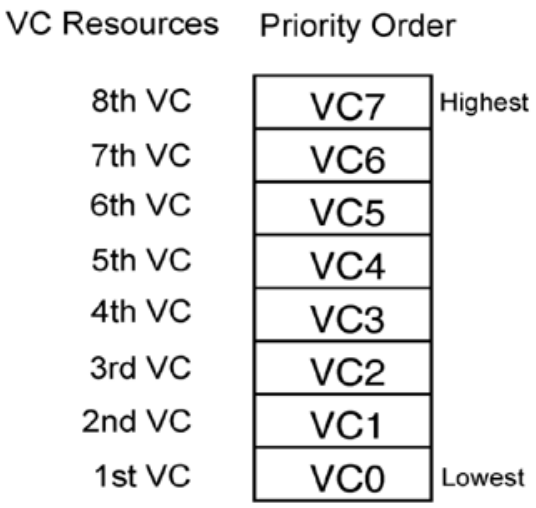
Postoje dva načina za ovo:

- Uređaj koji generiše pakete visokog prioriteta može ograničavati generisanje istih ostavljajući deo opsega za pakete nižeg prioriteta
- PCIe svičevi mogu da regulišu saobraćaj na izlaznom portu čime omogućavaju da paketi nižeg prioriteta dođu do linka

Takođe može se ograničiti broj VC kanala koji učestvuju u striktnim transakcijama.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 86 Primer arbitracije sa striktnim prioritetom**

Kod arbitracije VC sa podeljenim prioritetom virutelni kanali se dele na one sa striktnim prioritetom i one sa nižim prioritetom gde se primenjuju alternativne arbitracije (Slika 30).

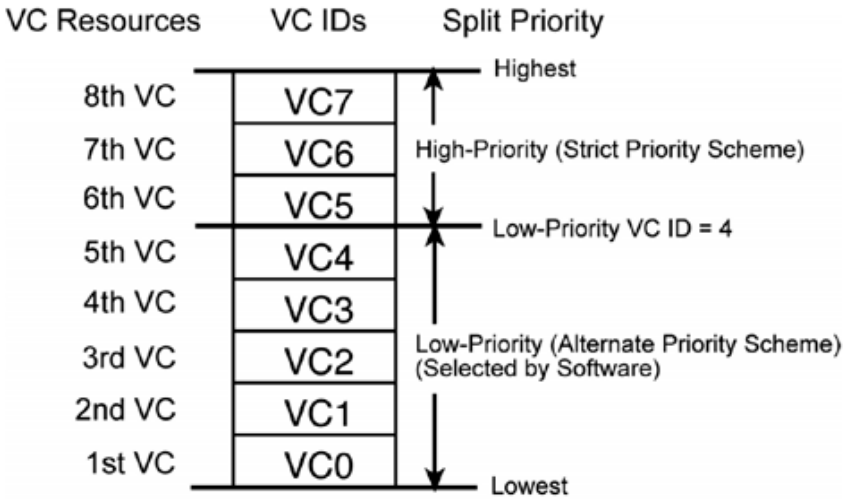
Arbitracija u nižoj grupi se bazira na dve šeme:

- Arbitracija bazirana na HW rešenju (npr. Round robin – RR)
- Arbitracija bazirana na Weighted Round Robin (WRR)

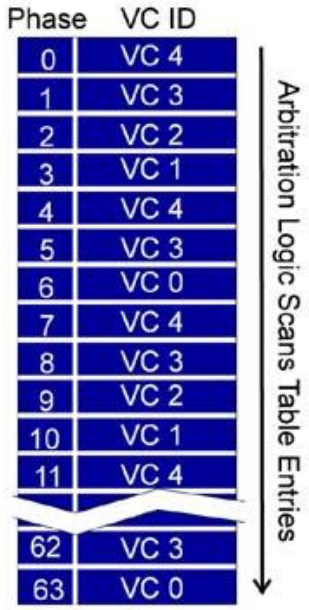
Arbitracija bazirana na HW rešenju je najčešće bazirana na RR principu ali se ne sprečavaju druge implementacje. Zasniva se na kružnom prolaženju kroz sve VC sa istim prioritetom. Arbitracija bazirana na WRR se zasniva na tabeli arbitracije u kojoj su navedeni redom VC ID sa kojih se uzima paket . Prolazak kroz tabelu je cikličan. Primer WRR tabele je prikazan na slici 31.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 87 Primer arbitracije sa podeljenim prioritetom**




**Slika 88 Primer arbitracije bazirane na Weighted Round robin principu**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 19.6.5 Arbitracija na portu

Arbitracija na portu je neophodna kada paketi sa više ulaznih portova stižu na jedan izlazni port

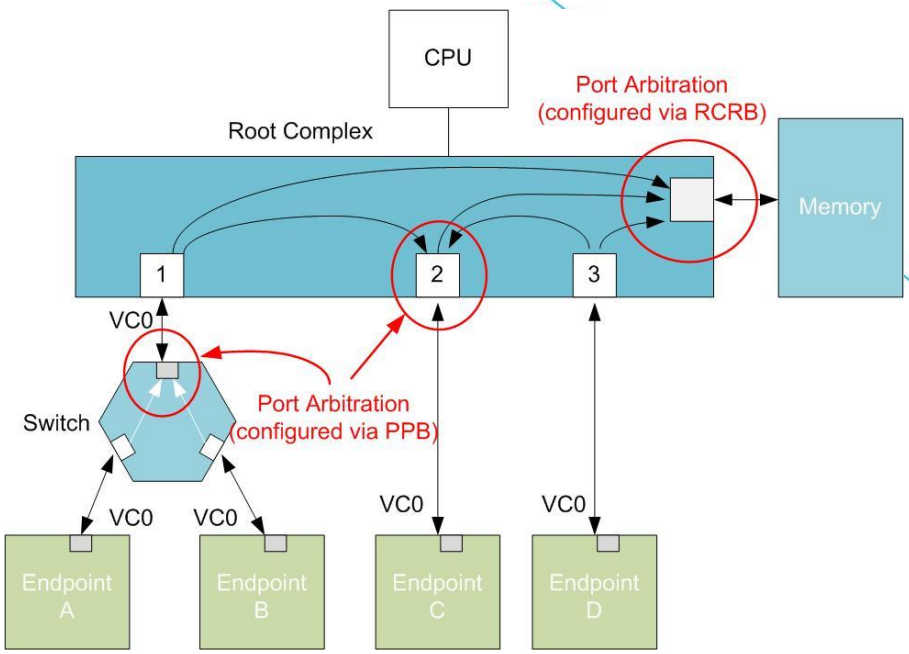
Ovo se dešava u sledećim situacijama (Slika 32):

- Izlazni port PCIe sviča
- Portovi root complex-a kada se obavljaju transakcije između uređaja povezanih na ove portove (peer to peer)
- Izlazni port root complex-a koji vodi ka memoriji

Arbitracija na portu zahteva konfiguraciju za PCI to PCI mostove koji postoje u PCIe svičevima i root complex blokovima. Arbitracija se obavlja individualno na različitim VC. Potrebna je posebna tabela za arbitraciju za svaki VC. Takođe potrebni su baferi za smeštanje paketa iz svakog ulaznog porta.

PCIe svič izvodi sledeće operacije arbitracije na portu (Slika 33):

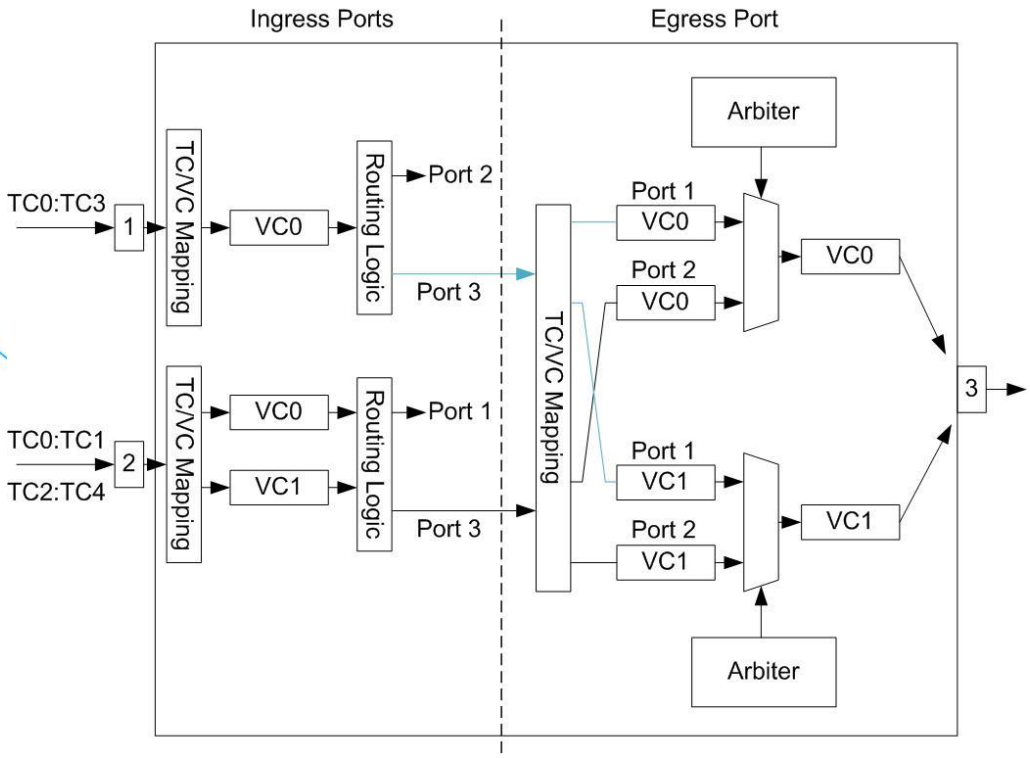
- Paketi koji pristižu na ulazne portove se preusmeravaju na odgovarajuće bafere pomoću TC/VC mapiranja
- Logika za rutiranje zatim preusmerava pakete na odgovarajući izlazni port
- Na ulazu u blok izlaznog porta TC/VC mapiranje preusmerava pakete u odgovarajući VC bafer
- Svaki ulazni port ima odgovarajući VC bafer u izlaznom portu
- Logika za arbitraciju porta određuje redosled slanja paketa iz različitih portova




**Slika 89 Slučajevi arbitracije na portu**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 90 Šema arbitracije na portu**

Postoji nekoliko mehanizama arbitracije po portu:

- Fiksiran hadverski metod
- Weighted Round Robin (WRR) u 32 faze
- WRR u 64 faze
- WRR u 128 faza
- WRR u 256 faza
- WRR baziran na vremenskoj raspodeli u 128 faza

Hardverska arbitracija ne zahteva nikakvu tabelu za rutiranje. Baziran je na principu Round Robin šeme ili nekog metoda sa jednakim prioritetima. Ovaj metod omogućava da se obezbedi progres u slanju sa svih portova. Međutim, u slučaju hardverske arbitracije nije obezbeđena podrška za izohroni prenos ili diferencirane servise.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

WRR arbitracija na portu je zasnovana na sličnom principu kao WRR implementacija arbitracije VC. Postoji tabela sa određenim brojem faza gde svaka specificira port sa koje treba uzeti paket. Ukoliko nema paketa sa tog porta momentalno se prelazi na sledeću fazu. PCIe specifikacija definiše četiri tipa tabele sa 32, 64, 128 ili 256 faza.

WRR sa vremenskim slotom se primenjuje kod izohronih prenosa paketa. Principski je slična WRR ali se prelazak na sledeću fazu vrši po isteku nekog vremenskog slota – 100ns. Na taj način transakcije se prihvataju u regularnim vremenskim intervalima. Maksimalno trajanje transfera je jedan ciklus prolaska kroz tabelu. Postoje dva slučaja kada se transakcija ne vrši: nema paketa sa tog porta ili faza sadrži adresu izlaznog porta. Maksimalan broj faza je 128 ali tabela može sadržati i manji broj faza.

### 19.7 PCIe Konfiguracija

PCIe uređaj se uvek vodi na nekoj magistrali (bus PCI Express magistrale). Mora imati implementiranu barem jednu funkciju i to funkciju nula. Uređaji sa više funkcija su multifunkcionalni uređai i oni imaju obaveznu funkciju 0 dok ostale ne moraju biti popunjene po redu.

PCIe Bus povezan na PCIe most u smeru na gore (prema root copmlex-u) se naziva primary bus. PCIe Bus povezan na PCIe most u smeru na dole je secondary bus. Po podizanju uređaja softver nije svestan topologije PCIe magistrale i ona se otkriva u procesu enumeracije. Softver koji se ivršava na procesoru pristupa konfiguraciji PCIe uređaja preko root complex-a. Konfiguracija PCIe uređaja je prostor od 4KB. Prvih 16DW u ovom prostoru su PCI header dok je sledećih 48DW PCI device specific prostor a ostatak je PCIe specifična konfiguracija. U tabelama 4 i 5 prikazani su konfiguracije Endpoint i PCIe svič uređaja respektivno.

Konfiguracione transakcije može da obavlja samo root complex. One se obavaljaju isključivo u smeru na dole. Transakcije se rutiraju po magistrali (bus), uređaju (device) i funkciji (function). Otkrivanje da li je funkcija podržana se izvodi upitom za Vendor ID polje u konfiguraciji. Rezultat 0xFFFF govori da funkcija nije podržana.

Polje Header Type otkriva koja je klasa uređaja:

- Vrednost 0 – funkcija nije PCI-to-PCI most
- Vrednost 1 – PCI-to-PCI bridge (aka P2P) povezuje dve magistrale
- Vrednost 2– funkcija je CardBus bridge.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

| | | | |
|---|---|---|---|
| Device ID | | Vendor ID | |
| Status | | Command | |
| Class Code | | | Revision ID |
| BIST | Header Type | Lat. Timer | Cache Line S. |
| Base Address Register 0 | | | |
| Base Address Register 1 | | | |
| Base Address Register 2 | | | |
| Base Address Register 3 | | | |
| Base Address Register 4 | | | |
| Base Address Register 5 | | | |
| Cardbus CIS Pointer | | | |
| Subsystem ID | | Sybsystem Vendor ID | |
| Expansion ROM Base Address | | | |
| Reserved | | | Cap Pointer |
| Reserved | | | |
| Max Lat. | Min Gnt. | Interrupt Pin | Interrupt Line |

**Tabela 6 Zaglavlje PCI konfiguracije u slučaju PCIe Endpoint uređaja**

| | | | |
|---|---|---|---|
| Device ID | | Vendor ID | |
| Status | | Command | |
| Class Code | | | Revision ID |
| BIST | Header Type | Lat. Timer | Cache Line S. |
| Base Address Register 0 | | | |
| Base Address Register 1 | | | |
| Base Address Register 2 | | | |

<table>
<tr><td colspan="4">Base Address Register 3</td></tr>
<tr><td colspan="4">Base Address Register 4</td></tr>
<tr><td colspan="4">Base Address Register 5</td></tr>
<tr><td colspan="4">Cardbus CIS Pointer</td></tr>
<tr><td colspan="2">Subsystem ID</td><td colspan="2">Sybsystem Vendor ID</td></tr>
<tr><td colspan="4">Expansion ROM Base Address</td></tr>
<tr><td colspan="3">Reserved</td><td>Cap Pointer</td></tr>
<tr><td colspan="4">Reserved</td></tr>
<tr><td>Max Lat.</td><td>Min Gnt.</td><td>Interrupt Pin</td><td>Interrupt Line</td></tr>
</table>

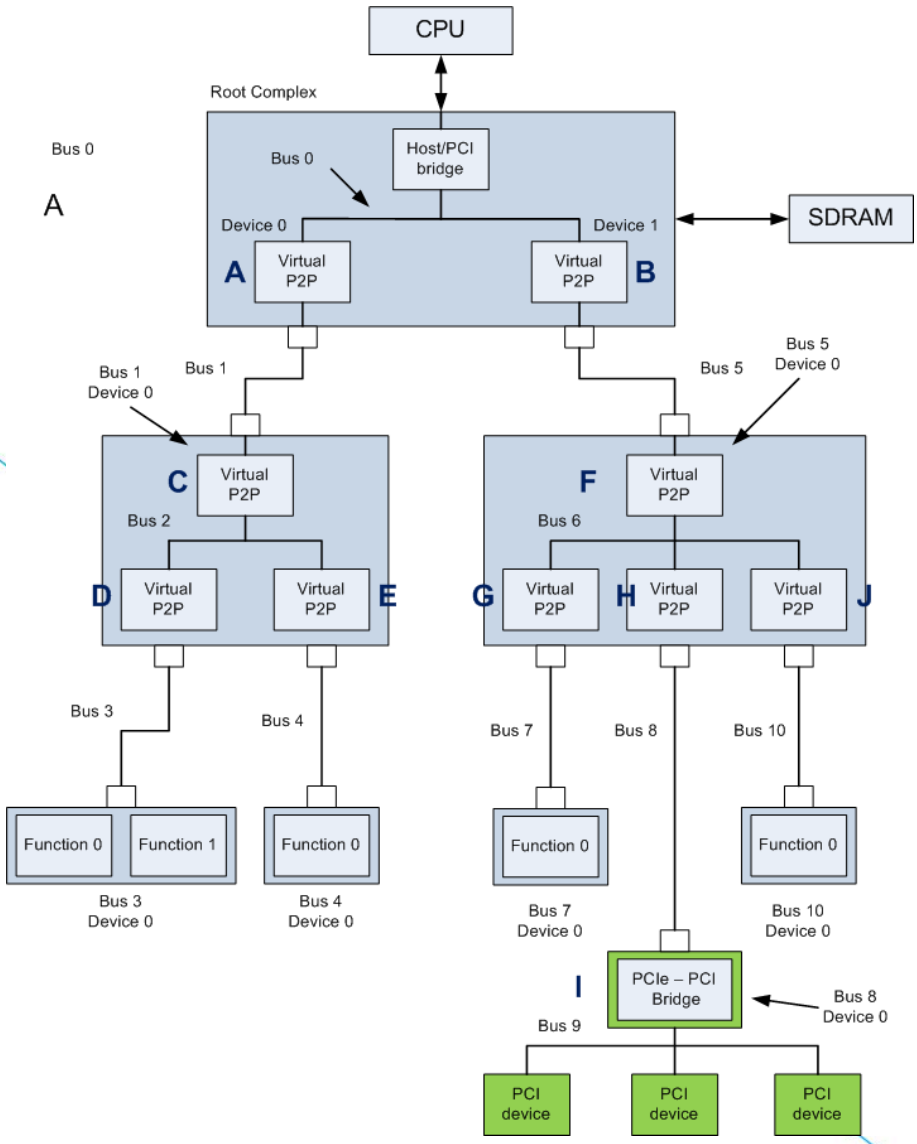
**Tabela 7 Zaglavlje PCI konfiguracije u slučaju PCIe Switch uređaja**

### 19.7.1 Enumeracija PCI Express uređaja

Enumeracija PCIe uređaja je proces kojim se ispituje postojanje uređaja na magistrali i organizacija magistrale kao i dodeljivanje resursa uredjajima. Ispod je opisan postupak enumeracije uredjaja sa sistem čija je arhitektura prikazana na slici 34.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**Slika 91 Primer enumeracije na PCI Express sistemu**

Enumeracija počinje od Uređaja A na magistrali 0 u okviru root complex-a. Ispituje se sadržaj Vendor ID registra za funkciju 0 i ako se vrati vrednost različita od 0xFFFF smatra se da uređaj postoji. Čitanjem Header type registra se utvrđuje da uređaj predstavlja PCI-PCI most.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Sada se vrši niz upisa u konfiguraciju mosta:

Primary Bus Number Register = 0

Secondary Bus Number Register = 1

Subordinate Bus Number Register = 1

Subordinate Bus Number je broj magistale najdalje niz ovaj most. Takođe se upisuje Subordinate Bus Number = 1 u registar Host/PCI mosta.

Sada se čita Capability registar čime se utvrđuje da je u pitanju root port root complex-a. Potraga se nastavlja u dubinu pre provere da li postoje sledeće funkcije (čitanje Vendor ID za bus – 1, device – 0, function – 0). Na magistrali 1 se nalazi svič C koji vrača Vendor ID. Čitanjem Header type registra se utvrđuje da uređaj predstavlja PCI-PCI most. Upitom u capability registar se vidi da se radi o upstream portu sviča.

Sada se vrši niz upisa u konfiguraciju mosta:

Primary Bus Number Register = 1

Secondary Bus Number Register = 2

Subordinate Bus Number Register = 2

Upisuje se Subordinate Bus Number = 2 u registar Host/PCI mosta kao i mosta A.

Pretraga se nastavlja na dole sa uređajem D – portom sviča (device 0). Upit vraća Vendor ID registra za za bus – 2, device – 0, function – 0. Čitanjem Header type registra se utvrđuje da uređaj predstavlja PCI-PCI most. Upitom u capability registar se vidi da se radi o downstream portu sviča.

Sada se vrši niz upisa u konfiguraciju mosta:

Primary Bus Number Register = 2

Secondary Bus Number Register = 3

Subordinate Bus Number Register = 3

Upisuje se Subordinate Bus Number = 3 u registar Host/PCI mosta, mosta A i mosta C.

Pretraga se nastavlja na dole sa uređajem na bus – 3 Endpoint uređajem. Upit vraća Vendor ID registra za za bus – 3, device – 0, function – 0. Čitanjem Header type registra se utvrđuje da uređaj predstavlja multifunkcionalni Endpoint. Upitom u capability registar se vidi da se radi o endpoint uređaju.

Zatim se proverava da li postoje dodatne funkcije i nalazi se da postoji samo funkcija 1.

Pošto je algoritam pretrage stigao do kraja ove grane vraća se jedan korak unazad i pokušava da nađe uređaj na bus – 2, device – 1, function – 0. To je uređaj E port sviča. Čitanjem Header type



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

registra se utvrđuje da uređaj predstavlja PCI-PCI most. Upitom u capability registar se vidi da se radi o downstream portu sviča.

Sada se vrši niz upisa u konfiguraciju mosta:

Primary Bus Number Register = 2

Secondary Bus Number Register = 4

Subordinate Bus Number Register = 4

Upisuje se Subordinate Bus Number = 4 u registar Host/PCI mosta, mosta A i mosta C.

Pretraga se nastavlja na dole sa uređajem na bus – 4 Endpoint uređajem. Upit vraća Vendor ID registra za za bus – 4, device – 0, function – 0. Čitanjem Header type registra se utvrđuje da uređaj predstavlja Endpoint sa jednom funkcijom. Upitom u capability registar se vidi da se radi o endpoint uređaju.

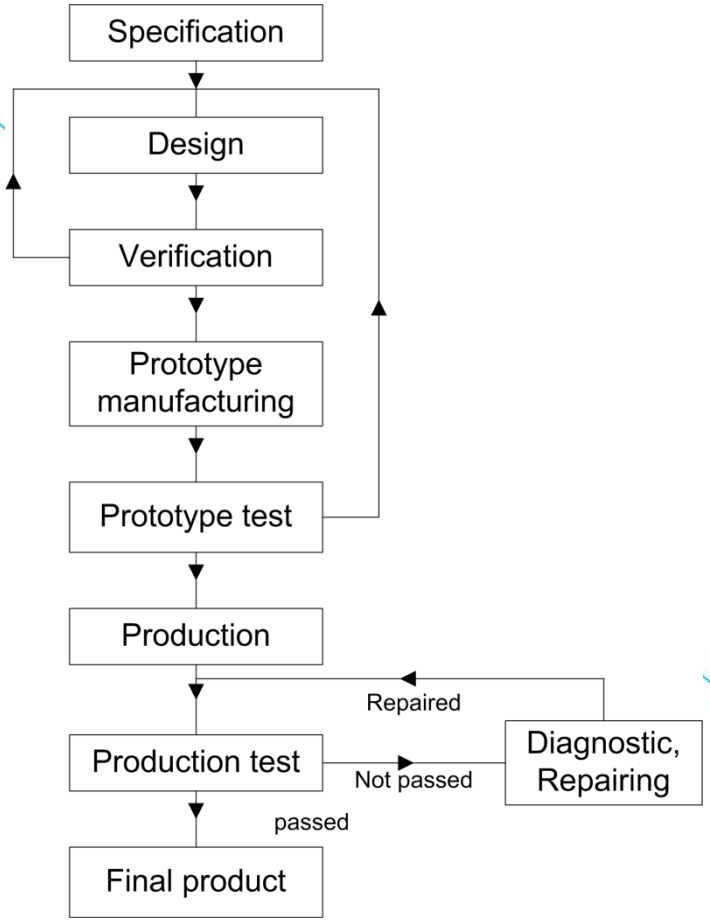
Algoritam se vraća nazad jedan korak i pokušava da nađe device – 2 na bus – 2. Kako nema sledećeg uređaja vraća se na bus 0 i nastavlja potragu po istom principu. Finalni rezultat algoritma je prikazan na slici 34.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 20. Automatsko testiranje i verifikacija dizajna

Verifikacija dizajna predstavlja proveru da li komponente u dizajnu el. uređaja i na kraju sam uređaj odgovaraju zadatim specifikacijama. Sa druge strane testiranje uređaja u proizvodnji predstavlja skup procedura kojima se proverava funkcionalnost uređaja u toku proizvodnje. U drugom slučaju se podrazumeva da je uređaj već verifikovan i da njegove karakteristike odgovaraju specifikacijama. Na slici 35 je prikazana podela na Specifikaciju, dizajn, verifikaciju i proizvodnju.




**Slika 92 Proces dizajna i proizvodnje uređaja po fazama**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

## 20.1 Verifikacija dizajna

Verifikacija dizajna se obično deli na nekoliko međusobno nezavisnih celina i po pravilu se završava funkcionalnim testom celokupnog uređaja. Da bi se omogućila uspešna verifikacija dizajna potrebno je u toku samog dizajna predvideti neophodne pristupne tačke i opcione komponente. Takođe potrebno je formirati verifikacione procedure i osmisliti način verifikacije pojedinih blokova i celog uređaja.

### 20.1.1 Prilagođenje dizajna – Test tačke

Test tačke su neophodne za proveru određenih parametara dizajna. Mogu se koristiti za proveru osnovnih fizičkih veličina (otpornost, kapacitivnost induktivnost, napon i struja) npr. prilikom provere napajanja. Takođe mogu se proveravati i složeniji signali (takt, adresne linije ili linije podataka).

### 20.1.2 Prilagođenje dizajna – Opcione komponente

Prilikom razvoja često je neophodno predvideti opcione komponente ukoliko postoji neizvesnost u pogledu funkcionisanja određenog bloka ili mogućnost odvajanja i nezavisnog testiranja pojedinih blokova. Npr. Izvore napajanja je potrebno odvojiti od potrošača kratkospojnicima tako da se mogu nezavisno proveravati.

### 20.1.3 Prilagođenje dizajna – Opcione komponente

Takođe, u slučaju korišćenja programabilne logike mogu se predvideti različiti vidovi komunikacije sa drugim logičkim kolima kao i opciona proširenja. Potrebno je predviteti i različite pozicije pull-up/down otpornika kojima se podešava određena funkcija čipa (I2C adresa, način rada internog napajanja, JTAG i test pinovi itd.).

### 20.1.4 Proces verifikacije – Ulazna kontrola

Proces verifikacije započinje proverom komponenti koje čine dizajn. Potrebno je proveriti da li su isporučene odgovarajuće elektronske komponente i u određenim slučajevima proveriti karakteristike istih na nekoliko jedinica. Potrebno je proveriti štampane ploče na mehaničke defekte i defekte u izradi. Ukoliko se koriste mehaničke komponente treba proveriti i njihovu izradu i kvalitet i da li odgovaraju specifikaciji.

Ukoliko su zadovoljeni ovi kriterijumi moguće je nastaviti sa asembliranjem ploče/ploča uređaja.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 20.1.5 Proces verifikacije – Asembliranje

Prvi korak prilikom verifikacije dizajna je optička provera. Proverava se kvalitet asembliranja (lemovi, pozicioniranje komponenti, mehaničke deformacije usled zagrevanja). Zatim se vrši provera prisutnosti komponenata u dizajnu prema spisku materijala za ulaganje u radnoj konfiguraciji (BOM).

Ove provere je najbolje vršiti nezavisno da ne bi došlo do preskakanja pojedinih komponenti.

### 20.1.6 Proces verifikacije – Omska provera

Nakon provere asemblirane ploče može se nastaviti sa omskom proverom dizajna. Blokovi napajanja treba da budu odvojeni od blokova potrošača i obe strane se nezavisno proveravaju. Mere se otpornosti grana napajanja prema masi a i međusobno čime se utvrđuje da li postoje kratki spojevi. Nakon provere asemblirane ploče može se nastaviti sa omskom proverom dizajna. Blokovi napajanja treba da budu odvojeni od blokova potrošača i obe strane se nezavisno proveravaju. Mere se otpornosti grana napajanja prema masi a i međusobno čime se utvrđuje da li postoje kratki spojevi. U specifičnim uređajima mogu se proveravati i ostale fizičke veliline (induktivnost, kapacitivnost itd.).

### 20.1.7 Proces verifikacije – Provera blok napajanja

Napajanja se testiraju nezavisno koliko je to moguće i to od napajanja za više napone. Često se napajanja sa višim naponom i većim strujnim kapacitetom koriste kao izvori za napajanja manjeg napona (12V, 5V, 3V3). Tada se prvo proveravaju ova napajanja dok su spojevi ka nižim izvorima raskinuti.

Napajanja se testiraju na:

- Nominalni napon
- Strujni kapacitet (meri se jednosmerni napon u radnom opsegu struja)
- Talasnost u celom opsegu struja
- Zagrevanje komponenti pri maksimalnom opterećenju

Ako je provera napajanja uspešno realizovana napajanje se povezuje na potrošače i uređaj se podiže u stanju reseta. Sada se mogu proveriti potrošnje pojedinih blokova i potvrditi da su u očekivanim granicama.

### 20.1.8 Proces verifikacije – Generatori Takta, PLL, Reset

Dok je sistem u resetu vrši se provera generatora takta (oscilatori, kristali) i PLL čipova. Potrebno je proveriti da li se oni nalaze u zahtevanim specifikacijama čipova koji se koriste na dizajnu. U ovom trenutku je moguće programirati PLL blokove koji su konfigurabilni. Proverava se reset na dolazak/nestanak napajanja, watchdog funkcija i sekvenca resetovanja.

### 20.1.9 Proces verifikacije – Funkcionalna provera

Ako su blokovi napajanja, taktovi i reset funkcionalnost proverene može se pristupiti programiranju programabilne logike (FPGA, CPLD) tj. Spuštanju test kodova u memoriju procesora. Pre svega je potrebno testirati memorije i zatim međusobnu komunikaciju blokova. Sada je moguće proveriti funkcije blokova nezavisno i u kombinaciji sa zavisnim modulima.

Na kraju se testira funkcionalnost celokuponog uređaja u simuliranim i realističnim uslovima rada.

### 20.1.10 Proces verifikacije – EMC test, Temperaturni opseg

Da bi uređaji dobili odgovarajuće međunarodne sertifikate neophodno je izvršiti EMC testove i to: Test na električno pražnjenje, zračenje uređaja kao i otpornost na spoljašnje EM polje. Takođe, potrebno je testirati ih na rad u traženom temperaturnom opsegu (komercijalni, industrijski ili vojni).

## 20.2 Automatsko testiranje u proizvodnji

Testiranje u proizvodnji po koracima otprilike odgovara verifikaciji sa tim da se ne vrše kompleksne provere koje se odnose na proveru dizajna. Ovaj proces se u proizvodnji srednjih/većih serija uređaja (od hiljadu pa na više jedinica) često automatizuje. Automatizacija testiranja pre svega omogućava veliko smanjenje verovatnoće da neispravan proizvod stigne do kupca. Automatizacija takođe omogućava eliminaciju grešaka operatera kao i veliku uštedu vremena u proizvodnji.

Na taj način mogu se ostvariti i velike uštede u novcu

### 20.2.1 Tipovi neispravnosti

Defekti u proizvodnji štampanih ploča mogu biti različiti: prekidi veza, kratki spojevi i mehanički defekti. Zatim može se javiti neispravnost komponenata. Greške u asembliranju su jedan od čestih uzroka neispravnosti. Javljaju se: loše uložene komponente, lemovi, neuložene komponente, pogrešna orjentacija, ulaganje pogrešnih komponenata kao i neuložene komponente. Greške operatera takođe mogu uzrokovati neispravnost uređaja – ESD (električno pražnjenje) i greške prilikom rukovanja.

### 20.2.2 Optička kontrola

Većina neispravnosti prilikom asembliranja se može otkloniti uređajima za automatsku optičku inspekciju. Ovde se koristi referentna ploča gde se sve pozicije komponenata snimaju kamerom i ove slike se koriste kao referenca za proveru kvaliteta. U slučaju neslaganja operater mora da potvrdi da je komponenta dobro uložena što se može memorisati za sledeće ploče (Slika 36).

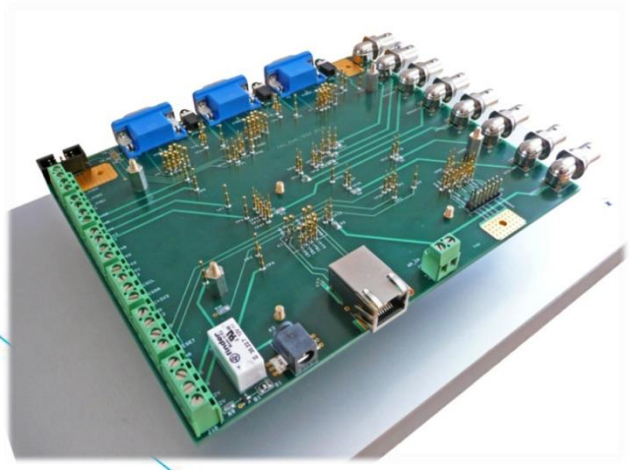
**Slika 93 Primer mašine za automatsku optičku inspekciju**

### 20.2.3 In Circuit Testing (ICT)

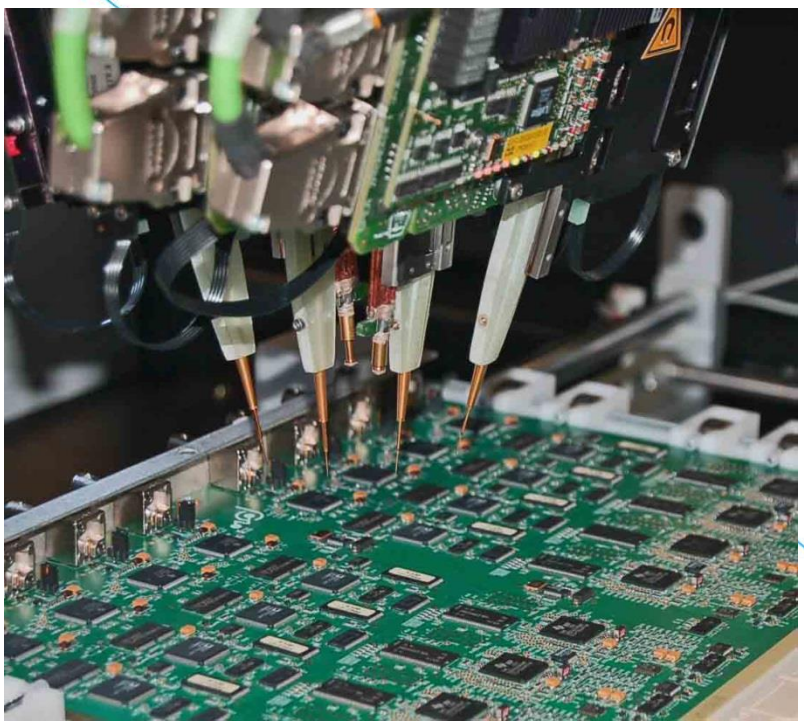
Ovaj tip testiranja proverava osnovne parametre kola (otpornost, induktivnost i kapacitivnost). Postoje dve metode za ovakvu proveru: Bed of nails tester (Slika 37) i Flying probe tester (Slika 38). Flying probe tester omogućava lak postavljanje test sistema uz nešto duže vreme testiranja. Bed of nails princip omogućava brže testiranje ali se mora dizajnirati test podnožje.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES



**Slika 94 Bed of nails test podnožje**



**Slika 95 Flying probe tester**

### 20.2.4 Boundary scan

Boundary scan testiranje se koristi za proveru interkonekcija između čipova korišćenjem test portova na samim čipovima kao i interne logike (JTAG Test Access Port, Boundary scan shift register). Moguće



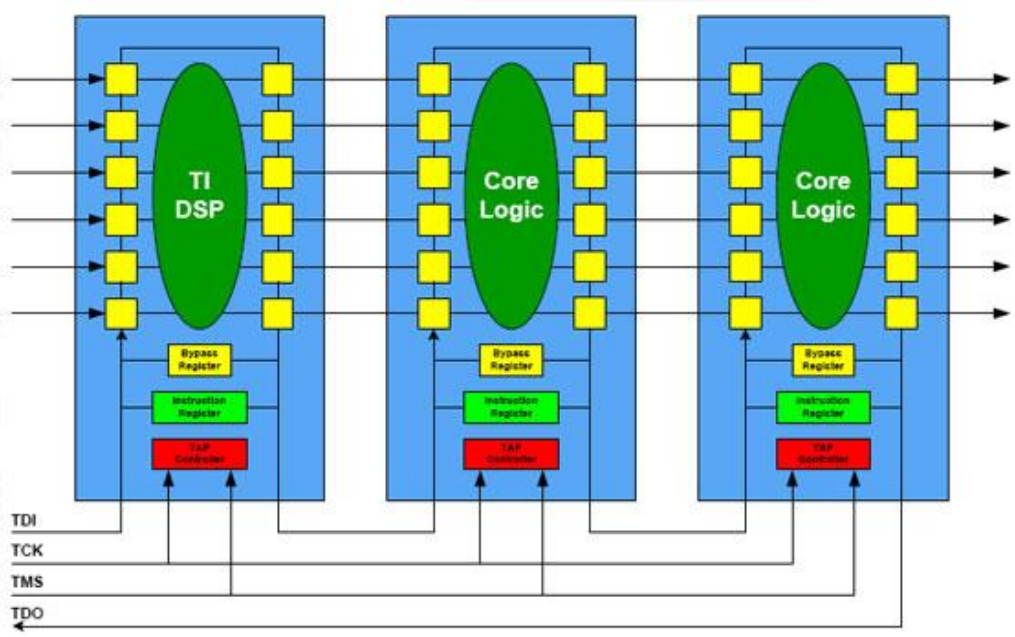
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

je povezati više čipova u JTAG lanac čime se omogućava istovremeno generisanje i akvizicija logičkih signala sa više čipova (Slika 39).

Testovi se zadaju korišćenjem BSDL (Boundary scan description language) jezika. Test okruženje se sastoji iz JTAG kontrolera i odgovarajućeg sovtvera za automatizaciju testa (kompanije XJTAG, JTAG Technologies, Corelis itd.).

### 20.2.5 Virtuelna instrumentacija

Za visoku automatizaciju testiranja moguće je koristiti virtuelnu instrumentaciju (instrumente u obliku PC periferijskih kartica) i odgovarajuća okruženja (npr. LabView/CVI). Ovakva rešenja omogućavaju obradu podataka na PC računarima – potpuno prilagođene programe testiranja kao i automatsko kreiranje izveštaja o testiranju sa svim relevantnim parametrima.




**Slika 96 Boundary Scan – Povezivanje uređaja**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 21 Literatura


[43] http://resources.infosecinstitute.com/system-address-map-initialization-in-x86x64-architecture-part-1-pci-based-systems/

[44] http://resources.infosecinstitute.com/system-address-map-initialization-x86x64-architecture-part-2-pci-express-based-systems/

[45] PCI express system architecture,
Ravi Budruk, Don Anderson, Tom Shanley Mindshare, Inc. 2003

[46] https://www.pcisig.com/developers/main/training_materials/get_document?doc_id=4e00a39acaa5c5a8ee44ebb07baba982e5972c67

[47] http://xillybus.com/tutorials/pci-express-tlp-pcie-primer-tutorial-guide-1

TVOJA KARIJERA U TVOJIM RUKAMA

# *Software Developer*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# Web programiranje 1

**Predavač: Slavimir Stošović**

**Ciljevi predmeta:**
Upoznavanje sa konceptom i metodologijom HTLM/HTML5, CSS3 i JavaScript jezikom i Graphical User Interface (GUI).

**Ishod predmeta:**
Ovladavanje osnovnim stvarima HTML programiranja i suštinom kao i tehnikama za stvarnje efektivnog grafičkog okruženja za korisnika.

**Teorijska nastava:**
Upoznavanje – Internet tehnologija, servisi, protokoli, klijent i server.
HTML Osnove – pravila, elementi i atributi, slike, stavke i tabele.
HTML5 – video , audio i canvas elementi.
CSS and CSS3 – Selektori, Pozadina i okviri, skale, tekstualni efekti, 2D/3D transofmacije, animacije, zaglavlje, navigaciona skala,…
JavaScript – Pregled JavaScript sintakse, promenjivih, operatora, funkcija.
Jquery – Osnove i implementacija
Tableless i moderan dizajn
Cross-platform and cross-browser kompatabilnost.

**Praktične vežbe:**
50% radnih sati. Implementacija modernog prikaza i dizajn uz korišćenje HTML, HTML5, CSS, CSS3 i JavaScript(jQuery)

**Kurikulum:**
Prva nedelja je predviđena za upoznavanje sa opštim pojmovima i hypertext markup language (HTML) sintaksa i tagovi. Sadržaj i tempo učenja zavisi od znanja i stečenog iskustva kandidata.
Za vreme druge nedelje, student će se upoznavati sa HTML5, i uporedo će učiti kako implementirati CSS u stranicu, sa ciljem da podese vizuelni izgled stranice.
U trećoj nedelji student će se upoznati sa JavaSript – om u cilju upoznavanja sa web programiranjem sa klijentove strane.
U završnoj nedelji kandidati će upotrebljavati JQuery biblioteke za kreiranje modernog grafičkog izgleda.
Na kraju svake nedelje kandidati će biti testirani.
Na kraju kursa, kandidati će biti testirani, praktičnim radom na računaru.
Sve teme sadrže veliki broj praktičnih vežbi.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Ceo kurs je koncipiran na praktičnom radu.

**Ocenjivanje:**

1. Nedeljno testiranje – ABCD test. Student treba da odgovori na 50 % ili više pitanja tačnim odgovorom. Test će biti ocenjivan od 1 do 3 , za prolaz moraju imati 2 ili 3 ocenu.
2. Finalni test – praktičan rad, rad na računaru. Student mora da obezbedi 60% i više funkionalnosti aplikacije
3. Studen će položiti samo ako položi sva 4 testa i obezbedi 60% i više funkcionalnosti aplikacije na praktičnom testu.

**Test**

Pitanja za test i teme projektnih zadataka neće biti objavljivane zbog poverljivosti.

# 1. Uvod

Savremene tehnologije menjaju ljudsko društvo te imaju mnogostruko veći značaj nego u prošlosti. Poslednju deceniju je obeležila ekspanzija korišćenja računarskih mreža u poslovne i privatne svrhe. Nastanak računarskih mreža se vezuje za ukrštanje dve tehnologije sa vrlo različitom istorijom i tradicijom: *računara* i *telekomunikacija*. Računarske mreže su na svom početku proširivale korišćenja računara ali danas služe za zadovoljavanje narasle ljudske potrebe za komunikacijom. One su bitan deo infrastrukture u postindustrijskom svetu. Sve veća potreba za razmenom informacija na globalnom sloju je jedan od najznačajnijih savremenih tehnoloških trendova. Kao odgovor na ovakve narasle potrebe velike svetske korporacije su ponudile rešenja stvaranjem globalnih računarskih mreža. Prve ovakve mreže pojavile su se sedamdesetih godina prošlog veka.

Savremene mreže su heterogene i u njima se primenjuju računari različitih proizvođača, što je na početku razvoja računarskih mreža bio ozbiljan problem. U to vreme nije postojao jedinstveni standard koji bi regulisao interfejse, formate i procedure.

Tek kada se u rešavanje ovog problema uključila Međunarodna organizacija za standarde ISO i objavila svoj referentni model mrežne arhitekture, počeo je ubrzani razvoj globalnih računarskih mreža. Ova arhitektura poznata je pod nazivom ISO OSI (eng. *Open Systems Interconnection*) što znači otvoreni sistem povezivanja.

Osnovna podela savremenih računarskih mreža je na dve velike grupe: *lokalne mreže,* mreže koje se karakterišu velikom brzinom prenosa podataka i informacija ali na relativno ograničenoj udaljenosti između računara, i *globalne mreže,* mreže koje nemaju ograničenja vezana za teritorijalnu pripadnost, ali zato obezbeđuju znatno manje brzine prenosa. Najveća, najznačajnija i najpoznatija globalna računarska mreža je *Internet*.

Internet nazivamo i "mrežom svih mreža" jer povezuje milione računara širom sveta. Internet je sastavljen od velikog broja lokalnih i globalnih mreža koje koriste TCP/IP protokole u međuračunarskim komunikacijama. Internet nije jedinstvena mreža sa centralizovanom kontrolom, već je povezani skup nezavisnih mreža koje koriste iste protokole, standarde i interfejse.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 2. Globalne računarske mreže

Internet kao globalna mreža pruža najrazličitije informacije u obimu koji se praktično ne može odrediti. Pronalaženje tačno određenog dokumenta na Internetu predstavlja izuzetan problem obzirom na more raspoloživih informacija. U cilju bržeg pronalaženja i jednostavnijeg pristupa željenim informacijama ustanovljeno je nekoliko osnovnih servisa. Korisnik doživljava Internet kao skup mrežnih usluga i servisa. Usluge i servisi Internet-a su uvek organizovani po klijent-server principu. Server je specijalni softver na nekom mrežnom računaru koji opslužuje zahteve korisnika. Onaj koji upućuje zahtev mora da startuje odgovarajući program, zvani klijent, na svom računaru. Klijent aplikacija koja želi neku uslugu, obraća se serveru slanjem poruke sa opisom zahtevanog zadatka. Server opslužuje zahtev i odgovara klijentu svojim porukama. Skup važećih poruka definisan je protokolom. Server može usluživati više klijenata odjednom. Usluge višeg sloja nazivaju se **mrežni servisi**. Neki od najvažnijih servisa opisani su u nastavku teksta.

# 3. Servisi na Internetu

**Word Wide Web** ili skraćeno **WWW** korisnicima pruža informacije u najatraktivnijem multimedijalnom obliku. Elektronska pošta i diskusione grupe su takođe veoma popularni servisi koji služe za razmenu poruka kao i mnogi drugi drugi oblici. Ovaj servis na Internetu je veoma brzo postao najpopularniji jer obezbeđuje lako, brzo i jednostavno pronalaženje željenog sadržaja na globalnoj računarskoj mreži. Za korišćenje naprednih mogućnosti Web-a razvijene su posebne korisničke aplikacije - pretraživači (eng. *Browsers*). Pretraživačima se može jednostavno posećivati razne Web lokacije - sajtove, koje sadrže tražene informacije u vidu tekstova, slika i drugih multimedijalnih sadržaja. Svaka lokacija na Web-u ima svoju sopstvenu adresu, poznatu kao URL (eng. *Uniform Resourse Location*). URL adresa je neophodna da bi se usmerio Web pretraživač na tačno određeno mesto na Internetu i pristupio željenom sadržaju. Unos nove Internet adrese može da Vas poveže sa sasvim drugim krajem sveta te se zato korišćenje Web-a često naziva „surfovanje".

**E-mail** - elektronska pošta je jedan od najstarijih i najčešće korišćenih servisa na Internetu. Slično kao kod tradicionalnog komuniciranja putem pisma, korisnik može komunicirati sa bilo kojim korisnikom Interneta sa neuporedivo kraćim vremenom prenosa poruka i to po neuporedivo nižoj



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

ceni. E - mail adresa je neophodna za ovaj vid komunikacije na Internetu, i nju dobija svaki korisnik otvaranjem naloga za pristup kod svog lokalnog Internet provajdera. Otvaranjem naloga za pristup Internetu dobijate posebno "poštansko sanduče" na računaru provajdera u koje će pristizati sve Vaše poruke. Korisnik mora povremeno proveriti njegov sadržaj i po potrebi prenese pristigle poruke na svoj računar. Primenjujući opisani koncept jasno je da pri tome nije važna međusobna udaljenost pošiljaoca i primaoca poruka. Jednostavnom tekstualnom sadržaju poruke mogu biti pridodati najrazličitiji prateći sadržaji kao što su slike, tonski i video zapisi.

Na **diskusionim grupama** koje se vode na Internetu možete se jednostavno uključiti odabirajući željenu temu. Diskusione grupe funkcionišu kao neka vrsta oglasne table - neko istakne poruku a onda svako ko to želi, može da ostavi svoj odgovor, i na taj način se priključi diskusiji. Teme koje o kojima se razmenjuju mišljenja obuhvataju praktično svaki vid ljudske delatnosti, od običnih iz svakodnevnog života pa sve do rasprava iz oblasti nauke, politike, religije ili neke druge stručne discipline. U slučaju usmerenih diskusija moderator (osoba koja je zadužena za održavanje reda na “tabli”) ima pravo da odbije objavljivanje poruke ako ona nije u skladu sa pravilima grupe. Korisnik se može opredeliti za aktivno ili pasivno učešće u raspravi kada samo prati šta drugi učesnici imaju da kažu na zadanu temu.

Mogućnost direktnog praćenja **radio i TV programa** putem Interneta je veoma popularana i mnogim korisnicima interesantna. Postoje radio stanice koje jedino preko ovog medijuma emituju svoj program! Za ovaj način emitovanja programa oslobađate se troška skupog održavanja emisione tehnike. Kao i kod drugih tehnologija koje su još u razvoju, praćenje ovakvih sadržaja je često skopčano sa određenim problemima. Najveći od njih je mala propusna moć većine komunikacionih kanala na Internetu.

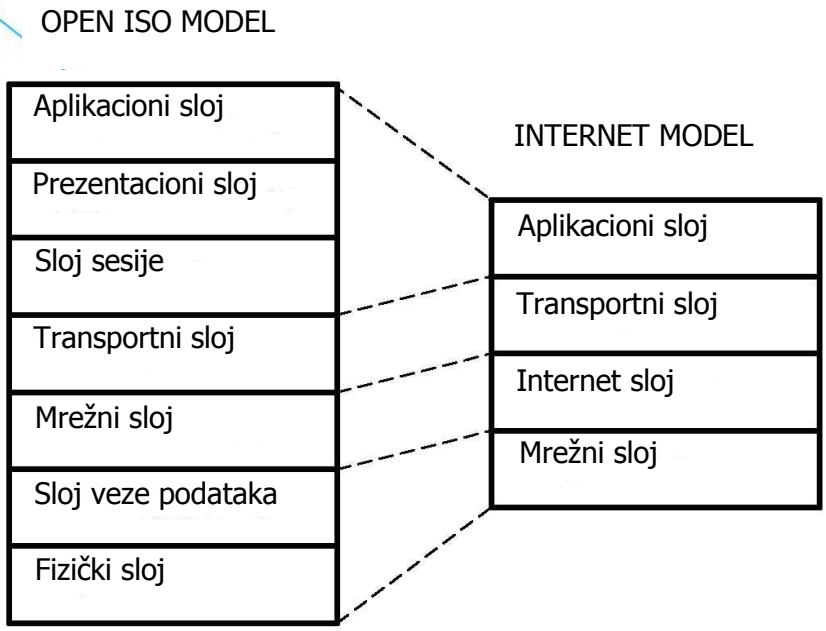
Alternativa živom audio i video prenosu na Internetu su tzv. **kanali** (eng. *channels*) koji omogućavaju da se željeni sadržaj automatski prenese do korisnikovog računara a kasnije omogući njegov pregled. Ovaj postupak je obično automatizovan te se npr. najnovije vesti svakog jutra prosleđuju vašem računaru da biste ih pregledati kada za to imate vremena.

# 4. TCP/IP protokoli

TCP/IP je skraćenica za "Transmission Control Protocol / Internet Protocol". Uloga ovih protokola je da osiguraju komunikaciju između svih računara na Interneta. Zahvaljujući ovom opšteprihvaćenom standardu, moguće je povezati bilo koja dva računara bez obzira na njihovu vrstu,

tip i druge specifičnosti. TCP/IP stek protokol ustvari obuhvata preko sto različitih protokola, a njihov broj stalno raste. TCP/IP protokoli povezuju tri osnovne komponente pri komuniciranju: otvorene procese na pojedinim računarima, same računare na kojima se obavljaju najrazličitiji poslovi i komunikacione mreže. Procesi su osnovne logičke celine koje imaju potrebu za komuniciranjem. Procesi se aktiviraju, izvršavaju i stopiraju na računarima koji mogu podržati veći broj procesa istovremeno.

Komunikacija među procesima se odvija putem komunikacione mreže na koju su računari povezani. ISO OSI je sedmoslojni referentni model koji opisuje komunikacionu arhitekturu na Internetu. Internet arhitektura podrazumeva četiri sloja protokola. Na slici 1.1 je prikazana struktura Internet protokola.




*Slika 1.1 Međusobni odnos OPEN ISO i Internet referencnog modela*

*Aplikacioni sloj* (Process/Application Layer) predstavlja protokole najvišeg sloja. U osnovi Internet aplikacioni sloj kombinuje funkcije dva najviša sloja OSI referentnog modela: prezentacionog i aplikacionog. Razlikuju se dve kategorije protokola aplikacionog sloja: korisnički protokoli koji pružaju usluge direktno krajnjem korisniku i protokoli za podršku koji obezbeđuju sistemske funkcije. Postoji veliki broj standardnih korisničkih protokola. Najpoznatiji korisnički protokoli su **TELNET** (rad na udaljenom računaru), **FTP** (prenos datoteka) i **SMTP** (elektronska pošta). Od protokola za podršku pomenimo **Domain Name System (DNS)** protokol koji se koristi za prevođenje naziva hostova u IP adrese.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

*Transportni sloj* (Host-Host Layer) obezbeđuje komunikacione usluge "s kraja na kraj" *(end-to-end)* koje koriste aplikacije. To znači da su poruke transportnog sloja od značaja samo za krajnje čvorove (ne i za komutacione međučvorove). Dva protokola ovog sloja su: **Transmission Control Protocol (TCP)** i **User Datagram Protocol (UDP)**. **TCP** je orijentisan na uspostavljanje virtuelne veze, i obezbeđuje pouzdanost tj. sekvencioniranje i kontrolu toka "s kraja na kraj". Virtuelna veza je logička veza između krajnjih učesnika u komunikaciji, koja izgleda kao fizička veza (najčešće sa izvesnim kašnjenjem u transportu). Napomenimo da svaki TCP paket, pored ostalih informacija, sadrži i dva šesnaestobitna broja koji se nazivaju izvorni i odredišni *port.* Pošto se komunikacija vrši između procesa na dva udaljena računara, izvorni port *(Source Port)* služi da identifikuje proces na izvornom hostu, dok odredišni port *(Destination Port)* identifikuje proces na odredišnom hostu. Pošto se na jednom računaru može izvršavati veći broj procesa, između dva hosta se može uspostaviti više virtuelnih veza istovremeno, a svaka je jednoznačno identifikovana sa pomenuta dva broja. Postoje brojevi koji su rezervisani za pristup pojedinim server-programima *(well-known port numbers).* **UDP** je protokol datagramskog tipa, tj. nema uspostavljanja virtuelne veze kao kod TCP, već se paketi šalju nezavisno jedan od drugog. Time je smanjena pouzdanost, ali je UDP jednostavniji od TCP.

*Internet sloj, s*vi protokoli transportnog sloja na *Internet-u* koriste IP *(Internet Protocol)* da bi preneli podatke od izvornog do odredišnog hosta. IP obezbeđuje usluge datagram tipa *(connectionless)* tj. ne pruža garancije za isporuku s kraja na kraj. Dakle, IP datagrami mogu da stignu do odredišnog hosta oštećeni, duplicirani, u lošem redosledu ili da čak uopšte ne stignu. Kada se traži pouzdana isporuka, za to su zaduženi slojevi iznad IP. IP obezbeđuje mehanizme za adresiranje, specifikaciju tipa usluge, fragmentaciju i sastavljanje datagrama, a sadrži i neke sigurnosne informacije.

*Sloj pristupa mreži* (Network Access Layer) sloj veze obezbeđuje komunikaciju hosta sa mrežama povezanim sa njim. Host mora da ima implementiran komunikacioni protokol koji se koristi kao interfejs sa tom mrežom. Taj protokol naziva se protokol sloja veze. Sloj veze ima zadatak da fizičku vezu učini pouzdanom korišćenjem mehanizama za kontrolu grešaka i da obezbedi aktiviranje, održavanje i deaktiviranje veze. Tabelarni prikaz funkcija pojedinih slojeva dat je u tabeli.

| IME SLOJA | FUNKCIJA | PRIMER |
|---|---|---|
| *Aplikacioni sloj* | Sadrži skup protokola definisanih različitim aplikacijama kako bi se obezbedila tražena usluga. | HTTP, telnet, e-mail (SMTP, POP) |
| *Transportni sloj* | Obavlja komunikaciju između krajnjih komunikacionih tačaka | Transmission control protocol (TCP) and user datagram |



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

| | | |
|---|---|---|
| | | protocol (UDP) |
| *Internet sloj* | Isporučuje i rutira datagrame između Internet čvorova. | Internet protocol (IP), Internet control message protocol (ICMP), address resolution protocol (ARP) |
| *Mrežni sloj* | Host-specifična implementacija transmisije datagrama. | Ethernet (IEEE 802.3), point-to-point protocol (PPP), AX.25 |

Osim opšte prihvaćenog protokola za prenos informacija veoma je važan sistem adresiranja. Da bi se povezao na Internet svakom računaru mora biti dodeljen identifikacioni jedinstveni broj poznatiji kao **IP adresa**. Ovu adresu čine brojevi koji se pišu u četiri niza kao npr. 159.97.6.175, i predstavlja binarni broj dužine četiri bajta (32 bita). Adresa počinje mrežnim brojem (network number) a ostatak se naziva "lokalna adresa". Pamćenje adresa u ovako kodiranom obliku nije podesno za čoveka te se formiraju tzv. URL ili Web adrese koje se sastoje od dve ili više reči razdvojenih tačkom.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# Vežba br.1 - HTML OSNOVE

**CILJ VEŽBE**

- Upoznati se sa načinima označavanja u HTML-u
- Razumevanje elemenata i atributa
- Naučiti osnovnu strukturu HTML dokumenta
- Savladati pisanje osnovnih tekstualnih elemenata kao i kreiranje veza sa drugim stranicama

**TEORIJSKE OSNOVE ZA IZRADU VEŽBE**

**HTML** označava englesku skraćenicu od **H**yper **T**ext **M**arkup **L**anguage i predstavlja jezik web stranica, poznat pod nazivom „hipertekstualni jezik za označavanje“. Veoma je važno da se razume da HTML nije programski jezik. To je jezik za označavanje koji postavlja uputstva ili oznake (eng. *tags*) oko teksta koji pretraživaču nalaže kako da na ekranu prikaže određeni sadržaj. HTML zapravo predstavlja skup instrukcija za Web pretraživač.

Web stranica se sastoji od niza HTML naredaba koje se unose u tekst datoteku korišćenjem programa za obradu teksta. Čitači Weba na računaru posetioca prate naredbe iz tekstualnog dokumenta i na ekranu prikazuju sadržaj stranice. Sadržaj Web stranice čine tekst, slike, video, zvuk i sve ostalo što se pomoću HTML-a na nju postavi.

Web čitači (eng. *web browsers*) su programi koji prevode HTML iz dokumenta web stranica i prikazuju tekst, slike, animacije na ekranu računara posetioca. Od verzije Web čitača (različiti proizvođači) donekle zavisi šta će posetilac videti na ekranu. Web server na kome se nalazi Web strana i Web čitač za komunikaciju koriste protokol za prenos hiperteksta HTTP (eng. *Hyper Text Transfer Protocol*).



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**ALATI ZA KREIRANJE WEB STRANA**

Postoje brojni besplatni ali i komercijalni alati pomoću kojih se mogu kreirati web strane, a koji su u skladu sa HTML standardima. Neki alati su toliko usavršeni da znanje HTML-a skoro nije ni potrebno, kao npr. Adobe Dreamweaver. Međutim, kada je potrebno da se kreira nešto van šablona, prilagodi strana ili uradi nešto izvan skupa komandi iz menija, neophodno je poznavanje HTML-a. Zbog toga je većina profesionalno izrađenih web strana kreirana ručno, bez upotrebe alata, jer je na taj način omogućen potpuni uticaj na izgled i ponašanje strane. Sve što je potrebno jeste najjednostavniji program za uređivanje teksta (Notepad, Wordpad ili sl.), pretraživač Web-a (Google Chrome, Mozilla Firefox, Internet Explorer, Opera,...) i poznavanje HTML jezika. Osim toga potrebno je i poznavanje CSS-a (eng. *Cascading Style Sheet*) za raspoređivanje elemenata na stranu odnosno vizuelno oblikovanje strane.

Ne treba mešati editore teksta sa softverskim programom za obradu teksta. Između njih postoje značajne razlike. Mnogi moderni proogrami za obradu teksta mogu da iz teksta izrade HTML dokument, ali takođe u njega uključuju brojne skrivene kodove koji mogu praviti probleme za neke pretraživače. Editori obogaćenog teksta (eng. *Rich Text Editor*) nemaju mnogo mogućnosti za formatiranje i ne dodaju dodatne kodove u izrađene dokumente.

U vežbama je korišćen editor teksta Notepad++, koji predstavlja poboljšani, besplatni softver za uređivanje teksta i može se preuzeti sa adrese http://notepad-plus-plus.org.

Svaki računar je podešen tako da otvara HTML dokumenta pomoću web pretraživača koji je odabran za standardni pretraživač. Dovoljno je samo da se dvaput klikne levim tasterom miša na bilo koju datoteku sa ekstenzijom .html ili .htm i datoteka će se otvoriti u web pretraživaču koji je određen kao standardni.

**OSNOVNI HTML ELEMENTI**

Uobičajeno je da oznake u HTML-u imaju početnu i završnu signaturu. Takve oznake se nazivaju uparene oznake (engl. *container tags*). Matematičkim znakom za manje "**<**" obeležava se početak uparene oznake. Iza početne asignacije sledi naziv oznake i atributi, a zatim i znak "**>**". Kraj dejstva uparene oznake obeležava se završnom oznakom. Svaka početna oznaka ima odgovarajuću uparenu završnu oznaku i pravi se matematičkim znakom za manje "**<**", kosom crtom (**/**) ispred naziva oznake, a zatim nazivom oznake i znakom veće "**>**".



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Početne oznake imaju oblik:

```
<naziv oznake [atributi]>
```

, a završne

```
</naziv oznake>
```

Naziv oznake čitaču definiše njenu namenu, a atributi (koji su opcioni) iza naziva oznake daju čitaču dodatne informacije vezano za tu oznaku. Oznake za početak i kraj pasusa (`<p>` odnosno `</p>`) primenjuju se na deo teksta. U ovom primeru oznaka `<p>` nalaže čitaču da prikaže sav tekst do oznake `</p>`, a nakon toga kreira jedan prazan red, primenjujući podrazumevana pravila formatiranja:

```
<p> Tekst između početnih i krajnjih oznaka. </p>
```

Osnovna struktura HTML dokumenta je univerzalna, i ista je za sve web strane, bez obzira na to da li se na ekranu pojavljuju u jednostavnom ili složenom obliku. Neki osnovni elementi zajednički za sve HTML dokumente kreiraju tu strukturu. U osnovne HTML elemente spadaju definicija tipa dokumenta, glavni element (zaglavlje) i telo elementa.

Prva linija svakog HTML dokumenta, bez obzira na to šta je još uključeno u dokument, mora biti DTD definicija ili definicija tipa dokumenta (eng. *Document Type Declaration*), odnosno DOCTYPE Declaration. DTD saopštava pretraživaču da očekuje HTML elemente a ne neki drugi markerski jezik ili drugi tip dokumenta. Zbog ove funkcije, pre DTD-a ne sme da postoji prazan prostor odnosno prazne linije. Ovo je jako bitno, jer ukoliko pretraživač prepozna prazne linije ispred DTD-a, on će se ponašati kao da DTD-a nema i rezultati mogu biti nepredvidljivi. U narednom primeru definicija tipa dokumenta identifikuje verziju HTML-a kojim je kodirana stranice. Ova informacija je važna čitačima kako bi korektno prikazali stranu. Primer ispravne definicije tipa dokumenta:

```
<!doctype html public "naziv verzije" "url">
```

| | |
|---|---|
| `html` | Označava kod kao i verziju HTML-a. |
| `public` | Označava dozvoljen javni pristup jeziku. |
| `naziv verzije` | Označava potpuno ime određene verzije HTML-a. |
| `url` | Označava adresu na Webu gde se nalazi javna definicija HTML koda. |



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

1. Najnovija verzija HTML-a nosi oznaku HTML5. Kod prethodnog standarda HTML-a postoje tri različite verzije:
2.

- HTML 4.01 Strict

Ova DTD verzija sadrži sve HTML elemente i atribute, ali ne sadrži elemente za prezentaciju i pozicioniranje (kao što su `font` i `center`). Frejmovi takođe nisu podržani.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

3.

- HTML 4.01 Transitional

Ova DTD verzija sadrži sve HTML elemente i atribute, uključujući i elemente za prezentaciju i pozicioniranje (kao što su `font` i `center`). Frejmovi nisu podržani, kao i u prethodnoj verziji.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

4.

- HTML 4.01 Frameset

Ova DTD verzija je identična verziji Transitional uz podršku za frejmove.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
```

Za detaljnije informacije, o tome koji su HTML/XHTML elementi podržani određenim standardom, pogledati **Dodatak A**, na kraju knjige.

Da bi ste proverili da li ste pravilno napisali vaš HTML/XHTML dokument možete koristiti deo sajta konzorcijuma W3 za proveru ispravnosti koda, na adresi http://validator.w3.org/ .

Par oznaka koje se pojavljuju na početku i na kraju HTML dokumenta, a između njih se nalaze sve oznake za definiciju izgleda Web stranice, su html oznake.

```
<html>
...
</html>
```

Zaglavlje ili glavni element se koristi za informacije o naslovu stranice, imenu autora i datuma postavljanja stranice i slede neposredno iza početnih `<html>` oznaka. Oznake za zaglavlje web stranice su:

```
<head>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
...
</head>
```

Naslov stranice se prikazuje na korisnikovom ekranu, i treba ga upisati između gore navedenih oznaka. Par ovih oznaka je:

```
<title>
...
</title>
```

Početna i završna oznaka tela stranice postavljaju se odmah iza zaglavlja web stranice. Tekst u telu oznake definiše čitaču šta i na koji način da prikaže na ekranu posetioca web stranice:

```
<body>
...
</body>
```

Često je poželjno da se unutar koda ostavi neki komentar, kao objašnjenje dela koda, datum kada je kod pisan ili ime autora. Postavljene oznaka komentara `<!-- -->` oko iskaza, govore čitaču da ignoriše te iskaze. Da bi napravili komentar, potrebno je da se tekst komentara obuhvati početnom i završnom oznakom komentara:

```
<! -- komentar -->
```

Za korektno prikazivanje specifičnih jezičkih karaktera unutar HTML strane, neophodno je da se web čitaču naznači koji skup karaktera koristi. Skup karaktera za prve web strane u prvim godinama razvoja Interneta bio je ASCII kod koji podržava brojeve od 0 do 9, mala i velika slova engleskog alfabeta i neke dodatne karaktere. S obzirom da mnoge zemlje koriste karaktere koji se ne nalaze unutar ASCII koda, osnovni skup karaktera je proširen u UNICODE koji sadrži sve neophodne karaktere. On se pojavljuje u dve verzije UTF-8 i UTF-16.

Preporučljivo je da se unutar `meta` taga (koji se nalazi unutar `head` taga) na vrhu HTML strane specificira koji skup karaktera se koristi, na sledeći način:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

Takođe, je potrebno prilikom snimanja `.html` dokumenta izabrati odgovarajuću enkripciju. Za *notepad++* u padajućem meniju `Encoding` je neophodno odabrati opciju `Encode in UTF-8`.

**PRVA WEB STRANA**

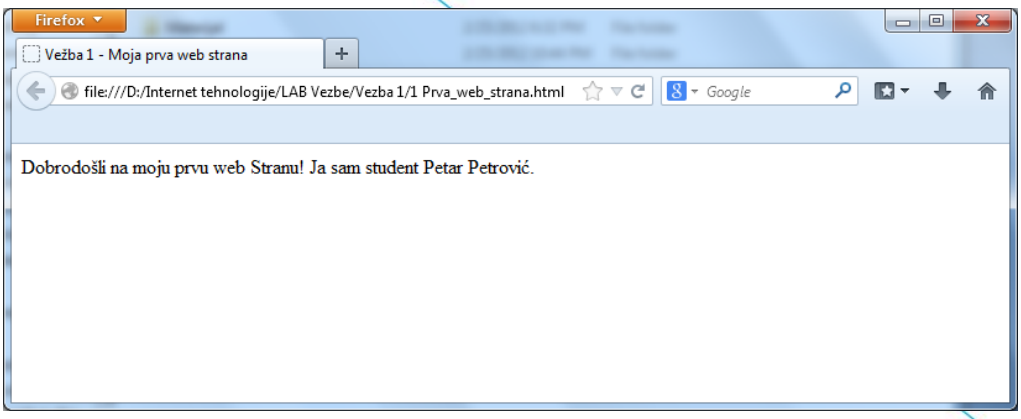
Da biste kreirali vašu prvu web stranu otvorite tekst editor (*notepad++*) i upišite sledeći sadržaj:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
   <title>Vežba 1 - Moja prva web strana</title>
   <meta http-equiv="Content-Type" content="text/html;
   charset=utf-8" />
</head>
<body>
      Dobrodošli na moju prvu web Stranu!
      Ja sam student Petar Petrović.
</body>
</html>
```

Na ekranu bi trebalo da dobijete sledeći prikaz:




**HTML OZNAKE ZA MARKIRANJE TEKSTA**

Internet se može posmatrati kao skup web strana koje se uglavnom sastoje od teksta. S obzirom na to da najveći deo web stranice čini tekst, osnovni zadatak HTML-a je označavanje teksta na pogodan način.

Kada se unosi neformatiran tekst u HTML dokument, nije bitno kako taj tekst unutar dokumenta izgleda, već kako će izgledati kada se pokrene u web čitaču. Razlog se nalazi u činjenici da čitač ne prepoznaje znakove za novi red (CR ili Enter) kao ispravne znake formatiranja. Svi čitači Weba



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

ignorišu znakove za novi red, odnosno za kraj reda (CR i LF) koje programi za obradu teksta umeću u HTML kod. Zato je za prelazak u novi red neophodno koristiti oznaku za prelom reda `<br/>`.

Za razliku od uparenih oznaka prazne oznake ne zahtevaju završnu signaturu. Prazne oznake se mogu porediti sa naredbama koje se odmah izvršavaju. Dobar primer za praznu oznaku je prelom reda `<br/>`, koja nalaže čitaču da pređe u naredni red, pa tek onda da prikaže ostatak stranice.

Bez obzira koliko blanko znaka postavite unutar HTML koda, unutar web čitača će se on videti kao jedan blanko znak. Zato, ukoliko želite više od jednog blanko znaka (*space* na tastaturi) morate da koristite znak za nelomivi razmak ` ` na mestu gde želite da čitač prikaže veći razmak.

Najopštije grupisanje teksta vrši se u pasusima. Pomoću HTML-a, pasus može da se koristi za logično grupisanje teksta, kao pri štampanju. Element za pasus zahteva početnu i završnu oznaku:

```
<p>...</p>
```

HTML pasusi nemaju uvučeni prvi red i podrazumevano su poravnati ulevo.

HTML oznaka `blockquote` formatira tekst uvlačenjem ili povećavanjem margina na levoj i desnoj strani teksta koji je blok čitača.

```
<blockquote>...</ blockquote>
```

Formatiranje teksta može se obaviti unošenjem početne oznake formata na sam početak teksta. Najčešće korišćene oznake formata su sledeće:

Za prikazivanje teksta **polucrnim** fontom.

```
<b>...</b>
```

Za prikazivanje teksta *kurzivnim* fontom.

```
<i>...</i>
```

Za prikazivanje teksta koji je <u>podvučen</u>.

```
<u>...</u>
```

HTML obezbeđuje šest različitih stilova za označavanje naslova i podnaslova. Da bi zadali veličinu fonta potrebno je oko teksta postaviti početne i završne uparene oznake nivoa naslova (od `<h1> ...</h1>` do `<h6> .. </h6>`). Broj iza slova "h" naznačava relativnu veličinu teksta naslova. Što je broj manji, to je naslav na ekranu veći.

Za kreiranje naslova prvog i drugog nivoa treba napisati sledeće.

```
<h1>Naslov poglavlja prvog nivoa</h1>

<h2>Naslov poglavlja drugog nivoa</h2>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Pre nego što je CSS (eng. *Cascading Style Sheet*) postao alat za formatiranje, o čemu će biti više reči kasnije, HTML je obezbeđivao osnovne načine formatiranja teksta i podešavanja izgleda na ekranu. Neki od tih načina su opstali i u verziji HTML5, dok su drugi zastareli (moderni pretraživači ih ne podržavaju). Neke od tih oznaka su sledeće:

`<cite> </cite>` Oznake citata ističu tekst, obično kurzivnim slovima.

`<code> </code>` Ove oznake prebacuju tekst koji obuhvataju u neproporcionalni (tzv. mono) font, kao što je `Courier`, da bi istakle tekst. Koriste se da bi označili HTML kod u dokumentu sa HTML primerom.

`<del> </del>` Oznake obrisanog teksta obeležavaju tekst kao da je obrisan, tako što ga čitač ispiše precrtanim slovima.

`<q> </q>` Oznake navoda okružuju tekst znacima navoda.

`<sub> </sub>` Ovo su oznake indeksa (prikazuju tekst niže od okolnog teksta).

`<sup> </sup>` Ovo su oznake eksponenta (prikazuju tekst iznad okolnog teksta)

`<em></em>` Ove oznake ističu tekst ispisujući okružene reči kurzivnim slovima

`<strong></strong>` Ove oznake ističu tekst ispisujući ga polucrnim kurzivnim slovima

Za promenu vrste ispisa karaktera (fonta) koristi se oznaka font i njen atribut `face` kao što je prikazano u nastavku:

```
<font face="Courier New" color="blue" size="+3">… </font>
```

Vrednost atributa `color` se može zadavati numerički ili simbolički. Podrazumevaju se simbolička imena boja na engleskom jeziku: `red`, `green`, `blue` i sl. Često se koristite i heksadecimalni brojevi za predstavljanje boje. Zapis u ovom obliku crvene boje je `#FF0000`. Drugi način numeričkog zadavanja boja je preko **`rgb(`**`red, green, blue)` koda boje oblika `rgb(255,123,30)`. Svaki broj može imati vrednost od 0 do 255.

Primer kako se ista boja može zadati na tri navedena načina pokazan je sledećom tabelom.

| Naziv boje | Paleta boja | RGB | HEX |
|---|---|---|---|
| Red | | 255 0 0 | #FF0000 |
| BlueViolet | | 138 43 226 | #8A2BE2 |
| Brown | | 165 42 42 | #A52A2A |



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

| | | | |
|---|---|---|---|
| Brown1 | | 255 64 64 | #FF4040 |
| Chartreuse | | 127 255 0 | #7FFF00 |
| Chartreuse1 | | 118 238 0 | #76EE00 |
| Chartreuse2 | | 102 205 0 | #66CD00 |
| Chartreuse3 | | 69 139 0 | #458B00 |
| Chocolate | | 210 105 30 | #D2691E |

Tabela 1. Tri različita načina definisanja boje

Atribut `size` služi za promenu veličine slova i može se kretati u opsegu od -7 (najmanji) do +7 (najveći). Ova veličina, zapravo predstavlja dimenziju fonta u odnosu na veličinu osnovnog fonta.

**SPECIJALNI ZNACI**

Da bi se specijalni znaci prikazali u prozoru čitača, u HTML kod se mora uneti numerička vrednost koja predstavlja taj simbol. Numerička vrednost zapravo predstavlja ASCII kod (dodatak B) željenog simbola. Čitaču Weba ćete reći da prikaže simbol tako što ćete ispred njegove numeričke vrednosti staviti znakove & i #, a posle koda tačku i zarez (;). Znači, da biste rekli čitaču da znak © prikaže na stranici, treba da unesete `©` u HTML kod Web stranice. Neki simboli imaju tzv. tekstualni zapis. Tekstualni zapis simbola (©) je `©`

Za detaljnije informacije o skraćenicama za prikazivanje specijalnih znaka i ostalih ASCII karaktera pogledati **Dodatak B**, na kraju praktikuma.

**HORIZONTALNA LINIJA**

Postavljanje horizontalne linije na Web stranicu može se obaviti primenom oznake `<hr>` na sledeći način.

```
<hr align="left" size="4" width="30" noshade>
```

Raspoloživi atributi za definisanje izgleda horizontalne linije su:

`align` određuje čitaču da poravna horizontalnu liniju uz jednu od ivica ili da je postavi na sredinu prozora aplikacije čitača.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

`size` određuje debljinu horizontalne linije u pikselima.

`width` atribut definiše dužinu linije. Vrednost ovog atributa se može zadavati u pikselima (npr. `width="300px"` ili samo `width="300"`) ili u procentima širine prozora čitača. Ako zadate vrednost 30% (npr. `width="30%"`), čitač će nacrtati horizontalnu liniju dužine približno 1/3 širine prozora čitača.

`noshade` ovaj atribut nalaže čitaču da prikaže liniju bez senke.

**HTML HIPERVEZE**

Osnovna prednost koju imaju Web stranice nad papirnim dokumentima je mogućnost hiperaktivne veze s drugim Web stranicama. Listanje stranica knjige zamenjuje se pritiskom posetioca na dugme hiperveze, čime se prelazi sa jedne Web stranice na drugu. Pri tome, stranice nisu ograničene na istu Web lokaciju.

Hipertekstualna veza je reč ili grupa reči koje posetilac treba da aktivira klikom kako bi čitač preuzeo označenu Web stranicu sa određene URL adrese nekog od Web servera.

URL adrese, predstavljaju duge redove karaktera koje vidite u traci sa adresom pretraživača svaki put kada posetite određeni izvor ili stranu. Čini ga nekoliko elemenata, kao što su protokol za prenošenje podataka, adresa servera mreže sa imenom domena, tačna lokacija na sereveru kao i određeno mesto samog izvora.

Da biste napravili hipertekstualnu vezu, stavite početnu (`<a>`) i završnu (`</a>`) oznaku za sidro odnosno mesto hiperveze oko teksta koji želite da posetilac pritisne kako bi aktivirao vezu. Dodajte atribut `href` (engl. skr. *hyperlink reference*) oznaci `<a>` kako bi označili putanju do datoteke koju čitač treba da učita prilikom aktiviranja hipertekstualne veze. Čitač Weba će podvući hipertekstualnu vezu na Web stranici.

```
<a href="http://google.rs" target="_blank"> Prelazak na google stranu.</a>
```

Pomoću ovog označavanja ćete tekst „Prelazak na google stranu" pretvoriti u hipervezu koja vodi na početnu stranu pretraživača google.

Atribut `target` ima četiri važeća podešavanja, pri čemu ispred svakog treba napisati donju crtu, a zatim sledi odredišno ime. Važeća podešavanja su:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

| | |
|---|---|
| `_blank` | otvara izvor u novom prozoru ili novoj kartici, ako se radi o interfejsu sa karticama i ako je pravilno konfigurisan, |
| `_parent` | otvara izvor u matičnom kontekstu za pretraživanje (prozor ili kartica) osim ako ih nema, kada otvara izvor u aktivnom kontekstu, |
| `_top` | otvara izvor u najvišem kontekstu za pretraživanje ili u aktivnom kontekstu, ukoliko nema najvišeg konteksta, |
| `_self` | je za sopstvene potrebe. |

Osim za prelaženje s dokumenta na dokument ili sa lokacije na lokaciju, hipertekstualna veza se može koristiti za preuzimanje datoteka sa Web lokacije. Sledeći kod nalaže čitaču da preuzme datoteku sa slikom corolla.zip kada posetilac pritisne jednu od reči "Ovde kliknite" unutar rečenice „Ovde kliknite da preuzmete sliku moje nove toyote“:

```
<a href="http://www.toyota.com/cars/corolla.zip">Ovde kliknite </a>

da preuzmete sliku moje nove "toyote".
```

Prethodni način zadavanja URL adrese predstavlja apsolutno adresiranje, gde je data celokupna adresa. Ovaj način se koristi u slučajevima kada želite da postavite vezu ka nekoj strani koju ste pronašli na Webu. Ovakva veza se još naziva i spoljna jer se odnosi na Web stranu koja se nalazi izvan vašeg servera, odnosno Web lokacije.

Osim apsolutnih koriste se i relativne adrese. Relativne adrese se koriste kada postavljate veze između strana na istoj web lokaciji. Kada se ne vidi „`http://`“, Web čitač traži dokument na istom serveru. Relativno ime putanje opisuje kako se stiže od lokacije tekućeg dokumenta (na kojoj se trenutno nalazite) do dokumenta koji treba povezati. Primeri relativnih putanja će biti pokazani na strukturi Web lokacije „kuhinjica“, čiji je dijagram strukture direktorijuma prikazan na sledećoj slici.

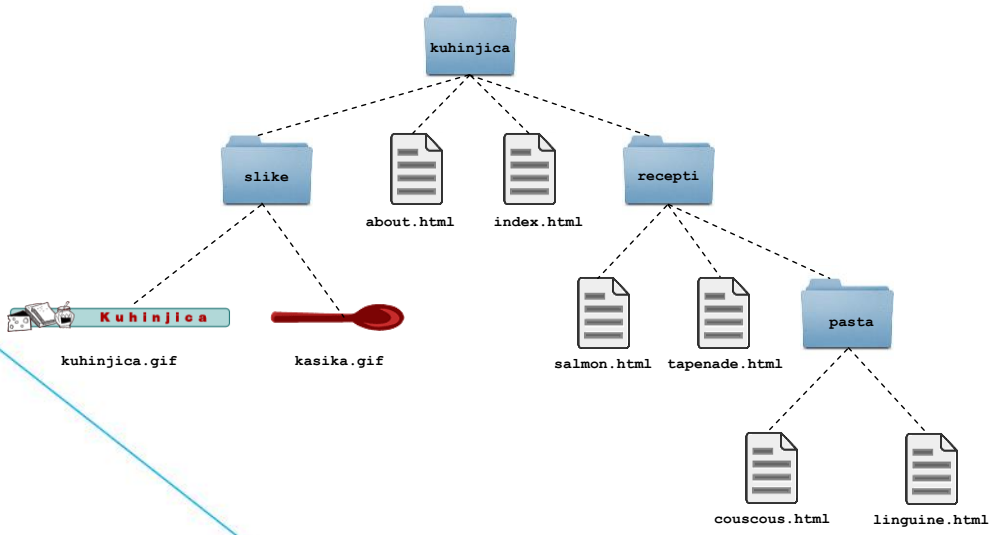
Najlakše se pišu ***relativne adrese unutar direktorijuma***. Kada se povezuju datoteke unutar istog direktorijuma dovoljno je da se samo napiše ime datoteke. Ako je URL samo ime datoteke, server ga traži u tekućem direktorijumu – onom u kojem je dokument sa vezom – za tu datoteku.

U narednom primeru je postavljena veza između početne strane (`index.html`) ka strani sa opštim informacijama (`about.html`). Obe datoteke su u istom direktorijumu (`kuhinjica`). Dakle, dovoljno je da napišete sledeće:

```
<a href="about.html">O ovoj lokaciji…</a>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Slika 1. Dijagram strukture direktorijuma, za sajt „kuhinjica"*

Često je potrebno da se postavi ***veza ka direktorijumu nižeg nivoa***. U prethodnom primeru datoteke sa receptima su u poddirektorijumu recepti. Da bi postavili vezu od datoteke `index.html` ka datoteci u direktorijumu recepti koja se zove `salmon.html` treba napisati URL adresu koja govori Web čitaču da u tekućem direktorijumu potraži direktorijum recepti i u njemu datoteku `salmon.html` na sledeći način:

```
<a href="recepti/salmon.html">Garlic Salmon</a>
```

Da bi postavili vezu ka datoteci `couscous.html`, koja se nalazi u poddirektorijumu `pasta`, treba napisati putanju kroz dva poddirektorijuma (`recepti`, zatim `pasta`) ka datoteci `couscous.html`.

```
<a href="recepti/pasta/couscous.html">Couscous recept</a>
```

Osim prethodnih primera moguće je i postavljanje ***veza ka direktorijumu višeg nivoa***. Sledeći primer pokazuje kako se postavlja veza ka početnoj strani (`index.html`) od strane `salmon.html`. Kako je `salmon.html` u poddirektorijumu `recepti`, moramo da se vratimo jedan nivo nagore, kako bismo našli `index.html`.

```
<a href="../index.html">Povratak na glavnu stranu</a>
```

Prvi deo putanje (`../`) govori Web čitaču da „ide jedan nivo iznad", slično pritiskanju dugmeta Up u Windows Exploreru, i da u tom direktorijumu potraži datoteku `index.html`.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Da bi se vratili dva nivoa gore, dvaput treba upotrebiti tačka-tačka-kosa crta (`../../`). Veza od strane `couscous.html` ka početnoj strani `index.html` treba da izgleda ovako:

```
<a href="../../index.html">Povratak na glavnu stranu</a>
```

Sva imena putanja koje su pokazana u prethodnim primerima predstavljaju ***relativne putanje u odnosu na dokument sa hipervezom***. Drugi način da napišemo ime putanje jeste da počnemo od osnovnog direktorijuma i navedemo imena svih poddirektorijuma dok ne dođemo do datoteke sa kojom želimo da se povežemo. Takvo ime putanje se naziva ***relativno u odnosu na koren lokacije***.

**Povezivanje sa određenim mestom na istoj web strani** (deo iste web strane) je proces koji se sastoji iz dva dela. Prvo treba odrediti krajnje odredište, a zatim postaviti vezu ka njemu.

Da bi kreirali odredište, potrebno je da se, pomoću atributa `id`, zada jedinstveno ime ciljnom elementu u dokumentu:

```
<h1 id="cilj">Ovo je prvi red odredišnog dela strane</a>
```

Nakon postavljanja identifikatora, treba postaviti vezu ka njemu. Kao i bilo koja druga veza, lokacija ove veze se kreira pomoću elementa `<a>` sa atributom `href`, s tim što ispred oznake imena odredišta treba postaviti znak `#`.

```
<a href="#cilj">Kliknite da biste prešli na odredište</a>
```

**Link za slanje pošte** ima većina web lokacija. Ona je najčešće kao veza „Contact us" koja otvara podrazumevani program za elektronsku poštu sa zaglavljem koje je spremno za unos poruke koju želite da pošaljete primaocu. Ovo se takođe kreira pomoću hiperveza. U delu za odredište `href`, umesto protokola `http` kao do sada piše se `mailto:` i adresa elektronske pošte i hiperveza će otvoriti poruku e-pošte koja će biti poslata na predviđenu e-adresu.

```
<a href="mailto:info@vtsnis.edu.rs">Kontaktirajte nas</a>
```

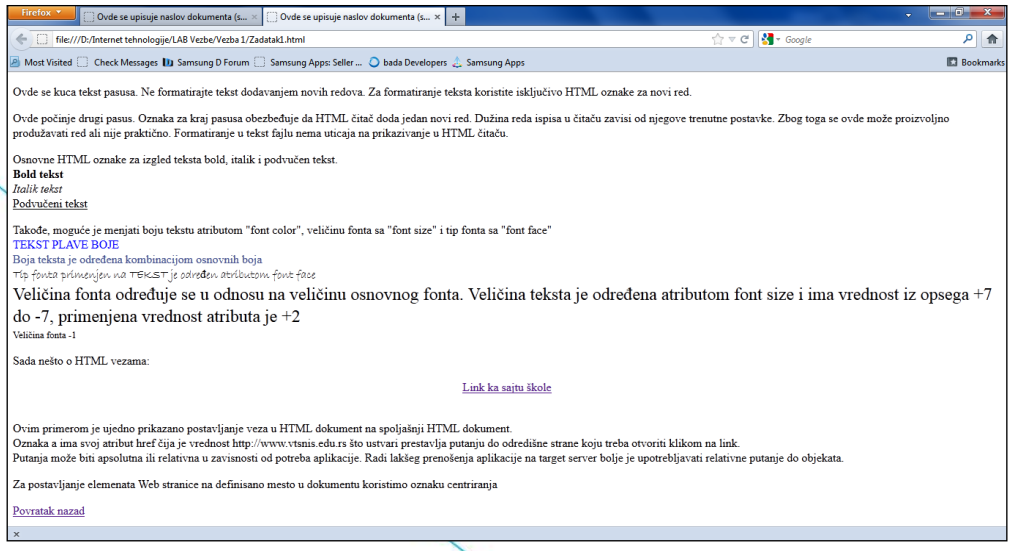


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

## ZADACI ZA VEŽBU

### *Zadatak 1:*

Formirati Web stranicu sa osnovnim web atributima kao na sledećoj slici.




*Web stranica sa osnovnim HTML oznakama i atributima*

Sledeći kod smestiti u poseban fajl sa ekstenzijom .htm ili .html.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>Ovde se upisuje naslov dokumenta (služi prilikom
  pretraživanja HTML stranica pretrazivačima) - ne štampa
  se!</title>
  <meta http-equiv="Content-Type" content="text/html;
  charset=utf-8" />
</head>
<body>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<p>Ovde se kuca tekst pasusa. Ne formatirajte tekst dodavanjem novih redova.Za formatiranje teksta koristite isključivo HTML oznake za novi red.</p>

<p>
<h1>Ovde počinje drugi pasus.</h1>
Oznaka za kraj pasusa obezbeđuje da HTML čitač doda
jedan novi red.
Dužina reda ispisa u čitaču zavisi od njegove trenutne
postavke. Zbog toga
se ovde može proizvoljno produžavati red ali nije
praktično.
Formatiranje u tekst fajlu nema uticaja na prikazivanje
u HTML čitaču.
</p>

<p>Osnovne HTML oznake za izgled teksta bold, italik i
podvučen tekst.
<br/>
<b>Bold tekst</b><br/>
<i>Italik tekst</i><br/>
<u>Podvučeni tekst</u><br/>
</p>

<p>
Takođe, moguće je menjati boju tekstu atributom "font
color", veličinu fonta sa "font size" i tip fonta sa
"font face"
<br/>
<font  color="blue"> TEKST PLAVE BOJE</font>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<br/>
<font color="#445595"> Boja teksta je određena kombinacijom osnovnih boja</font>
<br/>
<font face="Bradley Hand ITC"> Tip fonta primenjen na
TEKST je određen atributom font face</font>
<br/>
<font size="+2"> Veličina fonta određuje se u odnosu na
veličinu osnovnog fonta. Veličina teksta je određena
atributom font size i ima vrednost iz
opsega +7 do -7, primenjena vrednost atributa
je +2</font>
<br/>
<font size="-1"> Veličina fonta -1</font>
<br/>
</p>

<p>Sada nešto o HTML vezama:<br/>
<center>
<a href="http://www.vtsnis.edu.rs">
   Link ka sajtu škole
</a>
</center>
<br/>
</p>
<p>
Ovim primerom je ujedno prikazano postavljanje veza u
HTML dokument na spoljašnji HTML dokument.
<br/>
Oznaka a ima svoj atribut href čija je vrednost
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

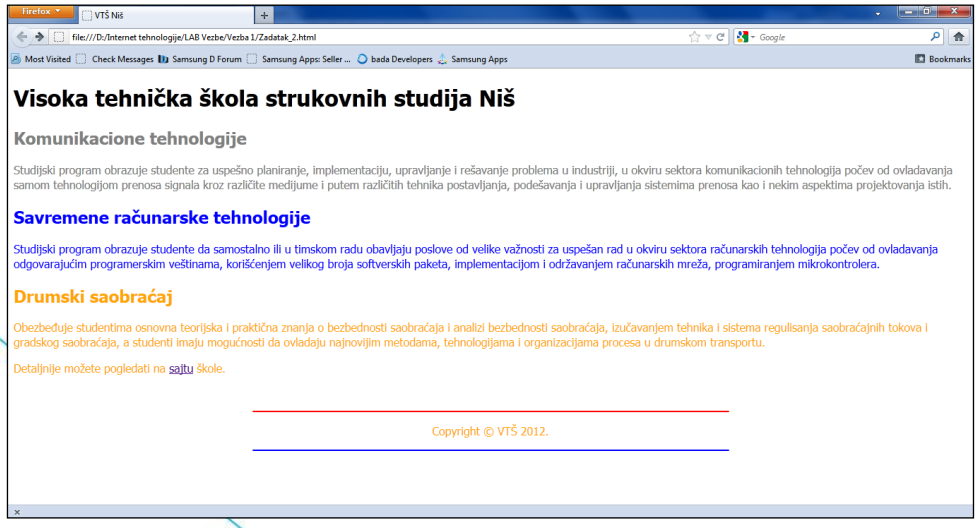
```
        http://www.vtsnis.edu.rs što u stvari prestavlja putanju
        do odredišne strane koju treba otvoriti klikom na  link.
        <br/>
        Putanja može biti apsolutna ili relativna u zavisnosti
        od potreba aplikacije. Radi lakšeg prenošenja aplikacije
        na target server bolje je upotrebljavati relativne
        putanje do objekata.
        </p>

        <p>Za postavljanje elemenata Web stranice na definisano
        mesto u dokumentu koristimo oznaku centriranja
        <center> </center>
        </p>

        <a href="Test_Proba.html">Povratak nazad</a>

</body>
</html>
```

***Zadatak 2:***

Formirajte Web stranicu koja sadrži osnovne podatke o Visokoj tehničkoj školi strukovnih studija u Nišu sa sadržajem o tri studijska programa kao na sledećoj slici. Uvežbati uparene i prazne oznake vezane za formatiranja teksta. Uočiti kako vrednosti atributa utiču na izgled i formatiranje stranice. Kreirati specijalni znak © korišćenjem oznake `©`



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




Sledeći kod smestiti u poseban fajl sa ekstenzijom .htm ili .html.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
      <title>VTŠ Niš</title>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
      <p><font face="Tahoma" color="#000000">
      <h1>Visoka tehnička škola strukovnih studija Niš</h1>
      </font>
      </p>

      <p>
      <font face="Tahoma" color="gray">
      <h2>Komunikacione tehnologije</h2>
      Studijski program obrazuje studente za uspešno
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
planiranje, implementaciju, upravljanje i rešavanje
problema u industriji, u okviru sektora komunikacionih
tehnologija počev od ovladavanja samom
tehnologijom prenosa signala kroz različite medijume i
putem različitih tehnika postavljanja, podešavanja i
upravljanja sistemima prenosa kao i nekim aspektima
projektovanja istih.
</font>
</p>

<p>
<font face="Tahoma" color="blue">
<h2>Savremene računarske tehnologije</h2>
Studijski program obrazuje studente da samostalno ili u
timskom radu obavljaju poslove od velike važnosti za
uspešan rad u okviru sektora računarskih tehnologija
počev od ovladavanja odgovarajućim programerskim
veštinama, korišćenjem velikog broja softverskih paketa,
implementacijom i održavanjem računarskih mreža,
programiranjem mikrokontrolera.
</font>
</p>

<p>
<font face="Tahoma" color="rgb(255,255,110)">
<h2>Drumski saobraćaj</h2>
Obezbeđuje studentima osnovna teorijska i praktična
znanja o bezbednosti saobraćaja i analizi bezbednosti
saobraćaja, izučavanjem tehnika i sistema regulisanja
```

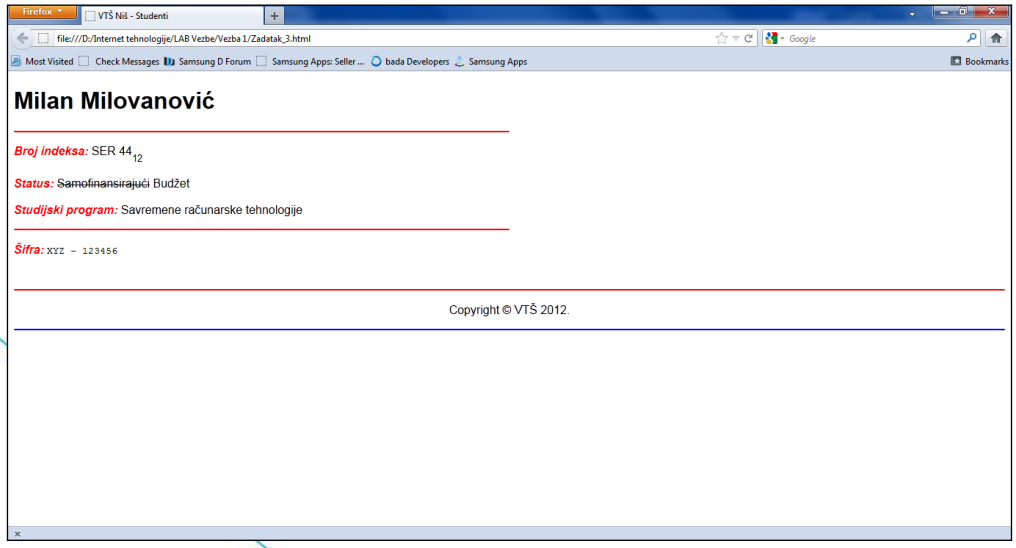
```
        saobraćajnih tokova i gradskog saobraćaja, a studenti
        imaju mogućnosti da ovladaju najnovijim metodama,
        tehnologijama i organizacijama procesa u drumskom
        transportu.
        </font>
        </p>

        Detaljnije možete pogledati na
        <a href="http://www.vtsnis.edu.rs" target="_blank">
        sajtu </a>škole.
        <br/>
        <br/>
        <br/>
        <hr width="50%" align="center" color="red"/>
        <p align="center">Copyright &copy; VTŠ 2012.</p>
        <hr width="50%" align="center" color="blue"/>
</body>
</html>
```

***Zadatak 3:***

Formirajte Web stranicu koja sadrži osnovne podatke o studentu, kao što je prikazano na sledećoj slici. Koristiti oznake za specijalne znake kao i formatiranje teksta (precrtan tekst, ) kao što je to opisani u teorijskim uvodu vežbe.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




Sledeći kod smestiti u poseban fajl sa ekstenzijom .htm ili .html.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
      <title>VTŠ Niš - Studenti</title>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
      <p><font face="Arial" color="#000000">
      <h1>Milan Milovanović</h1>
      </font>
      </p>
      <hr width="50%" align="left" color="red"/>

      <p><font face="Arial" color="red">
      <b><i>Broj indeksa:</i></b>
      <font color="black"> SER 44<sub>12</sub></font>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
</font>
</p>

<p><font face="Arial" color="red">
<b><i>Status:</i></b>
<font color="black">
<del> Samofinansirajući</del> Budžet</font>
</font>
</p>
<p>
<font face="Arial" color="red">
    <b><i>Studijski program:</i></b>
       <font color="black">
           Savremene računarske tehnologije
       </font>
</font>
</p>

<hr width="50%" align="left" color="red"/>

<p><font face="Arial" color="red">
<b><i>Šifra:</i></b>
<font color="black"><code> XYZ - 123456</code></font>
</font>
</p>
<br/>
<hr width="100%" align="center" color="red"/>
<p align="center">Copyright &copy; VTŠ 2012.</p>
<hr width="100%" align="center" color="blue"/>
```

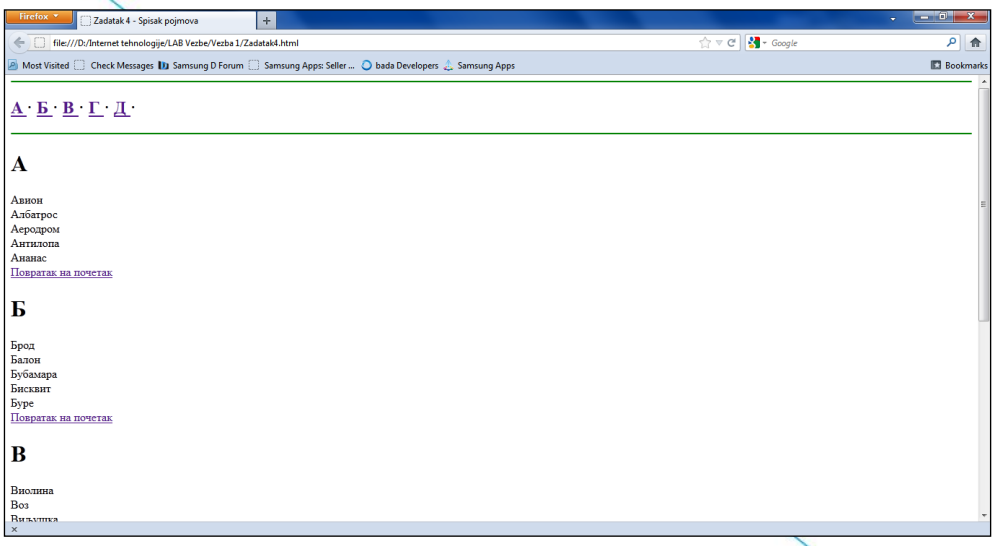


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
</body>

</html>
```

***Zadatak 4:***

Formirajte Web stranicu koja na vrhu sadrži veze ka odgovarajućem delu strane na kome se nalaze reči sa odabranim početnim slovom. Na kraju skupa reči koje počinju određenim slovom postaviti link koji vraća korisnika na početak strane. Primer izgleda strane i html koda je dat u nastavku.




Sledeći kod smestiti u poseban fajl sa ekstenzijom .htm ili .html.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

      <title>Zadatak 4 - Spisak pojmova</title>

      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

</head>


<body>

      <div id="start">

      <hr color="green">
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<h2>
      <a href="#startA"> А </a>&#183;
      <a href="#startB"> Б </a>&#183;
      <a href="#startV"> В </a>&#183;
      <a href="#startG"> Г </a>&#183;
      <a href="#startD"> Д </a>&#183;
</h2>
<hr color="green">
</div>

<div id="startA">
 <h1>А</h1>
 Авион <br/>
 Албатрос <br/>
 Аеродром <br/>
 Антилопа <br/>
 Ананас <br/>
 <a href="#start"> Повратак на почетак </a>
</div>

<div id="startB">
 <h1>Б</h1>
 Брод <br/>
 Балон <br/>
 Бубамара <br/>
 Бисквит <br/>
 Буре <br/>
 <a href="#start"> Повратак на почетак </a>
 </div>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<div id="startV">
 <h1>В</h1>
 Виолина <br/>
 Воз <br/>
 Виљушка <br/>
 Ветрењача <br/>
 Вагон <br/>
 <a href="#start"> Повратак на почетак </a>
</div>

<div id="startG">
 <h1>Г</h1>
 Галеб <br/>
 Гитара <br/>
 Голуб <br/>
 Гавран <br/>
 Гусеница <br/>
 <a href="#start"> Повратак на почетак </a>
</div>

<div id="startD">
 <h1>Д</h1>
 Дрво <br/>
 Детлић <br/>
 Дим <br/>
 Дабар <br/>
 Детелина <br/>
 <a href="#start"> Повратак на почетак </a>
```

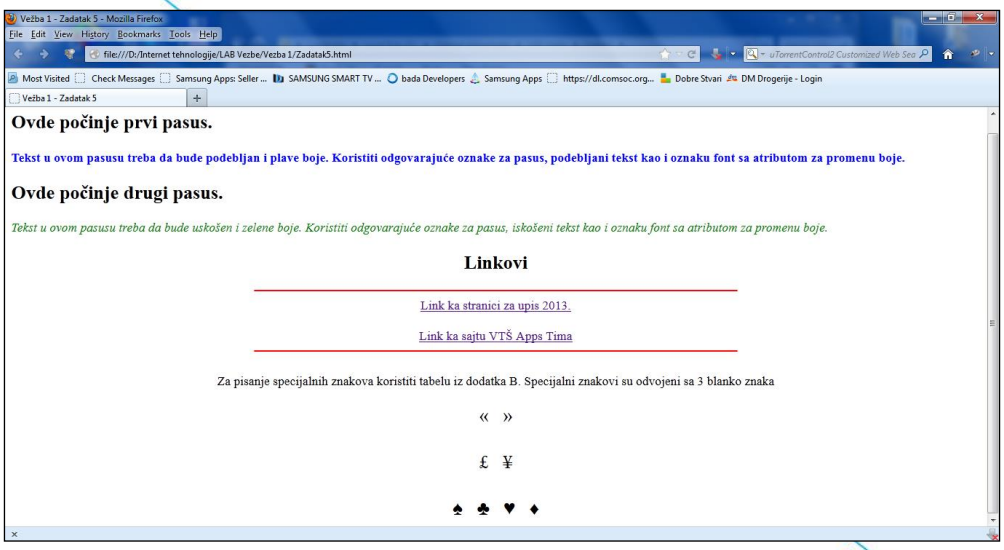


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
        </div>

</body>

</html>
```

***Zadatak 5:***

Formirajte Web stranicu koja na vrhu sadrži naslov drugog nivoa i pasus ispisan plavim podebljanim slovima. Nakon toga kreirati ponovo naslov drugog nivoa i pasus ispisan zelenim iskošenim slovima. Nakon linkova i crvenih horizontalnih linija širine 50% stranice, ispisati specijalne znake razdvojene sa tri blanko znaka, kao na slici. Specijalni znaci su dva puta veći od običnog teksta.




U nastavku napisati kod koji treba smestiti u poseban fajl sa ekstenzijom .htm ili .html.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

## PITANJA I ZADACI

1. Napišite minimalnu strukturu jednog HTML dokumenta.

2. Da li su sledeća imena datoteka prihvatljiva za ime Web dokumenta? Ako nisu objasnite zbog čega.

a. `Suncokret.html` DA NE _____

b. `index.doc` DA NE _____

c. `Kuvanje na pari.html` DA NE _____

d. `Tekstovi_pesama.html` DA NE _____

e. `igre/rubix.html` DA NE _____

f. `%bilokakavtekst.html` DA NE _____

3. Svi ovde navedeni primeri označavanja su pogrešni. Objasnite zbog čega i zatim ih napišite ispravno.

a. `<i>Čestitamo!<i>`



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

b.

```
<a href="datoteka.html">
    povezan tekst
</a href="datoteka.html">
```

c. `<p>Ovo je novi pasus<\p>`

4. Kako biste označili ovaj komentar u HTML dokumentu tako da ne bude prikazan u prozoru Web čitača?

```
Spisak proizvoda počinje ovde
```

5. Napišite HTML kod za kreiranje horizontalne linije crvene boje, širine 50% prozora web čitača i debljine linije 5px.

6. Koji element se koristi u HTML kodu kao zamena za blanko znak?



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

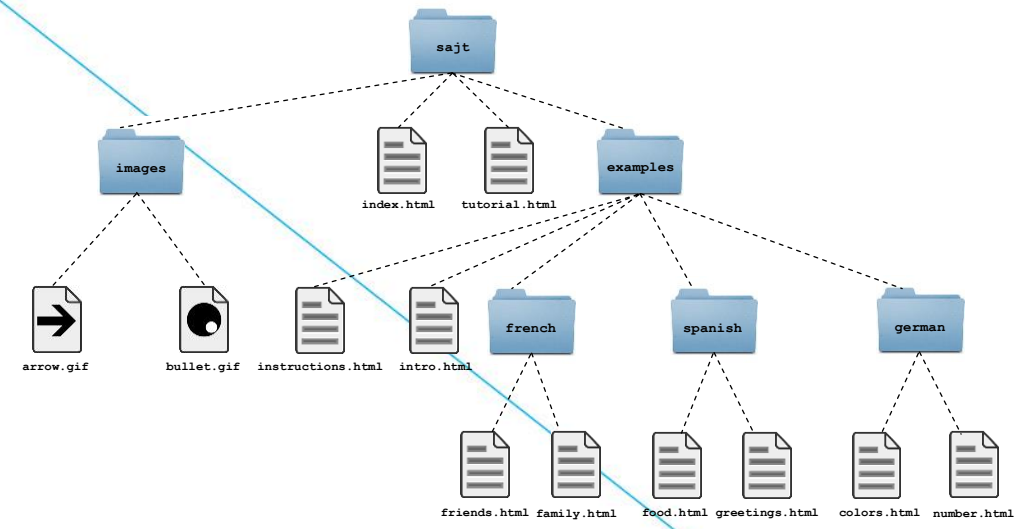
7. Navedite znakove koji se dobijaju pomoću ovih znakovnih entiteta:

— _____ & _____

  _____ © _____

· _____ ™ _____

8. Na osnovu dijagrama, na sledećoj slici, koji predstavlja strukturu direktorijuma sajta, napišite HTML kod za sledeće veze:




a) Napišite hipertekstualnu vezu koja vodi sa strane `tutorial.html` na stranu `family.html`.

b) Napišite hipertekstualnu vezu koja vodi sa strane `family.html` ka strani `numbers.html`, ali počnite od osnovnog direktorijuma.

c) Napišite hipertekstualnu vezu koja vodi sa strane `instructions.html` natrag do početne strane `index.html`.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

d) Napišite hipertekstualnu vezu koja vodi sa strane `intro.html`, ka Web lokaciji Visoke tehničke škole u Nišu (www.vtsnis.edu.rs)

e) Napišite hipertekstualnu vezu koja vodi ka strani `instructions.html` sa strane `greetings.html`.

f) Napišite hipertekstualnu vezu za vraćanje na početnu stranu `index.html` sa strane `colors.html`.

g) U datoteci `index.html`, napravite vezu ka stranici za predmet Internet tehnologije koja se nalazi na Web lokaciji Visoke tehničke škole u Nišu (www.vtsnis.edu.rs)

# Vežba br.2 – SLIKE, LISTE I FREJMOVI

## CILJ VEŽBE

- Upoznati se sa načinima dodavanja slika HTML strani
- Podešavanje atributa slike
- Kreiranje slike kao hiperaktivne veze
- Savladati kreiranje uređenih i neuređenih lista
- Kreiranje stranica sa frejmovi i podešavanje dimenzija frejmova

## TEORIJSKE OSNOVE ZA IZRADU VEŽBE

### DODAVANJE SLIKA

Web strana bez slika nije mnogo atraktivna. Slike se na Webu pojavljuju na dva načina: kao deo pozadine ili kao deo umetnutog sadržaja. Pozadinske slike se dodaju pomoću CSS-a o čemu će biti više reči u vežbi br. 5. Uneta slika može da se koristi na web strani na više načina: kao jednostavna slika, hiperveza ili mapirana slika.

Da bi web čitač mogao da ih prepozna, slike moraju da budu u odgovarajućem formatu, odnosno grafičke datoteke moraju imati imena sa odgovarajućom ekstenzijom tipa .gif, .jpg (ili .jpeg) i .png.

Upotrebom prazne oznake `<img>`, na Web stranicu se može postaviti slika. Oznaka `<img>` treba da sadrži i atribut `src`, koji čitaču predočava putanju do datoteke sa traženom slikom. Važe sve napomene koje su navedene za apsolutno i relativno adresiranje u prethodnoj vežbi (u delu za hiperaktivne veze).

```
<img src="kuca.jpg" width="300" height="300" alt="Slika kuće" border="1"/>
```

Kada prepozna element `img` web čitač traži od servera da mu pošalje grafičku datoteku sa specificirane adrese koju treba da prikaže na strani. S obzirom na brzinu današnjih mreža to bi trebalo da se dogodi u trenutku. Donja ivica slike se automatski postavlja u istu ravan kao osnovna ivica teksta. To je prazan element što znači da nema nikakav sadržaj. U XHTML-u (strožija pravila za pisanje HTML-a) prazni elementi moraju imati završnu oznaku, što znači da se oznaka `img` piše na kraju sa kosom crtom (`/`).



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Atribut `alt` sadrži alternativni tekst koji se prikazuje u slučaju da slika nije dostupna (pogrešno navedena putanja do slike, pogrešna ekstenzija i sl.).

Atributi `height` i `width` definišu dimenzije slike koja se prikazuje u prozoru čitača. Definisanjem ovih atributa u HTML kodu čitač može da prikaže tekst ostatka Web stranice dok čeka da se slike učitaju sa mreže. Ovi atributi praktično rezervišu mesto za slike, tako da se može prikazati tekst koji ih okružuje i pre njihovog učitavanja. Treba voditi računa da ove dimenzije mogu da se razlikuju od dimenzija originalne grafičke datoteke, ali radi optimizacije je najbolje da one budu jednake. Naime, nepotrebno je kroz mrežu prenositi informacije o slici čije su originalne dimenzije velike, a na strani atributima `height` i `width` prikazati umanjenu sliku.

Atribut `border` zadaje debljinu okvira oko slike. Vrednost atributa se izražava u pikselima. Uobičajeno je da ako sliku koristite kao hipervezu čitač oko nje nacrta okvir. Docrtavanjem okvira oko slike obaveštavamo posetioca da je slika u stvari hiperveza. Ako to nije slučaj atribut `border` postavite na vrednost nula (`border="0"`).

**LISTE PODATAKA**

Liste se koriste jer pregledno prikazuju informacije. HTML omogućava prikazivanje tri glavne vrste lista:

- **Uređene liste** – za označavanje elemenata liste se koriste slova abecede, rimski ili arapski brojevi,
- **Neuređene liste** – za označavanje elemenata liste se koriste simboli (kvadrat, krug ili disk),
- **Liste definicija** – koje se sastoje od izraza i definicija.

**Uređena lista** se kreira uparenom oznakom `<ol>...</ol>`. Za dodeljivanje elementa listi, treba umetnuti tekst svakog elementa između početne i završne oznake liste `<li>...</li>`. Čitač će prikazati arapske brojeve (1,2,3...) ispred svakog elementa liste. Za korišćenje slova ili rimskih brojeva, umesto arapskih brojeva, treba dopisati atribut `type` u oznaku uređene liste:

| | |
|---|---|
| `<ol type="1">` | Označava elemente liste arapskim brojevima. |
| `<ol type="a">` | Označava elemente liste malim latiničnim slovima. |
| `<ol type="A">` | Označava elemente liste velikim latiničnim slovima. |
| `<ol type="i">` | Označava elemente liste rimskim brojevima ispisanim malim slovima. |
| `<ol type="I">` | Označava elemente liste rimskim brojevima ispisanim velikim slovima. |



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Atribut `start` sa vrednošću koja je različita od `1` u oznaci `<ol>` obezbeđuje početak obeležavanja od željene vrednosti. Primer uređene liste koja počinje od broja 3, je definisan na sledeći način:

```
<ol type="1" start="3">
<li>Prva stavka</li>
<li>Druga stavka</li>
<li>Treća stavka</li>
</ol>
```

**Neuređene liste** pružaju mogućnost grupisanja tekstualnih elemenata kada redosled elemenata nije bitan, a za označavanje elemenata liste se koriste simboli. Da bi označili neuređenu listu, treba umetnuti elemente liste između početne i završne oznake neuređene liste `<ul>...</ul>`. Elementi neuređene liste se smeštaju između početne i završne oznake liste `<li>...</li>`. Podrazumevana oznaka elementa neuređene liste je puna crna tačka. Atribut `type` u oznaci `<ul>` definiše željeni simbol.

`<ul type="disc">` Ispred elementa prikazuje punu crnu tačku.

`<ul type= "square">` Ispred elementa prikazuje crni kvadrat.

`<ul type="circle">` Ispred elementa prikazuje crnu kružnicu.

Primer neuređene liste čije su stavke obeležene kvadratima dat je u nastavku:

```
<ul type="square">
<li>Prva stavka neuređene liste</li>
<li>Druga stavka neuređene liste </li>
<li>Treća stavka neuređene liste </li>
</ul>
```

**Lista definicija** je skup termina uz koje stoji opis, tj. objašnjenje termina. Čitač uvlači svaku definiciju u sledeći red, ispod termina. Lista definicija se kreira umetanjem termina i definicije između početne i završne oznake liste definicija `<dl>...</dl>`.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

U listi definicija koriste se ove oznake:

| | |
|---|---|
| `<dt>...</dt>` | Početna i završna oznaka termina definicije za kreiranje termina koji treba da bude opisan ili definisan. |
| `<dd>...</dd>` | Početna i završna oznaka opisa (objašnjenja) termina za kreiranje jednog ili više pasusa uvučenog teksta objašnjenja. |

Primer liste sa dve definicije:

```
<dl>
<dt>Kafa</dt>
<dd> Kafa je napitak koji se priprema kuvanjem prženih semenki biljke kafe, najčešće u vodi ili mleku.</dd>
<dt>Mleko</dt>
<dd> Mleko je najčešći naziv za tečnost bele boje koju proizvode mlečne žlezde ženki sisara koja se koristi u ishrani.</dd>
</dl>
```

**OKVIRI**

HTML okviri se koriste za prikazivanje više od jedne Web stranice u jednom prozoru čitača. Čitač učitava stranice u definisane okvire, a pri tome ne menja sadržaj Web stranica. Web stranice sa okvirom su se najčešće koristile za prikazivanje navigacionog menija. Uobičajeno je da se navigacioni meni nalazi u horizontalnom ili vertikalnom položaju na vrhu, odnosno uz levu ivicu prozora čitača. Stavljanje navigacionih menija u okvir obezbeđuje da se navigacioni meniji vide sve vreme na ekranu Web čitača.

Stranica koja sadrži okvire ne sadrži HTML kod iz svih stranica koje prikazujemo zajedno u prozoru čitača. Ovako definisana stranica samo nalaže čitaču da prikaže Web stranice koje su posebno kreirane u okvirima koji su definisani uparenom oznakom `<frameset>...</frameset>`. Ova uparena oznaka u stvari zamenjuju početnu oznaku tela `<body>` i završnu oznaku tela `</body>`, koja se primenjuje kod stranica bez okvira. Stranica sa okvirima nema ni telo koda ni sadržaj koji bi prikazala. Njena jedina svrha je da podeli prozor čitača na okvire (kao ramove slika) i da kaže čitaču putanju ili URL adresu Web stranica koje treba da prikaže u okvirima.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Atribut `cols` govori čitaču broj okvira kao kolone i širinu svakog okvira u pikselima. Ako je potrebno kreirati tri okvira kao kolone, atribut `cols` bi imao tri vrednosti i bio bi oblika `cols="x,y,z"`, a za četiri okvira, `cols="x,y,z,w"`. Ukoliko želite tri okvira kao redove umesto atributa `cols` koristi se atribut `rows` koji ima tri brojčane vrednosti `rows="x,y,z"`. Atribut `src` u oznaci `<frame>` sadrži putanju do Web stranice koju u okviru treba da prikaže, pošto učita stranicu s okvirima.

U sledećem primeru, čitač Weba će učitati stranicu `linkovi.html` u prvi okvir (`menu`), a `podaci.html` u drugi okvir (`content`).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<html>
      <head>
      <title>Primeri HTML dokumenata OKVIRI (Kolone)</title>
      </head>
      <frameset cols="200,700">
            <frame src="linkovi.html" name="menu">
            <frame src="podaci.html" name="content">
      </frameset>
</html>
```

Kada posetilac lokacije aktivira hipervezu u okviru menija (stranica `linkovi.html`), atribut target u oznaci `<a>` reći će čitaču u koji okvir da učita Web stranicu iz atributa `href` oznake `<a>`.

Primer otvaranja stranice `o_nama.html` u okviru pod nazivom `content`:

```
<a href ="o_nama.html" target ="content"> About Us </a>
```

Stranice sa okvirima traže da čitač učita i prikaže tri ili više Web stranica, što zahteva mnogo više vremena nego kad stranica ne bi imala okvire. Sa druge strane nemaju svi čitači mogućnost da prikazuju okvire. Da biste i takvim čitačima pružili priliku da prikažu Vaš HTML dokument postavite oznake `<noframes>` i `</noframes>` u dokument sa okvirima. Na sadašnjim Web stranama podela stranica na delove se češće vrši pomoću generičkih elemenata.

**GENERIČKI ELEMENTI (DIV i SPAN)**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

HTML sadrži dva generička elementa koji se mogu prilagoditi tako da opišu sadržaj. Element `div` (skraćenica od engl. *division* - podela) generički je element na nivou bloka, dok je element `span` generički element za umetanje teksta. Generičkom elementu se može dati ime, pomoću atributa `id` ili `class`.

Dodavanje istog skupa atributa većem broju susednih elemenata stranice može se obaviti postavljanjem željenih elemenata između početne i završne oznake `<div>...</div>`. Element `div` se koristi i za podelu teksta na blokove. Označavajući srodne elemente sa `div` i dodeljujući im opisna imena, elementima u grupi dajete kontekst. U sledećem primeru, element `div` je upotrebljen kao kutija u koju su stavljeni jedna slika i dva pasusa, grupisani pod imenom „`listing`". Svi elementi unutar elementa div su centrirani na strani atributom `align`. Češće se podešavanje atributa vrši unutar CSS fajla, o čemu će više biti reči u vežbi br. 5.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
   <head>
      <title>Primer za div</title>
      <meta http-equiv="Content-Type" content="text/html;
      charset=utf-8" />
   </head>
<body>
      <p>Ovo je deo teksta koji se nalazi ispred div-a</p>

      <div class="primer" align="center" style="border: 2px
      solid red">
            <img src="slike/internet.jpg"/>
            <p>Ovo je tekst unutar div-a koji je
            centriran.</p>
            <p>Ovo je drugi red unutar div-a koji je
            centriran.</p>
      </div>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
      <p>Ovo je deo teksta koji se nalazi iza div-a</p>
</body>
</html>
```

Elementom `span` se postiže isto što i elementom `div`, s tim što se koristi za umetnute elemente koji ne prelaze u novi red. Element `span` je umetnut, pa može da sadrži samo tekst i druge umetnute elemente. Drugim rečima u `span` element ne možete da stavite elemente bloka.

U narednom primeru obeleženi elementi `span` dodaju značenje onome što bi inače bio niz nasumice poređanih brojeva. Osim toga omogućavaju da primenimo isti stil za telefonske brojeve na čitavom web sajtu.

```
<ul>
      <li>Petar:
      <span class="telefon" style="color:lightblue;
      font-weight:bold">
           063-4660-990
      </span>
      </li>

      <li>Jovan:
      <span class="telefon" style="color:lightblue;
      font-weight:bold">
           063-4766-110
      </span>
      </li>

      <li>Milena:
      <span class="telefon" style="color:lightblue;
      font-weight:bold">
            063-7746-900
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
        </span>
        </li>
</ul>
```

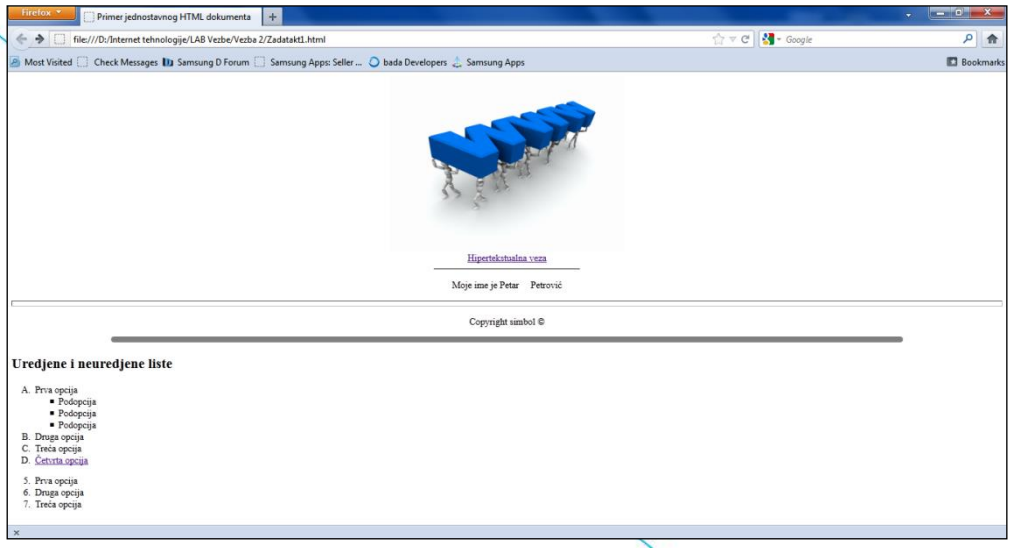


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

## ZADACI ZA VEŽBU

***Zadatak 1:***

Kreirati stranicu sa osnovnim HTML elementima kao na sledećoj slici. Osim osnovnih elemenata umetnuti sliku u dokument i kreirati uređenu i neuređenu listu. Jednu od opcija (npr. `D. Četvrta opcija`) postaviti da bude hiperaktivan veza.




Kod za formiranje stranice sa prethodne slike dat je u nastavku. Ovaj kod smestiti u dokument i snimiti ga sa odgovarajućom ekstenzijom.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<html>
   <head>
      <title>Primer jednostavnog HTML dokumenta</title>
      <meta http-equiv="Content-Type" content="text/html;
      charset=utf-8" />
   </head>
      <body>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<center>

<img src="slike/internet.jpg"><br/>
<a href="Zadatakt1_Cilj.html">Hipertekstualna veza</a>
<hr width="250">
<p>Moje ime je Petar    &nbspč;Petrović
</p>
<hr width="100%" size="10">
<p>Copyright simbol &#169;</p>
<hr width="80%" size ="10" noshade>

</center>

<h2>Uredjene i neuredjene liste</h2>

<ol type="A">
  <li>Prva opcija</li>
        <ul type="square">
          <li>Podopcija</li>
        <li>Podopcija</li>
        <li>Podopcija</li>
        </ul>
  <li>Druga opcija</li>
  <li>Treća opcija</li>
  <li><a href="Zadatakt1_Cilj.html">
      Četvrta opcija
      </a>
  </li>
</ol>
```

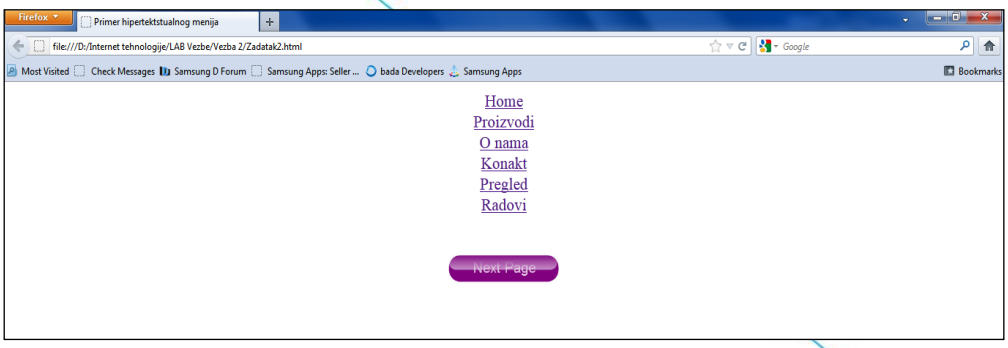


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
      <ol type="1" start="5">
      <li>Prva opcija</li>
      <li>Druga opcija</li>
      <li>Treća opcija</li>
      </ol>
</body>
</html>
```

***Zadatak 2:***

Koristeći HTML oznake formirati Web stranicu koje predstavlja meni. Za ove potrebe koristiti HTML oznaku za hipertekstualni poziv. Iskoristiti mogućnost da slika bude hiperveza.




Kod za kreiranje stranice koja predstavlja meni dat je u nastavku. Snimiti ga u okviru dokumenta sa odgovarajućom ekstenzijom.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
      <title>Primer hipertekstualnog menija</title>
      <meta http-equiv="Content-Type" content="text/html;
      charset=utf-8" />
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
</head>
<body>
  <center>
  <a href="Zadatak2_Cilj1.html" target="right">Home</a><br/>
  <a href="Zadatak2_Cilj2.html" target="right">Proizvodi</a>
  <br/>
  <a href="Zadatak2_Cilj3.html" target="right">O nama</a><br/>
  <a href="Zadatak2_Cilj4.html" target="right">Konakt</a><br/>
  <a href="Zadatak2_Cilj5.html">Pregled</a><br/>
  <a href="Zadatak2_Cilj6.html" target="left">Radovi</a><br/>
  <br/>
  <a href="http://www.vtsnis.edu.rs" target="right">
     <img src="slike/dugme.jpg" width="150">
  </a>
  </center>
</body>
</html>
```

Kod za kreiranje ciljne stranice na koju ukazuje veza unutar menija dat je u nastavku. Snimiti ga u okviru posebnog dokumenta sa odgovarajućom ekstenzijom.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<html>
   <head>
      <title>
         Primer jednostavnog HTML dokumenta
      </title>
      <meta http-equiv="Content-Type" content="text/html;
      charset=utf-8" />
   </head>
```

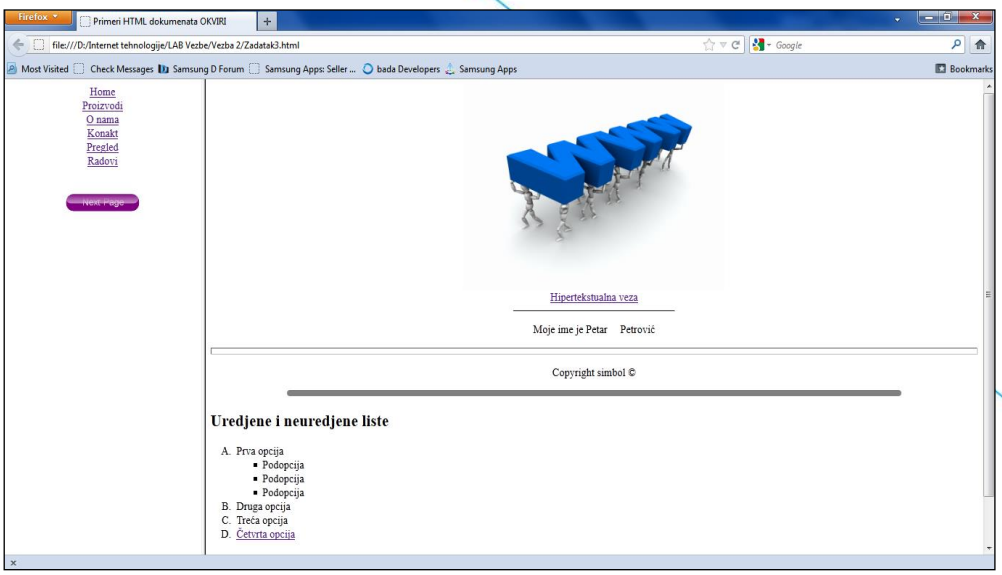


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<body>
      <center>
      <p>
         <font size="+7" color="#1E90FF">
         <b>HOME</b>
      </p>
      </center>
</body>
</html>
```

***Zadatak 3:***

Formirati novu Web stranicu sa okvirima. U levi okvir učitati stranicu koja predstavlja meni sa hipertekstualnim vezama (stranica kreirana u Zadatku 2.). U desni okvir učitavati sadržaje izabranih stranica, kao na sledećoj slici (stranica kreirana u Zadatku 1. i druge).




Primer HTML koda za formiranje stranice sa okvirima dat je sledećim kodom. Ovaj kod smestiti u poseban fajl sa ekstenzijom `.htm` ili `.html`.

```
<html>
```

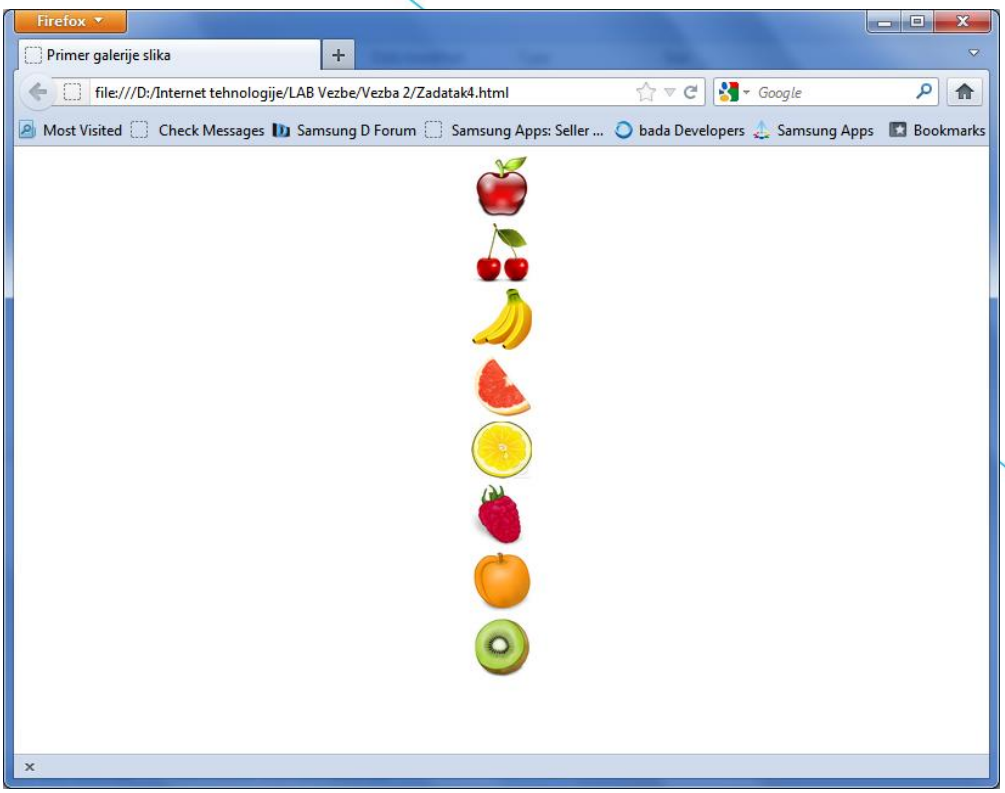


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<head>
      <title>Primeri HTML dokumenata OKVIRI</title>
</head>
      <frameset cols="200,800">
            <frame src="Zadatak2.html" name="left">
            <frame src="Zadatak1.html" name="right">
      </frameset>
</html>
```

***Zadatak 4:***

Kreirati stranicu koja se koristi kao galerija slika (ukupno 8 slika) kao što je prikazano u nastavku. Postaviti svaku malu sliku unutar hiperaktivne veze, tako da se klikom na malu sliku otvara ista takva slika većih dimenzija.






Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Napisati kod za kreiranje galerije slika.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

## Pitanja i zadaci

1. Koji atribut mora biti sadržan u svakom elementu `img`?

2. Napišite kod za dodavanje slike `primer.jpg` koja se nalazi u istom direktorijumu kao i tekući dokument.

3. Čemu služe atributi `height` i `width`. Da li je neophodno da stoje uz oznaku `img`? Kakva je prednost dodavanja atributa `height` i `width` svakoj slici na strani?

4. Šta sve može biti problem ako se slike ne vide u Web čitaču? Navedite bar tri objašnjenja.

5. Napisati html kod za dodavanje slike `veza.jpg` kao hiperaktivne veze. Slika se nalazi u podfolderu `Slike` foldera u kome se nalazi `.html` dokument.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

6. Koja je razlika između oznake `ul` i `ol`?

7. Kako se Web stranica može podeliti na više delova?

8. Koje su prednosti i mane Web stranica sa okvirima?



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# Vežba br. 3 - HTML TABELE

## CILJ VEŽBE

- Upoznati se sa načinima kreiranja tabela pomoću HTML-a.
- Upoznati se sa mogućim primenama tabela u Web dizajnu.
- Savladati podešavanje rastojanja između sadržaja ćelija i njenih ivica.
- Savladati podešavanje rastojanja između ćelija.
- Kreiranje složenijih tabela u kojima su spojene ćelije po redovima ili kolonama.
- Podešavanje različitih atributa oznaka za tabele.

## TEORIJSKE OSNOVE ZA IZRADU VEŽBE

**HTML tabele** u okviru jedne strane mogu da se koriste u najrazličitije svrhe. Pre svega koriste se za tabelarno prikazivanje podataka, ali i za raspoređivanje elemenata u okviru strane. Veoma su korisne u situacijama kada skup okvira postane previše glomazan za rad, pa tabele predstavljaju koristan dizajnerski alat za kreiranje rasporeda elemenata.

Tabele (eng. *table*) se sastoje iz kolona i redova koji se sastoje od niza ćelija u kojima su smešteni pojedinačni podaci ili sadržaj. Sadržaj mogu činiti kako tekst, tako i slike, multimedijalni sadržaj, hiperveze i sl.

HTML tabele se kreiraju pomoću oznaka za redove i kolone koje su smeštene unutar oznake za početak i kraj kreiranja tabele:

```
<table>
 ……
</table>
```

Nakon pomenute oznake, neophodno je podesiti redove u tabeli. Red HTML tabele označava se pomoću sledećeg elementa:

```
<tr>
……
</tr>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Svaki red HTML tabele sadrži nizove horizontalnih ćelija tabele, sa podacima, u kojima se nalaze stvarni podaci. Ćelije se označavaju pomoću sledećih elemenata:

```
<td>.........</td>
```

Dakle, hijerarhija elemenata počinje elementom `<table>.........</table>`, koji sadrži sve ostale elemente tabele. Element za redove tabele (`<tr>.........</tr>`) sadrži ćelije sa podacima iz tabele. Ćelije sa podacima iz tabele (`<td>.........</td>`) sadrže stvarne podatke koji se prikazuju u tabeli. Za kreiranje Tabele 1, koja se sastoji od 2 reda i 3 kolone, potrebno je napisati sledeći HTML kod.

Tabela 1.

| Prezime | Ime | Broj indeksa |
|---|---|---|
| Petrović | Petar | SER 11/11 |

```
<table border="1">
      <tr>
            <td>Prezime</td>
            <td>Ime</td>
            <td>Broj indeksa</td>
      </tr>
      <tr>
             <td>Petrović</td>
             <td>Petar</td>
             <td>SER 11/11</td>
      </tr>
</table>
```

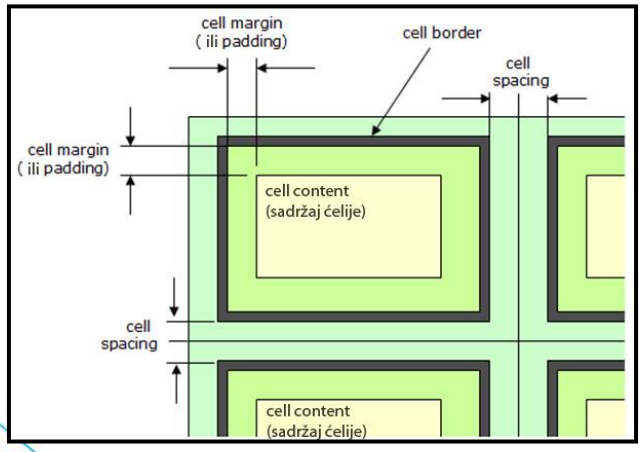
Unutar početne oznake `<table>` može da stoji atribut `border` koji služi za podešavanje širine ivice tabele. Vrednost atributa border se zadaje u pikselima, kao što je prikazano u prethodnom primeru.

Preporučljivo je da se za postavljanje širine okvira (ivice ili linije) tabele u pikselima koriste pravila CSS-a, a ne atribut elementa `<table>`. Međutim ako ga uopšte ne podesimo, tabela možda neće biti vidljiva u svim pretraživačima.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Veoma je bitno podesiti rastojanje između sadržaja ćelije i njenih okvira, što predstavlja margine ćelije tabele (*engl. cell padding*) kao i razmak između ćelija (*eng. cell spacing*). Ovo je ilustrovano na Slici 1.




*Slika 1. Cell spacing i Cell padding*

Da bi zadali da rastojanje između sadržaja i ivice ćelije bude 10 piksela, a rastojanje između ćelija bude 20 piksela, treba napisati sledeće:

```
<table border="1" cellpadding="10" cellspacing="20">
```

Zaglavlja tabele su redovi ili grupe redova koji predstavljaju naslove kolona ili redova. Tekst može biti podebljan ili naglašen na različite načine kako bi se razlikovao od samih tabelarnih podataka. Najčešće korišćena oznaka je `<th>` što daje efekat crnih podebljanih slova. Primer:

```
  <table>
      <tr>
            <th>Prezime</th>
            <th>Ime</th>
            <th>Broj indeksa</th>
      </tr>
      <tr>
             <td>Petrović</td>
             <td>Petar</td>
             <td>SER 11/11</td>
      </tr>
</table>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Kada se tabela koristi zajedno sa drugim sadržajem kao slika kojom se nešto objašnjava, često je potrebno kreirati naslov tabele. Pomoću elementa `<caption>...</caption>` tabeli se dodaje naslov, odnosno natpis. Kada se koristi, element `<caption>` mora da bude prvi element u elementu `<table>`. U suprotnom, natpis tabele se neće prikazati. Ovaj element nema unapred definisano formatiranje, pa se određeni izgled mora posebno definisati. Primer koji ilustruje prethodno rečeno, dat je u nastavku:

```
<table>
      <caption>Naslov tabele 1 - Spisak studenata</caption>
      <tr>
            <th>Prezime</th>
            <th>Ime</th>
            <th>Broj indeksa</th>
      </tr>
      <tr>
             <td>Petrović</td>
             <td>Petar</td>
             <td>SER 11/11</td>
      </tr>
</table>
```

Često je neophodno da se jedna ćelija unutar tabele spoji sa sledećom ćelijom iz sledeće kolone ili reda, ili da jedna ćelija zauzme ceo red ili kolonu. Spajanje više kolona se postiže upotrebom atributa `colspan` (*engl. column spanning = spajanje kolona*) koji se dodaje oznaci `<th>` ili `<td>`. Spajanje više redova se postiže upotrebom atributa `rowspan` (*engl. row spanning = spajanje redova*) koji se dodaje oznaci `<th>` ili `<td>`. Redovi i kolone mogu biti spojeni bilo kako, ali se mora voditi računa o konačnom broju redova ili kolona kako bi struktura tabele bila ispravna.

Tabela 2.

| | Prezime | Ime | Broj indeksa |
|---|---|---|---|
| 1. godina | Petrović | Petar | SER 11/11 |
| | Jovanović | Jovan | SER 12/11 |



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Za realizaciju Tabele 2. potrebno je uneti sledeći kod:

```
<table border="1">
      <tr>
            <td> </td>
            <td>Prezime</td>
            <td>Ime</td>
            <td>Broj indeksa</td>
      </tr>
      <tr>
             <td rowspan="2">1. godina</td>
             <td>Petrović</td>
             <td>Petar</td>
             <td>SER 11/11</td>
      </tr>
      <tr>
             <td>Jovanović</td>
             <td>Jovan</td>
             <td>SER 12/11</td>
      </tr>
</table>
```

U kodu se može uočiti da treći red ima jednu oznaku `<td>...</td>` manje, a razlog za to je što se ćelija, iz prethodnog drugog reda i prve kolone, prostire na dva reda (`rowspan="2"`).

Tabela 3.

| Prezime i ime | | Broj indeksa |
|---|---|---|
| Petrović | Petar | SER 11/11 |
| Jovanović | Jovan | SER 12/11 |

Za realizaciju Tabele 3. potrebno je uneti sledeći kod:

```
<table border="1">
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
        <tr>
              <td colspan="2">Prezime i ime</td>
              <td>Broj indeksa</td>
        </tr>
        <tr>
               <td>Petrović</td>
               <td>Petar</td>
               <td>SER 11/11</td>
        </tr>
        <tr>
               <td>Jovanović</td>
               <td>Jovan</td>
               <td>SER 12/11</td>
        </tr>
</table>
```

U kodu se može uočiti da prvi red ima jednu oznaku `<td>...</td>` manje, a razlog za to je što se ćelija iz prvog reda i prve kolone prostire na dve kolone (`colspan="2"`).

Pored do sada navedenih atributa, HTML tabele mogu da sadrže i sledeće atribute.

- `bordercolor` – koji služi za podešavanje boje svih ivica tabele. Postavlja se unutar oznake `</table>`. Vrednost se može zadati heksadecimalnom notacijom, pomoću rgb vrednosti ili nazivom boje, kao što je pokazano u vežbama 1. i 2.

```
<table border="1" bordercolor="red">
```

- `width i height` – atributi koji služe za podešavanje širine i visine ćelije ili cele tabele. Širina se može zadati u pikselima (apsolutno) ili u procentima (relativno) što zapravo predstavlja procenat od širine čitača Weba.

```
<table border="1" bordercolor="red" width="400" height="200">
```

Da bi nacrtali tabelu preko celog prozora čitača, čija je linija crvene boje, treba napisati sledeće:

```
<table border="1" bordercolor="red" width="100%" height="100%">
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- `bgcolor` – je atribut za podešavanje boje pozadine tabele ili pojedinačne ćelije. Sledeći primer boji pozadinu cele tabele u plavo:

```
<table border="1" bordercolor="red" width="400" height="200" bgcolor="#0000FF">
```

- `background` - čitač Weba će koristiti datoteku slike dodeljenu ovom atributu kao "tapete" za prekrivanje pozadine tabele. Upotrebom ovog atributa u oznaci `<td>` slikom se prekriva pozadina ćelije. U sledećem primeru prekriva se pozadina tabele slikom iz poddirektorijuma images pod nazivom `slika.gif`.

```
<table border = "1" background="images/slika.gif">
```

- `align` - Horizontalno poravnanje sadržaja ćelije. *Upotrebljava se u oznaci ćelije <td>.* Može se zadati vrednosti `left`, `right` i `center`.
- `valign` - Centrira sadržaj ćelije vertikalno između vrha i dna. `Top`, `middle`, `baseline` i `bottom` su dozvoljene vrednosti ovog atributa.

Na kraju, ako je potrebno da se u tabelu umetne slika ili hipertekstualna veza, treba dodati oznaku za sliku ili vezu između odgovarajućih oznaka `<td>` i `<td>`, kako bi ih smestili u željenu ćeliju.

Za dodavanje slike u ćeliju tabele, treba postaviti oznaku `<img>` između početne oznake `<td>` i završne oznake `</td>`:

```
<table>
      <tr>
          <td><img src="image.jpg" ></td>
      </tr>
</table>
```

Da slika u tabeli bude i hiperaktivna veza, oko oznake `<img>` `treba` staviti početnu i završnu oznaku sidra veze `<a>` i `</a>`

```
<table>
      <tr>
          <td>
              <a href="info.htm"><img src="image.jpg"></a>
          </td>
      </tr>
</table>
```

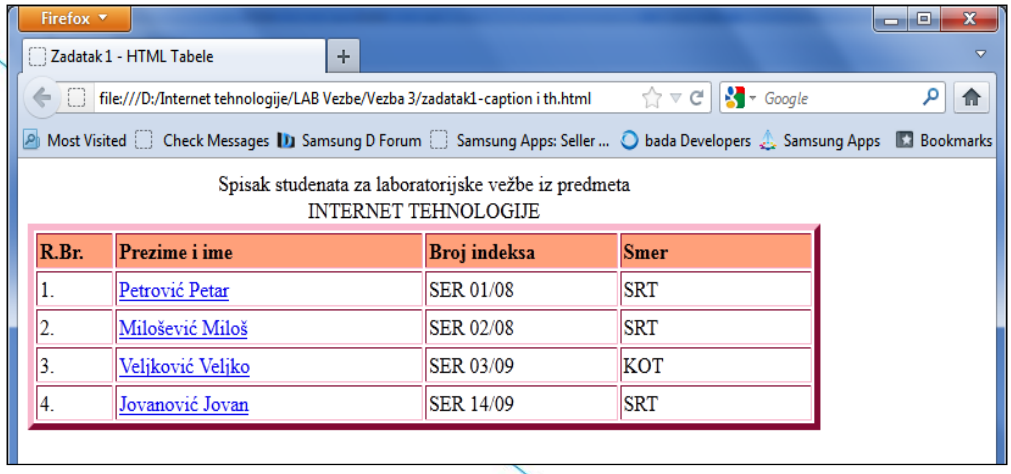


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

## ZADACI ZA VEŽBU

***Zadatak 1:***

Iskoristiti tabele za smeštanje osnovnih podataka o studentima vaše grupe – prvi red kreirati kao zaglavlje tabele koja će sadržati matične podatke o studentu: Redni broj, prezime i ime, br.indeksa, smer. Proceniti širine kolona u procentima, a čitava tabela treba da ima širinu 650 piksela. Upotrebite oznake za formiranje zaglavlja i naslova tabele. Realizujte nekoliko tipova okvira tabele.




Primer HTML koda za formiranje stranice kao na prethodnoj slici dat je sledećim kodom. Ovaj kod smestiti u poseban fajl sa ekstenzijom .htm ili .html.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
      <head>
          <title>Zadatak 1 - HTML Tabele</title>
          <meta http-equiv="Content-Type" content="text/html; charset=utf-
8" />
      </head>
      <body>
         <table border="5" bordercolor="#ee115b"
         cellpadding="2" cellspacing="2" bgcolor="#ffffff"
          width="650">
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<caption>
Spisak studenata za laboratorijske vežbe iz
predmeta<br/>INTERNET TEHNOLOGIJE
</caption>
<tr align="left" valign="middle" bgcolor="#ffa07a">
        <th width="10%">R.Br.</th>
        <th width="40%">Prezime i ime</th>
        <th width="25%">Broj indeksa</th>
        <th width="25%">Smer</th>
</tr>
<tr align="left" valign="middle">
        <td width="10%">1.</td>
        <td width="40%"><a href="pp.html">Petrović
        Petar</a></td>
        <td width="25%">SER 01/08</td>
        <td width="25%">SRT</td>
</tr>
<tr align="left" valign="middle">
        <td width="10%">2.</td>
        <td width="40%"><a href="mm.html">Milošević
        Miloš</a></td>
        <td width="25%">SER 02/08</td>
        <td width="25%">SRT</td>
</tr>
<tr align="left" valign="middle">
        <td width="10%">3.</td>
        <td width="40%"><a href="vv.html">Veljković
        Veljko</a></td>
        <td width="25%">SER 03/09</td>
```

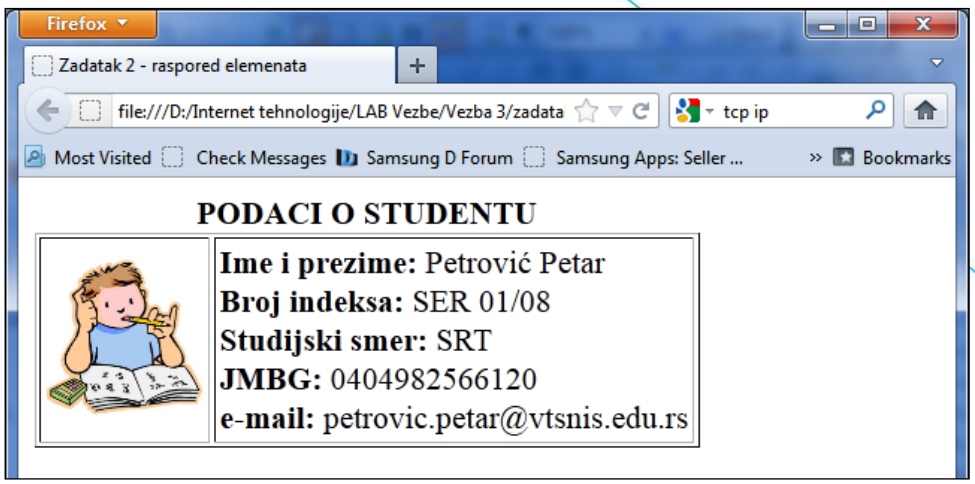


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
                <td width="25%">KOT</td>                                    </tr>
        <tr align="left" valign="middle">
                <td width="10%">4.</td>
                <td width="40%"><a href="jj.html">Jovanović
                Jovan</a></td>
                <td width="25%">SER 14/09</td>
                <td width="25%">SRT</td>
         </tr>
       </table>
     </body>
</html>
```

***Zadatak 2:***

Upotrebiti tabelu u svrhu rasporeda objekata na Web stranici. Formirajte preglednu tabelu sa podacima o studentu i unesite fotografiju u ćeliju tabele. Izvršiti poravnanje objekata na Web stranici pomoću tabela. Primer je prikazan na sledećoj slici.




Primer HTML kod za formiranje stranice kao na prethodnoj slici dat je sledećim kodom. Ovaj kod smestiti u poseban fajl sa ekstenzijom .htm ili .html.



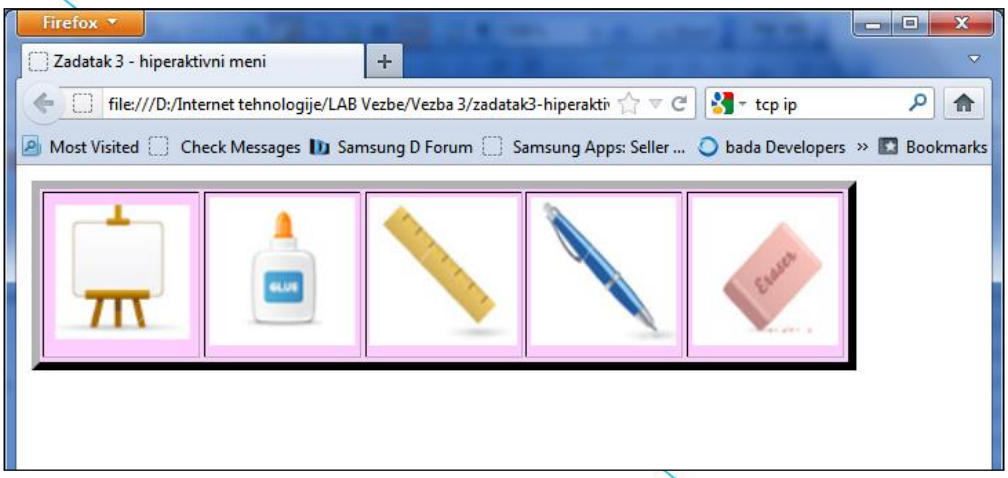
Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
      <head>
          <title>Zadatak 2 - raspored  elemenata</title>
          <meta http-equiv="Content-Type" content="text/html;
          charset=utf-8" />
      </head>
      <body>
            <table border="1" cellpadding="2" cellspacing="2"
            bgcolor="#ffffff">

            <caption><b>PODACI O STUDENTU</b></caption>

            <tr align="left" valign="middle">
            <td height="40" align="center" valign="middle">
            <img src="Slike/student.jpg" width="80"
            height="80">
            </td>

            <td height="40"><b>Ime i prezime:</b>  
            Petrović Petar<br/>
            <b>Broj indeksa:</b> SER 01/08<br/>
            <b>Studijski smer:</b> SRT<br/>
            <b>JMBG:</b> 0404982566120<br/>
            <b>e-mail:</b>
             petrovic.petar@vtsnis.edu.rs<br/>
            </td>
            </tr>
            </table>
```

```
        </body>
</html>
```

***Zadatak 3***

Formirajte Web stranicu koja će upotrebom tabele formirati hiperaktivni meni (klikom na svaku sličicu otvara se posebna web strana). U nekom od korisničkih programa formirajte sličice koje odgovaraju zahtevanom meniju ili ih preuzmite sa nekog od pretraživača. Ovako formirane sličice postavite u horizontalni odnosno vertikalni meni na Vašoj Web stranici.




Web stranica sa menijem formiranim tabelom i sličicama se kreira na sledeći način:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
        <head>
            <title>Zadatak 3 - hiperaktivni meni</title>
            <meta http-equiv="Content-Type" content="text/html;
            charset=utf-8" />
        </head>
        <body>
              <table border="5" cellpadding="2" cellspacing="2"
```

```
width="327" bgcolor="#ffccff">
<tr height="20">
<td width="80" height="80">
      <a href="zadatak3_Test_1.html">
      <img height="80" width="80"
      src="slike/board.png">
      </a>
</td>
<td width="80" height="80">
      <a href="zadatak 3_Test_2.html">
      <img height="80" width="80"
      src="slike/glue.png">
      </a>
</td>
<td width="80" height="80">
      <a href="zadatak 3_Test_3.html">
      <img height="80" width="80"
      src="slike/ruler.png">
      </a>
</td>
<td width="80" height="80">
      <a href="zadatak 3_Test_4.html">
      <img height="80" width="80"
      src="slike/pen.png">
      </a>
</td>
<td width="80" height="80">
      <a href="zadatak 3_Test_5.html">
      <img height="80" width="80"
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
                    src="slike/rubber.png">
                    </a>
            </td>
            </tr>
            </table>
      </body>
</html>
```

Web strana, koja se otvara aktiviranjem jedne od stavki menija, prikazana je na sledećoj slici.

*Učitana Web stranica iz menija formiranog tabelom i sličicama*

HTML kod ze kreiranje strane sa prethodne slike, upisati u nastavku.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

***Zadatak 4***

Korišćenjem atributa `rowspan` i `colspan` formirajte Web stranice sa tabelama kao na sledećim slikama. Povežite linkove iz tabele sa stranicama koje ste kreirali u zadatku 3 (koje objašnjavaju pojmove iz hiperaktivnog menija).

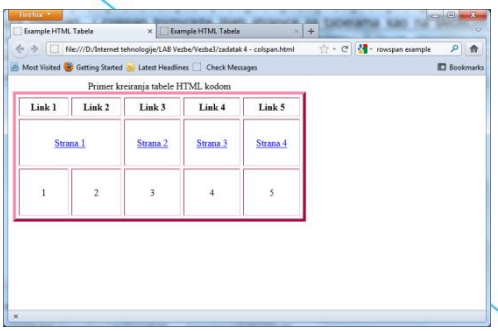



Tabela u kojoj se ćelija prostire na dve kolone (opcija `colspan`)

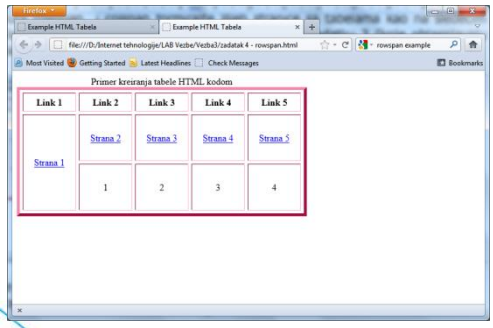



Tabela u kojoj se ćelija prostire na dva reda (opcija `rowspan`)

Primer koda za tabelu u kojoj se ćelija prostire na dve kolone (opcija `colspan`), dat je u nastavku:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
    <head>
        <title>Example HTML Tabela</title>
        <meta http-equiv="Content-Type" content="text/html;
        charset=utf-8" />
    </head>
    <body>
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
        <table border="5" bordercolor="#ee115b"
        cellpadding="5" cellspacing="5" width="500"
        bgcolor="#ffffff">
        <caption>
        Primer kreiranja tabele HTML kodom
        </caption>
        <tr align="center" valign="middle" height="20">
              <th>Link 1</th>
              <th>Link 2</th>
              <th>Link 3</th>
              <th>Link 4</th>
              <th>Link 5</th>
        </tr>
        <tr align="center" valign="middle" height="80">
              <td colspan="2"><a href="Test_1.htm">
              Strana 1</a></td>
              <td><a href="Test_1.htm">Strana 2</a></td>
              <td><a href="Test_1.htm">Strana 3</a></td>
              <td><a href="Test_1.htm">Strana 4</a></td>
        </tr>
        <tr align="center" valign="middle" height="80">
              <td>1</td>
              <td>2</td>
              <td>3</td>
              <td>4</td>
              <td>5</td>
        </tr>
        </table>
  </body>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
</html>
```

Primer koda za tabelu u kojoj se ćelija prostire na dva reda (opcija `rowspan`), dat je u nastavku:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
      <head>
          <title>Example HTML Tabela</title>
          <meta http-equiv="Content-Type" content="text/html;
          charset=utf-8" />
      </head>
      <body>
            <table border="5" bordercolor="#ee115b"
            cellpadding="5" cellspacing="5" width="500"
            bgcolor="#ffffff">
            <caption>
            Primer kreiranja tabele HTML kodom
            </caption>
            <tr align="center" valign="middle" height="20">
                  <th>Link 1</th>
                  <th>Link 2</th>
                  <th>Link 3</th>
                  <th>Link 4</th>
                  <th>Link 5</th>
            </tr>
            <tr align="center" valign="middle" height="80">
                  <td rowspan="2"><a href="Test_1.htm">
                  Strana 1</a></td>
                  <td><a href="Test_1.htm">Strana 2</a></td>
```

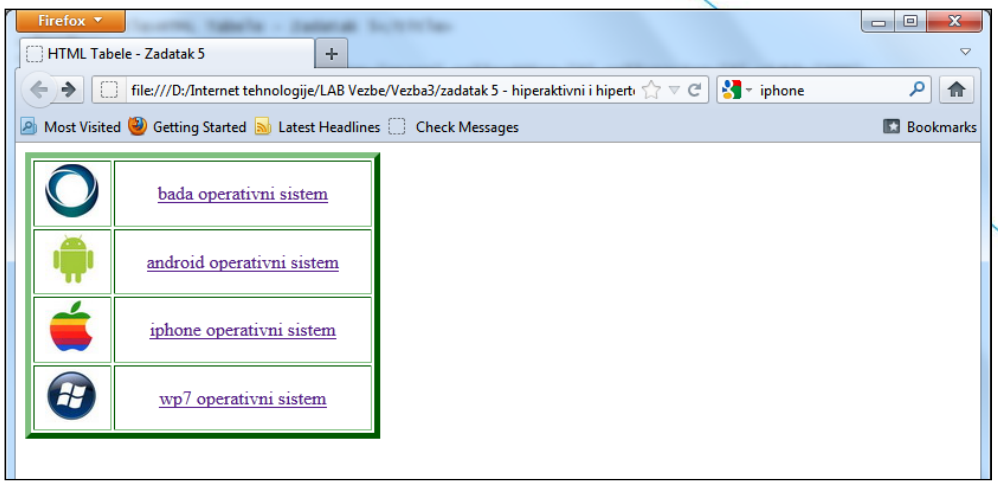


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
                <td><a href="Test_1.htm">Strana 3</a></td>
                <td><a href="Test_1.htm">Strana 4</a></td>
                <td><a href="Test_1.htm">Strana 5</a></td>
            </tr>
            <tr align="center" valign="midle" height="80">
                <td>1</td>
                <td>2</td>
                <td>3</td>
                <td>4</td>
            </tr>
            </table>
        </body>
</html>
```

***Zadatak 5***

Formirajte Web stranicu koja će upotrebom tabele formirati hiperaktivni i hipertekstualni meni. Klikom na svaku sličicu ili klikom na odgovarajući link otvara se posebna web strana. U nekom od korisničkih programa formirajte sličice koje odgovaraju zahtevanom meniju.




*Web stranica sa asocijativnim slikama i hipertekstualnim vezama*



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Kod za realizaciju prethodne stranice upisati u nastavku.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

*Pitanja i zadaci:*

1. Koje oznake se koristi za kreiranje tabele?

2. Koje su osnovne mogućnosti HTML-a za poravnavanje sadržaja ćelija tabela? Koja je razlika između atributa `align` i `valign`?

3. Koji se atributi koriste za vizuelno sređivanje tabela?

4. Pronađite 5 grešaka u narednom HTML dokumentu.

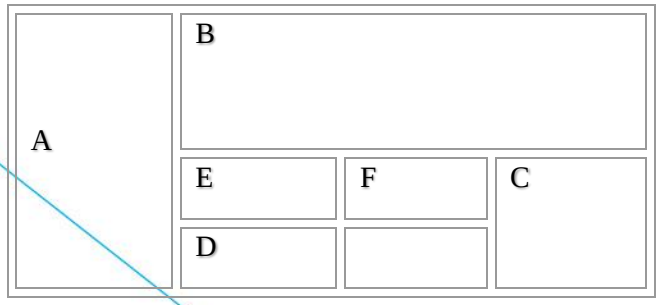
```
<caption>TV program za Čevtrak, 16.02.2012.god<caption>
<table>
      Četvrtak uveče
      <tr></tr>
            <th>7:30</th>
            <th>8:00</th>
            <th>9:30</th>
      <tr>
            <td>Jutarnje vesti</td>
            <td>Crtani filmovi</td>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
            <td>Kefalica</td>
      <tr>
            <td>by VTŠ tv</td>
      </tr>
</table>
```

5. Napisati HTML kôd za kreiranje tabele kao na slici.






Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# Vežba br.4 - HTML OBRASCI

## CILJ VEŽBE

- Upoznati se sa primenom obrazaca na Web stranama.
- Pravilno korišćenje HTML oznaka i atributa kojima se pravi obrazac.
- Kreiranje različitih kontrola obrasca.
- Priprema obrazaca za potrebe provere popunjenosti svih traženih polja na klijentskom računaru.

## TEORIJSKE OSNOVE ZA IZRADU VEŽBE

Iako se većina stranica sastoji od teksta, slika i hiperveza, skoro sve Web lokacije imaju bar po jednu stranicu na kojoj se popunjavaju podaci i te informacije se vraćaju operateru WEB lokacije.

Obrazac (eng. *form*) je skup polja koja se koriste za prikupljanje podataka ili interakciju sa korisnicima. Nakon popunjavanja formulara, posetilac Web stranice vrši potvrdu unetih podataka i ti podaci se zatim šalju serveru na obradu. Forme sadrže različite vrste objekata (polja ili kontrola) kao što su tekst polja, padajuće liste, polja za potvrdu, opciona dugmad, dugmad za slanje podataka, dugmad za brisanje unetih podataka i sl. Kada se serveru proslede podaci iz formulara, vrši se njihova obrada odgovarajućom skriptom ili aplikacijom i kao potvrda uspešnosti popunjenog formulara pojavljuje se naredna strana sa rezultatima obrade.

Prilikom kreiranja stranice sa obrascima, najpre se definiše koje podatke od korisnika treba prikupiti. Zatim se dodaje forma kojoj pripadaju sva polja i u nju se dodaju pojedinačna polja. Moguće je postaviti pravila unosa podataka u polja (dopuštene vrednosti, format unosa, obavezna polja,...). Za obradu podataka u formama potrebno je da postoji određen skript na serverskoj strani (npr. *CGI script – Common Gateway Interface*), a najjednostavniji način obrade je prosleđivanje podataka na određenu e-mail adresu. Takođe, kontrola popunjenih polja se može izvršiti i na klijentskoj strani, pre slanja na server pomoću JavaScript-a.

Kreiranje obrasca se vrši pomoću elementa `form`:

```
<form>
 ……
</form>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Obrascu se može dodeliti ime upotrebom atributa `name`. Ovaj atribut omogućava da se, ukoliko postoji više obrazaca na strani, oni međusobno razlikuju. Ime se daje tako da asocira na podatke u obrascu. Pomoću ovog imena se pristupa obrascu u JavaScript-u. Ukoliko ime nije definisano, svakom obrascu se pristupa prema njegovom rednom broju na stranici. Prvom obrascu se pristupa kao `documents.forms[0]`.

```
<form name="frmMojObrazac">
 ……
</form>
```

Kako će se formular obraditi određuje se atributom `action`, koji sadrži naziv dokumenta ili programskog skripta kome se podaci prosleđuju na obradu. Najčešće je to neka dinamička strana (PHP, CGI skript, ASP, JSP ili sl.).

```
<form name=" frmMojObrazac "
action="http://www.vtsnis.edu.rs/skript/mycgi.pl">
 ……
</form>
```

Ako želimo neke podatke da prosledimo na e-mail adresu `action` atribut glasi:

```
<form name="frmMojObrazac" action="mailto:info@vtsnis.edu.rs">
 ……
</form>
```

Način prosleđivanja podataka na server podešava se atributom `method` koji može imati vrednosti:

`post` – unete vrednosti polja (parovi atribut-vrednost) se integrišu u HTTP zahtev-request i ne vide se u URL. Izaziva neke prateće pojave, kao što je promena podataka koji se čuvaju u bazi podataka.

`get` - unete vrednosti polja (parovi atribut-vrednost) se dodaju URL adresi koja je data atributom `action`. Koristi se kada obrada obrasca ne menja podatke čuvane na WEB serveru.

```
<form name="frmMojObrazac" action="mailto:info@vtsnis.edu.rs"
method="post">
 ……
</form>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Element `form` takođe poseduje `target` atribut koji određuje u kom prozoru će se otvoriti strana koja obrađuje formu.

`_blank` – novi prozor

`_parent` – prozor koji je roditelj (iznad trenutnog)

`_self` – isti prozor u kome je i forma

`_top` – krajnji prozor iznad trenutnog (ako ima frejmova).

Forma kao element se ne vidi u čitaču, već služi samo za smeštanje svih polja koja čine celinu sa istim načinom obrade. Sadržaj forme je ono što se nalazi između početnog i završnog `form` taga. Tekst (opis obrasca) koji čitač prikazuje kada posetilac zadrži pokazivač miša na bilo kom mestu unutar obrasca se definiše atributom `title`.

```
<form name="frmMojObrazac" action="mailto:info@vtsnis.edu.rs" method="post"
title="Opis obrasca">
 ……
</form>
```

Osim navedenih atributa oznaka `form` može sadržati i atribute `onSubmit` i `onReset`.

`onSubmit` definiše naziv funkcije (tj. Skripta u HTML kodu web stranice) koju čitač treba da izvrši ako posetilac pritisne dugme SUBMIT. Koristi se za potvrdu podataka iz obrasca pre nego što oni budu poslati serveru na obradu. Ako funkcija vrati vrednost `false`, čitač neće poslati podatke iz obrasca, ako vrati vrednost `true`, čitač podatke šalje serveru da ih on obradi.

`onReset` definiše naziv funkcije (tj. Skripta u HTML kodu web stranice) koju čitač treba da izvrši ako posetilac pritisne dugme RESET. Ako funkcija vrati vrednost `true`, čitač će obrisati sve podatke unete do tada i podesiti njihove prvobitne vrednosti. Ako funkcija vrati vrednost `false` čitač neće podesiti kontrole na njihove prvobitne vrednosti.

Atribut `enctype` govori čitaču weba koju metodu kodiranja da koristi za prosleđivanje podataka obrasca serveru. Na primer sledeći kod šalje podatke u formatu `ImePolja = Vrednost`:

```
<form name="frmMojObrazac" action="mailto:info@vtsnis.edu.rs" method="post"
title="Opis obrasca" enctype="text/plain">
 ……
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
</form>
```

Sledeći kod šalje podatke u formatu, `ImePolja&Vrednost`:

```
<form name="frmMojObrazac" action="mailto:info@vtsnis.edu.rs" method="post"
title="Opis obrasca"
enctype="application/x-www-form-urlencoded">
 ……
</form>
```

**5.**

## 6. DODAVANJE KONTROLA U OBRAZAC

Većina kontrola se u obrazac dodaje pomoću taga `input`. Atributi koji su neophodni da se definišu su:

`type` – tip polja i

`name` - jedinstveno ime polja.

Atribut `type` uzima jednu od vrednosti: `button`, `checkbox`, `radio`, `text`. Na primer:

```
<input type="text" name="ImeStudenta">
```

umeće tekst polje koje ima naziv ImeStudenta. Kada korisnik na Web stranici upiše adresu u ovo tekst polje, i izvrši potvrdu formulara promenljiva `ImeStudenta` će sadržati tekst koji je korisnik upisao u ovo polje. Takav par atribut-vrednost (`ImeStudenta = Marko`) se šalje serveru na obradu.

### TEKST POLJE (TEXTBOX)

Tekst polje se u obrazac dodaje pomoću taga `input` čiji atribut `type` ima vrednost `text`:

```
<input type="text" name="txtIme">
```

Dodatni atributi su:

- `value` – inicijalna vrednost polja

```
<input type="text" name="txtIme" value="Marko" >
```

Rezultat je:

Ime: Marko

- `size` – veličina odnosno širina tekst polja u karakterima

```
<input type="text" name="txtIme" size="10" >
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<input type ="text" name ="txtIme" size="20" >
<input type ="text" name ="txtIme" size="30" >
```

Rezultat je:

Ime: Marko
Ime: Marko
Ime: Marko

- `maxlength` – maksimalan broj karaktera koji se može uneti u polje. Na primer za poštanski broj nekog mesta potrebno je max 5 cifara:

```
<input type="text" name="txtIme" maxlength="5" >
```

Jedna vrsta tekst polja je **polje za lozinku**. Ovo polje na ekranu "maskira" unos sa tastature. Polje za lozinku je tip tekstualnog polja, a razlika je što se umesto `type="text"` piše `type="password"`.

```
<input type="password" name="txtPassword" >
```

Naravno, tekst koji se nalazi pored textboxa i objašnjava šta treba uneti, dodaje se kao običan tekst u paragrafu.

**7.**

**8. Višeredno tekstualno polje - TextArea**

Tekstualno polje omogućava unos više linija teksta. Ona se dodaje pomoću posebnog elementa `textarea`, koji se koristi umesto `input` taga.

```
<textarea name="txArAdresa" >
```

Tekstualna oblast ima određenu širinu i visinu, i ako se posebno ne navedu one iznose 20 karaktera širine i dve linije teksta po visini:

Adresa: 111111111111111111111
111111111111111111111

Atributi ovog elementa su:

`cols` – širina, zadaje se brojem vidljivih karaktera

`rows` – visina, zadaje se brojem redova

**9.**

***10.* Dugme za potvrdu - Checkbox**

Checkbox omogućava potvrdu neke opcije. Dodaje se pomoću `input` taga sa atributom `type="checkbox"`:

```
<input type="checkbox" name="chToMail">
```

Atribut `name` je izabran tako da asocira da je u pitanje checkbox (`ch`), a `ToMail` govori o tome da se vrši slanje na e-mail adresu.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Atributom `value` se definiše vrednost koju će imati promenljiva kojom je nazvan checkbox, ako je izvršena potvrda dugmeta. Često se postavljaju vrednosti kao što su `1, da, true` ili slično:

```
<input type="checkbox" name="chToMail" value="DA" >
```

☑ Da, želim da se registrujem.

Kada se izvrši potvrda ovog dugmeta i potvrdi ceo formular, promenljiva `chToMail` dobija vrednost `DA (chToMail=DA)`. Tekst pored dugmeta objašnjava šta se potvrđuje.

Ako se ne navede atribut `value`, promenljivoj se dodeljuje vrednost `ON: (chToMail=ON)`. Ukoliko želimo da dugme inicijalno bude potvrđeno koristi se atribut `checked`:

```
<input type="checkbox" name="chToMail" value="DA" checked>
```

### 11. Opciono ili radio dugme

Ova kontrola omogućava izbor jedne iz grupe opcija, kao što je prikazano na sledećoj slici.

Pol: ◉ muški ○ ženski

Dodaje se u formular korišćenjem `input` taga tipa `radio`:

```
<input type="radio" name="rbPol">
```

Naravno, potrebno je polju dodeliti ime pomoću atributa `name`.

Ovaj kod umeće jedno dugme. Da bi više dugmadi predstavljali jednu grupu, od kojih je moguće izabrati samo jedno dugme (više opcija koje se međusobno isključuju), potrebno je da sva dugmad imaju isti naziv odnosno istu vrednost za `name` atribut. Dakle, za izbor pola (muški-ženski) potrebno je imati 2 opciona dugmeta koja imaju isti naziv.

```
<input type="radio" name="rbPol">
<input type="radio" name="rbPol">
```

Svakom dugmetu treba dodeliti vrednost koju će promenljiva sa nazivom dugmeta dobiti u slučaju izbora jedne od dve opcije. Atribute `value` ćemo postaviti na vrednosti `M` i `Z` koji asociraju na muški i ženski pol:

```
<input type="radio" name="rbPol" value="M">
<input type="radio" name="rbPol" value="Z">
```

Ovaj kod samo prikazuje dva dugmeta u Web čitaču, ali potrebno je dodati i tekst pored dugmadi da bi se znalo šta se izabira:

```
 <p>Pol:
 <input type="radio" name="rbPol" value="M"> muški
<input type="radio" name="rbPol" value="Z"> ženski
</p>
```

Nakon izbora jedne od opcija promenljiva `rbPol` dobija vrednost `M` ili `Z`: na primer `rbPol=M`.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Ako je potrebno, jedna od opcija može inicijalno biti selektovana, dodavanjem atributa `checked`:

```
<input type="radio" name="rbPol" value="M" checked>
```

Pol: muški ženski

### 12. Padajuća lista

Ova kontrola omogućava izbor iz grupe opcija, ali zauzima manje mesta od opcionih dugmadi. Postoje dve vrste izbornih listi: izbor iz menija ili izbor iz liste. U drugom slučaju moguće je podesiti da korisnik može da izabere i više od jedne opcije.

Studijski program:
Komunikacione Tehnologije
Savremene Računarske Tehnologije
Komunikacione Tehnologije
Drumski Saobraćaj
Industrijsko inženjerstvo
Zaštita životne sredine
Građevinsko inženjerstvo

Oznaka kojom se ove kontrole dodaju u formular je `select`. Lista ima svoj naziv, određen atributom `name`. Pojedine opcije u listi se dodaju pomoću elementa `option` unutar `select` elementa.

```
<select name="stProgram">
    <option>Savremene računarske tehnologije</option>
    <option>Komunikacione tehnologije</option>
    <option>Drumski saobraćaj</option>
    <option>Industrijsko inženjerstvo</option>
    <option>Zaštita životne sredine</option>
    <option>Građevinsko inženjerstvo</option>
</select>
```

Ovo je tip izbora iz menija. Vidljiva je samo jedna opcija a potrebno je pomoću klizača pregledati ostale i neku izabrati.

Kada se iz ovakve liste izabere jedna od opcija (recimo `Komunikacone tehnologije`) i izvrši potvrda formulara, promenljiva `stProgram` uzima vrednost: `stProgram = Komunikacone tehnologije`. Vrednosti koje dobijaju promenljive se mogu razlikovati od teksta koji se prikazuje u listi tako što se upotrebe atributi `value` za svaku od opcija:

```
<select name="stProgram">
    <option value="1">
       Savremene računarske tehnologije
    </option>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
    <option value="2">
       Komunikacione tehnologije
    </option>
    <option value="3">
       Drumski saobraćaj
    </option>
    <option value="4">
       Industrijsko inženjerstvo
    </option>
    <option value="5">
       Zaštita životne sredine
    </option>
    <option value="6">Građevinsko inženjerstvo
    </option>
</select>
```

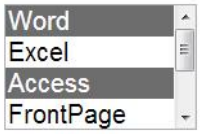
U ovom slučaju nakon izbora i potvrde formulara, promenljiva `stProgram` dobija jednu od vrednosti `1,2,3,4, 5` ili `6` i to je vrednost koja se serveru prosleđuje na obradu.

Ako želimo da neka od opcija bude inicijalno postavljena kao prva i izabrana (recimo, pretpostavka da je najveći broj korisnika na studijskom program Savremene računarske tehnologije) potrebno je kod te opcije postaviti atribut `selected`.

```
<option value="1" selected>Savremene računarske tehnologije
</option>
```

**Izbor iz liste** se razlikuje u tome što je vidljivo više od jedne opcije, i što se može izabrati više opcija. Ovakva lista ima određenu visinu, odnosno vidljivo je nekoliko opcija (u ovom slučaju 3), a do ostalih se dolazi upotrebom klizača. Izbor više opcija vrši se selektovanjem opcija uz istovremeno držanje tastera Ctrl.

Da bi se formirala ovakva lista, potrebno je `select` elementu dodeliti atribut `size` ili atribut `multiple`. Atribut `size` definiše visinu odnosno broj vidljivih opcija (ne radi u Chrome pretraživaču za vrednosti 2,3,4), a ukoliko postoji atribut `multiple`, on govori da je moguće vršiti izbor više od jedne opcije. Naravno, i ovde je moguće izvršiti inicijalno izabiranje jedne ili više opcija dodavanjem atributa `selected` pored željene opcije.




```
<select name="selProgram" size="3" multiple>
   <option selected>Word</option>
   <option>Excel</option>
   <option selected  >Access</option>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
    <option>FrontPage</option>
    <option>DreamWeaver</option>
    <option>Photoshop</option>
    <option>CorelDraw</option>
</select>
```

### 13. Dugmad

Dugmad su sastavni deo formulara i služe za određene akcije: potvrda formulara (`submit`), brisanje – poništavanje unetih podataka u polja formulara (`reset`) i druge akcije koje se definišu upotrebom skript jezika.

Potvrdi Obriši

Dugme se dodaje korišćenjem `input` taga tipa: `submit`, `reset` ili `button`.

`Submit` dugme vrši potvrdu formulara. Atribut `value` određuje labelu odnosno tekst ispisan na dugmetu.

```
<input name="btnPotvrdi" type="submit" value="Potvrdi">
```

Potvrdi

`Reset` dugme vrši za brisanje unetih podataka u formular. Ovo se najčešće koristi u toku popunjavanja formulara ako dođe do greške u popunjavanju.

```
<input name="btnObrisi" type="reset" value="Obrisi">
```

Obriši

Dugme tipa `button` se koristi za aktiviranje određenog skripta, npr. zatvaranje aktivnog prozora, štampanje stranice i sl.

```
<input type="button" name="btnStampa" value="Odštampaj stranu">
```

### 14. Odštampaj stranu

### 15. Dugme za potvrdu sa slikom

Umesto klasičnog Windows dugmeta za potvrdu može se dodati slika koja će se koristiti kao dugme za potvrdu. Tip `input` polja je `image`, a kao naziv se mora definisati `submit`. Datoteka sa slikom se navodi u okviru `src` atributa.

```
<input name="Submit" type="image" src="Dugme.gif" >
```

CLICK HERE



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 16. Skrivena polja

Nekada je potrebno prikupiti podatke sa Web strane, pri čemu ti podaci ne moraju biti vidljivi na strani. Tada se koriste skrivena polja koja su takođe deo formulara i čije vrednosti se prosleđuje zajedno sa vrednostima ostalih polja prilikom potvrđivanja formulara.

Skriveno polje se umeće na stranu na sledeći način:

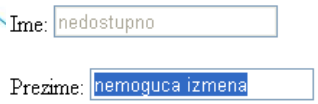
```
<input type="hidden" name="hiddPolje" value="...">
```

Vrednosti koje se prikupljaju u skrivenim poljima obično su vezane za sistemske informacije, informacije o Web strani, korisniku i sl.

### 17. Dodatni atributi input taga

Većina polja može imati atribute koji omogućavaju zabranu promene sadržaja nekog polja . To je atribut `readonly="true"`. Ovo se koristi ako želimo da neka vrednost bude prikazana na ekranu u textbox-u, recimo današnji datum, ali da ga korisnik ne može menjati

Takođe, pomoću atributa `disabled`, onemogućava se pristup nekom polju. Slično je kao kod atributa `readonly`, ali korisnik čak ne može ni pristupiti tom polju.



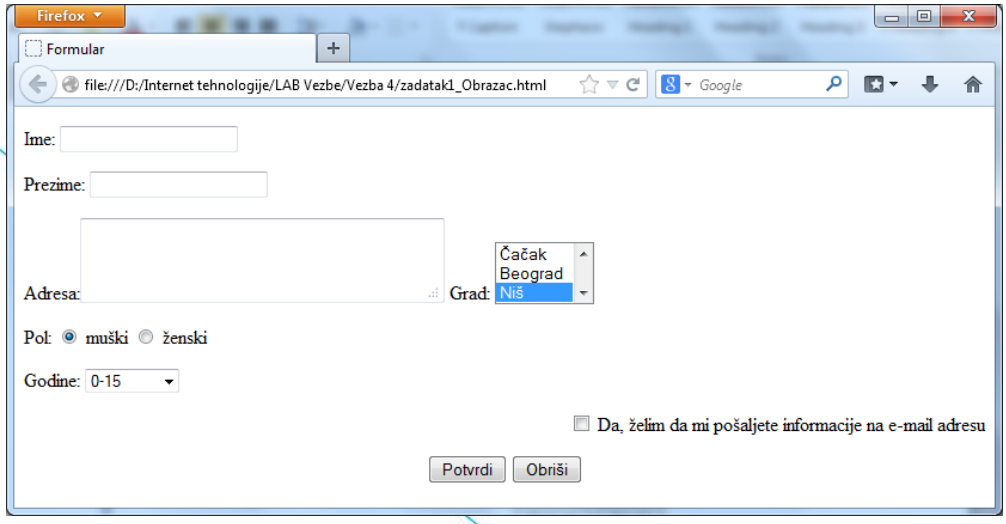



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

## ZADACI ZA VEŽBU

***Zadatak 1:***

Kreirati formular koji služi za prikupljanje osnovnih podataka sa elementima koji su prikazani na sledećoj slici. Pratiti korake koji slede.




1. Prvo napisati neophodne HTML tagove.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Formular</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
...
</body>
</html>
```

2. Dodati oznaku za obrazac `form`:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Formular</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
</head>
<body>
   <form name="frmPodaci" method="get" action="">
   ...
   </form>
</body>
</html>
```

3. Sledeće korake uraditi upisivanjem navedenih oznaka između početne i završne oznake `form`. U paragraf uneti tekstualnu labelu `Ime` i `textbox` kontrolu obrasca za unos imena:

```
<form name="frmPodaci" method="get" action="">
  <p>Ime: <input type="text" name="txtIme"></p>
  ...
</form>
```

4. U narednom paragrafu uneti `textbox` kontrolu obrasca za unos prezimena:

```
<p> Prezime: <input type="text" name="txtPrezime"></p>
```

Rezutat koraka 3. i 4. je prikazan na sledećoj slici.

Ime:

Prezime:

5. Postaviti tekstualnu oblast za unos adrese sa `3` reda i `35` karaktera širine:

```
<p>Adresa:
<textarea name="txtAdresa" cols="35" rows="3"></textarea>
```

Adresa:

6. U istom paragrafu dodajte listu za izbor grada, inicijalno je izabran `Niš`. Atributom `size` određuje se da je broj vidljivih stavki u listi `3`.

```
  Grad:
    <select name="selGrad" size="3" id="selGrad">
      <option>Čačak</option>
      <option>Beograd</option>
      <option selected>Niš</option>
      <option>Novi Sad</option>
    </select>
  </p>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

ICT CLUSTER ACADEMY

TVOJA KARIJERA U TVOJIM RUKAMA

Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA

Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
    <title>Zadatak 2 - studentski obrazac</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>

<body>
<form name="form1" method="get" action="">

  <p>Ime: <input name="txtIme" type="text" > </p>
  <p>Prezime: <input name="txtPrezime" type="text" > </p>

  <p>Adresa:
  <textarea name="txtAdresa" cols="35" rows="3">
  </textarea>

    Grad:
    <select name="selGrad" size="3" >
      <option selected>Čačak</option>
      <option>Beograd</option>
      <option>Niš</option>
      <option>Novi Sad</option>
    </select>
  </p>

  <p>Pol:
  <input name="rbPol" type="radio" value="M" checked> muški
  <input type="radio" name="rbPol" value="Z"> ženski
  </p>

  <p>Godine:
    <select name="selGodine">
      <option >0-15</option>
      <option>16-30</option>
      <option>31-45</option>
      <option>preko 45</option>
    </select>
  </p>

  <p align="right">
  <input name="chToMail" type="checkbox"/>
    Da, želim da mi pošaljete informacije na e-mail adresu
  </p>
```



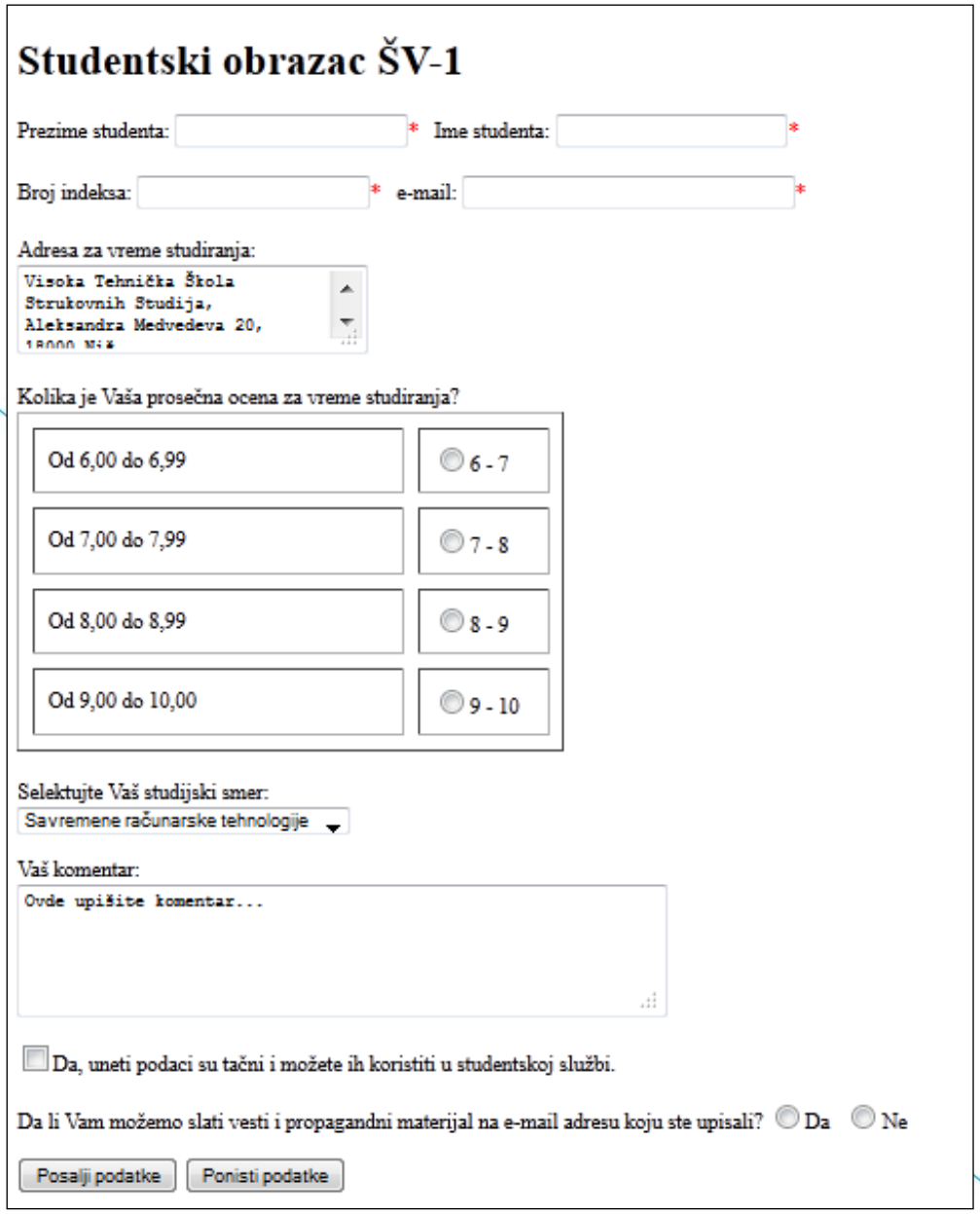
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
  <p align="center">
    <input name="btnPotvrdi" type="submit" value="Potvrdi">
    <input name="btnObrisi" type="reset" value="Obriši">
  </p>

</form>
</body>
</html>
```

**Zadatak 2:** Formirati obrazac za prikupljanje studentskih podataka koji je prikazan na sledećoj slici.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




HTML kod za formiranje obrasca dat je sledećim kodom. Ovaj kod zapamtite u fajl sa ekstenzijom .htm ili .html.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Zadatak 2 - Studentski obrazac</title>
      <meta http-equiv="Content-Type" content="text/html;
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
    charset=utf-8" />
</head>
<body>
  <h1>Studentski obrazac ŠV-1</h1>
  <form name="MojObrazac"
  title="Ovo je tekst koji se ispisuje kada se zadrzi
  pokazivač miša na obrascu"
  onsubmit="return ValidateForm(MojObrazac)"
  action="http://www.htmlcodetutorial.com/cgi-bin/mycgi.pl"
  method="POST"
  enctype="application/x-www-form-urlencoded">

  Prezime studenta:
  <input type="text" name="LastName" id="LN" size="20">
  <font color="red">*</font>   

  Ime studenta:
  <input type="text" name="FirstName" id="FN" size="20">
  <font color="red">*</font><br/><br/>

  Broj indeksa:
  <input type="text" name="IndeksNo" id="IN" size="20">
  <font color="red">*</font>  

  e-mail:
  <input type="text" name="Email" id="EM" size="30">
  <font color="red">*</font><br/><br/>

  Adresa za vreme studiranja:<br/>
  <textarea name="PoljeAdresa" rows="2" cols="30">
  Visoka Tehnička Škola Strukovnih Studija,
  Aleksandra Medvedeva 20, 18000 Niš
  </textarea>

  <br/><br/>

  Kolika je Vaša prosečna ocena za vreme studiranja?
  <table border="1" cellpadding="10" cellspacing="10"
  width="411">

  <tr>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
    <td width="244">Od 6,00 do 6,99</td>
    <td width="84"><input type="radio" name="Mark"
    value="6-7" />6 - 7</td>
  </tr>

  <tr>
  <td width="244">Od 7,00 do 7,99</td>
  <td width="84"><input type="radio" name="Mark"
  value="7-8" />7 - 8</td>
  </tr>

  <tr>
  <td width="244">Od 8,00 do 8,99</td>
  <td width="84"><input type="radio" name="Mark"
  value="8-9" />8 - 9</td>
  </tr>

  <tr>
  <td width="244">Od 9,00 do 10,00</td>
  <td width="64"><input type="radio" name="Mark"
  value="9-10" />9 - 10</td>
  </tr>
  </table>
<br/>
Selektujte Vaš studijski smer:<br/>
<select size="1" name="Group">
  <option value="SRT">Savremene računarske tehnologije
  </option>
  <option value="KOT">Komunikacione tehnologije</option>
  <option value="INI">Industrijsko inženjerstvo</option>
  <option value="DRS">Drumski saobraćaj</option>
  <option value="GIN">Građevinsko inženjerstvo</option>
</select>

<p>Vaš komentar:<br/>
  <textarea name="Comments" rows="4" cols="60">
  Ovde upišite komentar...</textarea>
</p>

<input type="checkbox" name="Yes/No checkbox" value="DA" unchecked>
Da, uneti podaci su tačni i možete ih koristiti u studentskoj službi.<br/>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<p>Da li Vam možemo slati vesti i propagandni materijal na e-mail adresu
koju ste upisali?
<input type="radio" value="DA" name="Contact_Ok" />Da 
<input type="radio" value="NE" name="Contact_Ok"  />Ne
</p>
<p>
<input type="submit" value="Posalji podatke"/>
<input type="reset" value="Ponisti podatke"/>
</p>
</form>
</body>
</html>
```

**Pitanja i zadaci:**

1. Šta je HTML forma?

2. Gde se najčešće vrši obrada podataka prikupljenih u Web formi, i u kom obliku se podaci prosleđuje?

3. Koji su atributi `form` taga?

4. Koji tagovi se koriste za dodavanje tekstualnih polja u formama?

5. Koje od ovih obrazaca bi trebalo slati pomoću metode GET, a koje pomoću metode POST:

a) Obrazac za pristup bankovnom računu preko Interneta _____

b) Obrazac za slanje dizajna majica štamparu _____

c) Obrazac za pretraživanje arhive novinskih članaka _____

d) Obrazac za prikupljanje dužih tekstova _____



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

6. Koji element obrasca je najbolji za sledeće poslove? Tamo gde je odgovor `input`, obavezno navedite i vrednost atributa `type`. Za neke poslove postoji i više tačnih odgovora.

a) Izaberite svoj horoskopski znak od 12 ponuđenih.

b) Bolujete li od srčanih tegoba (da ili ne).

c) Napišite kritiku knjige.

d) Izaberite omiljene ukuse sa liste od 8 ponuđenih.

7. Svaki od navedenih primera koda sadrži grešku. Pronađite je.

a) Pogrešno napisan kod:

```
<input name=”zemlja” value=”Ovde navedite zemlju.”/>
```

Tačno napisan kod:

b) Pogrešno napisan kod:

```
<checkbox name=“boja“ value=“plava“/>
```

Tačno napisan kod:

c) Pogrešno napisan kod:

```
<select name=“lilihip“>
      <option value=“narandža“/>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
      <option value="grejpfrut"/>

      <option value="višnja"/>

</select>
```

Tačno napisan kod:

d) Pogrešno napisan kod:

```
<input type="lozinka"/>
```

Tačno napisan kod:

e) Pogrešno napisan kod:

```
<textarea name="tekst" height="6" width="100"> Vaša priča. </textarea>
```

Tačno napisan kod:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# Vežba br. 5 – HTML5

**CILJ VEŽBE**

- Upoznati se sa HTML5 osnovama i definicijom HTML5 dokumenta.
- Upoznati se sa HTML5 oznakama u obrascima.
- Savladati dodavanje multimedijalnog sadržaja HTML5 oznakama.
- Atributi za kontrolu multimedijalnog sadržaja.

**TEORIJSKE OSNOVE ZA IZRADU VEŽBE**

HTML5 predstavlja verziju script jezika kojom je HTML dobio na funkcionalnosti. Prethodne verzije koristile su pomoć drugih skript jezika, što je predstavljalo glavni nedostatak. Upotrebom HTML5 verzije i sam CSS za kontrolisanje izgleda Web strane je mnogo efikasniji, jer je HTML5 fokusiran samo na strukturu, a ne na izgled strane.

HTML je dizajniran tako da pruži sve što je potrebno za rad prilikom izrade Web strana, bez potrebe za dodatnim plugin-ovima. Napredne mogućnosti animiranja aplikacija, pokretanja muzičkih i video sadržaja, čine da HTML5 postane platforma za različite uređaje (računare, tablet računare, laptopove, pametne telefone i televizore). Takođe, može se koristiti i za kreiranje Web aplikacija koje rade i u „offline" režimu. Na njegovom razvoju radila je velika grupa kompanija: AOL, Apple, Google, IBM, Microsoft, Mozilla, Nokia, Opera i mnogi drugi. Zapravo predstavlja proizvod saradnje konzorcijuma W3C, (World Wide Web Consortium) i organizacije WHATWG (Web Hypertext Application Technology Working Group). WHATWG je radio na unapređenju Web formi i aplikacija, dok je W3C razvijao XHTML 2.0. Na početku razvoja HTML5 standarda postavljeno je nekoliko ključnih pravila:

- Nove funkcionalnosti moraju biti zasnovane na HTML, CSS, DOM i JavaScript tehnologijama
- Potreba za eksternim plugin-ovima (kao što je Flash) mora da bude redukovana
- Otklanjanje grešaka mora da bude lakše nego u prethodnim verzijama
- HTML5 mora da bude nezavisan od uređaja na kome se primenjuje
- Proces razvoja mora da bude javno dostupan

HTML5 ima veoma jednostavnu `<!doctype>` deklaraciju:

```
<!DOCTYPE html>
```

Na osnovu toga minimalna struktura jednog HTML5 dokumenta je prikazana u nastavku:



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<!DOCTYPE                                                                    html>
<html>
<head>
<meta                                                                charset="UTF-8">
<title>Naslov                                                       dokumenta</title>
</head>
<body>
   Sadržaj                                                                dokumenta…
</body>
</html>
```

Svi vodeći Web čitači (Safari, Chrome, Firefox, Opera, Internet Explorer) podržavaju HTML5 I nastoje stalno da dodaju nove tagove u svoje poslednje verzije.

**HTML5 U OBRASCIMA**

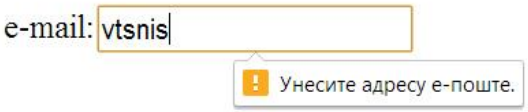
Nova polja obrazaca bila su početak specifikacije koja je postala HTML5. Proširenja se dobrim delom sastoje od novih vrednosti atributa `type` elementa `input`. Prethodna verzija, HTML4 nalaže da ako se ne zada vrednost atributa type ili se navede nepoznati tip, čitači podrazumevaju vrednost `<input type="text">`. Zato će zastareli Web čitači koji ne razumeju nova proširenja usvojiti podrazumevane parametre i omogućiti korisniku da unosi podatke u polje za tekst.

Jedan od problema sa HTML4 obrascima jeste što su oni samo prosta polja. U Vežbi 6 i 7 je prikazan način na koji se pomoću JavaScript-a vrši provera polja obrasca. HTML5 obrasci omogućavaju lakši način provere valjanosti unetih podataka za neke od najznačajnijih tipova podataka, čime je značajno olakšan posao provere.

U nastavku je dat pregled nekih od novih ulaznih tipova za obrasce kao i novih atributa.

**Ulazni tip email**

Pomoću oznake `<input type="email">` Web čitaču se saopštava da ne bi smeo da dozvoli da se obrazac pošalje ako korisnik nije upisao nešto što liči na valjanu elektronsku adresu. Kao sa svim ulaznim tipovima, korisnik može da pošalje obrazac s praznim poljem, sem ako postoji atribut `required`.






Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Atribut `multiple` ukazuje na to da vrednost polja može biti lista ispravnih elektronskih adresa razdvojenih zarezom.

**Ulazni tip date**

Često je neophodno da se u obrascu unese datum, ali to može biti i problematično zbog postojanja različitih formata (DD-MM-YYYY, MM-DD-YYYY, YYYY-MM-DD,…). Mnogi programeri pišu na JavaScript-u birače datuma koji se razlikuju po izgledu, upotrebljivosti i pristupačnosti od jedne strane do druge. Pomoću oznake `<input type="date">` nalažemo Web čitaču da prikaže kontrolu za izbor datuma.

**Ulazni tip time**

Pomoću oznake `<input type="time">` može se korisniku omogućiti unos vremena u 24-časovnom formatu i provera valjanosti upisanog podatka. I uvom slučaju zavisi od čitača kako će kontrola izgledati.

**Ulazni tip month**

Oznaka `<input type="month">` omogućava unos i proveru ispravnosti unosa meseca. Iako se vrednost meseca interno čuva kao broj iz opsega 1-12, čitač sada može da ponudi metodu izbora po imenima meseci i to lokalizovano (za francusku lokalizaciju piše Janvier umesto Januar).

**Ulazni tip number**



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Oznaka `<input type="number">` omogućava proveru valjanosti numeričkog unosa. Nije predviđena za brojeve telefona jer se oni često pišu sa razmacima, zagradama, znakom plus.

Pomoću atributa `min` i `max` vrši se ograničavanje vrednosti koje se mogu uneti i ne može se poslati obrazac ako je u njega unet broj koji nije iz definisanog opsega.

```
<input type="number" mini="2" max="10">
```

Atribut upravlja nivoom granularnosti unosa. Dakle, ako želite da korisnik unosi procenat u opsegu od 0 do 100 sa korakom 5 onda treba napisati sledeće:

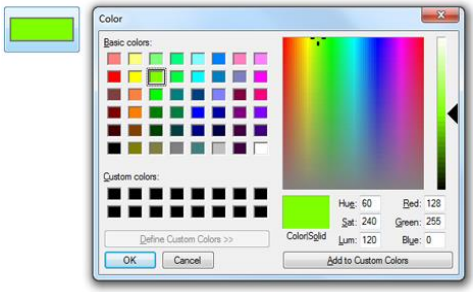
```
<input type="number" mini="0" max="100" step="5">
```

**Ulazni tip range**

Pomoću oznake `<input type="range">` prikazuje se klizač. Pre pojave nog taga u HTML5 klizaču su mogli da se formiraju pomoću JavaScript-a. Pošto takvi klizači nisu bili prirodni deo čitača, programeri su morali da budu veoma obazrivi i da se dodatno trude oko koda kako bi obezbedili pristupačnost sa tastature.

**Ulazni tip color**

Tip `<input type="color">` omogućava da korisnik unese vrednost boje pomoću birača boje (engl. *Color picker*).




**HTML5 VIDEO I AUDIO**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

HTML5 nudi otvoren standard za prenošenje multimedijalnih sadržaja na Webu sa svojim izvornim video i audio elementima i interfejsima za programiranje aplikacija.

Najjednostavniji način dodavanja video sadržaja HTML5 stranici jeste pomoću narednog segmenta koda:

```
<video src="film.webm"></video>
```

Oznaka tipa (ekstenzija) `.webm` označava da je video u WebM formatu.

Dodavanje audio sadržaja pomoću HTML5 sintakse može se izvršiti na sledeći način:

```
<audio src="zvuk.mp3"></audio>
```

Oznaka tipa (ekstenzija) `.mp3` označava da je video u mp3 formatu.

Ipak prethodni primer za video prikazuje na ekranu samo prvi kadar filma. Razlog je to što nije naglašeno da se video sadržaj reprodukuje, niti su obezbeđene komande za reprodukovanje ili pauziranje sadržaja.

Kao i u slučaju elementa `img` atribut `src` ukazuje na audio ili video sadržaj koji će čitač da reprodukuje pod uslovom da podržava dati format kodeka. S obzirom da ne podržavaju svi Web čitači iste formate, da bi obezbedili da multimedijalni sadržaj bude vidljiv većini korisnika bez obzira na to koji čitač koriste, potrebno je obezbediti više izvornih datoteka različitih formata. U Tabeli 10.1 dat je pregled podrške za video kodeke među modernim čitačima.

| | **WEBM**<br>**VP8 kodek** | **MP4**<br>**H.264 kodek** | **OGV**<br>**OGG THEORA kodek** |
|---|---|---|---|
| **Opera** | DA | NE | DA |
| **Firefox** | DA | NE | DA |
| **Chrome** | DA | DA | DA |
| **IE9+** | DA (ali kodek se mora ručno instalirati) | DA | NE |
| **Safari** | NE | DA | NE |

***Tabela 1.*** Podrška za video kodeke među modernim čitačima

U Tabeli 10.2 dat je pregled podrške za audio kodeke među modernim čitačima.

| | **.OGG/.OGV**<br>**VORBIS kodek** | **MP3** | **MP4/M4A**<br>**AAC kodek** | **WAV** |
|---|---|---|---|---|
| **Opera** | DA | NE | NE | DA |



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

| | | | | |
|---|---|---|---|---|
| **Firefox** | DA | NE | NE | DA |
| **Chrome** | DA | DA | DA | DA |
| **IE9+** | NE | DA | DA | NE |
| **Safari** | NE | DA | DA | DA |

***Tabela 2.*** Podrška za audio kodeke među modernim čitačima

Na osnovu tabela može se zaključiti da ako se obezbede i WebM i H.264 video kao i Vorbis i MP3 verzija audio sadržaja, on će biti dostupan u svim modernim Web čitačima.

U tom slučaju potrebno je navesti dva (ili više) atributa `src` na sledeći način:

```
<video controls>
<source src="film.webm" type=’"video/webm"; codecs="vp8, vorbis"’>
<source src="film.mp4" type=’"video/mp4";
codecs="avc1.42E01E, mp4a.40.2"’>
</video>
```

U nastavku će biti dati atributi za kontrolu video i audio sadržaja.

### Atribut autoplay

Atribut `autoplay` nalaže Web čitaču da video ili audio sadržaj reprodukuje čim se Web stranica učita.

```
<video src="film.webm" autoplay></video>
<audio src="zvuk.mp3" autoplay></audio>
```

To baš i nije preporučljivo jer ćo to iritirati korisnike mobilnih uređaja jer verovatno ne žele da upotrebljavate njihov mrežni protok a da prethodno nisu eksplicitno zatražili da pogledaju video.

### Atribut controls

Umesto automatskog pokretanja mnogo je bolje da se korisniku ponude kontrole, što se može uraditi jednostavno dodavanjem atributa `controls`.

```
<video src="film.webm" controls></video>
<audio src="zvuk.mp3" controls></audio>
```

Naravno, kontrole se razlikuju od zavisnosti od čitača jer specifikacija ne preporučuje kako bi trebalo da izgledaju niti šta da rade. I pored toga većina čitača se pridržava opštih normi za kontrole: postoji



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

mogućnost za pokretanje i pauziranje reprodukcije, traka za pretraživanje (engl. *Seek bar*) i kontrola jačine zvuka.

Ako element `audio` nema atribut `controls` neće se videti ništa vizuelno na stranici ali će se zvuk reprodukovati. On se nalazi u DOM-u i može se potpuno kontrolisati pomoću JavaScript-a i novih interfejsa za programiranje aplikacija.

**Atribut poster**

Atribut `poster` ukazuje na sliku koju će čitač koristiti dok se video sadržaj preuzima ili dok korisnik ne naloži pokretanje reprodukovanja sadržaja.

```
<video src="film.webm" poster="slika.jpg"></video>
```

On otklanja potrebu za dodatnim trikovima poput prikazivanja slike, a potom i njenog uklanjanja nakon pokretanja videa, pomoću JavaScript-a. Ako se ne navede atribut `poster`, čitač će prikazati prvi kadar filma. Ovaj atribut nije primenljiv na element audio.

**Atribut muted**

Atribut `muted` omogućava da multimedijalni element podrazumevano bude potpuno utišan, pa korisnik mora da pojača zvuk. U sledećem primeru se video sadržaj automatski reprodukuje ali je ton sasvim utišan:

```
<video src="film.webm" controls autoplay muted></video>
```

**Atributi height, width**

Atributi `height` i `width` saopštavaju čitaču veličinu video sadržaja u pikselima. Ne mogu se pridružiti elementu audio. Ako se izostave čitač koristi izvornu širinu video sadržaja, odnosno izvornu širinu slike koje je postavljena u atributu poster. Ako ništa od toga nije moguće čitač koristi podrazumevanu vrednost za širinu od 300px.

```
<video src="film.webm" controls width="480" height="270"></video>
```

**Atribut loop**

Atribut `loop` je još jedan logički atribut. Njime se ostvaruje petlja reprodukcije multimedijalnog sadržaja. To znači da po završetku reprodukcije počinje ponovna reprodukcija istog sadržaja.

```
<video src="film.webm" autoplay loop></video>
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<audio src="zvuk.mp3" autoplay loop></audio>
```

**Atribut preload**

Postoji mogućnost da se audio ili video sadržaj unapred učita tako da počne da se baferuje tokom učitavanja stranice.

```
<video src="film.webm" controls preload></video>
```

Specifikacija predviđa tri moguće vrednosti atributa preload.

`preload="auto"`

Ovo je sugestija čitaču da bi trebalo da počne da preuzima celu datoteku

`preload="none"`

Sugeriše čitaču da ne učitava sadržaj dok korisnik ne aktivira kontrole

`preload="metadat"`

Sugeriše čitaču da preuzme samo dopunske metapodatke (dimenzije, prvi kadar, listu zapisa, trajanje,…)



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**ZADACI ZA VEŽBU**

***Zadatak 1:*** Kreirati obrazac prikazan na sledećoj slici pomoću odgovarajućih HTML5 oznaka. Tekstualna polja su obavezna. U polju za unos e-maila vrši se provera postojanja znaka @. U polju za unos broja može se uneti vrednost od 0 do 10 sa korakom 2. Kontrola za izbor opsega je klizač. Ako je polje pravilno popunjeno pojavljuje se znak ✔. Sve dok je polje nepravilno popunjeno prikazuje se znak ✖. Kod praznog obrasca svako tekstualno polje je obeleženo znakom ✱.

Primer HTML5 koda za realizaciju Web stranice sa obrascem dat je u nastavku.

```
<!DOCTYPE html>

<html>

<head>
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
</head>
<body>
<form action="" method="post">
            <label for="name">Ime:</label>
            <input type="text" name="name" required placeholder="Ime" />
            <label for="email">e-mail:</label>
            <input type="email" name="email" required
            placeholder="email@primer.com" />
            <label for="website">Website:</label>
            <input type="url" name="website" required
            placeholder="http://www.primer.com" />
            <label for="number">Broj:</label>
            <input type="number" name="number" min="0" max="10" step="2"
            required placeholder="Broj < 10">
            <label for="range">Opseg:</label>
            <input type="range" name="range" min="0" max="10" step="2" />
            <label for="message">Poruka:</label>
            <textarea name="message" required></textarea>
            <input type="submit" value="Posalji" />
</form>
</body>
</html>
```

Za podešavanje izgleda obrasca dodati sledeći CSS:

```
body {font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;}
form {
width:300px;
margin: 20px auto;
}

input, textarea {
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
background-color:#fff;
border:1px solid #ccc;
font-size:20px;
width:300px;
min-height:30px;
display:block;
margin-bottom:16px;
margin-top:8px;
padding-left:8px;
border-radius:5px;
transition: all 0.5s ease-in-out;
}
textarea {
min-height:200px;
}
input:focus, textarea:focus {
box-shadow:0 0 25px #ccc;
transform: scale(1.05);
}
input:not(:focus), textarea:not(:focus) {
opacity:0.5;
}
input:required, textarea:required {
background:url("asterisk_orange.png") no-repeat 280px 7px;
}
input:valid, textarea:valid {
background:url("tick.png") no-repeat 280px 5px;
}
input:focus:invalid, textarea:focus:invalid {
background:url("cancel.png") no-repeat 280px 7px;
```

```
}
input[type=submit] {
padding:10px;
background:none;
opacity:1.0;
}
```

***Zadatak 2:*** Kreirati pomoću odgovarajućih HTML5 oznaka Web stranu na kojoj postoji mogućnost kontrole video i audio sadržaja bez automatskog pokretanja.

Primer HTML5 koda za realizaciju Web stranice iz zadatka 2 data je u nastavku:

```
<!DOCTYPE html>
<html>
<head>
    <title>Video i audio</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
      <video controls="controls">
        <source src="film.mp4"/>
        <source src="film.webm"/>
      </video>
      <br/>

      <audio controls="controls">
        <source src="music.mp3"/>
        <source src="music.ogg"/>
      </audio>
</body>
</html>
```

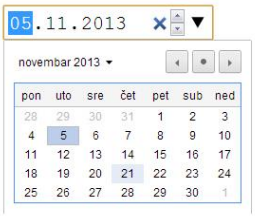


Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**Pitanja i zadaci:**

1. Napisati deklaraciju HTML5 dokumenta.

2. Napisati HTML5 oznake za dodavanje elemenata obrasca koji su na slikama.



3. Napisati HTML5 oznaku za dodavanje polja koje ima mogućnost provere valjanosti numeričkog unosa. Odgovarajućim atributima podesiti da polje omogućava unos u opsegu od 0 do 20 sa korakom 4.

Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

5. Objasnite svaki red koda koji je napisan u nastavku.

```
<video controls>
<source src="film.webm" type=’"video/webm"; codecs="vp8, vorbis"’>
<source src="film.mp4" type=’"video/mp4"; codecs="avc1.42E01E, mp4a.40.2"’>
</video>
```

6. U kojem Web čitaču će se sigurno reprodukovati sadržaj koji je naveden u sledećem kodu?

```
<video src="film.mp4"></video>
```

_____

```
<audio src="zvuk.ogg" autoplay></audio>
```

_____

7. Napisati oznaku za reprodukciju klikom na odgovarajuću kontrolu video sadržaja `MojFilm.mp4`. Podesiti da se pre pokretanja reprodukcije umesto prvog kadra vidi slika `Start.jpg`.



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# Vežba br. 6 – CSS (Kaskadne Liste Stilova)

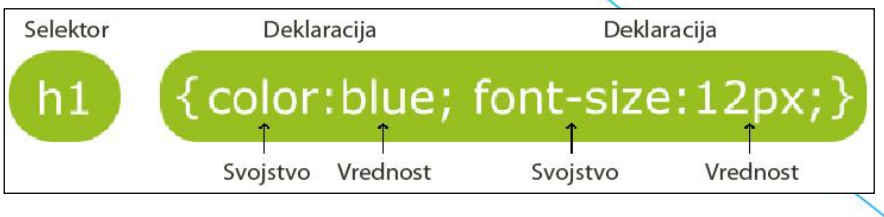
## CILJ VEŽBE

- Upoznati se sa načinima dodavanja CSS pravila HTML strani.
- Razumevanje osnovnih osobina i vrednosti.
- Pisanje CSS dokumenata.
- Kreiranje stranica sa definisanim CSS-om.

## TEORIJSKE OSNOVE ZA IZRADU VEŽBE

**CSS** (*Cascading Style Sheet*) predstavlja skup pravila formatiranja kojima se kontroliše izgled sadržaja Web strane. Osim osnovnog podešavanja teksta (font, veličina, stil) moguće je jednobrazno podesiti margine, okvire, način preloma teksta i objekata za sve strane u prezentaciji. Pojam "kaskadne" odnosi se na mogućnost primene više stilova na istoj strani.

CSS pravilo formatiranja sastoji se iz dva dela: **selektora** i **deklaracije**. Selektor je naziv stila ili taga (kao što je `td, p, h1`,...) a deklaracija opisuje stil. Sama deklaracija se takođe sastoji iz dva dela: svojstva (property) i vrednosti (value) (kao što je font-size i vrednost 12px).




Najveća prednost upotrebe CSS-a je u jednostavnosti ažuriranja, jer se ažuriranjem CSS stila, automatski ažurira izgled svih strana na koje se taj stil primenjuju.

Primena CSS stila se vrši unutar `head` elementa Web strane i to na jedan od sledeća tri načina:

1) Navođenjem selektora i deklaracija u samoj html strani (najčešće u okviru `head` taga):

```
<style type="text/css">
<!--
body {
      font-family: Arial, Helvetica, sans-serif;
      font-size: 14px;
      font-style: italic;
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
        background-color: #FF0000;
        text-align: justify;
}
option {
        color: #0000CC;
}
-->
</style>
```

U ovom primeru `body` je selektor, a deklaracioni parovi atribut-vrednost su navedeni unutar vitičastih zagrada. Navedeni stil se odnosi na `body` element, definišu se font, veličina i stil slova. Zatim se postavlja pozadinska boja na crvenu (`#FF0000`), i osnovno poravnanje teksta na obostrano poravnato (`justify`).

2) Navođenjem veze sa posebnom CSS datotekom u kojoj su sadržani selektori i deklaracije:

```
<link rel="stylesheet" type="text/css" href="stil.css">
```

Datoteka sa stilom `stil.css` sadrži selektore i deklaracije i upotreba ovih "eksternih" šema stilova je mnogo bolja, jer se može unutar više strana koristiti isti stil.

Ukoliko se na strani pojedini elementi dodatno formatiraju to ručno formatiranje će imati prednost u odnosu na ono koje je definisano u listi stilova. Na primer, ako je paragraf stilom podešen da bude uvek poravnat obostrano, a mi u nekom paragrafu dodamo atribut da je poravnat levo, taj paragraf će imati levo poravnanje.

3) Navođenjem pravila u okviru same html oznake za koju ta pravila važe korišćenjem atributa `style`. Primer pokazuje kako se može izmeniti boja teksta u okviru pasusa i izvršiti levo poravnjanje teksta.

```
<p style="color:sienna;margin-left:20px">
This is a paragraph.
</p>
```

Lista svih CSS svojstava podeljenih u grupe, njihovo objašnjenje kao i podatak u kojoj verziji CSS-a je prvi put definisano svojstvo dati su u Dodatku C na kraju praktikuma.

**DODAVANJE CSS PRAVILA POMOĆU KLASE I ID**



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Za dodavanje CSS stila određenom HTML elementu, mogu se koristiti dve vrste selektora `id` i `class`.

Selektor `id` se koristi za dodavanje stila jednom jedinstvenom element. On koristi vrednost `id` atributa HTML elementa i definiše se pomoću znaka #. Stil definisan u nastavku se primenjuje na element koji ima atribut `id="para1"`.

```
#para1{text-align:center; color:red;}
```

Selektor class se koristi za određivanje stila za grupu elemenata. To omogućava da se postavi zajednički stil za vise elemenata na jednom mestu. Selektor `class` koristi HTML atribut `class` i definiše se znakom tačka (.). U primeru koji sledi, svi HTML elementi se atributom `class="center"` će biti centrirani.

```
.center {text-align:center;}
```

Takođe, može se specificirati da se stil primenjuje na samo određene HTML elemente. U sledećem primeru, svi `p` elementi sa atributom `class="center"` će biti centrirani.

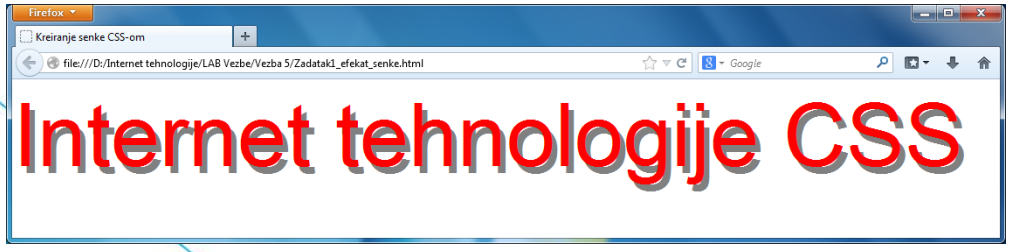
```
p.center {text-align:center;}
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**ZADACI ZA VEŽBU**

**Zadatak 1:** Koristeći pravila CSS-a dodajte senke iza teksta zaglavlja kako bi formirali trodimenzioni efekat. Formirajte jedno pravilo za određivanje mesta i izgled zaglavlja. Drugo pravilo treba da određuje izgled i mesto osenčenog teksta. Izgled Web stranice sa ovako formatiranim zaglavljem dat je na sledećoj slici.




Primer HTML koda sa dodatim CSS-om dat je u nastavku. Definisana su pravila `.banner` i `.shadow`.

```
<html>
<head>
  <title>Kreiranje senke CSS-om</title>
  <style type="text/css">
  <!--
            .banner
            {
            position:absolute;
            top:7px;
            left:1px;
            font-size:100px;
            font-family:"Arial";
            color:red;
            }

            .shadow
            {
            position:absolute;
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
            top:15px;
            left:7px;
            font-size:100px;
            font-family:"Arial";
            color:gray;
            }
  -->
  </style>
</head>
<body bgcolor="white">
  <div class="shadow">Internet tehnologije CSS</div>
  <div class="banner">Internet tehnologije CSS</div>
</body>
</html>
```

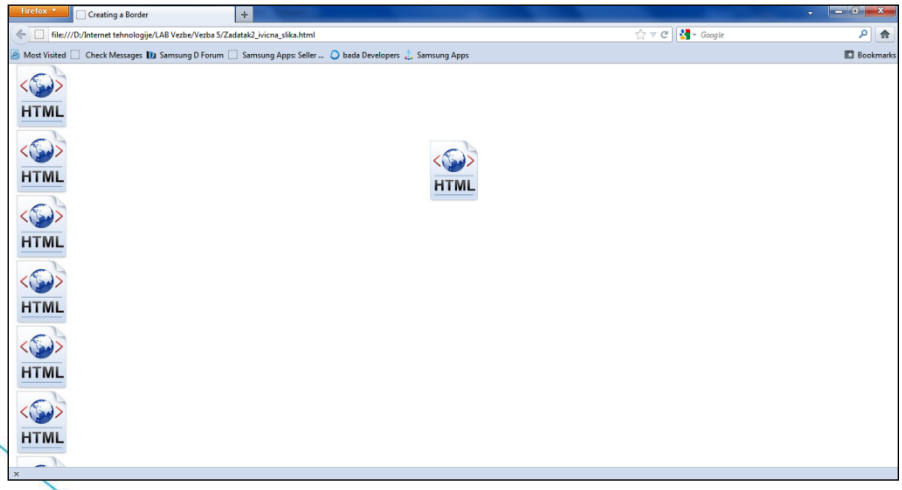
Premestiti CSS definisan u zaglavlju html strane u poseban CSS fajl i povezati novu stranu sa posebnim css fajlom. Koristiti oznaku:

```
<link rel="stylesheet" type="text/css"
href="poseban_fajl.css">
```

**Zadatak 2:** Koristeći pravila CSS-a napravite klasu stila `body` korišćenjem svojstva `background-image`. Ako ne naznačite poziciju smeštanja `.gif` sličice ona se smešta uz levu ivicu Web stranice.



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Ivična slika Web stranice formirana CSS-om*

Primer CSS pravila dat je u nastavku. Sličan kod smestite u fajl sa ekstenzijom .html.

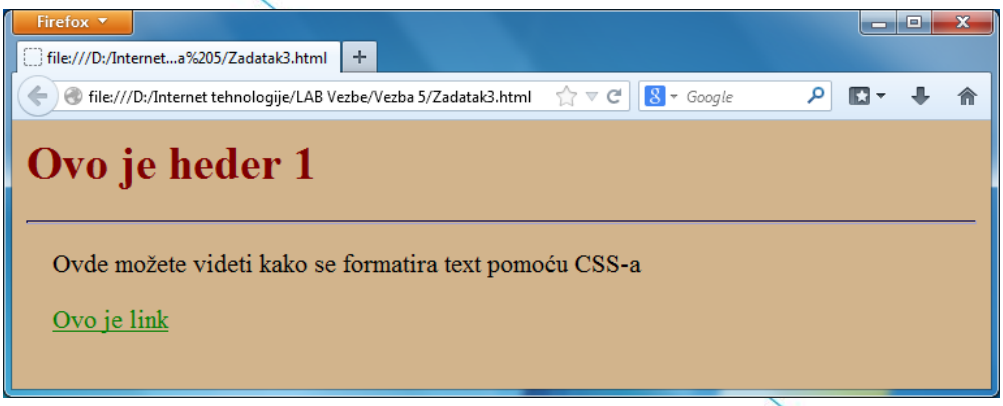
```
<html>
<head>
<title>Creating a Border</title>

<style type="text/css">
<!--
Body
{
  background-image:url('slike/ime_slike.jpg');
  background-repeat:repeat-y
}
-->
</style>

</head>
<body>
      <p> </p>
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
      <p> </p>
      <p> </p>
      <p align="center">
        <img border="0" src="slike/ime_slike.jpg"
        width="50" height="50" />
     </p>
</body>
</html>
```

**Zadatak 3:** Koristeći pravila CSS-a napravite stranicu `Zadatak3.html` koja je uređena spoljašnjim opisom stila `zad.css`. Izgled stranice je dat na sledećoj slici.




**Zadatak3.html**

```
<html>
<head>
   <link rel="stylesheet" type="text/css" href="zad3.css"/>
</head>
<body>
 <h1>Ovo je heder 1</h1>
 <hr/>
 <p>Ovde možete videti kako se formatira text pomoću CSS-a</p>
 <p>
 <a href="http://www.vtsnis.edu.rs" target="_blank">
```



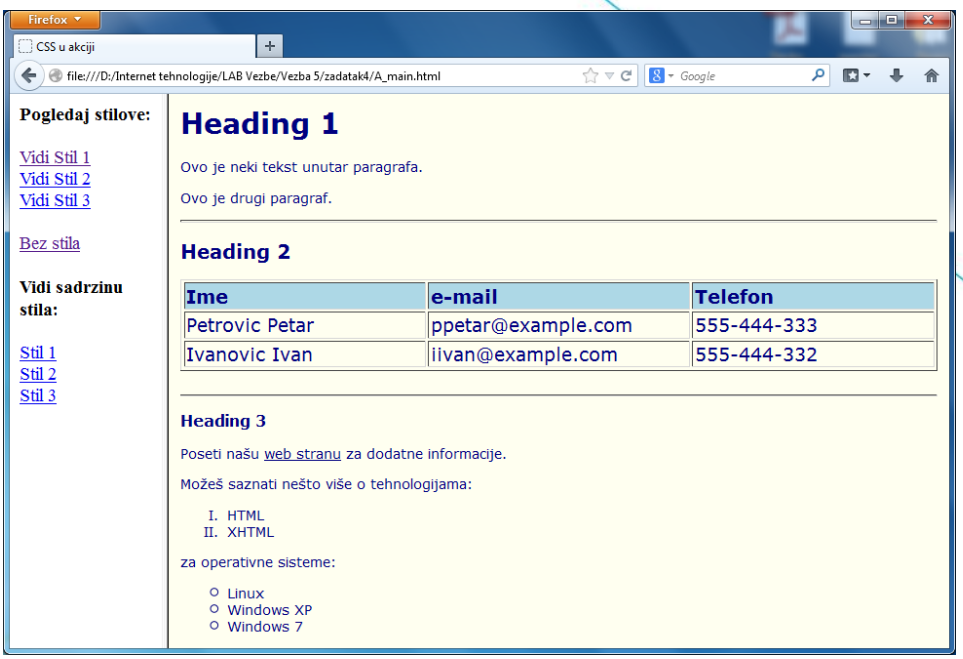
Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
    Ovo je link
 </a>
 </p>
</body>
</html>
```

**zad3.css**

```
body {background-color:tan;}
h1 {color:maroon;font-size:20pt;}
hr {color:navy;}
p {font-size:11pt;margin-left:15px;}
a:link {color:green;}
a:visited {color:yellow;}
a:hover {color:black;}
a:active {color:blue;}
```

**Zadatak 4:** Koristeći pravila CSS-a i frejmova kreirati stranice kao na slici:






Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**A_main.html**

```
<html>
<head>
  <title>CSS u akciji</title>
</head>
  <frameset cols="130,*" frameborder="1">
  <frame name="contents" src="demo_style_content.htm"
  noresize="noresize">
  <frame name="main" src="demo_style_main.htm">
  <noframes>
  <body>
  <p>Ova stranica sadrzi frejmove. Vaš browser ne podrzava
     frejmove.
  </p>
  </body>
  </noframes>
</frameset>
</html>
```

**demo_style_content.html**

```
<html>
<head>
  <meta http-equiv="content-type" content="text/html;
  charset=ISO-8859-1">
</head>
<body>
      <b>Pogledaj stilove:</b>
      <br/>
      <br/>
      <a href="demo_style_main1.htm" target="main">
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
      Vidi Stil 1</a><br/>
      <a href="demo_style_main2.htm" target="main">
      Vidi Stil 2</a><br/>
      <a href="demo_style_main3.htm" target="main">
      Vidi Stil 3</a><br/>
      <br/>
      <a href="demo_style_main.htm" target="main">
      Bez stila</a>
      <br/>
      <br/>
      <b>Vidi sadržinu stila:</b>
      <br/>
      <br/>

      <a href="style1.css" target="main">Stil 1</a><br/>
      <a href="style2.css" target="main">Stil 2</a><br/>
      <a href="style3.css" target="main">Stil 3</a><br/>
</body>
</html>
```

**demo_style_main.html, demo_style_main1.html, demo_style_main2.html, demo_style_main3.html**

```
<html>
<head>
  <link rel="stylesheet" href="style1.css" type="text/css">
</head>
<body>
      <h1>Heading 1</h1>
      <p>Ovo je neki tekst unutar paragrafa.</p>
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<p>Ovo je drugi paragraf.</p>
<hr>

<h2>Heading 2</h2>
<table width="100%" border="1">
<tr>
      <th align="left">Ime</th>
      <th align="left">e-mail</th>
      <th align="left">Telefon</th>
</tr>
<tr>
      <td width="25%">Petrović Petar</td>
      <td width="25%">ppetar@example.com</td>
      <td width="25%">555-444-333</td>
</tr>
<tr>
      <td width="25%">Ivanović Ivan</td>
      <td width="25%">iivan@example.com</td>
      <td width="25%">555-444-332</td>
</tr>
</table>
<br/>
<hr>

<h3>Heading 3</h3>
<p>Posetite našu
<a href="http://www.vtsnis.edu.rs/">
web stranu</a>
za dodatne informacije.
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
        </p>
        <p>Možete saznati nešto više o tehnologijama:</p>
        <ol>
                <li>HTML</li>
                <li>XHTML</li>
        </ol>

        <p>i operativnim sistemima:</p>
        <ul>
                <li>Linux</li>
                <li>Windows XP</li>
                <li>Windows 7</li>
        </ul>
</body>
</html>
```

**style1.css**

```
body
{
font-size:75%;
font-family:verdana,arial,'sans serif';
background-color:#FFFFF0;
color:#000080;
margin:10px;
}

h1 {font-size:200%;}
h2 {font-size:140%;}
h3 {font-size:110%;}
```

```
th {background-color:#ADD8E6;}

ul {list-style:circle;}

ol {list-style:upper-roman;}

a:link {color:#000080;}
a:hover {color:red;}
```

**style2.css**

```
body
{
font-size:75%;
font-family:"lucida calligraphy",arial,'sans serif';
background-color:#DCDCDC;
color:#8A2BE2;
margin:10px;
}

h1 {font-size:200%;}
h2 {font-size:140%;}
h3 {font-size:110%;}

th {background-color:#D3D3D3;}
td {background-color:#FFFAF0;}

a:link {color:#8A2BE2;text-decoration:none;}
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
a:hover {color:red;font-weight:bold;text-decoration:none;}
a:visited {text-decoration:none;}
```

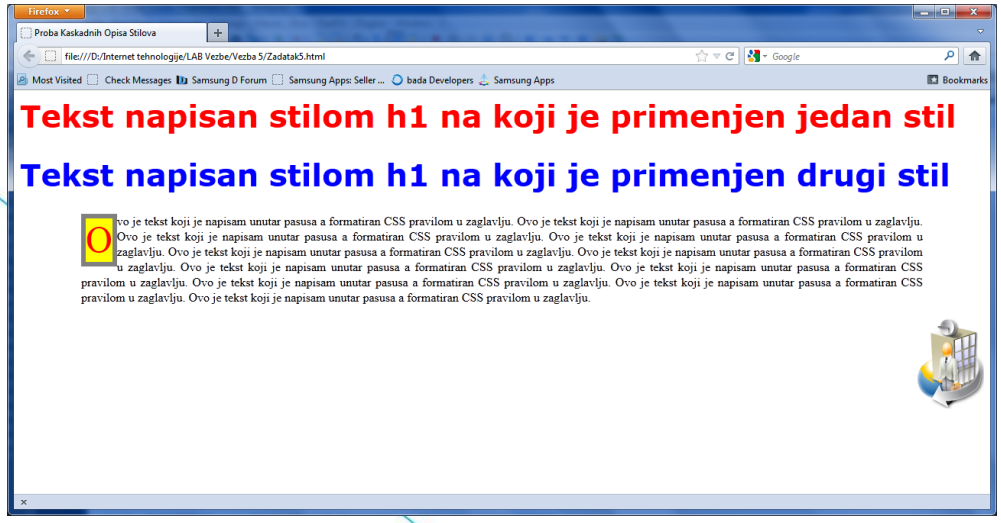
**style3.css**

```
body
{
font-size:75%;
font-family:verdana,arial,'sans serif';
background-image:url('gradient.png');
background-repeat:repeat-x;
background-color:#FFFFF0;
color:#000080;
margin:70px;
}

h1 {font-size:200%;}
h2 {font-size:140%;}
h3 {font-size:110%;}

th {background-color:#ADD8E6;}

ul {list-style:circle;}
ol {list-style:upper-roman;}

a:link {color:#000080;}
a:hover {color:red;}
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**Zadatak 5:** Koristeći pravila CSS-a i svojstvo `float` rasporediti elemente na Web stranici kao što je prikazano na sledećoj slici. Formirajte klasu `.fancyLetter` za ulepšavanje početnog slova. Slovo kome hoćete da date poseban izgled stavite unutar oznake `<span>`.




Raspored elemenata Web stranice pomoću CSS-a

**Zadatak5.html**

```
<html>
<head>
<title>Proba Kaskadnih Opisa Stilova</title>
<link rel="stylesheet" href="zad5.css" type="text/css">
</head>
<body>

<h1 class="prvistil">Tekst napisan stilom h1 na koji je primenjen jedan stil</h1>

<h1 class="drugistil">Tekst napisan stilom h1 na koji je primenjen drugi stil</h1>
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<p><span class="fancyletter">O</span>vo je tekst koji je napisam unutar pasusa a formatiran CSS pravilom u zaglavlju.


Ovo je tekst koji je napisan unutar pasusa a formatiran CSS pravilom u zaglavlju.
Ovo je tekst koji je napisan unutar pasusa a formatiran CSS pravilom u zaglavlju.
Ovo je tekst koji je napisan unutar pasusa a formatiran CSS pravilom u zaglavlju.
Ovo je tekst koji je napisan unutar pasusa a formatiran CSS pravilom u zaglavlju.
Ovo je tekst koji je napisan unutar pasusa a formatiran CSS pravilom u zaglavlju.
Ovo je tekst koji je napisan unutar pasusa a formatiran CSS pravilom u zaglavlju.
Ovo je tekst koji je napisan unutar pasusa a formatiran CSS pravilom u zaglavlju.
</p>


<img src="slike/partneri.png">


</body>
</html>
```

**zad5.css**



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**Pitanja i zadaci**

1. Šta označava termin CSS?

2. Kakvu strukturu ima CSS?

3. Koji su načini definisanja CSS-a?

4. Navesti primer podešavanja stila za ćeliju tabele.

5. Koje će boje biti pasus kada se primeni sledeći ugrađeni opis? Zašto?

```
<style type="text/css">
   p{color:purple;}
   p{color:green;}
   p{color:gray;}
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
</style>
```

6. Kako se određuje CSS preko atributa `class` a kako preko atributa `id`.

7. Šta se definiše CSS svojstvom `font-weight` i koje vrednosti može da ima?



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# Vežba br. 7 – Java Script

## CILJ VEŽBE

- Upoznati se sa načinima dodavanja Java Script koda unutar HTML strane.
- Obrada događaja na HTML strani pomoću Java Script-a.
- Savladavanje jednostavnijih funkcionalnosti Java Script-a.
- Pozivanje JavaScript funkcije koju je definisao korisnik.

## TEORIJSKE OSNOVE ZA IZRADU VEŽBE

JavaScript predstavlja skript jezik kojim se mogu automatizovati postupci unutar WEB stranice. JavaScript se naziva skript jezik zato što omogućava pisanje skripta (niza naredaba) koji čitač WEB-a treba da izvrši. JavaScript je po sintaksi veoma sličan C, C++ i Javi i omogućava zadavanje grupe naredaba. I pored toga, on se ne može koristiti za pravljenje spoljnog programa odnosno izvršnog fajla koji se izvršava nezavisno od čitača.

Bez skript jezika, kao što je JavaScript, stranice nemaju mogućnost obrade ili bilo kakav način komunikacije sa posetiocima stranice. JavaScript se koristi na milionima WEB strana kako bi im dodali funkcionalnosti kao što su validacija obrazaca, komunikacija sa klijentom, animacija elemenata na strani i još mnogo toga. Jezgro jezika JavaScript-a podržava brojeve, znakovne nizove i logičke vrednosti kao osnovne tipove podataka. JavaScript se najčešće koristi u čitačima Weba pa se jezgro opšte namene proširuje objektima koji omogućavaju skriptovima interakciju sa korisnikom.

Ovakva ugrađena verzija JavaScript-a izvršava skriptove koji su ugrađeni u HTML Web stranice i zbog toga se zove klijentski JavaScript, kako bi se naglasilo kako se skriptovi izvršavaju na klijentskim računarima a ne na Web serveru. Svaki čitač koji je noviji od Netscape 4.5 i Internet Explorera 4 podržava najnoviju verziju JavaScripta, što praktično znači da je malo verovatno da ćemo u praksi naići na neodgovarajući interpreter.

Kada se interpreter JavaScripta umetne u čitač Weba, rezultat je klijentski JavaScript. U klijentskom JavaScriptu kombinuju se mogućnosti interpretera JavaScripta da izvršava skriptove sa objektnim modelom dokumenta definisanim u čitaču Weba. JavaScript se uključuje u sadržaj HTML dokumenta i omogućava unapređenje HTML strana.

## UMETANJE JAVA SCRIPTA



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

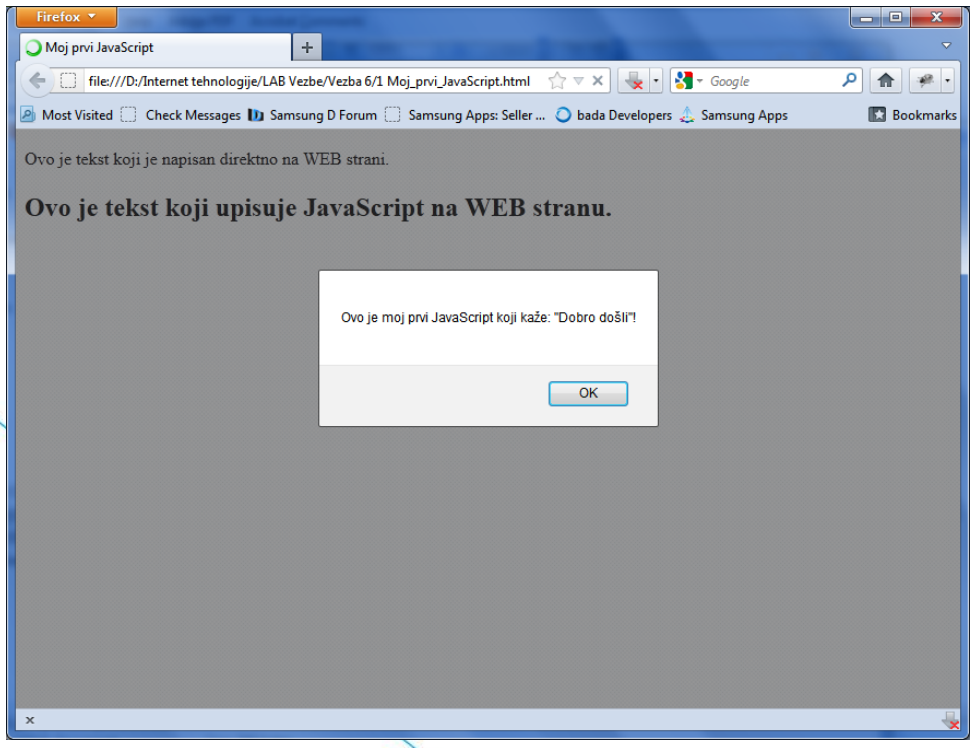
Da bi smo dodali kôd JavaScript-a unutar HTML stranice koristimo oznake `<script>` i `</script>`. Unutar oznake `script` umeće se atribut `language` kojim se tačno definiše o kome skript jeziku se radi. U sledećem primeru prikazan je jedan jednostavan kôd umetnut u HTML dokument koji ispisuje jednu rečenicu.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Moj prvi JavaScript</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
</head>
<body>
      <p>Ovo je tekst koji je napisan direktno na WEB strani.</p>

      <script type="text/javascript" language="JavaScript">
      <!--
      document.write("<h2>Ovo je tekst koji upisuje JavaScript na WEB stranu.</h2>");
      alert ("Ovo je moj prvi JavaScript koji kaže: \"Dobro došli\"!");
      // -->
      </script>

      <p>WEB browser je prikazao JavaScript popup prozor koji ispisuje tekst. Ovaj pasus se ispisuje tek nakon pritiskanja dugmeta OK unutar prozora za obaveštenje.</p>
</body>
</html>
```

Kada se prethodni kôd unese u čitač koji podržava JavaScript dobija se izlaz kao što je to prikazano na slici 1.



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




***Slika 1**. Rezultat izvršavanja prethodnog koda*

U prethodnom primeru, `document.write` znači da će čitač izvršiti metodu (funkciju) `write` povezanu sa objektom `document` (tj. tekućom web stranicom). Tako JavaScript dodaje u HTML kod sledeći sadržaj `("<h2>Ovo je tekst koji upisuje JavaScript na WEB stranu.</h2>",` koji će ispisati naslov drugog nivoa sa sledećim sadržajem: *„Ovo je tekst koji upisuje JavaScript na WEB stranu.*".

Deo u kome piše:

```
alert ("Ovo je moj prvi JavaScript koji kaže: \"Dobro došli\"!");
```

prikazuje iskačući prozor u kome se ispisuje navedena rečenica. Prozor se uklanja pritiskanjem dugmeta OK unutar prozora. Nakon toga čitač nastavlja da prikazuje na WEB stranu sve ono što je navedeno u nastavku (u ovom primeru ispisuje sledeći pasus).

**SAKRIVANJE SKRIPTA OD ČITAČA KOJI NE PODRŽAVAJU JavaScript**

Dakle, JavaScript se unutar HTML dokumenata uvek mora naći unutar `script` oznaka. Ukoliko bi se desilo da se gornji kôd izvršava u nekom starijem pretraživaču koji nema ugrađenu podršku za JavaScript, gornji skript bi se nepravilno prikazao i čitač bi prema specifikaciji HTML jezika ignorisao



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

sve oznake „<" i „>" koje ne zna da protumači i sadržaj JavaScript-a bi bio u potpunosti vidljiv (bio bi ispisan kao običan tekst na HTML strani).

Danas je prava retkost da neki od WEB čitača ne podržava JavaScript. Ipak, zbog čitača koji ne prepoznaju oznake `<script>` i `</script>`, da bi se osigurali od nepravilnog prikazivanja potrebno je sam skript stavljati između oznake za početak komentara (`<!--`) i kraj komentara (`-->`).

**REZERVISANE REČI U JavaScript-u**

Kada se zadaje ime funkciji ili promenljivoj ono ne sme biti u sukobu sa rezervisanim rečima JavaScripta, datim u Tabeli 2. Rezervisane reči imaju posebno značenje u JavaScriptu. Reč `function` označava početak deklaracije funkcije, dok reč `return` saopštava čitaču da se vrati na mesto odakle je pozvao funkciju. Ako pokušate da koristite rezervisane reči za nazive funkcija ili promenljivih, čitač Weba će prekinuti izvršavanje skripta ili funkcije kada se susretne sa nepravilnom upotrebom rezervisane reči.

| | | | | | |
|---|---|---|---|---|---|
| abstract | continue | finally | instanceof | protected | throws |
| boolean | default | float | int | public | transient |
| break | delete | for | interface | return | true |
| byte | do | function | long | short | try |
| case | double | goto | native | static | typeof |
| catch | else | if | new | super switch | var |
| char | extends | implements | null | synchronized | void |
| class | false | import | package | this | while |
| const | final | in | private | throw | with |

JavaScript razlikuje mala i velika slova što podrazumeva da se prilikom pisanja kôda u JavaScript-u o tome mora strogo voditi računa a takođe, važno je napomenuti da zanemaruje razmake načinjene tabulatorom kao i prelome redova. Iza jednostavnih JavaScript naredbi najčešće se koristi tačka-zarez (`;`) kao što je to slučaj u jezicima C i C++ ili Java, a označava odvajanje jedne naredbe od druge, ali se može izostaviti ukoliko je svaka naredba posebno napisana u svakom redu.

**DEKLARISANJE PROMENLJIVIH**

Promenljiva je ime povezano sa vrednošću i ona čuva, odnosno sadrži neku vrednost. Promenljive omogućavaju čuvanje podataka koje kasnije koristimo u programima. Za razliku od većine



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

programskih jezika, ne treba da zadate tip promenljive. JavaScript određuje automatski tip promenljive na osnovu podatka koji smeštate u promenljivu.

Za deklarisanje promenljive u skrpitu, zadaje se ime promenljive, posle ključne reči `var`, kao što je prikazano u nastavku.

```
var NaslovKnjige, BrojPoglavlja;

var Izdavac;
```

Da istovremeno zadate promenljivoj i početnu vrednost, iza njenog imena stavite operator dodele (=) i početnu vrednost, čime automatski određujete tip promenljive.

```
var StringVrednost = "10", var IntegerVrednost = 20;
```

Kada govorimo o znakovnim promenljivama njihova vrednost mora biti unutar navodnika kako je to napisano u prethodnom kodu. Unutar navodnika se mogu koristiti i specijalni znaci. Tabela 1. prikazuje listu specijalnih znakova koje možemo koristiti u JavaScript stringovima.

| **Znak** | **Značenje** |
|---|---|
| \b | backspace (jedan karakter unazad) |
| \f | form feed |
| \n | new line (prelazak u novi red) |
| \r | carritage return |
| \t | tab (pomeranje za jedan tabulator udesno) |
| \" | quote (prikazivanje znaka navoda) |
| \\ | backslash character (prikazivaje znaka \) |

***Tabela 1**. Specijalni znakovi za stringove*

Osim toga, tip promenljive može da se promeni zavisno od operacije u kojoj koristite promenljivu.

```
var StringVrednost = "10", var IntegerVrednost = 20;

var SpojenaVrednost = StringVrednost + IntegerVrednost;
```

Rezultat prethodne operacije je „1020" (tipa String) jer je prva promenljiva tipa string.

```
var StringVrednost = "10", var IntegerVrednost = 20;

var DodataVrednost = IntegerVrednost + StringVrednost;
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Rezultat prethodne operacije je „30" (tipa Int – celobrojna vrednost) jer je prva promenljiva tipa Int.

Postoje sledeća ograničenja prilikom zadavanja imena promenljivama:

- Ime promenljive ne sme biti u sukobu sa rezervisanim rečima JavaScripta,
- Imena promenljivih moraju počinjati slovom ili donjom crtom ( _ ), i mogu sadržati brojeve i slova nakon prvog znaka,
- Imena promenljivih ne mogu sadržati znakove za razmak ili interpunkciju,

U nazivima treba voditi računa o malim i velikim slovima (eng. *case-sensitive*). To znači da je `var` rezervisana reč a `Var` nije. Promenljiva `StringVrednost` se razlikuje od promenljive `stringvrednost`.

**OBLAST VAŽENJA PROMENLJIVIH**

Oblast važenja promenljivih predstavlja oblast u kojoj važi definicija te promenljive. Globalna promenljiva ima globalnu oblast važenja, što znači da je definicija važeća u čitavom programu na JavaScriptu. Ako je promenljiva definisana u funkciji onda je njena oblast važeća samo u funkciji, odnosno telu funkcije. To su takozvane lokalne promenljive i imaju lokalnu oblast važenja. Parametri funkcije takođe se smatraju lokalnim promenljivama i njihova oblast važenja je telo funkcije. Ukoliko deklarišemo lokalnu promenljivu sa jednakim nazivom kao globalnu onda je u telu funkcije dostupna samo lokalna promenljiva. Kod JavaScripta ne možemo ograničiti oblast važenja unutar bloka.

**OPERATORI**

Operatori dozvoljavaju kombinovanje promenljivih i/ili slovnih vrednosti u izraze koji daju vrednost. Većina operatora u JavaScript-u je vrlo slična operatorima C, C++ i Jave. JavaScript ima više operatora za razne aritmetičke, logičke i operacije sa stringovima. Ovde će biti navedeni samo najznačajniji operatori a to su za: aritmetički operatori, operatori pridruživanje, operatori za komparaciju i logički operatori.

**Aritmetički operatori**

Aritmetički operatori koji zahtevaju dve promenljive, dve vrednosti ili dva izraza (po jedan sa svake strane operatora) su prikazani u Tabeli 2.

| Operator | Namena | Primer |
|---|---|---|
| + | Sabiranje | izraz1 + izraz2 |
| - | Oduzimanje | izraz1 - izraz2 |



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

| | | |
|---|---|---|
| * | Množenje | izraz1 * izraz2 |
| / | Deljenje | izraz1 / izraz2 |
| % | Deljenje po modulu (vraća ostatak pri deljenju) | izraz1 % izraz2 |

***Tabela 2**. Aritmetički (binarni) operatori*

Unarni operatori, koji rade sa jednim operandom prikazani su u Tabeli 3.

| Operator | Namena | Primer |
|---|---|---|
| ++ | Povećanje za 1 | izraz1 + izraz2 |
| -- | Smanjenje za 1 | izraz1 - izraz2 |
| - | Promena znaka | izraz1 * izraz2 |
| ! | Komplement (operator negacije) | izraz1 / izraz2 |

***Tabela 3**. Unarni operatori*

Kada se dvostruki plus (++) ili dvostruki minus (--) stavi ispred promenljive, kao prefiks, operacija se obavlja pre dodele vrednosti. Obrnuto, ako se stavi operator povećanja ili smanjivanja nakon promenljive, kao sufiks, operacija se obavlja nakon dodeljivanja vrednosti. Npr.

```
i=10;
j=i++; //j=9, i=11
j=--i; //i=9, j=9
```

Operator komplement (negacije) – znak uzvika (!) – vraća negaciju logičke vrednosti promenljive (tačno ili netačno) pri čemu se menja vrednost promenljive. Npr.

```
RezultatTesta = true;
document.write(RezultatTesta); //tačno
document.write(!RezultatTesta); //netačno
```

**Operatori pridruživanja**

Operatorom pridruživanja zadaje se vrednost promenljivoj sa leve strane od operatora (=), tako što se izračunava vrednost izraza sa desne strane operatora.



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Primer:

```
x  =  y + 5;
```

U ovom primeru prvo će biti izračunat zbir vrednosti koju trenutno sadrži promenljiva `y` i broja `5`, a onda će se tako izračunata vrednost dodeliti promenljivoj `x` u koju će biti smešten rezultat.

U JavaScriptu postoji i kraći način pisanja naredbi pridruživanja kao što je to prikazano u Tabeli 4.

| Skraćeno | Značenje |
|---|---|
| x += y | x = x + y |
| x -= y | x = x - y |
| x *= y | x = x * y |
| x /= y | x = x / y |
| x++ | x = x + 1 |
| x-- | x = x - 1 |

***Tabela 4**. Operatori pridruživanja*

**Operatori poređenja (komparacije)**

Operatorima poređenja vrši se poređenje dve vrednosti i na osnovu rezultata poređenja dobija logička vrednost `true` ili `false` u zavisnosti da li je poređenje tačno ili netačno. Vrednosti koje se porede mogu biti numeričke ili string vrednosti. Kada se porede stringovi, poređenje se vrši na osnovu standardnog leksikografskog redosleda slova. Operatori poređenja opisani su u sledećoj tabeli.

| Operator | Opis | Primer |
|---|---|---|
| Jednako<br>(= =) | Daje vrednost `true`<br>ako su operandi jednaki. | x == y;<br>je true ako je x jednako y |
| Nije jednako<br>(!=) | Daje vrednost `true`<br>ako operandi nisu jednaki. | x != y;<br>j e true ako x nije jednako y |



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

| | | |
|---|---|---|
| Veće od<br>(>) | Daje vrednost `true` ako je<br>levi operand veći od desnog | x > y;<br>je true ako je x veće od y |
| Veće ili jednako (>=) | Daje vrednost `true` ako je levi operand veći ili jednak desnom | x >= y;<br>je true ako je x veće ili jednako y |
| Manje od<br>(<) | Daje vrednost `true` ako je<br>levi operand manji od desnog | x < y;<br>je true ako je x manje od y |
| Manje ili jednako<br>(<=) | Daje vrednost `true` ako je levi operand manji ili jednak desnom | x <= y;<br>je true ako je x manje ili jednako y |
| Uslovni operator<br>(?:) | Daje vrednost navedenu posle znaka `?` ako je uslov u zagradi `true`;<br>Daje vrednost posle znaka : ako je uslov u zagradi jednak `false` | odgovor = (x>0) ? „vece od nula" : „manje od nula";<br>je true ako je x manje ili jednako y |

***Tabela 5**. Operatori poređenja*

Operatori poređenja, kao što pokazuje ime njihove grupe, omogućuje da poredite jedan izraz, promenljivu ili slovnu vrednost sa drugom. Operatori poređenja (prikazani u Tabeli 5.) se obično koriste u uslovnim iskazima (kao što su `if` i `if-else`) i upravljačkim petljama (kao što su `while` i `for`).

**Logički operatori**

Logički operatori najčešće izvode logičke algebarske operacije i često se koriste zajedno sa operatorima poređenja da bi se izvela složena poređenja sa više promenljivih. Rezultat logičkih operacija je logička vrednost. Najčešće se koriste u naredbama `if`, `while` i `for`. Tabela 4 prikazuje najčešće korišćene logičke operatore.

| Operator | Upotreba | Opis |
|---|---|---|
| And (&&) | `Izraz1 && Izraz2` | Rezultat je true ako su oba izraza, Izraz1 i Izraz2 true. U suprotnom je false. |
| Or (\|\|) | `Izraz1 \|\| Izraz2` | Rezultat je true ako su ili Izraz1 ili Izraz2 true, a false samo ako su oba izraza Izraz1 i Izraz2 false. |



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

| | | |
|---|---|---|
| Not (!) | `!Izraz` | Rezultat je false ako je Izraz true, a true ako je Izraz false. |

***Tabela 6**. Logički operatori*

**NAREDBE**

Da bi se nešto uradilo moramo upotrebiti naredbu u JavaScriptu koja je nalik celovitoj rečenici ili komandi. Najjednostavnija vrsta naredaba u JavaScriptu jesu izrazi sa sporednim efektima. Primer jedne naredbe može biti sledeći kôd:

```
poruka = "Zdravo" + ime;
```

JavaScript omogućava i da se od više naredbi napravi jedna naredba, odnosno blok naredbi. Ovo je moguće uraditi tako što se proizvoljan broj naredbi napiše unutar velikih zagrada čineći jedan blok kao što je to pokazano u narednom primeru:

```
{
        x = Math.PI;
        cx = Math.cos(x);
        alert("cos(" + y + ")= " + cx);
}
```

Kao što se vidi u prethodnom primeru ovaj blok kôda, iako se ponaša kao jedinstvena naredba, ne završava se tačka-zarezom. Osnovne naredbe unutar bloka se završavaju tačka-zarezom ali ne i sam blok.

**IF NAREDBA**

IF naredba predstavlja osnovnu naredbu za kontrolu toka programa. Omogućava JavaScriptu da sam odlučuje, odnosno uslovno izvršava naredbe. Ova naredba postoji u dva oblika.

Prvi oblik je:

```
if ( izraz )
naredba;
```

U ovom obliku računa se vrednost izraza. Ako je izraz `true` izvršava se naredba, ako nije, naredba se ne izvršava.

Drugi oblik naredbe if je:



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
if ( izraz )
   naredba1;
else
   naredba2;
```

U ovoj naredbi koristi se odredba `else` koja se izvršava ako `izraz` ima vrednost `false`.

**ELSE IF naredba**

Ako je potrebno izvršiti jednu ili više mogućnosti u kôdu koristi se naredba `else if`, kao što je pokazano u sledećem primeru:

```
if ( n ==1) {
// Izvršavaju se naredbe ovog bloka
}
else if (  n ==2) {
// Izvršavaju se naredbe ovog bloka
}
else
// Ako ni jedan uslov nije zadovoljen, izvršava se ovaj deo
```

**JavaScript FUNKCIJE**

Funkcije su jedan od fundamentalnih gradivnih blokova JavaScript programa. Funkcija predstavlja skup naredbi koje izvršavaju neki specifičan zadatak. Funkcija se sastoji od sledećih delova:

- Ključne reči : `function`
- Naziva (imena) funkcije
- Liste argumenata smeštene u malim zagradama i sa zapetama kao separatorima
- Niza naredbi unutar velikih zagrada: `{ }`

Kada se definiše funkcija, važno je da se vodi računa o sintaksi. Za razliku od HTML-a, JavaScript je osetljiv na velika i mala slova, pa je važno da funkcijske naredbe budu unutar velikih zagrada `{ }`, da argumenti budu razdvojeni zapetama, kao i da koristite znak tačka-zarez na kraju svake naredbe.



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Takođe je važno da se uoči razlika između definisanja i pozivanja funkcije. Definisanje funkcije imenuje funkciju i specificira šta će biti urađeno kada se funkcija pozove. Funkcija se definiše unutar `<script>...</script>` tagova, koji su smešteni unutar `<head>...</head>` tagova. Pri definisanju funkcije takođe se deklarišu i promenljive koje ćemo pozivati u toj funkciji.

Primer definisanja funkcije:

```
function poruka() {
            alert("Ovo je poruka");
}
```

U ovom primeru potrebno je uočiti zagrade koje se nalaze odmah posle imena funkcije. One su obavezne čak i kada između njih nema ničega. Ako želimo da pri pozivanju funkcije prenesemo neki parametar (argument), onda takve parametre stavljamo između ovih zagrada. Parametri su dodatne informacije koje mogu biti različite pri raznim pozivima funkcije. Parametri se smeštaju u promenljive i njima se pristupa kao i svim drugim promenljivama.

Kada programeri pišu naredbe, obično grupišu srodne naredbe (iskaze) u funkciju. Svaka funkcija ima jedinstveno ime i obavlja neki zadatak. Sledeći iskaz definiše funkciju `Dobrodosli` koja prikazuje poruku *„Dobrodošli na moj sajt!"* u okviru za poruke.

```
<html>
<head>
      <title> Pozivanje JavaScript funkcije</title>
      <script type="text/javascript">
            function Dobrodosli()
            {
                  alert("Dobrodošli na moj sajt!");
                  return;
            }
      </script>
</head>
<body>
<script language="text/JavaScript">
<!--
      Dobrodosli();
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
-->
</script>
<p>Ovaj tekst, je napisan u HTML dokumentu i pojavljuje se nakon pritiskanjem miša na dugme OK unutar upozorenja koje se pokreće prilikom pokretanja strane. Upozorenje se pojavljuje pozivanjem odgovarajuće funkcije</p>
</body>
</html>
```

Kod pozivanja JavaScript funkcija mogu se proslediti vrednosti u funkciju preko promenljivih koje se zovu *parametri*. Niz parametara funkcije nalazi se u zagradama, posle imena u deklaraciji funkcije. Npr. sledeći kod deklariše funkciju sa dva parametra – `strA` i `strB`:

```
function racunanjePovrsine(strA, strB)
{
      P=strA*strB;
      alert("stranica A = " + strA + "; stranica B = " + strB);
      return(strA*strB)/2;
}
```

Da bi prosledili vrednosti funkciji `racunanjePovrsine`, uključuju se parametri u poziv funkcije na sledeći način:

```
PovrsinaTrougla=racunanjePovrsine(a, 10);
```

Kada čitač pozove funkciju `racunanjePovrsine`, on će proslediti vrednost promenljive `a` funkciji `racunanjePovrsine` preko parametra `strA`, a broj `10` funkciji preko parametra `strB`. Ukratko, čitač dodeljuje vrednosti parametrima funkcije prema njihovom redosledu. Prema tome, ako promenljivoj `a` dodelite vrednost `20` i pozovete funkciju `racunanjePovrsine`, funkcija će prikazati poruku sa tekstom „`stranica A = 20; stranica B = 10`". Za razliku od funkcije u prethodnom primeru, funkcija `racunanjePovrsine` vraća vrednost pozivaocu. Da bi dobili vrednost iz funkcije, treba umetnuti vrednost koju želimo da vratimo odmah nakon ključne reči `return`, kao što je prikazano u poslednjem redu, pre vitičaste zagrade.

Celokupan kod za prethodno opisan primer dat je u nastavku:

```
<html>
<head>
<title> Pozivanje JavaScript funkcije</title>
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<script language="javascript" type="text/javascript">
    function racunanjePovrsine(strA, strB)
      {
            P=strA*strB;
            alert("stranica A = " + strA + "; stranica B = " + strB);
            return(strA*strB)/2;
      }
</script>
</head>
<body>
<script language="javascript" type="text/javascript">
<!--
      var P, strA, strB, PovrsinaTrougla;
      var a=20;
      PovrsinaTrougla = racunanjePovrsine(a,10);
      document.write("<h2>Površina trougla sa prethodno definisanim stranicama je: " + PovrsinaTrougla + "</h2>");
-->
</script>
<p>Ovaj tekst, je napisan u HTML dokumentu i pojavljuje se nakon izvršavanja JavaScript koda.</p>
</body>
</html>
```

**OBRADA DOGAĐAJA U JavaScriptu**

U prethodnim poglavljima je pokazano kako se poziva JavaScript funkcija, upisujući ime funkcije kao iskaz u skriptu ili kao deo izraza. Ponekad je potrebno pozvati funkciju kao odgovor na neki događaj (učitavanja strane, klik na dugme, klik na link). Specijalni atributi HTML oznaka koriste se za povezivanje događaja i JavaScript iskaza koji ih obrađuju. Ako je potrebno da JavaScript saopšti čitaču Weba da izvrši funkciju koja je deklarisana na stranici, mora se povezati funkcija sa određenim atributom obrade  događaja. Spisak svih atributa obrade događaja dat je u Tabeli 5.



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Za dodavanje JavaScript funkcije događaju, prvo izaberite atribut obrade tog događaja i onda odlučite koje objekte Web stranice želite da atribut obrade događaja prati. Umetanjem atributa obrade događaja `onClick` u oznaku `<input>` dugmeta, saopštava se rutini za obradu događaja `onClick` da prati pritiskanje dugmeta mišem. Kada posetilac pritisne dugme, atribut obrade događaja saopštava čitaču Weba da izvrši funkciju `youClickedIt()`:

```
<form> <input type="button" value="Click Me!" onClick="youClickedIt()">
</form>
```

U nastavku je data tabela sa atributima obrade događaja i opisom kada događaj nastaje, kao i veza sa određenim oznakama.

| **Atribut Obrade Događaja** | **Događaj nastaje kada** |
|---|---|
| ***onAbort*** | Posetilac prekine učitavanje slike (na primer, pritiskanjem hiperveze ili dugmeta Stop čitača). Oznaka: `<img>` |
| ***onBlur*** | Prozor, okvir ili element obrasca izađe iz fokusa.<br>Oznake: `<body>`, `<frame>`, `<frameset>`, `<ilayer>`, `<input>`, `<layer>`, `<select>` |
| ***OnChange*** | Posetilac menja vrednost u elementu obrasca kao što je jednoredno ili višeredno polje za unos teksta, ili bira novi element u padajućoj listi. Oznake: `<input>`, `<select>` |
| ***OnClick*** | Posetilac pritisne hipervezu ili element obrasca kao što je polje za potvrdu, dugme, radio-dugme, dugme Reset ili dugme Submit. Oznake: `<a>`, `<area>`, `<body>`, `<input>` |
| ***onDoubleClick*** | Posetilac dvaput pritisne hipervezu ili element obrasca kao što je polje za potvrdu, dugme, radio-dugme, dugme Reset ili dugme Submit.<br>Oznake: `<a>`, `<area>`, `<body>`, `<input>` |
| ***onDragDrop*** | Posetilac povlači objekat i pušta ga u prozor čitača Weba.<br>Oznake: `<body>`, `<frame>`, `<frameset>` |
| ***onError*** | Čitač naiđe na grešku kada učitava sliku ili Web stranicu.<br>Oznake: `<body>`, `<frame>`, `<frameset>`, `<img>` |
| ***onFocus*** | Prozor ili element obrasca uđe u fokus. Oznake: `<body>`, `<frame>`, `<frameset>`, `<ilayer>`, `<input>`, `<layer>`, `<select>` |



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

| | |
|---|---|
| ***onKeyDown*** | Posetilac pritisne taster na tastaturi. Oznake: `<a>`, `<body>`, `<frame>`, `<frameset>`, `<img>`, `<input>` |
| ***OnKeyPress*** | Posetilac pritisne i drži pritisnut taster u polju za unos teksta elementa obrasca. Oznake: `<a>`, `<body>`, `<frame>`, `<frameset>`, `<img>`, `<input>` |
| ***onKeyUp*** | Posetilac otpusti taster u polju za unos teksta elementa obrasca. Oznake: `<a>`, `<body>`, `<frame>`, `<frameset>`, `<img>`, `<input>` |
| ***onLoad*** | Čitač učita Web stranicu ili sliku.<br>Oznake: `<body>`, `<frame>`, `<frameset>`, `<img>`, `<layer>` |
| ***onMouseDown*** | Posetilac pritisne taster miša. Oznake: `<a>`, `<body>`, `<input>` |
| ***onMouseOut*** | Posetilac pomeri pokazivač miša van područja mapirane slike ili sa hiperveze. Oznake: `<a>`, `<ilayer>`, `<layer>` |
| ***onMouseOver*** | Posetilac pomeri pokazivač miša u područje mapirane slike ili na hipervezu. Oznake: `<a>`, `<ilayer>`, `<layer>` |
| ***onMouseUp*** | Posetilac otpusti taster miša. Oznake: `<a>`, `<body>`, `<input>` |
| ***onMove*** | Posetilac ili skript pomere prozor čitača. Oznake: `<body>`, `<frame>`, `<frameset>` |
| ***onReset*** | Posetilac pritisne dugme Reset u obrascu. Oznaka: `<form>` |
| ***onResize*** | Posetilac ili skript promene veličinu prozora čitača.<br>Oznake: `<body>`, `<frame>`, `<frameset>` |
| ***onSelect*** | Posetilac izabere tekst u višerednom ili jednorednom polju za unos teksta. Oznaka: `<input>` |
| ***onSubmit*** | Posetilac pritisne dugme Submit u obrascu.<br>Oznaka: `<form>` |
| ***OnUnload*** | Pre nego što čitač zameni jednu Web stranicu drugom.<br>Oznake: `<body>`, `<frame>`, `<frameset>` |

**Tabela 7.** *Opis atributa obrade događaja*

Ako je potrebno da čitač pozove funkciju kada posetilac pomeri pokazivač miša preko hiperveze, treba umetnuti atribut obrade događaja `onMouseover` u oznaku `<a>` hiperveze:



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<a href="http://www.vtsnis.edu.rs" onMouseover="youAreOnIt()">
      Link ka sajtu škole
</a>
```

**JAVASCRIPT OBJEKTI**

JavaScript je objektno orjentisan skript jezik. Kada pomoću pretraživača pristupimo nekom HTML dokumentu na Webu, onda se kreira niz JavaScript objekata sa svojstvima i mogućnostima koje se baziraju na HTML-u i drugim delovima sadržaja učitanog dokumenta. Ovi objekti su povezani hijerarhijski u skladu sa strukturom HTML stranice.

Od svih objekata najčešće su u upotrebi objekti `window` i `document`. Objekat `window` ima metod koji nam omogućava da kreiramo novi window pomoću `open()` i `close()` metoda. On nam takođe omogućava da kreiramo male prozore (eng. *message boxe*) korišćenjem `alert()`, `confirm()`, i `prompt()` metoda. Svaki od ovih malih prozora prikazuje tekst koji se stavlja između zagrada. Na primer, naredba:

```
alert ("Ovo je prozor sa porukom upozorenja")
```

će otvoriti mali prozor sa porukom u kome će biti tekst koji je postavljen u navodnike.

Pozivanje metode nekog objekta vrši se preko imena objekta iza čega sledi tačka (.) a onda ime metode.

```
document.write("Poruka");
```

Kada se koristi metoda `document.write()` za upisivanje teksta na Web stranicu, obavezno treba uključiti HTML oznake za formatiranje koje izgleda. Da biste prelomili dugačak tekst, podelite tekst na nekoliko znakovnih nizova nadovežite ih operatorom +:

```
document.write("<p><b>Welcome to the Web page" +
"with my first JavaScript script.</b></p>");
```

Objekat `document` predstavlja HTML stranicu. Objekat `document` sadrži nizove u kojima su memorisane sve komponente od kojih se sastoji sadržaj vaše web stranice kao što su slike, linkovi, forme itd. Možemo pristupati svim tim elementima i sa njima povezanim metodama preko ovih nizova.

Sledeći JavaScript kod, prikazaće naslov Web stranice, URL i datum poslednje izmene:

```
document.write ("<b>Web Page Title: </b>" +
document.title + "<br/>");
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
document.write ("<b>URL/Web Address: </b>" +
document.URL + "<br/>");

document.write ("<b>Last modified on: </b>" +
document.lastModified + "<br/>");
```

Osim svojstava koja opisuju Web stranicu i njen izgled, objekat `document` sadrži i nizove svojstva objekata umetnutih na stranicu:

- sidra,
- aplete,
- ugrađene elemente,
- obrasce,
- okvire,
- slike,
- veze i dodatne module.

Čitač Weba smešta informacije o svakom objektu na Web stranici u niz koji se odnosi na ime objekta. Na primer, čitač smešta:

- u niz `document.plugins`, svojstva dodatnih modula
- u niz `document.links`, svojstva hiperveza
- u niz `document.images`, svojstva slika.

Objektima iz ove predefinisane hijerarhije se može pristupati, i oni se mogu modifikovati. Da pristupimo nekom od elemenata, moramo specificirati njegov naziv kao i nazive svih njegovih predaka unazad sve do objekta document. Tačka, (.), se koristi kao separator u toj hijerarhiji imena. Generalno govoreći, svaki element ili objekat ima ono ime koje je dato u `name` atributu odgovarajućeg HTML taga. Na primer, sledeća konstrukcija se odnosi na vrednost (`value`) tekstualnog podatka (`text1`) koji je deo forme čije je ime *`forma`* a koja je deo tekućeg dokumenta.

```
document.forma.text1.value
```

Istom tom elementu se može pristupiti i preko gore pomenutih nizova. U gornjem primeru, ako je forma sa nazivom `forma` bila prva forma u dokumentu i ako je `text1` bio treće polje u formi, onda bi isti element mogao da bude referisan i na sledeći način:

```
document.forms[0].elements[2].value
```

Funkcije (metode) objekata mogu biti pozivane na sličan način. Na primer, sledeća naredba resetuje drugu formu u dokumentu.



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
document.forms[2].reset();
```

Ako se oznaci `<img>` pridruži atribut ime `name= "logo"`:

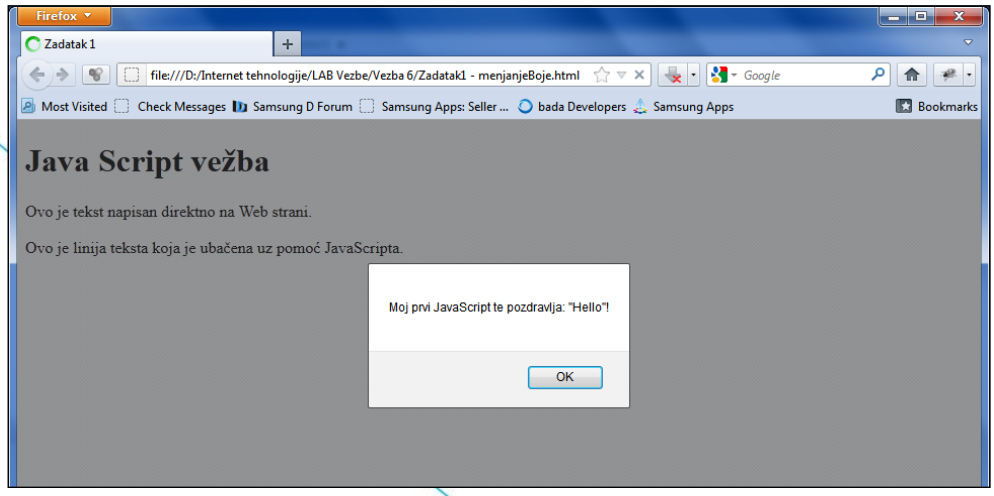
```
<img name="logo" width="78" height="61" src="images/picture.GIF">
```

Možete zameniti ovu sliku sliku na Web stranici drugom slikom sledećim iskazom:

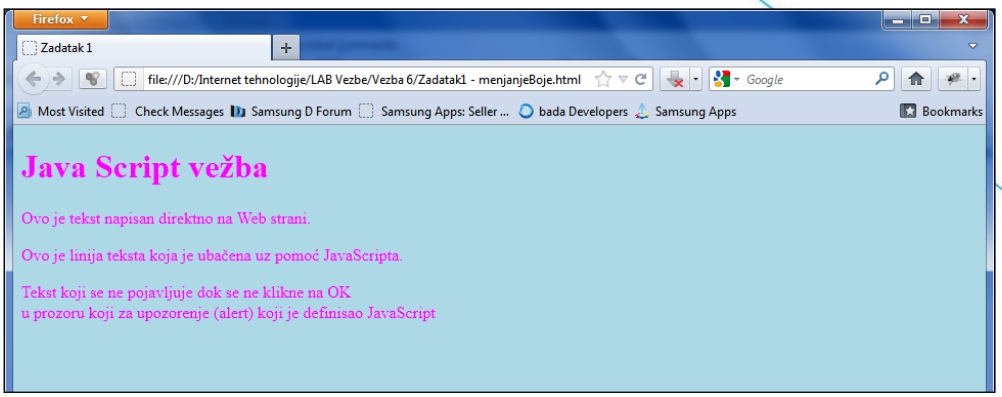
```
document.logo.src = "images/VTS_Slike.JPG"
```

## ZADACI ZA VEŽBU

**Zadatak 1:** Kreirati HTML stranu sa sadržajem kao na sledećoj slici. Napisati JavaScript kod koji će prilikom pokretanja strane aktivirati prozor sa porukom: "`Moj prvi JavaScript te pozdravlja: Hello!`"




Nakon klika na dugme OK u okviru prozora sa upozorenjem JavaScript dodaje na stranu sledeći pasus: "Tekst koji se ne pojavljuje dok se ne klikne na OK u prozoru koji za upozorenje (`alert`) koji je definisao JavaScript" i menja boju pozadine u plavu (`lightblue`), a boju teksta u roze (`magenta`), kako što je prikazano na sledećoj slici.




Za realizaciju zadatka 1, potrebno je uneti sledeći kod:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html>
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<head>
<title>Zadatak 1</title>
  <meta http-equiv="Content-Type" content="text/html;
  charset=utf-8" />
</head>
<body>
<h1>Java Script vežba</h1>
<p>Ovo je tekst napisan direktno na Web strani.</p>

<script language="JavaScript">
<!--
document.write("<p>Ovo je linija teksta koja je ubačena uz pomoć JavaScripta.</p>");
alert("Moj prvi JavaScript te pozdravlja: \"Hello\"!");
document.bgColor="lightblue";
document.fgColor="magenta";
//-->
</script>
<p>Tekst koji se ne pojavljuje dok se ne klikne na OK<br/>u prozoru koji za upozorenje (alert) koji je definisao JavaScript</p>
</body>
</html>
```

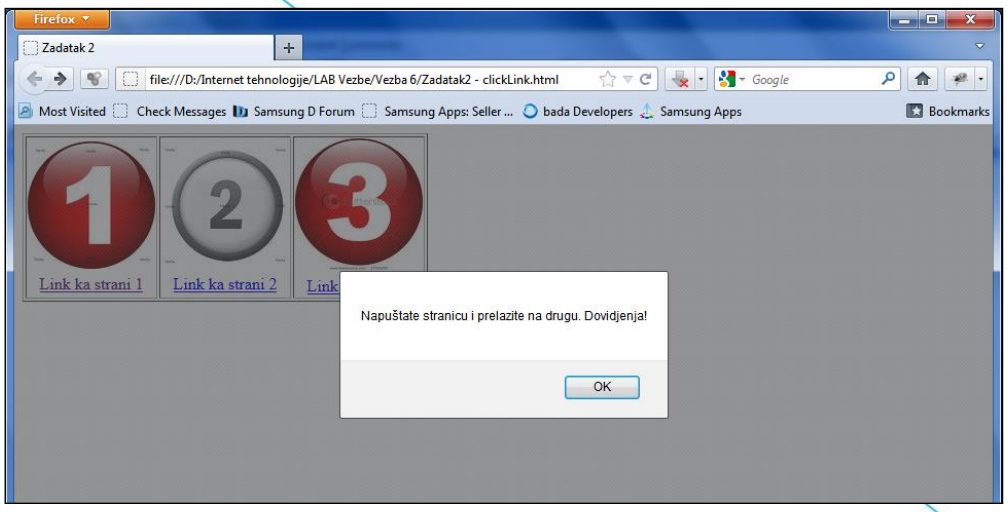
Napisati kako treba modifikovati `body` tag, da bi se pomenuta poruka pojavljivala prilikom svakog napuštanja aktuelne stranice odnosno zamenom nekom drugom stranicom.

**Zadatak 2:** Kreirati HTML stranu sa sadržajem kao na sledećoj slici.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Napisati JavaScript kod koji će, prilikom klika na neki od linkova, a pre prelaska na ciljanu stranu, aktivirati prozor sa porukom: "Napuštate stranicu i prelazite na drugu. Dovidjenja!".




Za realizaciju zadatka 2 potrebno je uneti sledeći kod:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

<title>Zadatak 2</title>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<script type="text/javascript" language="JavaScript">
<!--
      function Poruka()
      {
      alert ("Napuštate stranicu i prelazite na drugu. " +
             "Dovidjenja!");
      }
// -->
</script>

</head>
<body>
<table border="1">
      <tr align="center">
      <td>
      <a href="strana1.html" onclick="Poruka()">
         <img src="slike/animacija_1.gif"><br/>
          Link ka strani 1
      </a>
      </td>
      <td>
      <a href="strana2.html" >
         <img src="slike/animacija_2.gif"><br/>
          Link ka strani 2
      </a>
</td>
      <td>
      <a href="strana3.html" >
         <img src="slike/animacija_3.gif"><br/>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
            Link ka strani 3
        </a>
        </td>
        </tr>
</table>
</body>
</html>
```

**Zadatak 3:** Napisati funkcije u JavaScriptu "`BojaPozadine1()`", "`BojaPozadine2()`" i "`BojaPozadine3()`" za promenu boje pozadine Web stranice pomoću objektnog modela Web stranice (`document`) i njenog svojstva `bgColor`. Za poziv realizovanih funkcija postaviti tri dugmeta sa odgovarajućim natpisima o promeni pozadine.

Za realizaciju zadatka 3 potrebno je uneti sledeći kod:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
        <title>Zadatak 3 - promena boje pozadine</title>
        <meta http-equiv="Content-Type" content="text/html;
        charset=utf-8" />

        <script language = "Javascript">
                function BojaPozadine1()
                {
                    document.bgColor="yellow"
                }

                function BojaPozadine2()
                {
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
                document.bgColor="lightblue"
            }

            function BojaPozadine3()
            {
                document.bgColor="red"
            }
        </script>
</head>
<body>
<input type="button" onClick="BojaPozadine1()" value="Žuta"/><br/>
<input type="button" onClick="BojaPozadine2()" value="Plava"/><br/>
<input type="button" onClick="BojaPozadine3()" value="Crvena"/><br/>
</body>
</html>
```

Izmeniti prethodni kod, tako da ima istu funkcionalnost ali da se poziva samo jedna funkcija `BojaPozadine(boja)`, a da se parametru `boja` prosleđuje vrednost boje pozadine. Napisati kod na odgovarajućem mestu unutar sledećeg koda.

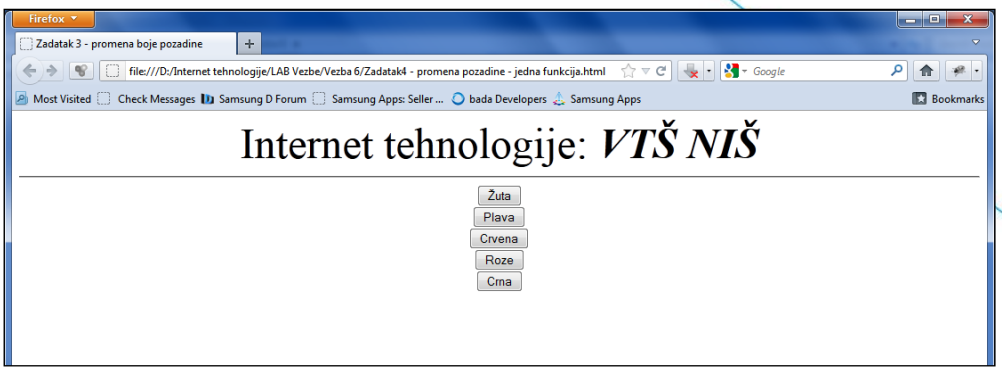
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
      <title>Zadatak 3 - promena boje jednom f-jom</title>
      <meta http-equiv="Content-Type" content="text/html;
      charset=utf-8" />
      <script language = "Javascript">
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
        </script>
</head>
<body>
<input type="button" onClick=BojaPozadine('yellow') value="Žuta"/><br/>
<input type="button" onClick=BojaPozadine('lightBlue') value="Plava"/><br/>
<input type="button" onClick=BojaPozadine('red') value="Crvena"/><br/>
</body>
</html>
```

**Zadatak 4:** Kreirati HTML stranu sa sadržajem kao na sledećoj slici. Napisati JavaScript kod koji će omogućiti promenu boje pozadine strane, klikom na određeno dugme. Koristiti događaj klika na dugme (`onClick`) a naziv boje proslediti funkciji u vidu parametra.




Za realizaciju zadatka 3 potrebno je uneti sledeći kod:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

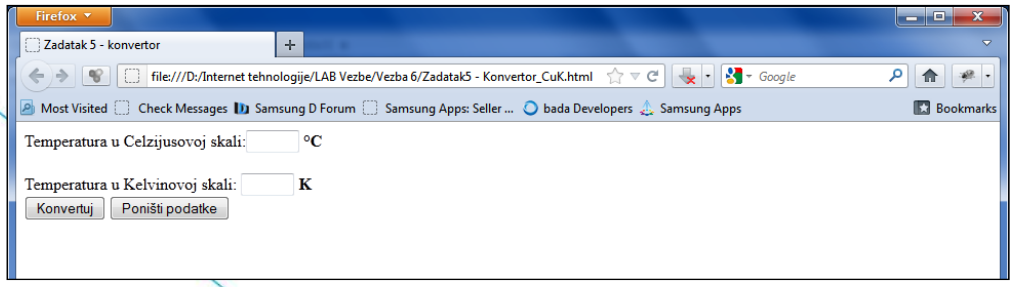
```
        <title>Zadatak 3 - promena boje pozadine</title>
        <meta http-equiv="Content-Type" content="text/html;
        charset=utf-8" />

<script language = "Javascript">
        function pageBackground(bojaPozadine)
        {
            document.bgColor= bojaPozadine;
        }
</script>
</head>
<body>
<form name="Forma">
<center>
<font size="+4">Internet tehnologije:
<i><b>VTŠ NIŠ</i></b></font>
<hr width="100%">
<input type="button" onClick=pageBackground('yellow') value="Žuta"/>
<br/>
<input type="button" onClick=pageBackground('lightblue') value="Plava"/>
<br/>
<input type="button" onClick=pageBackground('red') value="Crvena"/>
<br/>
<input type="button" onClick=pageBackground('pink') value="Roze"/>
<br/>
<input type="button" onClick=pageBackground('black') value="Crna"/>
<br/>
</form>
</center>
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
</body>

</html>
```

**Zadatak 5:** Kreirati HTML stranu koja će koristiti kao konvertor unete brojčane vrednosti temperature u Celzijusovoj skali u temperaturu u Kelvinovoj skali. Sračunata vrednost temperature u Kelvinovoj skali se upisuje u odgovarajuće tekstualno polje. Klikom na dugme `Konvertuj` proverava se da li je polje za temperaturu u Celzijusovoj skali popunjeno. Ako u polju ništa nije upisano pojavljuje se odgovarajuće obaveštenje.




Odgovarajući kod, za kreiranje konvertora, je dat u nastavku:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Zadatak 5 - konvertor</title>
  <meta http-equiv="Content-Type" content="text/html;
  charset=utf-8" />
  <script type="text/javascript" language="JavaScript">
  <!--
    function Konvertuj(Form)
    {
        if(Form.Celzijus.value==="")
        {
         alert("Molim Vas upišite temperaturu u Celzijusovoj"+
         "skali!");
         document.bgColor="white";
         return(false);
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
        }
        else
        {
         document.bgColor="lightBlue";
         Form.Kelvin.value=parseInt(Form.Celzijus.value)+273;
         return (true);
        }
   }
// -->
</script>
</head>
<body>
<form name="ObrazacTemperature">
Temperatura u Celzijusovoj skali:
<input type="text" name="Celzijus" id="Cel" size="5"/>
<b> &deg;C</b><br/><br/>
Temperatura u Kelvinovoj skali:
<input type="text" name="Kelvin" id="Kel" size="5"/>
<b> K</b><br/>
<input          type="button"          onClick=Konvertuj(ObrazacTemperature)
value="Konvertuj" />
<input type="reset" value="Poništi podatke"/>
</form>
</body>
</html>
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

## Pitanja i zadaci

1. Unutar kog HTML elementa treba smestiti JavaScript kod?

2. Koja je korektna JavaScript sintaksa da bi se na HTML strani ispisalo „VTŠ Niš"?

a. `("VTŠ Niš")`

b. `response.write("VTŠ Niš")`

c. `document.write("VTŠ Niš")`

d. `„VTŠ Niš"`

3. Na koji način se može definisati JavaScript kod za jednu HTML stranu?

4. Koja su osnovna pravila za davanje imena promenljivama u JavaScript-u?

5. Koja je vrednost promenljive `SpojenaVrednost` nakon sledećeg JavaScript koda?

```
var StringVrednost = "10", var IntegerVrednost = 20;

var SpojenaVrednost = StringVrednost + IntegerVrednost;
```

Promenljiva `SpojenaVrednost` ima vrednost _____.

```
var StringVrednost = "10", var IntegerVrednost = 20;
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
var DodataVrednost = IntegerVrednost + StringVrednost;
```

Promenljiva `SpojenaVrednost` ima vrednost _____.

6. Koje su prednosti JavaScript funkcija? Navesti način definisanja funkcije koja se naziva `PrimerFunkcija` koja treba da primi vrednosti parametara `param1` i `param2`.

7. Navesti operatore poređenja koji se koriste u JavaScripta i objasniti njihovu primenu.



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# Vežba br. 8 – Java Script (II deo)

## CILJ VEŽBE

- Upoznati se sa definisanjem JavaScript koda u okviru spoljašnjih fajlova.
- Pozivanje JavaScript funkcije koju je definisao korisnik.
- Savladati uslovno izvršavanje i kontrolu toka programa.
- Izvršavanje iskaza određeni broj puta.
- Korišćenje objektnog modela.
- Upoznati se sa tehnologijom provere unetih podataka iz obrazaca sa Web strane.
- Primena skript funkcije za proveru unetih podataka svih vrsta polja na obrascu.

## TEORIJSKE OSNOVE ZA IZRADU VEŽBE

### DEFINISANJE JavaScript-a U OKVIRU SPOLJAŠNJIH FAJLOVA

JavaScript može biti definisan i okviru spoljašnjih fajlova. Spoljašnji JavaScript fajlovi mogu sadržati skript koji se primenjuje na nekoliko različitih HTML strana. Fajl koji sadrži spoljašnji JavaScript ima ekstenziju `.js` i nije potrebno da sadrži oznake `<script>` i `</script>`. Da bi povezali HTML stranu sa spoljašnjim fajlom koji sadrži JavaScript (npr. `ABC.js`) treba uneti sledeći kod u zaglavlje HTML strane (između oznaka `<head>` i `</head>`).

```
<script type="text/javascript" src="ABC.js"></script>
```

Primer sa definisanim skriptom u spoljašnjem fajlu (`skript.js`) bi izgledao ovako:

```
<html>
<head>
      <title> Jednostavan JavaScript </title>
      <script type="text/javascript" src="skript.js"></script>
</head>
<body>
<p>Ovaj tekst, napisan u telu HTML dokumenta</p>
<script language="JavaScript">
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<!--
   Funkcija();
//-->
</script>
</body>
</html>
```

dok bi fajl `skript.js` (koji treba smestiti u isti direktorijum kao i .html strana) imao sledeći sadržaj:

```
document.write("<B>Ovaj tekst napisan je u JavaScriptu. </B>");
```

**DODAVANJE KOMENTARA U JavaScript**

Unošenje komentara unutar JavaScripta omogućeno je preko dvorstruke kose crte `//` što omogućava unos komentara koji je ispisan u jednom redu. Ako želimo da unesemo višeredni komentar koristimo `/*` i `*/` na početku i kraju višerednog komentara, respektivno. Kada Web čitač prikaže HTML dokument svaki komentar koji je napisan kao što je to gore navedeno, neće biti vidljiv.

**NIZOVI**

Niz predstavlja uređen skup vrednosti, gde se svaka od tih vrednosti naziva elemenat a svaki elemenat ima numerički opisanu poziciju tzv. index. S obzirom da JavaScript nije strogo tipizirani jezik, element niza može biti proizvoljnog karaktera što znači da ne moraju svi elementi niza biti istog tipa. Elementi niza mogu čak sadržati i druge nizove. Kada deklarišemo niz, moramo da mu deklarišemo naziv, a zatim i koliko će vrednosti sadržati. Svaka pojedinačna vrednost niza će biti memorisana u po jednom elementu niza. Indeks niza startuje od vrednosti 0. To znači da će prva vrednost u nizu biti smeštena sa indeksom 0, druga sa indeksom 1, itd. To znači, ukoliko nameravamo nizu dodeliti 10 elemenata, prvi član niza će biti sa indeksom 0, a poslednji član niza sa indeksom 9.

Deklarisanje niza, sa imenom `Niz`, koji sadrži 10 članova, vrši se na sledeći način:

```
Niz = new Array(10);
```

Inicijalno, svi elementi niza po njegovom deklarisanju dobijaju vrednost nula. Ako želimo dodeliti vrednosti elementima niza onda to radimo na sledeći način:

```
Niz[0] = 23;
Niz[1] = 34;
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
.
.
.
Niz[9] = 75;
```

Stavljajući u uglaste zagrade [ ] indeks elementa, možemo svakom elementu zadati željenu vrednost.

Ukoliko smo niz definisali tako da ima 3 elementa a potom uradimo dodelu tako da stavimo indeks 20, neće doći do greške kao u nekim programskim jezicima. JavaScript će automatski proširiti prethodno deklarisani niz kao da ima 20 elemenata.

## NAREDBE ZA KONTROLU TOKA PROGRAMA

### SWITCH naredba

Ako sve grane zavise od jedne promenljive onda je najbolje koristiti naredbu `switch`. Opšti oblik naredbe `switch` je:

```
switch ( n ){
case 1:
// Ako je n=1 izvršava se ovaj blok
break;
case 2:
// Ako je n=2 izvršava se ovaj blok
break;
case 3:
// Ako je n=3 izvršava se ovaj blok
break;
default:
// Izvršava se ovaj deo koda
}
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Naredba `switch` prvo računa vrednosti izraza a onda traži labelu `case` koja odgovara toj vrednosti. Ukoliko nađe takvu labelu, počinje izvršavanje kôda u prvoj labeli posle labele `case`. Ukoliko odgovarajuća vrednost ne postoji počinje izvršavanje prve naredbe posle labele `default`.

**WHILE petlja**

Naredba `while` je osnovna naredba za ponavljanje akcija. Sintaksa while naredbe ima sledeći oblik:

```
while ( izraz ){
naredba
}
```

Naredba `while` prvo računa vrednost izraza u zagradi. Ukoliko je ta vrednost `false`, JavaScript prelazi na sledeću naredbu u programu, u suprotnom, izvršava se naredba koja čini telo petlje i ponovo se računa vrednost izraza u zagradi.

**DO/WHILE petlja**

Petlja `do/while` je slična petlji `while`, osim što se uslovni izraz petlje računa na kraju umesto na početku petlje. To znači da će se telo petlje izvršiti najmanje jednom. Sintaksa `do/while` petlje je sledeća:

```
do
naredba
while ( izraz )
```

**FOR petlja**

Naredba `for` omogućava pravljenje petlji koje su najčešće pogodnije od petlji sa naredbom `while`. Opšti oblik `for` petlje je:

```
for ( inicijalizacija ; uslov ; ažuriranje )
naredba
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Inicijalizacija, provera uslova i ažuriranje čine ovaj oblik petlje. Inicijalizacija se računa pre početka petlje i predstavlja početnu vrednost brojača. Uslov sadrži neki od operatora poređenja i određuje trenutak do kada će se petlja izvršavati. Izraz `uslov` računa se pre svake iteracije i određuje da li će se telo petlje izvršiti. Ukoliko je njegova vrednost `true`, izvršava se naredba u telu petlje i na kraju računa se izraz `uvećanje`. Ažuriranje predstavlja način na koji se brojač menja nakon svakog prolaska kroz petlju. Obično se koristi operator ++ ili – što znači da se brojač nakon svakog prolaska kroz petlju uvećava odnosno smanjuje za 1.

Sledeća petlja će se izvršiti 6 puta za vrednosti brojača `i=0,1,2,3,4 i 5`.

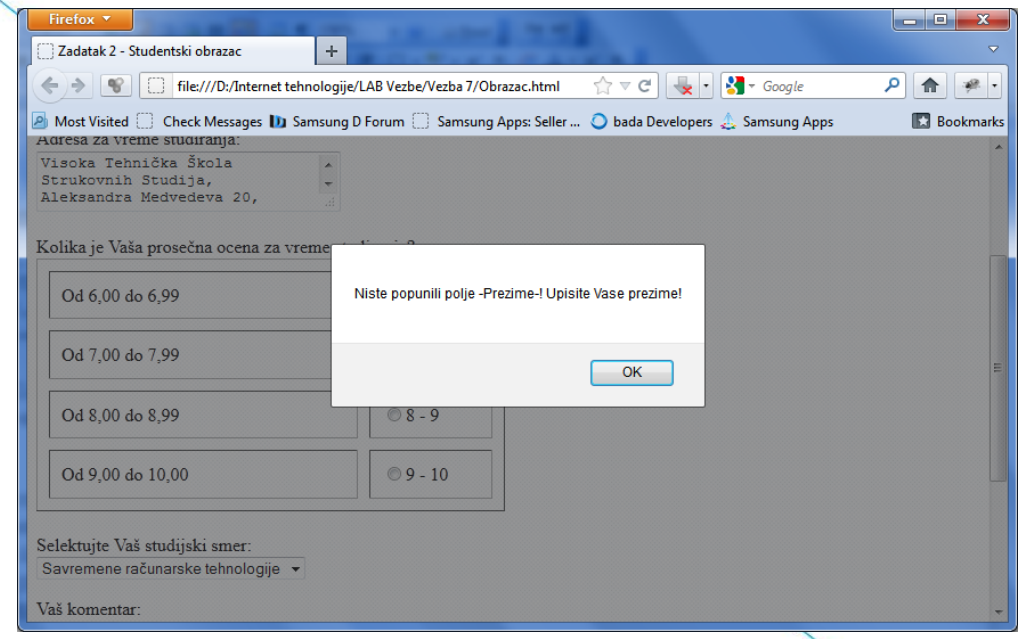
```
for (int i=0; i<5; i++)

naredba
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

## ZADACI ZA VEŽBU

**Zadatak 1:** Za obrazac koji je kreiran u okviru Zadatka 2 vežbe br. 4 (HTML obrasci) izraditi funkcije za proveru unešenih podataka u jednoredno i višeredno polje. Povezati HTML stranu sa spoljašnjim JavaScript fajlom koji ćete nazvati `Validacija.js`




*Poruka čitača korisniku da nije popunio sva polja*

HTML kod za obrazac koji je kreiran u okviru Zadatka 2 vežbe br. 4 je ovde ponovljen radi preglednosti.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
      <title>Zadatak 2 - Studentski obrazac</title>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>

<body>
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
  <h1>Studentski obrazac ŠV-1</h1>
  <form name="MojObrazac" title="Ovo je tekst koji se ispisuje kada se zadrži pokazivač miša na obrascu" onsubmit="return ValidateForm(MojObrazac)" action="http://www.htmlcodetutorial.com/cgi-bin/mycgi.pl" method="POST" enctype="application/x-www-form-urlencoded">

Prezime studenta: <input type="text" name="LastName" id="LN" size="20"><font color="red">*</font>  
Ime studenta: <input type="text" name="FirstName" id="FN" size="20"><font color="red">*</font><br/><br/>
Broj indeksa: <input type="text" name="IndeksNo" id="IN" size="20"><font color="red">*</font>  
e-mail: <input type="text" name="Email" id="EM" size="30"><font color="red">*</font><br/><br/>
Adresa za vreme studiranja:<br/>

<textarea name="PoljeAdresa" rows="2" cols="30">Visoka Tehnička Škola Strukovnih Studija, Aleksandra Medvedeva 20, 18000 Niš
</textarea>

<br/><br/>
Kolika je Vaša prosečna ocena za vreme studiranja?
      <table border="1" cellpadding="10" cellspacing="10" width="411">
<tr>
<td width="244">Od 6,00 do 6,99</td>
<td width="84"><input type="radio" name="Mark" value="6-7" />6 - 7</td>
</tr>

<tr>
<td width="244">Od 7,00 do 7,99</td>
<td width="84"><input type="radio" name="Mark" value="7-8" />7 - 8</td>
</tr>

<tr>
<td width="244">Od 8,00 do 8,99</td>
<td width="84"><input type="radio" name="Mark" value="8-9" />8 - 9</td>
</tr>

<tr>
<td width="244">Od 9,00 do 10,00</td>
<td width="64"><input type="radio" name="Mark" value="9-10" />9 - 10</td>
```

```
</tr>
</table>
<br/>
Selektujte Vaš studijski smer:<br/>
<select size="1" name="Group">
<option value="SRT">
Savremene računarske tehnologije</option>
<option value="KOT">Komunikacione tehnologije</option>
<option value="INI">Industrijsko inženjerstvo</option>
<option value="DRS">Drumski saobraćaj</option>
<option value="GIN">Građevinsko inženjerstvo</option>
</select>

<p>Vaš  komentar:<br/><textarea  name="Comments"  rows="4"  cols="60">Ovde
upišite komentar...</textarea></p>

<input type="checkbox" name="YesNoCheckbox" value="DA" unchecked>Da, uneti
podaci su tačni i možete ih koristiti u studentskoj službi.<br/>
      <p>Da li Vam možemo slati vesti i propagandni materijal na e-mail
adresu koju ste upisali?
<input type="radio" value="DA" name="Contact_Ok" />Da  
<input type="radio" value="NE" name="Contact_Ok"  />Ne
</p>
<p>
<input type="submit" value="Posalji podatke" />
<input type="reset" value="Ponisti podatke"/>
</p>
</form>
</body>
</html>
```

Da bi povezali HTML stranu sa obrascem sa odgovarajućim spoljašnjim JavaScript fajlom, u prethodno navedeni kod dodajte sledeće (unutar `head` taga):

```
<script type="text/javascript" src="validacija.js"></script>
```

Sledeći kod smestiti u fajl `validacija.js`

```
//*********************************************************
//**            Funkcija Validacije Obrasca              **
```



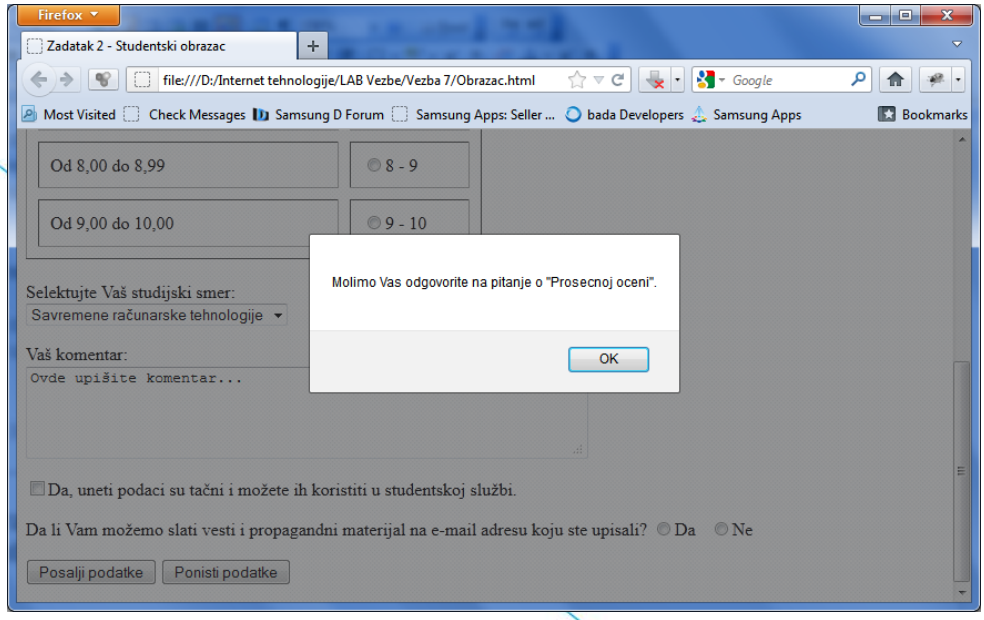
Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
//*******************************************************
function ValidateForm(Forma)
{
    //*************************************************************
    //** Proverava da li su prva cetiri polja sa imenom
    //** LastName, FirstName, IdeksNo i Email popunjena.
    //** Ako je polje prazno, obrazac vraca false,
    //** sto onemogucava slanje podataka iz obrasca.
    //** true omogucava slanje podataka iz obrasca.
    //**************************************************************
    if (Forma.LastName.value==="")
    {
       alert("Niste popunili polje -Prezime-!" +
              "Upišite Vaše prezime!");
        return (false);
    }

    if (Forma.FirstName.value==="")
    {
        alert("Niste popunili polje -Ime-!" +
        "Upišite Vaše ime!");
        return (false);
    }

    if (Forma.IndeksNo.value==="")
    {
        alert("Niste popunili polje -Broj indeksa-!" +
            "Upisite Vaš broj indeksa!");
        return (false);
    }

    if (Forma.Email.value==="")
    {
        alert("Niste popunili polje -Email-!" +
        "Upišite Vašu E-mail adresu!");
        return (false);
    }

return(true);
}
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**Zadatak 2:** Izraditi funkcije za proveru unešenih podataka u radio dugmadima obrasca. Za slučaj da korisnik ne obeleži stavku ostavi nepopunjena polja informisati ga odgovarajućom porukom.




*Poruka čitača korisniku da nije obeležio polje unutar radiobutton grupe*

Unutar funkcije ValidateForm a ispod prethodno unetog JavaScript koda za proveru popunjenosti textbox-ova uneti sledeći kod:

```
//*********************************************************
//**          FUNKCIJA ProveraRadioDugmadi
//**
//** Proverava da li je bilo koje "radio dugme",
//**           za obelezavanje oznaceno.
//**       Ako jeste vraca vrednost true i
//** prekida for petlju. Ako ne utvrdi u petlji
//** da je bilo koje dugme oznaceno, onda vraca
//** vrednost false i onemogucava slanje obrasca
//*********************************************************
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

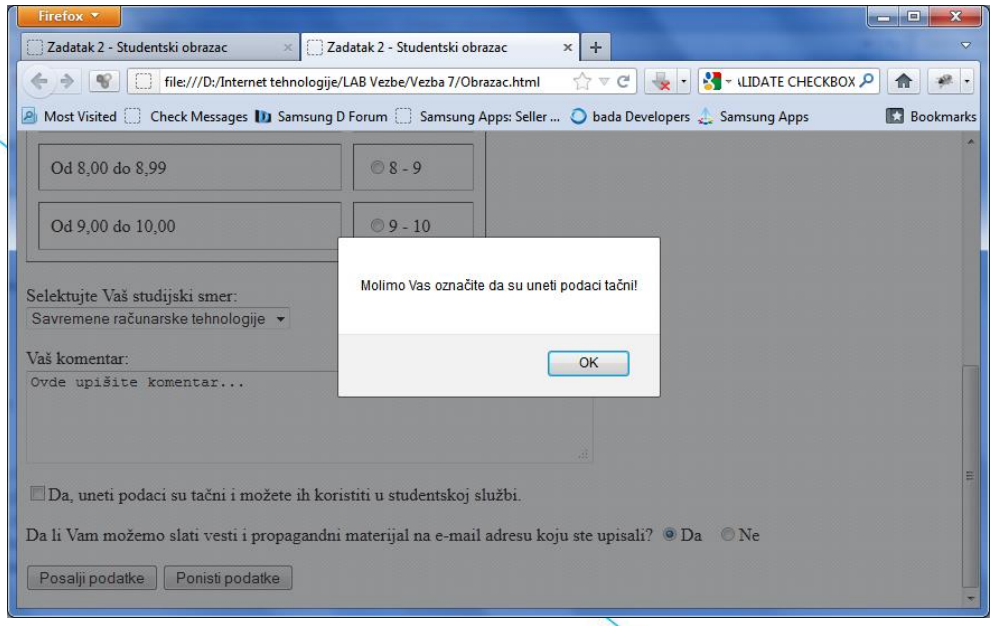
```
function ProveraRadioDugmadi(RadioGroup)
{
  for (i = 0; i <RadioGroup.length; i++)
  {
      if (RadioGroup[i].checked)
            {
                  return (true);
            }
  }
  return (false);
}
//Pozivanje univerzalne funkcije za radio dugmad koja se
//odnose na prosecnu ocenu
if (!ProveraRadioDugmadi(Forma.Mark))
{
  alert("Molimo Vas odgovorite na pitanje" +
        " \"Prosecnoj oceni\".");
  return (false);
}

//Pozivanje univerzalne funkcije za radio dugmad koja se
//odnose na kontakt pitanje
if (!ProveraRadioDugmadi(Forma.Contact_Ok))
{
 alert("Molimo Vas selektujte jednu od opcija" +
 " \"Da\" ili \"Ne\" iz \"Kontakt\" opcije.");
 return (false);
}
return(true);
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
}
```

**Zadatak 3:** Izraditi funkcije za proveru unetih podataka u check boksovima.

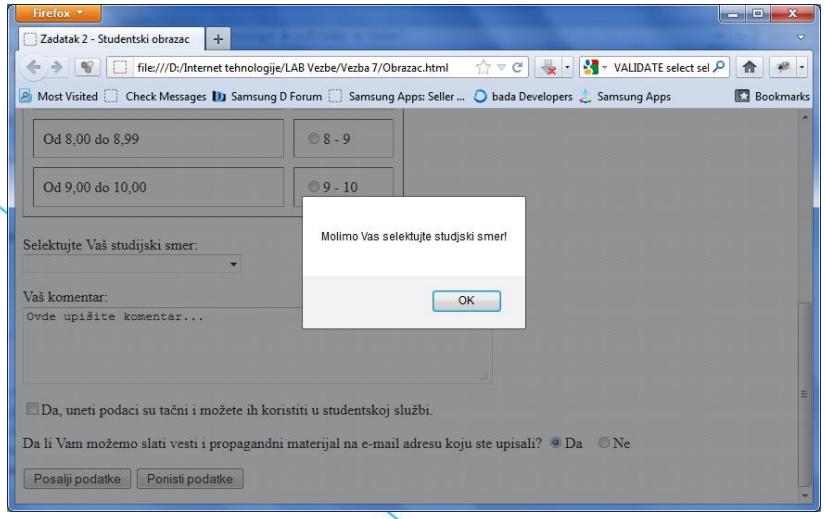



```
//*********************************************************
    //**          FUNKCIJA ProveraCheckBox-a
    //**
    //**     Proverava da li je oznacen checkbox,
    //**           Ako nije vraca vrednost false.
//*********************************************************


      if(Forma.YesNoCheckbox.checked == false){
         alert("Molimo Vas označite da su uneti" +
               "podaci tačni!");
         return (false);
      }
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**Zadatak 4:** Izraditi funkcije za proveru unetih podataka u padajućim listama.

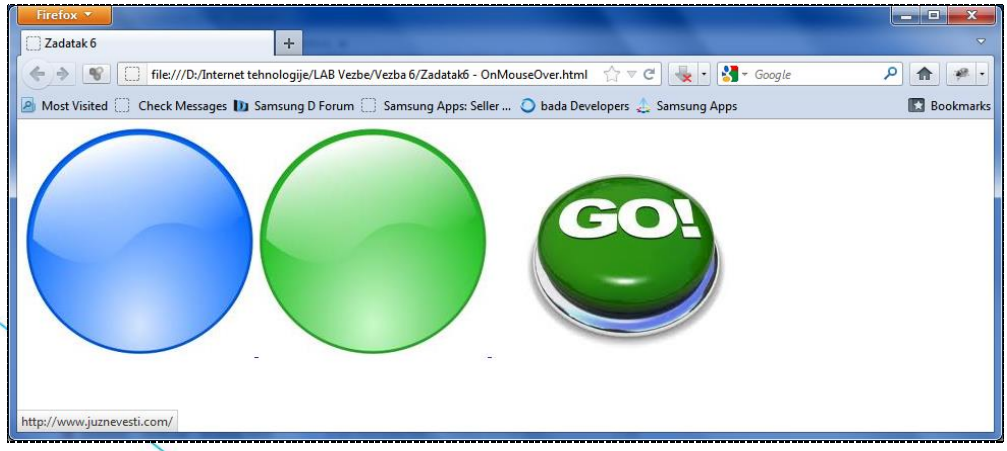



```
//*********************************************************

    //**          FUNKCIJA ProveraPadajuceListe

    //**

    //** Proverava da li je izabran smer iz padajuce liste,

    //**       Ako je selektovana prva stavka (prazna)

    //**                    vraca vrednost false.

    //*********************************************************

      if(Forma.Group.options[0].selected == true){

         alert("Molimo Vas selektujte studjski smer!");

         return (false);

      }
```

**Zadatak 5:** Kreirati HTML stranu koja će koristiti kao hiperaktivni meni, ali tako da se slika menja kada se pređe pokazivačem miša preko slike koja predstavlja stavku menija. Kada se pokazivač skloni vraća se originalna slika. Efekat je prikazan na sledećoj slici.



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Zadatak 6</title>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

  <script type="text/javascript" language="JavaScript">
      <!--
      function postaviPozadinu(redniBrSlike){
        putanjaPozadine = "slike/go.jpg";
        document.images[redniBrSlike].src=putanjaPozadine;
      }

      function ukloniPozadinu(redniBrSlike){
        putanjaSlike = "slike/slika" + redniBrSlike + ".jpg";
        document.images[redniBrSlike].src=putanjaSlike;
      }
      //-->
  </script>
</head>
```

```
<body>
<a href="http://www.blic.rs">
<img src="slike/slika0.jpg" border="0"
onMouseOver="postaviPozadinu(0)" onMouseOut="ukloniPozadinu(0)"/>
<a/>
<a href="http://www.novosti.rs">
<img                    src="slike/slika1.jpg"                    border="0"
onMouseOver="postaviPozadinu(1)" onMouseOut="ukloniPozadinu(1)"/>
<a/>
<a href="http://www.juznevesti.com">
<img src="slike/slika2.jpg" border="0"
onMouseOver="postaviPozadinu(2)" onMouseOut="ukloniPozadinu(2)"/>
<a/>
</body>
</html>
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**Pitanja i zadaci**

1. Na koji način se kreiraju komentari unutar JavaScript-a? Navesti primer jednorednog i višerednog komentara.

2. Napišite kod za dodavanje spoljašnjeg JavaScript dokumenta pod nazivom `Primer2` jednoj .html strani. Unutar kod taga treba smestiti ovaj kod.

3. Da li spoljašnji JavaScript fajla treba da sadrži `<script>` tag?

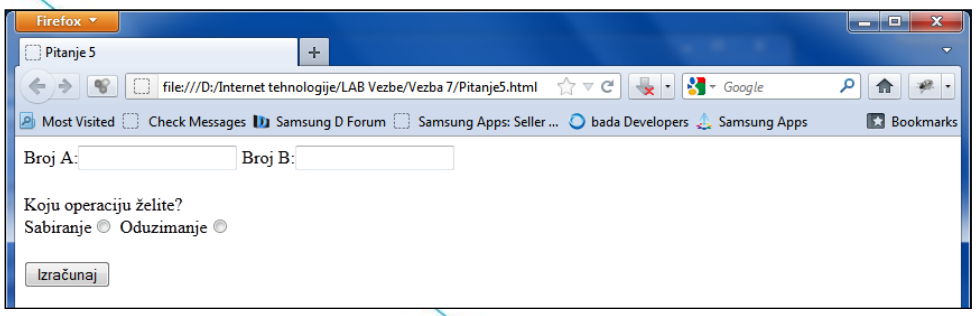
4. Deklarisati niz od 10 članova pod nazivom `MojNiz1`. Definisati vrednost prvog, trećeg i petog člana sa `14`, `18` i `25` respektivno.

5. Deklarisati niz od 10 članova pod nazivom `MojNiz2`. Definisati da vrednost svakog člana niza bude jednaka njegovom indeksu (`0,1,...,9`).



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

6. Napisati JavaScript koji će za unete brojčane vrednosti u dva tekstualna polja, sračunati zbir ili razliku u zavisnosti od označenog tekst box-a. Rezultat ispisati na web strani. Slika i .html kod su dati u nastavku.




.html strana

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
      <title>Pitanje 5</title>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <script type="text/javascript" src="pitanje5.js"></script>
</head>
<body>
<form name="Forma">
 Broj A:<input type="text" name="poljeA"/>
 Broj B:<input type="text" name="poljeB"/>
 <br /><br />
```



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
 Koju operaciju želite? <br />
 Sabiranje<input type="radio" name="Operacija"/>
 Oduzimanje<input type="radio" name="Operacija"/>
 <br /><br />
 <input type="button" value="Izračunaj"   onClick="Racunanje(Forma)"/>
 <br /><br />
 <div id="Rez"></div>
 </form>
</body>
</html>
```

pitanje5.js



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 5. Literatura


1. J. N. Robbins, „Naučite Web dizajn“, O`Reilly, Mikro knjiga – prevod, 2008.

2. J. Hill, J. A. Brannen, „briljantno HTML5 i CSS3“, PortaLibris, CET – prevod, 2011.

3. K. Jamsa, K. King, A. Anderson, „HTML i Web dizajn“, Osborne, McGraw Hill, Mikro knjiga – prevod, 2003.

4. R. Greenlaw, E. Hep, „In-line/On-line: Fundamentals of the Internet and World Wide Web“, McGraw Hill, 2002.

5. Z. Veličković, „Internet tehnologije: Praktikum laboratorijskih vežbi”, XPRINT, Niš 2009.

6. W. Choi, „Od početka PHP”, Wrox Press Ltd, CET-prevod, 2002.

7. V. Milutinović, I. Vujović, “Advances in the Internet technology: concepts and systems”, Academic Mind, Belgrade, 2003.

8. M. Millholn, J. Castrina, “Faster Smarter Web Page Creation”, Microsoft Press, CET-prevod, 2003.

9. T.A. Powell, „Web dizajn: kompletan priručnik”, Osborne, McGraw Hill, 2000.

10. D. Goodman, “JavaScript Biblija”, IDG Books

11. H.E Wiliams, D. Lane, „Web aplikacije i baze podataka: PHP I MySQL”, Mikro knjiga

12. M. Stanković, D. Radulović, D. Ristić, „Mrežni servis World Wide Web”, Elektronski fakultet Niš, 1996.

13. N. Pits, „Osnove XML”, Kompjuter biblioteka 2000.

14. J.C. Meloni, „Naučite MySQL za 24 časa”, Kompjuter biblioteka 2002.

15. http://www.w3schools.com, januar 2012.

16. http://w3.org, jaunar 2012.

17. B. Lawson, R. Sharp, „HTML5 za programere“, Mikro knjiga, Beograd 2012.




Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# WEB PROGRAMIRANJE 2

**Predavač: Zoran Veličković**

**Ciljevi premeta:**

Izučavanje naprednih programerskih Web tehnologija za razvoj bogatih Internet aplikacije RIA (engl. *Rich Internet Applications*) baziranih na ASP .NET-u.

**Ishodi predmeta:**

Očekuje se da će nakon završenog kursa polaznici:

1. Biti u stanju da razviju Web aplikacije u ASP .NET MVC 4 okruženju.
2. Potpuno ovladti programerskim veštinama koje se primenjuju na strani klijenta/servera.
3. Moći da analiziraju detalje programa i kvalitet programskog rešenja.

**Teorijska nastava:**

1. Osnove Interneta. Klijent - server arhitektura. Web stranica, Web aplikacija. Statičke-dinamičke Web stranice. HTML, XML, XHTML. Arhitektura Web aplikacija. Programiranje na strani klijenta. JavaScript. Programiranje na strani servera. PHP, ASP, ASP .NET.

2. Microsoft-ove tehnologije i Web programiranje. ASP .NET razvojna platforma M.S. Visual Studio (VS) kao intergrisano razvojno i izvršno okruženje. Programski jezik C# i klase za razvoj Web aplikacija. HTML serverske kontole. Serverske Web kontrole. Web Pages. RAZOR. MVC (engl. *Model-View-Controller*).

3. MVC 4 (Model-View-Controller) koncept Web aplikacije. Benefiti korišćenja. Anatomija MVC Web aplikacije u VS-u. Kontroleri u MVC. Rute u MVC-u. Klasa Controllers. Klasa ActionResult. Kreiranje MVC Controlera u VS-u. Formiranje ruta u MVC-u. Formiranje akcionih metoda. Views. View engine. Razor View engine. Helper metode. ViewData i ViewBag metode. MVC i mobilne aplikacije.

4. Modeli u MVC-u. Baze podataka i MVC. Microsoftov koncept MVC modela. Razvoj Data Model-a. Entity Framework Model. Business Model. Model Binding. Validacija podataka u MVC-u. AJAX i JQuery tehnologije. Ručna validacija podataka. Validacija sa Data Annotations. JQuery selektori. Obrada događaja sa jQuery. Ajax forme. Ajax ActionLinks. AjaxRequest sa jQury. Bezbednost. Autentifikacija i autorizacija.

5. Testiranje i instaliranje MVC aplikacije na serveru.

**Praktičane vežbe:**

**Od 40% do 50%** radnih sati.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Instalacija i podešavanje Visal Studia 2013 za razvoj Web aplikacija. Anatomija Web aplikacije u VS-u. Tipovi fajlova koje generiše VS pri razvoju MVC 4 Web aplikacije.
- Kreiranje MVC Controlera u VS-u. Formiranje ruta u MVC-u. Formiranje akcionih metoda.
- Razvoj Internet aplikacije sa više View-a za podršku desktop i mobilnim aplikacijama u VS-u.
- Modeli u MVC-u. Baze podataka i MVC. Microsoftov koncept MVC modela. Razvoj Data Model-a. Entity Framework Model. Business Model. Model Binding.
- Ručna validacija podataka. Validacija sa Data Annotations. JQuery selektori. Obrada događaja sa jQuery. Ajax forme. Ajax ActionLinks. AjaxRequest sa jQury.
- Bezbednost. Autentifikacija i autorizacija.
- Testiranje i instaliranje MVC aplikacije na serveru.

**Sadržaj kursa:**

Predavač će kompletirati kurs izabranim poglavljima prema sopstvenom nastavnom planu i programu.

**Trajanje kursa:**

Kurs će trajati 30 školskih časova (26 za predavanja i 4 za testove). Predavanja će biti završena u blok sistemu u jednom mesecu (pola dana petak, pola dana subota)

**Kurukulim:**

U prvoj nedelji kursa će biti objašnjen koncept Web programiranja. Generalno će biti razmatrane programerske tehnologije na strani klijenta i na strani servera. Poseban akcenat će biti stavljen na programske jezike C# i JavaScript, koji su podržani ASP.NET tehnologijom. Nivo prezentacije će biti prilagođen prethodnim znanjima kandidata.

Koncept MVC-a koji podržava Microsoftova verzija VS-a 2013 će biti prikazan na jednostavnom primeru gde će se videti sve prednosti koje on donosi. Detaljno će biti izučena anatomija MVC Web aplikacije koje se formira u VS 2013. Prvi test će se sprovesti na kraju ove sedmice.

U drugoj nedelji će biti obrađene osnovne komponente MVC-a: Controller i View. Kroz praktičan primer će se proučavati i implementirati ove komponente koje su deo šablona Internet application koja ima podršku i za desktop i mobilne uređaje.

U trećoj nedelji će biti obrađena treća komponenta MVC-a - Model. Kroz praktičan primer će biti implementiran poslovni model i biće razmatran rad MVC-a sa bazama podataka. Drugi test će biti sproveden krajem ove nedelje.

Četvrta nedelja je posvećena podršci MVC-a AJAKS-u i jQueri tehnologijama. Za ove potrebe biće razvijene specijalizovane klase razvijene da podrže ove tehnologije. Biće realizovan obrazac koji je podržan od strane ovih tehnologija. Završni test će biti sprovedena krajem ove nedelje.

**Ocenjivanje:**

1. *Jednostavna MVC aplikacije*. Studenti treba da naprave jednostavnu MVC aplikaciju na osnovu MVC šablona Internet aplikacija. Projekat će se bodovati od 1 do 3. Student je prošao test samo ako dobija ocenu 2 ili 3.
2. *Poslovni model MVC 4 aplikacije*. Ovaj deo kursa će biti vrednovan ocenom od 1 do 3. Student je prošao test samo ako dobija ocenu 2 ili 3.
3. Konačni ABCD test. Student će proći test ako je tačno odgovorio više od 50% pitanja.

Student će uspešno završiti kurs samo ako je položio sva tri testa.

**Testovi**

Test pitanja i teme projektnih zadatka neće biti objavljeni iz razloga poverljivosti.

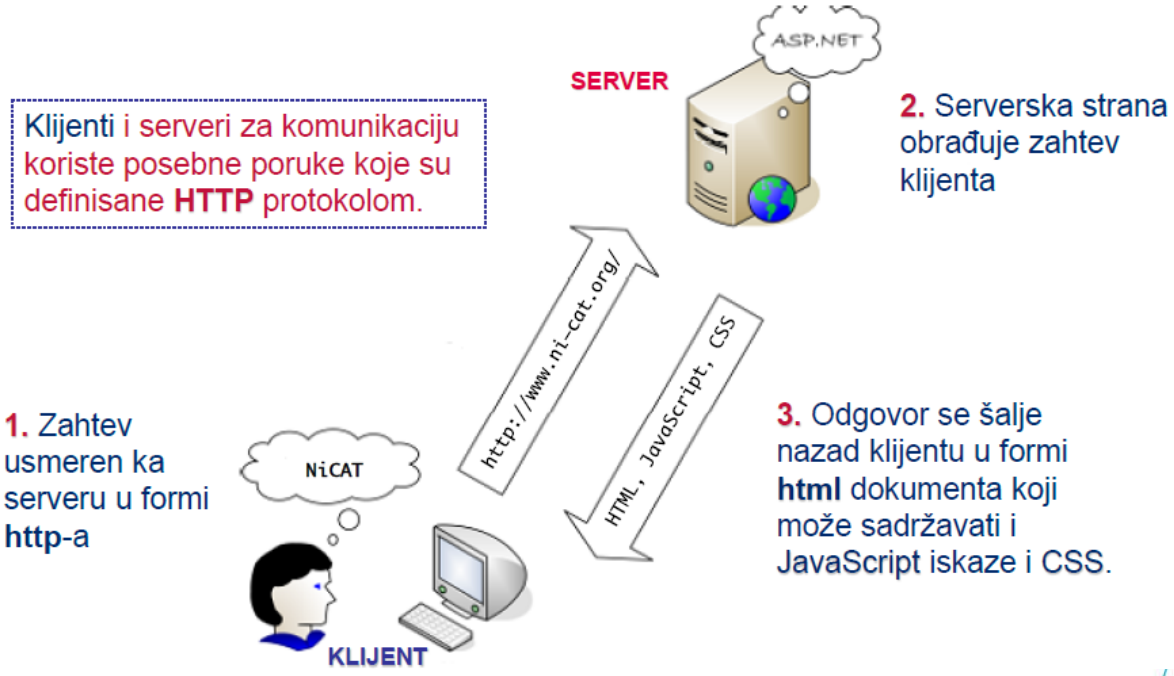


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 1. Uvod

Internet je globalna računarska mreža koja koristi TCP/IP stek protokola za međumrežnu komunikaciju. Sa druge strane, poznata je činjenica da je Internet zasnovan na klijent-server tehnologiji koja podrazumeva razmenu podataka između dve krajnje komunikacione tačke. Jedna komunikaciona tačka se naziva klijentom (započinje proces komunikaciije slanjem upita), dok se druga naziva serverom (pruža odgovore na postavljena upite). Klijenti i serveri za međusobnu komunikaciju koriste posebne poruke (metode) koje su definisane HTTP protokolom. Na slici 1.1. je prikazana bazična klijent-server tehnologija koja je realizovana u 3 koraka:

1. Klijent upućuje zahtev usmeren ka serveru u formi HTTP upita.
2. Serverska strana obrađuje zahtev klijenta.
3. Odgovor se šalje nazad klijentu u formi .html dokumenta koji može sadržavati JavaScript iskaze i CSS.



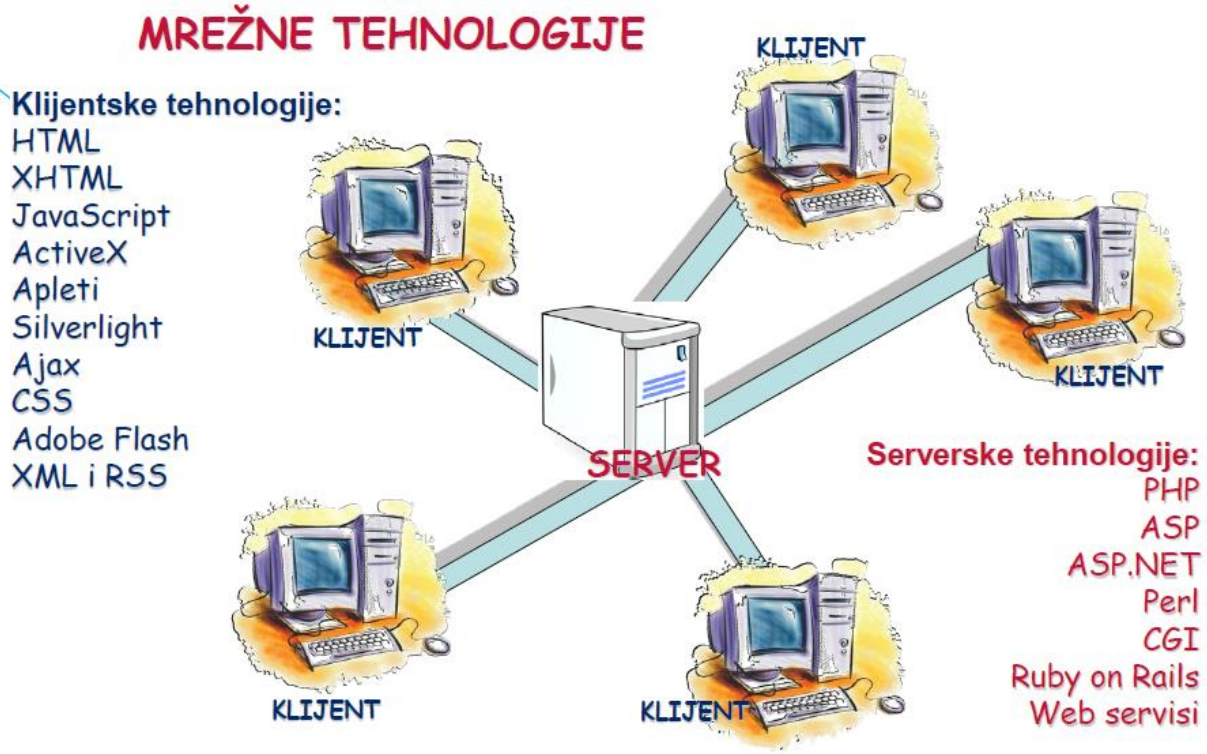

*Slika 1.1. Klijent – server arhitektura na kojoj je baziran Internet*

Za potrebe realizacije klijent server-tehnologije razvijeno je mnoštvo programerskih tehnologija i altata. Prema prikazanoj arhitekturi, jasno je da se ovi alati i tehnologije mogu globalno podeliti na



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

one koji se odnose na klijentsku, odnosno, na serversku stranu Internet komunikacije. Na slici 1.2. su prikazane neke od programerskih tehnologija koje su jasno svrstane na serversku ili klijentsku stranu. U klijentske tehnologije se mogu ubrojati: HTML, XHTML, JavaScript, ActiveX, Apleti, Silverlight, Ajax, CSS, Adobe Flash, XML i RSS, dok se u serverske tehnologije mogu svrstati: PHP, ASP, ASP.NET, Perl, CGI, Ruby on Rails, Web servisi, ...




*Slika 1.2. Klijent – server arhitektura na kojoj je baziran Internet*

Podela prikazanih tehnologija je izvršena na osnovu mesta izvršavanja programskog koda. Ovo zapravo znači da se klijentske tehnologije izvršavaju na klijentskom računaru, dok se serverske tehnologije izvršavaju na serveru. Ovo ima za posledicu da programiranje Web aplikacije zahteva kreiranje dela Web aplikacije na klijentskoj, odnosno na serverskoj strani. Benefiti ove podele su brojni, ali za programere je značajno da se u ovom konceptu Web aplikacija može razvijati i testirati parcijalno.

Da bi Web aplikacije imale funkcionalnost desktop aplikacija (što je glavni uslov zadovoljstva korisnika Web aplikacijom) uobičajeno je korišćenje više programerskih tehnologija i na klijentskoj i na serverskoj strani.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Na slici 1.3. su prikazane karakteristike klijentskog modela, dok je na slici 1.4. prikazan serverski programski model.

Klijentski programski model se karakteriše sledećim osobinama:

- **Klijentske** tehnologije **ne zahtevaju** izvršavanje programa na serverima.
- Korisnik preuzima **kompletnu** aplikaciju i prebacuje je u svoj Web čitač (engl. Browser) gde se ona izvršava u lokalu.
- Problem **različitog** izvršnog klijentskog okruženja predstavlja potencijalni problem.

*Slika 1.3. Klijentski programski model*

Serverski programski model se karakteriše sledećim osobinama:

- **Servrski programski model** ne zahteva izvršavanje programa na klijentu (zahteva se samo prezentacija sadržaja).
- U ranim danima Web programiranja serveri su morali da pokrenu novu, **zasebnu instalaciju aplikacije** za svaki primljeni zahtev.
- Dobra strana **Web programiranja** serverskog modela je što aplikacije ne zahtevaju instalaciju!



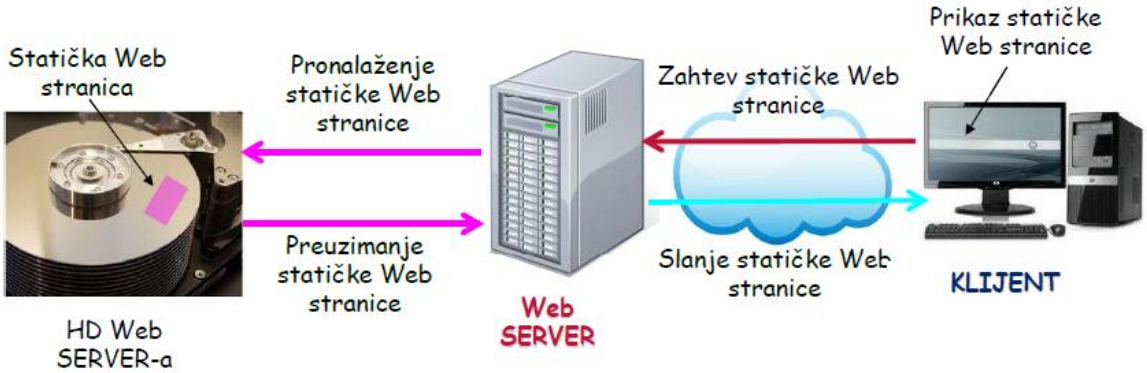
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

*Slika 1.4. Serverski programski model*

U ranim danima Web programiranja Web stranice su bile statičke, dakle već dizajnirane i smeštene na hard disk servera. Prilikom prijema zahteva za pristupom statičkoj stranici ona se nalazi na hard disku i kao takva prosleđuje klijentu.

Sa druge strane, koncept programiranja Web aplikacija na serverskoj strani je omogućilo kreiranje Web stranica tek u trenutku postavljanja upita. Takve Web stranice se nazivaju dinamičkim Web stranicam. Dakle, tražena Web stranica **nije** unapred dizajnirana, već se to obavlja u trenutku realizacije upita.

Načini dobavljanja statičkih i dinamičkih Web starnica su prikazani na slikama 1.5. i 1.6. respektivno. Sa slika se jasno može uočiti razlika između mesta izvršavanja programskog koda.

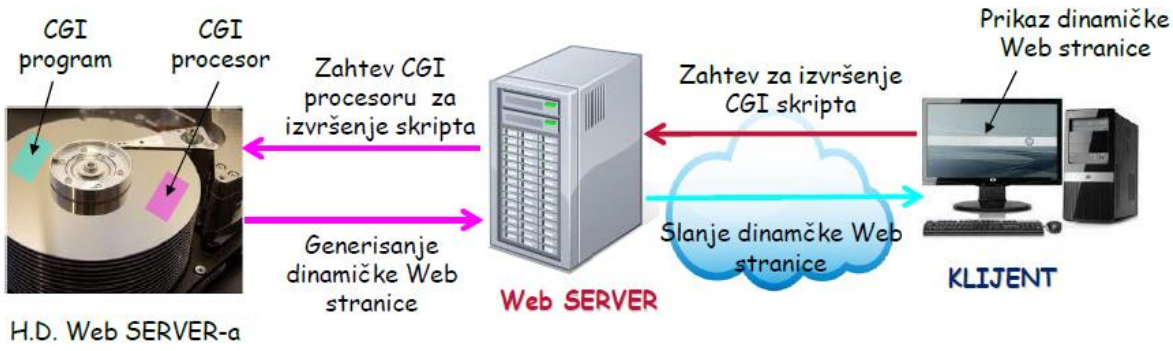



*Slika 1.5. Dobavljanje statičke Web stranice*

Primetite da se prilikom dobaljanja statičke Web stranice ne izvršava nikakav programski kod na serveru, već se samo pronalazi i prosleđuje zahtevani resurs. Niz akcija za dobavljanje statičke Web stranice je: Zahtev statičke Web stranice - Pronalaženje statičke Web stranice - Preuzimanje statičke Web stranice - Slanje statičke Web stranice - Prikaz statičke Web stranice.



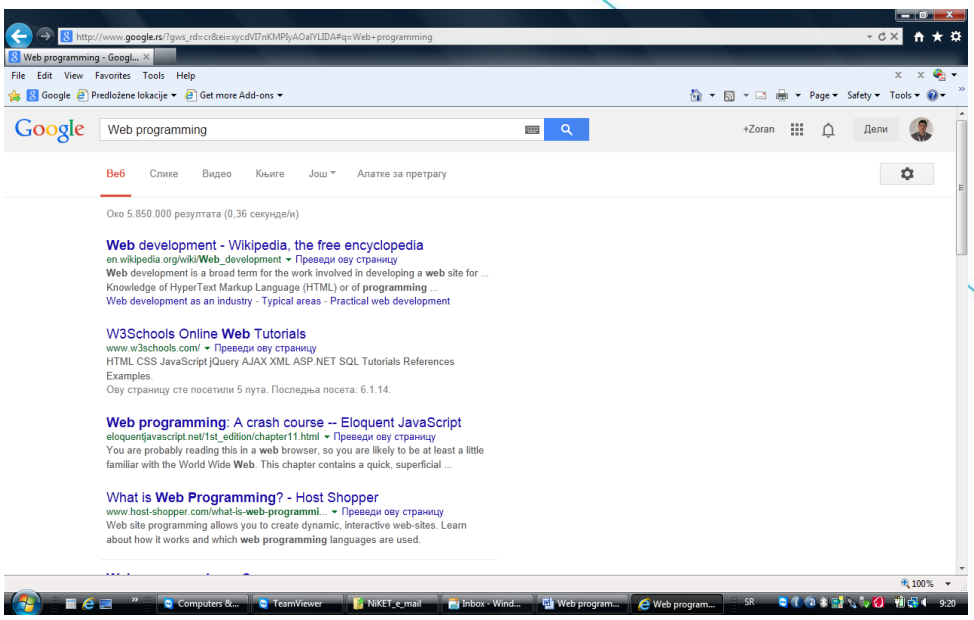
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Slika 1.6. Dobavljanje dinamičke Web stranice*

Niz akcija za dobavljanje dinamičke Web stranice je: Zahtev dinamičke Web stranice - Zahtev za izvršavanjem skripta – Generisanje dinamičke Web stranice - Slanje statičke Web stranice - Prikaz dinamičke Web stranice.

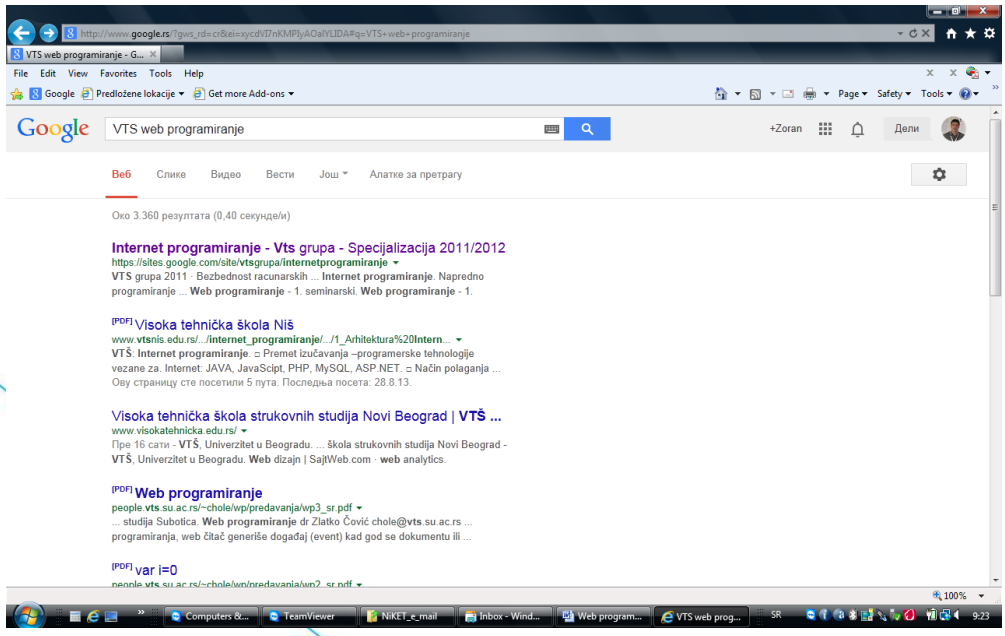
Treba uočiti da se prilikom dobaljanja dinamičke Web stranice izvršava CGI programski kod na serveru, što ima za posledicu generisanje Web stranice u trenutku upita. Benefit koji se ovim može ostvariti je adaptacija Web stranice prema karakteristikama Web čitača, odnosno, formiranje Web sadržaja na osnovu upita klijenta. Očigledan primer ovog koncepta je specifičan zahtev upućen Google-u. Google vraća Web stranicu koja je specifična i određena sadržinom upita. Dakle, Web stranica nije već pripremljena i smeštena na hard disk servera, već se izgrađuje u vreme izvršenja. Na slici 1.7. i 1.8. prikazani su izgledi Web stranice u funkciji sadržine upita.




*Slika 1.7. Sadržaj dinamičke Web stranice za upit Web programming.*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Slika 1.8. Sadržaj dinamičke Web stranice za upit VTS Web programming.*

Savremene Web aplikacije su zasnovane upravo na konceptu dinamičkih Web stranica. Na ovom kursu će biti razmatrane Microsoft-ove tehnologije za dizajn atraktivnih Web aplikacija.

Za razvoj Web aplikacija biće korišćeni Microsoft-ovi razvojni alati koji su sadržani u Visual Studiu. Posle upoznavanja osnovnih koncepata, do kraja kursa biće izgrađena Web aplikacija u MVC razvojnom okruženju koje podržava Visual Studio 2012 i 2013.

Za uspešan rad na ovom projektu je potrebno znanje i objektno-orijentisanog programiranja, a posebno poznavanje .NET tehnologije i programskog jezika C#. Pored znanja C#, neophodno je poznavanje i osnova sledećih Internet tehnologija: JavaScripta, CSS-a, HTML–a 5 kao i arhitekture dinamičkih Web stranica. Dodatna literatura koja pokriva ove oblasti data je u spisku korišćene literature.

# 2. Osnove ASP .NET tehnologije

Microsoft je svoje viđenje računarskeih tehnologija u budućnosti pretočio u koncept pod nazivom **.NET** (objavljen 2000-te). **.NET** je skup proizvoda i servisa koji omogućavaju rad sa heterogenom računarskom opremom.

**.NET** kombinuje dve važne mrežne tehnologije:

- **HTML** sa njegovim izuzetnim prezentacionim mogućnostima i
- **XML** sa njegovim velikim potencijalom za rad sa metapodacima.

Razvojno okruženje **.NET Framework** obuhvata pet tehnologija:

- **Visual Studio .NET** - razvojno i izvršno okruženje;
- **CLR** (engl. Common Language Runtime), zajednički izvršni podsistem za jezike koji opslužuje aplikacije u trenutku izvršavanja;
- **Programski jezici** koji koriste izvršni podsistem **CLR**-a i biblioteke klasa;
- **F(B)CL** - biblioteke klasa (engl. Framework (Base) Class Library);
- **MS ASP.NET** tehnologija za razvoj Web aplikacija i Web servisa.

Za nas je u ovom trenutku značajna podrška razvoju Web aplikacija. Na platformi .NET-a je razvijeno i nekoliko tehnologija za podršku dizajna Web stranica, odnosno, Web aplikacija. Osnovni koncept ASP .NET-a se može sagledati kroz nekoliko činjenica:

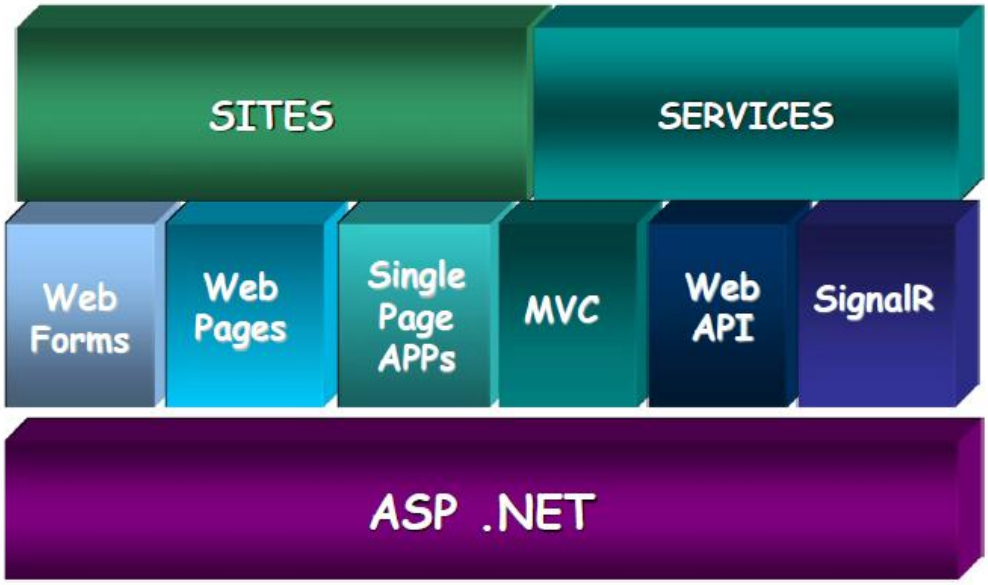
- **ASP .NET** je bazična platforma na kojoj funkcionišu (i rade) sve **ASP .NET** aplikacije.
- **ASP .NET** je bazična tenologija orijentisana ka Web-u zasnovana na otvorenim programerskim standardima: **XML, XHTML**, **HTTP**, **SOAP**, **JSON** i **WSDL**.
- Iako je Microsoft najčešće razvijao sopstvene programerske tehnologije, uvideo je značaj podrške otvorenim programerskim tehnologija u svojim proizvodima!
- Tako je Microsoft razvio **tri** bazične tehnologije za izgradnju Web aplikacija na platformi **ASP .NET**-a:
    1. **ASP .NET Web Forms** (originalni programski model)
    2. **ASP .NET Web Pages**
    3. **ASP .NET MVC**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Na slici 2.1. je prikazana struktura Microsoft-ovog koncepta razvoja Web aplikacija. Sa slike 2.1 treba primetiti da se u osnovi svih MS tehnologija za Web nalazi .NET, odnosno ASP .NET.

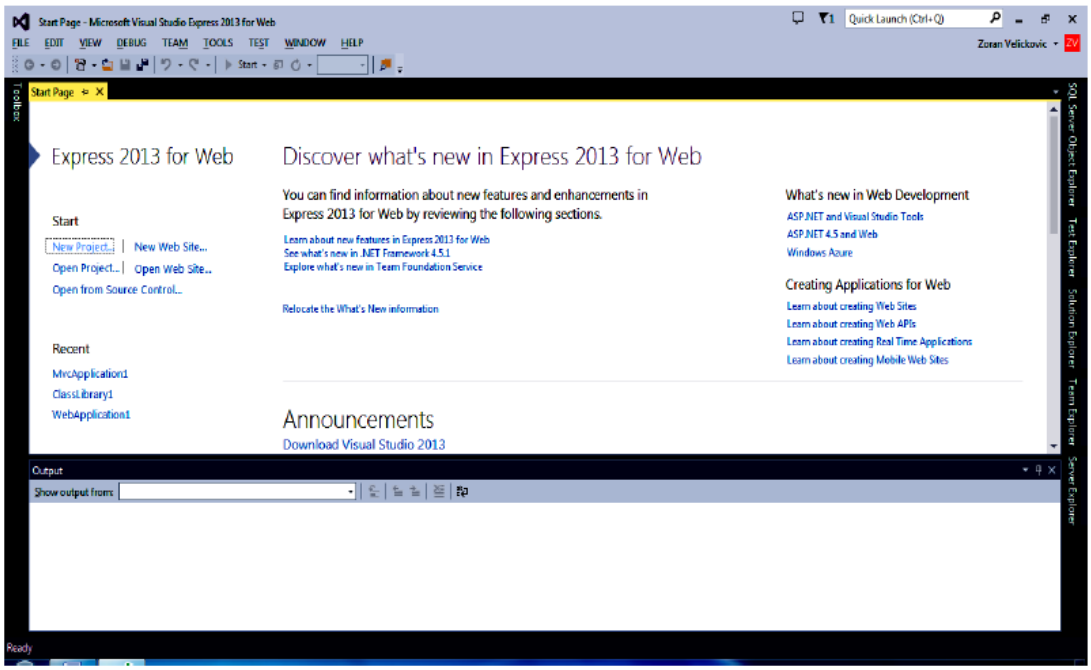
- **ASP .NET** je Web platforma koja obezbeđuje sve neophodne servise za izgradnju serverski orijentisanih Web aplikacija.
- **ASP.NET** je izgrađen na **.NET** framework-u, tako da su svi servisi **.NET**-a dostupni u ASP .NET aplikacijama.
- Aplikacije mogu biti pisane na **bilo kom** programskom jeziku koji je kompatibilan sa **CLR**-om (engl. Common language runtime), što uključuje Visual Basic .**NET** kao i **C#**.




*Slika 2.1. ASP.NET baza za sve Web tehnologije*

- Za kreiranje **ASP.NET** Web aplikacija najčešće se koristi **Visual Studio**.
- Programerski alati koje poseduje **Visual Studio** su posebno kreirani za razvoj Web aplikacija i poznati su pod imenom **Visual Web Developer**.
- **Visual Web Developer Express** je besplatna razvojna platforma koja uključuje osnovni skup alata za dizajn Web stranica iz Visual Studija.

Izgled prve stranice **Visual Web Developer Express** 2013 je prikazan na slici 2.2.

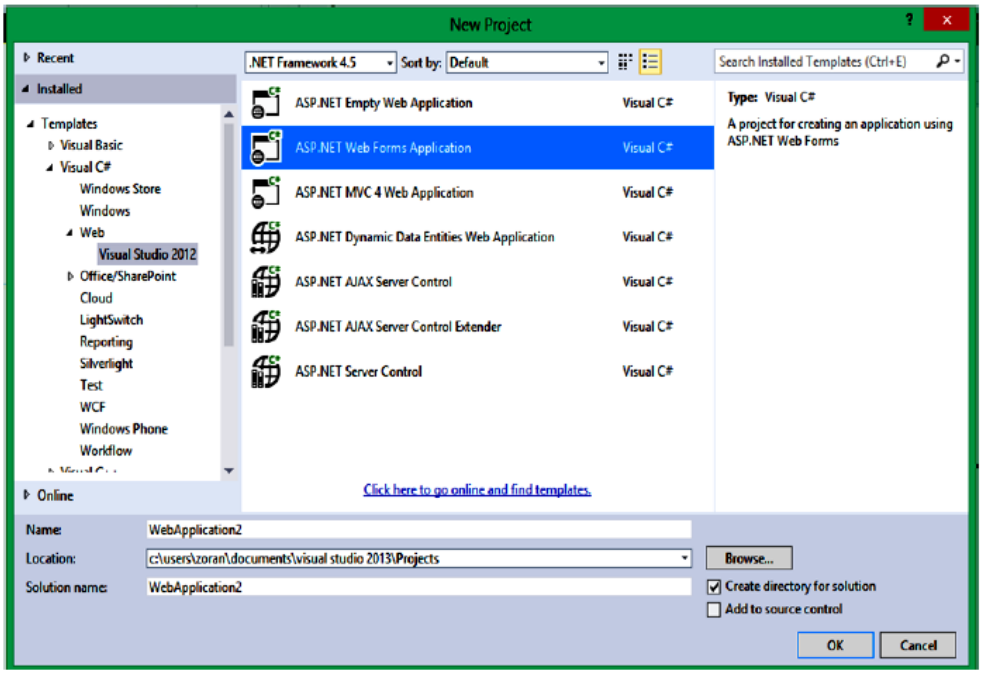



*Slika 2.2. Izgled prve stranice Visual Web Developer Express 2013*

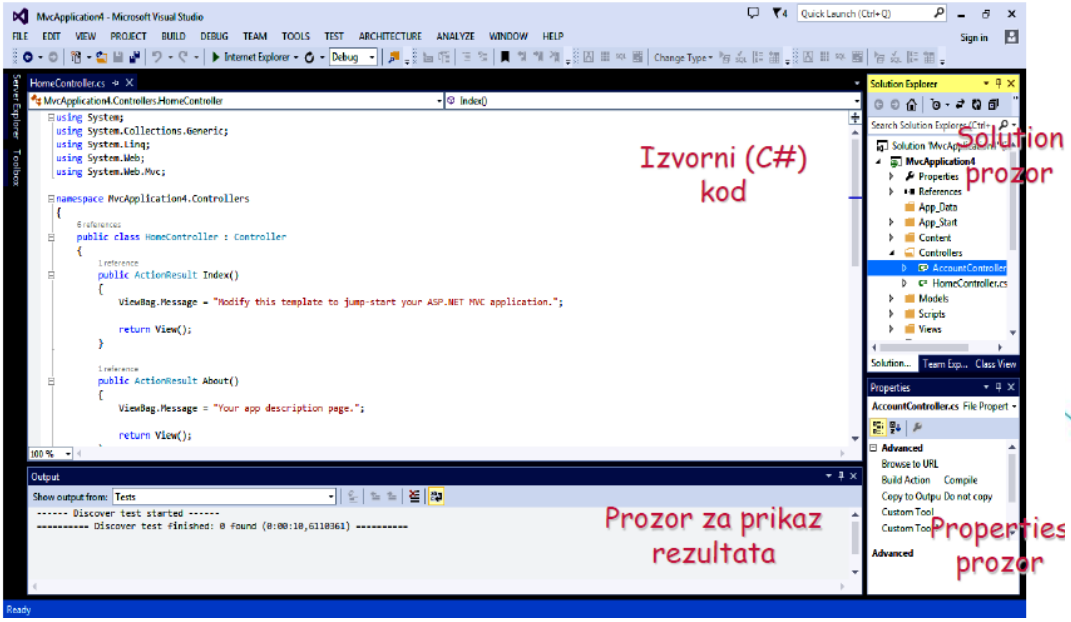
Na slici 2.3 prikazane su tipovi podržanih projekata u Visual Studiu 2013 za razvoj Web aplikacija. Na ovom kursu iz Web programiranja najviše pažnje je posvećeno razvoju Web aplikacija u ASP .NET MVC 4 okruženju. Naravno, pored ove klase aplikacija, podržani su i ostale klase Web aplikacija kao što su **ASP .NET Web forms**, **ASP .NET Web Pages** i **RAZOR** i **ASP .NET MVC**. Na slici 2.4. su prikazani su delovi osnovnog ekrana iz Visual Studia pri razvoju (Web) aplikacija.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Slika 2.3. Tipovi podržanih projekata u Visual Studiu 2013 za razvoj Web aplikacija*




*Slika 2.4. Izgled razvojnog i izvršnog okruženja Visual Studio 2013.*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Na slici 2.4. se izdvajaju važne celine neophodne za razvoj aplikacija u Visual Studiu. Označeni su delovi ekrana stranice koja se odnosi na izvorni kod, tzv. Solution prozor, tzv. Properties prozor i prozor za prikaz rezultata (kompajliranja ili izvršenja). U nastavku će ukratko biti prikazane najvažnije karakteristike svake od MS tehnologija za razvoj Web aplikacija.

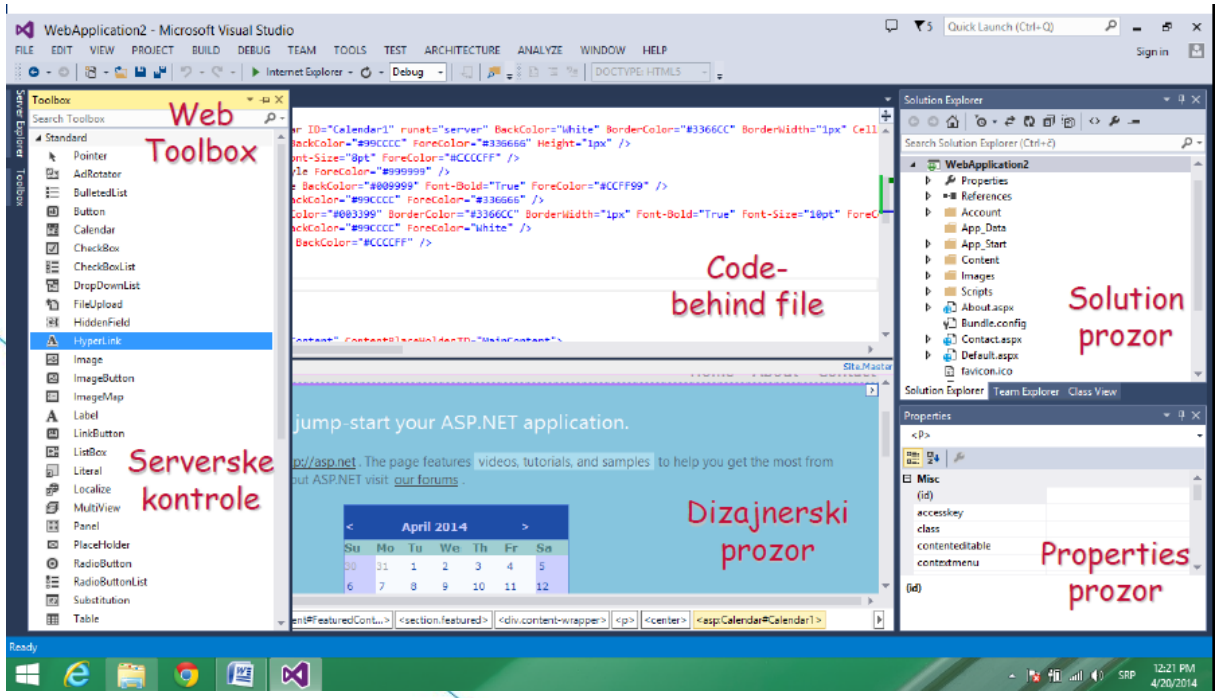
**ASP .NET Web forms**

- **ASP .NET Web Forms** omogućavaju razvoj Web aplikacija na isti način na koji se kreiraju klasične **Windows Forms** aplikacije.
- Iz **Web Toolbox**-a Visual Studio-a se povlače **serverske kontrole** koje se postavljaju na Web stranicu.
- Ovaj koncept podrazumeva interfejs baziran na serverskim kontrolama odnosno, fajlovima koji ih prate:
    - **User interface** (ui) fajl: uključuje jezik za označavanje Web dokumenta kojim se definiše izgled Web stranice i koje se Web kontrole koriste. Ekstenzija ovog fajla je **.aspx**.
    - **Code-behind fajl**: uključije programski kod koji opslužuje događaje i interakcije između objekata na Web stranici. Ova vrsta fajla ima ekstenziju **aspx.cs** ako se razvoj aplikacije obavlja u programskom jeziku **C#**.

Na slici 2.5. prikazan je proces razvoja **ASP .NET Web forms** aplikacije u Visual Studiu. Na levoj strani se nalazi Web Toolbox u kome se nalaze kontrole karakteristične za beb dizajn odakle ih dizajner povlači na radnu površinu, što ima za posledicu formiranje izvornog koda Web stranice.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Slika 2.5. Izgled razvojnog okruženja Visual Studio 2013 pri razvoju **ASP .NET W**eb forms aplikacija.*

**ASP .NET WEB PAGES i RAZOR**

**ASP .NET Web Pages** predstavlja MS framework za izgradnju Web aplikacija kod koga se koristi **RAZOR** sintaksa za dizajniranje izgleda Web stranice. **ASP.NET RAZOR** koristi jednostavnu programsku sintaksu koja omogućava **ugradnju** serverski-baziranog koda u Web stranice.

- **RAZOR** stranice imaju ekstenziju **.cshtml** kada se za razvoj aplikacije koristi programski jezik **C#** (mi ćemo koristiti ovu verziju), odnosno **.vbhtml** ako se koristi Visual Basic .NET.
- Zbog toga što se kod ugrađuje u samu Web stranicu, u ovom konceptu **RAZOR**-a ideja „code-behind file“ **nije primenjiva**! Za razliku od **ASP .NET Web forms,** povlačenje serverskih kontrola u Web stranicu NIJE MOGUĆE! Ako bolje pogledate primetićete da je Web Toolbox prazan.
- Dakle, **RAZOR** stranice su sačinjene od **HTML** koda i **ne koristi se** serverske kontrole kao u Web Formama (primer: **<asp:Button />**).
- Za kreiranje **ASP .NET Web Pages** se može koristiti Visual Studio, ali je osnovni alat za to **MS Webmatrix 2**.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**ASP .NET MVC**

ASP.NET **MVC** je **slobodano** i potpuno podržano radno okruženje (engl. framework) za izgradnju Web stranica zasnovanih na šablonu (engl. pattern) **Model-View-Controller**. Kao i ASP.NET Web Forms, ASP.NET MVC je izgrađen na vrhu **ASP.NET** framework-a. Koncept .NET-a omogućava da mnogi API-i koji su korišćeni u tradicionalnim ASP.NET Web Forms aplikacijma su sada **dostupn**i i ASP .NET MVC konceptu.

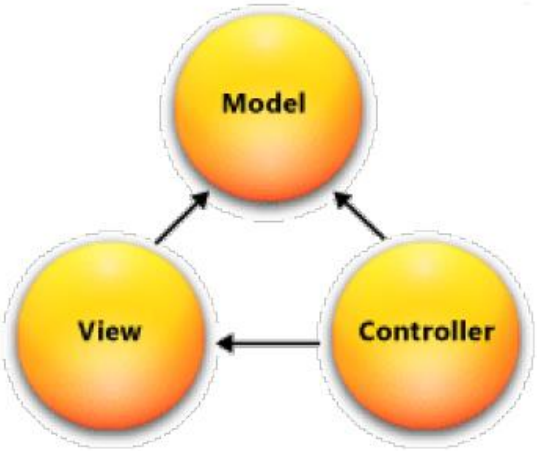
- Osnovni benefit koji donosi "**patern dizajn**" je izolacija programske logike od interakcije korisnika.
- **Patern dizajn** uspostavlja jasnu arhitekturu aplikacije što ima uticaja na poboljšano održavanje aplikacije i njeno testiranje.
- **Prva verzija** ASP.NET MVC-a se pojavljuje 13.03.2009. sa potpuno implementiranim MVC konceptom na bazi ASP.NET-a. Ova verzija poseduje routing engine, Helper methode, data binding ...
- **ASP.NET MVC 2** se pojavio u martu 2010. kada je dodat skup UI Helpers metoda, strongly typed HTML Helper metode; asynchronous controllers; validacija na klientu i serveru ...
- Deset meseci kasnije pojavljuje se **ASP.NET MVC 3**, koji uključuje **.NET 4**, data annotations (meta podaci) i **RAZOR** engine, **JavaScript** dobija podršku, **jQuery** validaciju i **JSON** binding.
- **ASP.NET MVC 4** dolazi u avgustu 2012., kada se uključuje u Visual Studio 2012.
- Oktobra 2013., pojavljuje se verzija **ASP .NET MVC 5** koja je podržana u Visual Studiu 2013.

**Model-View-Controller** (**MVC**) je arhitektura koja Web aplikaciju deli u **tri** osnovna dela: **Model**, **View** i **Controller**. **ASP.NET MVC** framework pruža *alternativu* ASP.NET Web Formama za kreiranje Web aplikacija. **ASP.NET MVC framework** je jednostavno, testibilno okruženje integrisano sa postojećim ASP .NET mogućnostima (kao što su *master pages* i ***membership*** – bazirana autentifikacija). **MVC** framework je definisan u *System.Web.Mvc* sklopu.

Na slici 2.6. je prikazana interakcija između osnovnih delova **MVC** Web aplikacije.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Slika 2.6. Interakcije između osnovnih delova Web aplikacije.*

**ZAHTEV ZA PROCESIRANJEM U ASP .NET MVC-u**

Jedna od najvažnijih koncepta u **ASP .NET MVC** aplikaciji je da **ne postoji** relacija između zahteva za nekom Web stranicom i fizičkog fajla na Web serveru. Kod tradicionalnih Web aplikacija (po MS-u Web Forms i Web Pages) svaki zahtev za stranicom se *preslikava* u poziv fizičkom fajlu na serveru. Tako, ako se traži *http://myapp/mypage.aspx*, Web server interpretira zahtev tako što u svom fajl-sistemu traži Web stranicu sa zahtevanim imenom, tj. *mypage.aspx*. Po pronalaženju traženog fajla, server vraća klijentu fajl sa **html** kodom. Potom, klijent interpretira Web sadržaj određen HTML kodom same stranice i prikazuje ga u prozoru Web čitača.

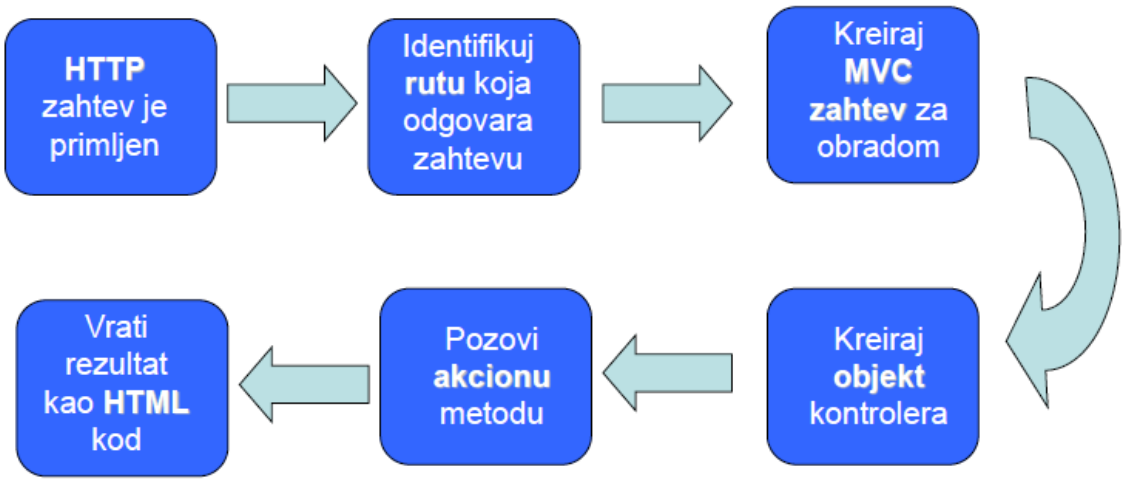
Na slici 2.7. prikazan je tok procesiranja zahteva za Web stranicom u MVC 4 kontekstu. U kontekstu **MVC** aplikacije, MVC komponenta koja se naziva „**routing engine**“, vrši proveru slaganja zahteva sa specificiranom **rutom**.

- **Ruta** se u **MVC** konceptu definiše korišćenjem **string uzorka** koji određuje način pronalaženja odgovarajućeg kontrolera, odnosno, klasa u kontroleru čije će metode konačno obraditi zahtev korisnika za resursom.
- Kada se **ruta** identifikuje, „routing engine“ kreira **poziv hendleru** čiji je zadatak da kreira objekt kontrolera sa metodama koje procesiraju korisnički zahtev.
- Metode koje obrađuju korisničke zahteve se nazivaju „**Akcione metode**“.
- Kontroler zapravo poziva neku od svojih akcionih metoda koja konačno obrađuje korisnički zahtev.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Kada se završi obrada zahteva, akcione metode generišu rezultat koji se vraća nazad klijentu.
- Najčešće je rezultat rada akcionih metoda **HTML kôd** (videćete kasnije i neke druge povratne rezultate) koji se potom prikazuje na klijentu.




*Slika 2.7.Procesiranje zahteva u ASP .NET MVC konceptu.*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 3. MVC kontroleri

Svaki pojedinačni HTTP zahev se opslužuje specifičnim **kontrolerom**. U ASP .NET MVC-u je implementiran koncept kontrolera pomoću .NET **klasa** koje poseduju **metode** za procesiranje HTTP zahteva. Već je rečeno da se te metode nazivaju akcionim metodama i one najčešće vraćaju objekte tipa **Actionresult**.

Procesiranje HTTP zahteva započinje mapiranjem zahteva u **MVC kontroler** i neku od njegovih **akcionih metoda**. Posle mapiranja odgovarajuće akcione metode ona se izvršava, a pozivaocu se vraća objekt tipa **Actionresult** kao rezultat rada akcione metode.

Jedan od najčešćih povratnih tipova akcionih metoda su objekti tipa **View** koji sadrže HTML kod koji se kasnije renderuje u klijentskom browseru.

**ROUTING ENGINE**

**Ruta** se formira od:

- **imena rute** (prvi parameta);
- **šablona rute** (drugi parametar) i
- **podrazumevane vrednosti šablona rute** (kao treći parametar).

- Ime rute je simbolička **oznaka** definisane rute.
- Šablon rute se sastoji od **tri** "čuvara mesta" (engl. placeholders), od kojih je svaki određen (omeđen) vitičasrim zagradama **{ }**.
- Ruting sistem nema nikakva saznanja o kontrolerima i njihovim metodama, već samo vrši njihovo **poređenje** zahteva sa definisanom rutom.

Ako ruta postoji, zahtev se dalje prosleđuje metodi u kontroleru na izvršenje.

- **Šablon rute** identifikuje ime **kontrolera** (pomoću svojstva placeholder) u koji se mapira **HTTP** zahtev (**{controller}**).
- **Akcioni metod** se identifikuje pomoću svojstva **action** placeholdera u klasi kontrolera i biće izvršen (**{action}**).
- Dodatni, treći, parametar identifikuje dopunske informacije - parametre koje koristi pozvana akciona metoda (**{id}).**
- Primetite da **HTTP** zahtev ne sadrži reč "***controller***" (po konvenciji) iako se ona nalazi u imenu klase.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Tako, http zahtev http://myapp/home/about se mapira u **home*controller*** klasu, dok je akcioni metod **about**.
- Primer: zahtev http://myapp/product/details?Id=1 se mapira u klasu **product*controller*** sa akcionom metodom **details** koja prihvata **id** parametar sa definisanom vrednošću.

**URL ŠABLON**

- URL šablononom se definišu **segment**i koji sačinjavaju korektan **URL** resurs.
- Segmenti od kojih su sačinjeni korektni **URL**-ovi su razdvojeni "slash" znakom (**/**), a "placeholder"-i su definisani imenom koje je okruženo vitičastim zagradama:

  **{controller}/{action}/{id}**
- Svaki **URL uzorak** se može formirati od jednog ili više placeholder-a odvojenih znakom crtice (-) (engl. hyphen).
- U sledećoj tabeli je prikazano nekoliko URL šablona.

| URL Pattern | Matching URL Request |
|---|---|
| {controller}/{action}/{id} | /Pet/Display/Fido |
| {section}/{language}-{country}/{operation} | /Admin/en-US/Users |
| blog/{postYear}-{postMonth}-{postDay}/{postName} | /blog/2013-04-15/MissingPetsSummary |

**ČITANJE DELOVA URL-a**

- Delovi segmenata iz URL zahteva se mogu pročitati korišćenjem *Page.RouteData* svojstva u telu Akcionih metoda koje procesiraju zahteve.
- Svojstvo *RouteData* je tipa *System.Web.Routing.RouteData*.
- **RouteData** ima svojstvo **Values**, tipa System.Web.Routing.RouteValueDictionary, koje sadži sve vrednosti iz URL segmenata.
- Za čitanje vrednosti iz **{id}** segment podrazumevane rute u Akcionoj metodi treba uraditi sledeće:

  **object id = RouteData.Values["id"];**
- Prevođenje u željeni tip podatka se obavlja jednostavno **cast** operatorom

  **string id = (string)RouteData.Values["id"];**

**ROUTING ENGINE U ŠABNLONU INTERNET APP**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Šablon Internet aplikacije u **ASP .NET MVC 4** koji formira Visual Studio uključuje **dve podrazumevane** rute sa kojima će aplikacija raditi.

- **Prva ruta** govori da treba ignorisati resurse sa ekstenzijom **.axd** (scriptresource.axd i webresource.axd).
- Ovi resursi **ne postoje** kao fizički fajlovi u ovoj aplikaciji i ne mogu biti vraćeni klijentu kao odgovor.
- **Druga ruta** se koristi za mapiranje zahteva u odgovarajući kontroler, odnosno **akcionu metodu**.
- Primer podrazumevane rute se može naći u folderu *App_start/routeconfig.cs*.
- Rute se kreiraju prilikom **prvog** startovanja aplikacije.
- Program koji se zadužen za rute je Application_start i nalazi se u Global.asax.cs fajlu.
- **Kod ASP .NET MVC 4**, Application_start događaj poziva statičku metodu *RegisterRoutes()* iz routeconfig klase locirane u *App_start* folderu.
- Metod prihvata postojeće tabele ruta kao parametre tako da ruta može biti dodana.

Podrazumevana ruta u Internet App. je data na slici 3.1. Ovaj programski kod se podrazumevano dodaje MVC 4 Web aplikaciji.

```
public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults:
                new { controller = "Home",
                action = "Index",
                id = UrlParameter.Optional } );
        }
```

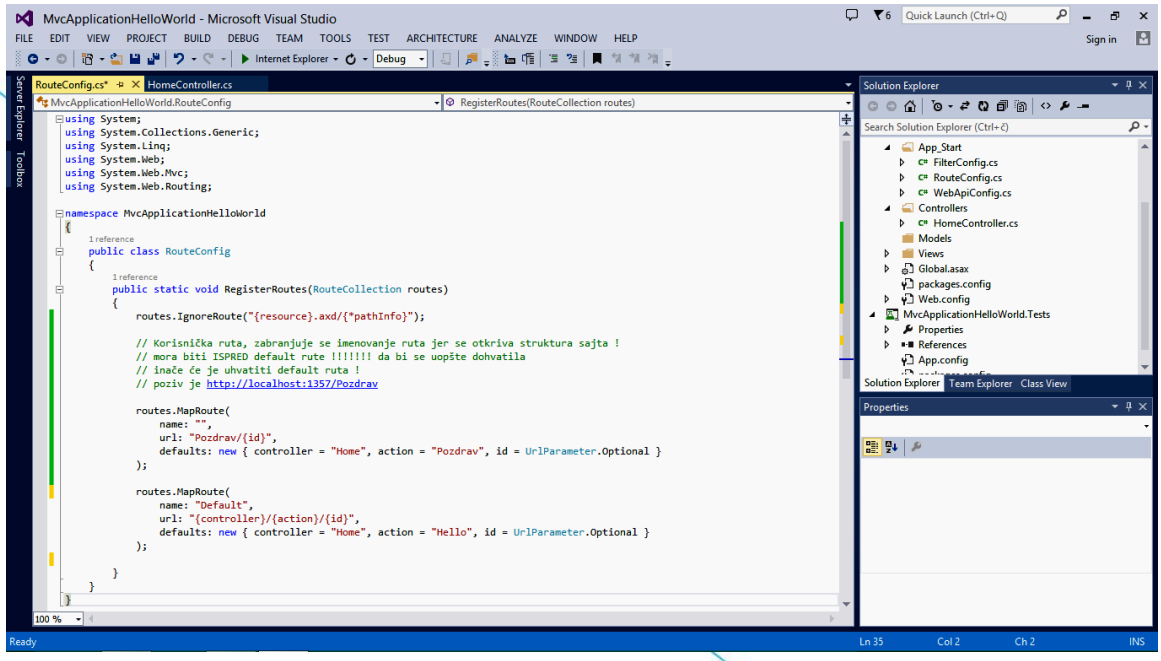


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

}

*Slika 3.1. Programski kod za podrazumevane rute u Internet MVC 4 App.*

Na slici 3.2. dat je prikaz podrazumevane rute u Visual Studiu, a na slici 3.3. su primeri podrazumevanih ruta sa podrazumevanim imenima kontrolera i akcionih metoda.




*Slika 3.2. Programski kod za podrazumevane rute u Internet MVC 4 App u Visual Studi-u.*

**PRIMERI PODRAZUMEVANE RUTE**

| Zahtev | Vrednost parametra | Značenje |
|---|---|---|
| http://domain/ | controller = Home<br>action = Index | **Index** je metoda u **HomeController** klasi |
| http://domain/Pet | controller = Pet<br>action = Index | **Index** je metoda u **PetController** klasi |
| http://domain/Pet/Details | controller = Pet<br>action = Details | **Details** je metoda u **PetController** klasi |
| http://domain/Pet/Details/1 | controller = Pet<br>action = Details<br>id = 1 | **Details** je akciona metoda u **PetController** klasi kojoj se prosleđuje vrednost **1** kao **id** parametar |

José Rolando Guay Paz, **Beginning ASP.NET MVC 4**, Apress, 2013.

*Slika 3.3. Primeri podrazumevanih ruta.*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**KREIRANJE KONTROLERA**

- Prilikom kreiranja kontrolera u ASP.NET MVC treba voditi računa o podrazumevanom šablonu pri dodeljivanju imena klasi **<Controller*Name*>Controller**.
- Klase kontrolera se nalaze u fajlu **<Controller*Name*>Controller.cs**, tako ako je ime klase **Pet**, njeno podraumevano ime je **Pet*Controller***.
- Metode iz klase kontrolera određuju njegovu funkcionalnost i nazivaju se actionim methodama i deklarišu se kao **ActionResult**.

Ove metode vraćaju objekte tipa **ActionResult**, kojih ima nekoliko tipova.

- Raspoložive su sledeće klase izvedene iz klase **ActionResult**:

ViewResult,

PartialViewResult,

ContentResult,

EmptyResult,

FileResult,

HttpUnauthorizedResult,

JavaScriptResult,

JsonResult,

RedirectResult,

RedirectToRouteResult.

Primer C# koda za kreiranja kontrolera je prikazan na slici 3.4., dok su naslici 3.5 prikazani primeri akcioni metoda.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
namespace HaveYouSeenMe.Controllers
    {
        public class PetController : Controller
            {
                // Izvorni kod akcionih metoda
                . . .
            }
    }
```

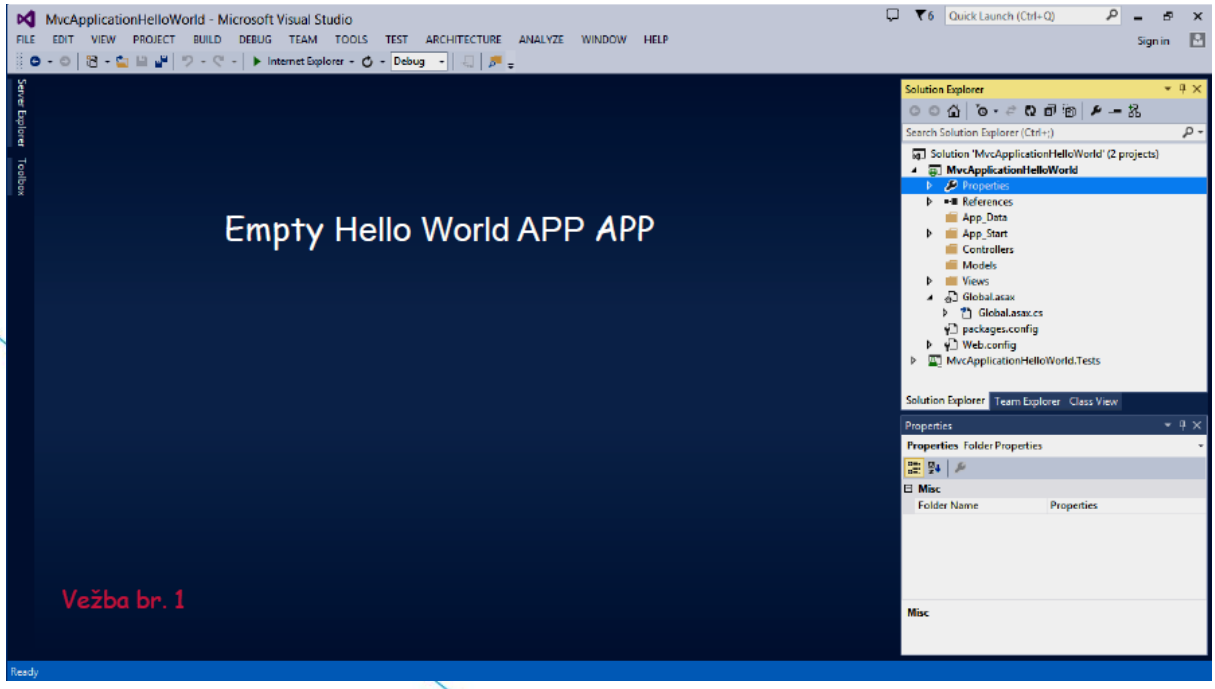
*Slika 3.4. Programski kod za podrazumevane rute u Internet MVC 4 App u Visual Studi-u..*

**PRIMER AKCIONE METODE**

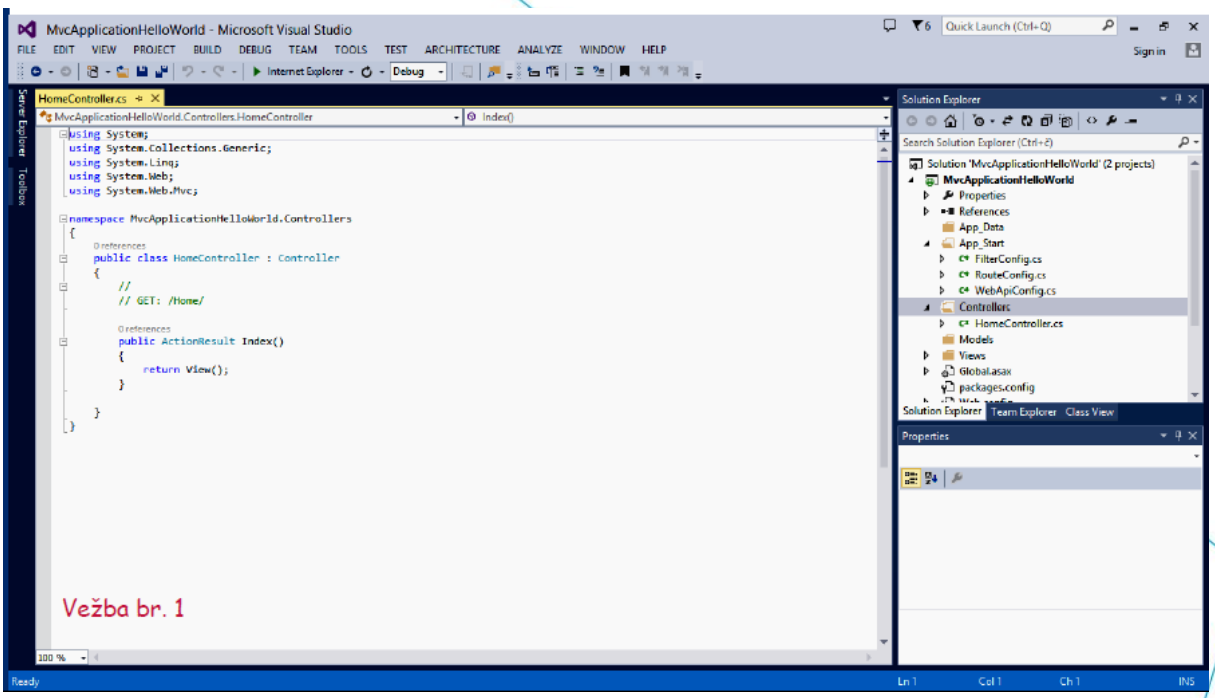
```
public ActionResult Display()
{
    var name = (string)RouteData.Values["id"];
    var model = PetManagement.GetByName(name);
        if (model == null)
            return RedirectToAction("NotFound");
        return View(model);
}

public ActionResult NotFound()
{
    return View();
}
```

*Slika 3.5. Programski kod Akcione metode.*

Primer kreiranja Hello Word bazne aplikacije u MVC 4 okruženju je dat kroz nekoliko slika ekrana razvojnog okruženja (slike 3.6., 3.7. i 3.8.). Ova jednostavna Web aplikacija poseduje samo jedan kontroler i jednu jednostavnu akcionu metodu. Kompletan izvorni kod i slike razvoja aplikacije su date u prilogu.



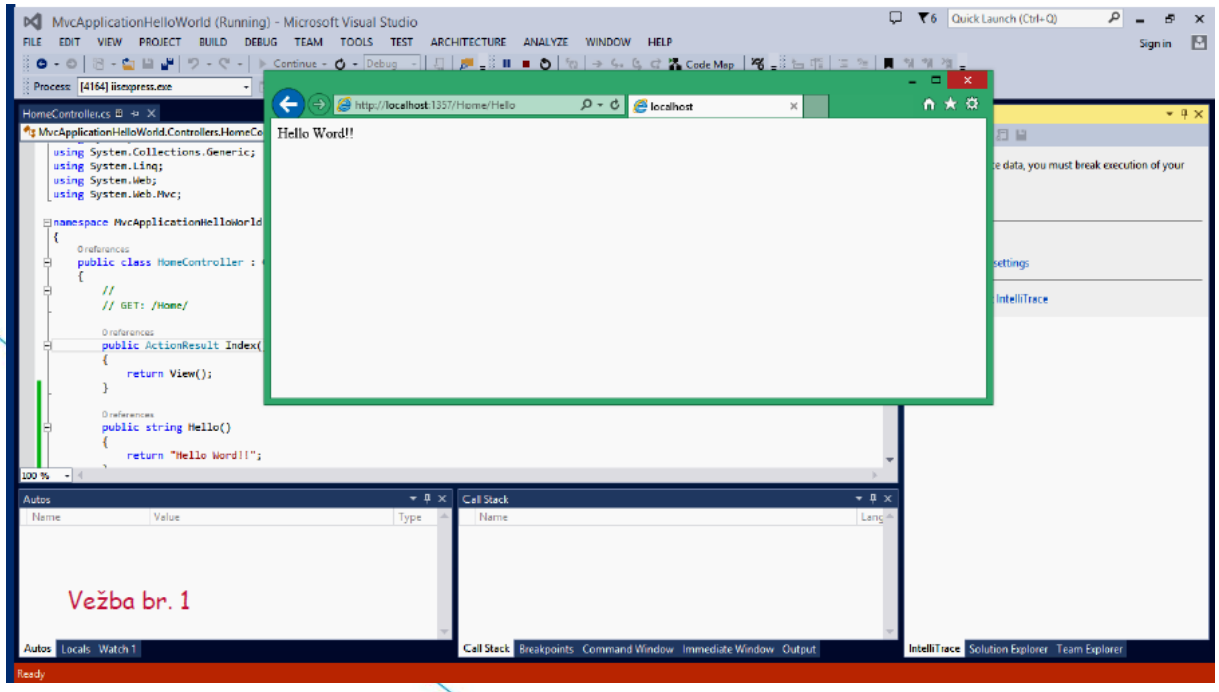

*Slika 3.6. Hello Wporld APP, IDE okruženje.*




*Slika 3.7. Programski kod Akcione metode u IDE-u.*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Slika 3.8. Poziv na izvršenje HelloWorld App.*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 4. MVC views

**KREIRANJE VIEW-a**

Da bi se Web sadržaju učinili dinamičnim, renderovanje Web stranica se prepušta **Web serveru**. Za potrebe dizajniranja dinamičkih Web stranica neophodno je načiniti **view engines** (motor-mašine) koje manipulišu podacima na serveru. **View engines** (enđžins) poseduju specifičnu sintaksu za rad sa **server-side** elementima i renderovanje HTML-a u Web čitačima.

Najčešće povratna vrednost iz akcionih metoda je tipa **ViewResult**. Ovaj povratni tip indicira da bi **ASP.NET MVC** trebalo da pozove stranicu na renderovanje i vrati **HTML** koda nazad klijentu.

- Ove stranice se smeštaju u **Views** direktorijumu unutar aplikacije.
- **Views** često podrazumevaju **server-side kôd** koji proizvodi finalni HTML.
- Prilikom kreiranja **View**-a u VS-u, neophodno je specificirati **view endžin** koji će se koristiti za procesiranje **server-side** koda.

**VIEW ENGINE**

ASP.NET MVC uključuje **dva** view endžina:

- **ASPX** view (prvi pit se pojavljuje u ASP.NET MVC 1 i koristi sintaksu koja je korišćena u MS platformi Web Forms).
- **RAZOR** view engine (nova view endžin koja koristi jednostavniju sintaksu).

| **ASPX** engine | **RAZOR** engine |
|---|---|
| <%=DateTime.Now %> | @DateTime.Now |
| <ul><br><% foreach (var t in tweets) { %><br><li><%= t.content %></li><br><% } %><br></ul> | <ul><br>@foreach (var t in tweets) {<br><li>@t.content</li><br>}<br></ul> |



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

ASP.NET MVC 4 ima i podršku za tzv. **Display Modes** (o njemu nešto kasnije) uz pomoć koga aplikacija može izabrati način renderovanja (ovo je značajno za pristup sajtu različitim Web čitačima pa čak i mobilnim tel.).

**TIPOVI VIEW-a**

- Primer: Akciona metoda NotFound() se poziva uvek kada se u sistemu ne pronađe informacija o traženom ljubimcu (aplikacija koja će biti razvijena do kraja kursa).
- Zapravo, kada se pozove Akciona metoda NotFound() treba ispisati (prikazati) obaveštenje o nepostojanju resursa.
- Da bi se kreirao izgled Web stranice sa ovim obaveštenjem, najlakše je u **MVC** konceptu kreirati **View** za ovu Akcionu metodu.

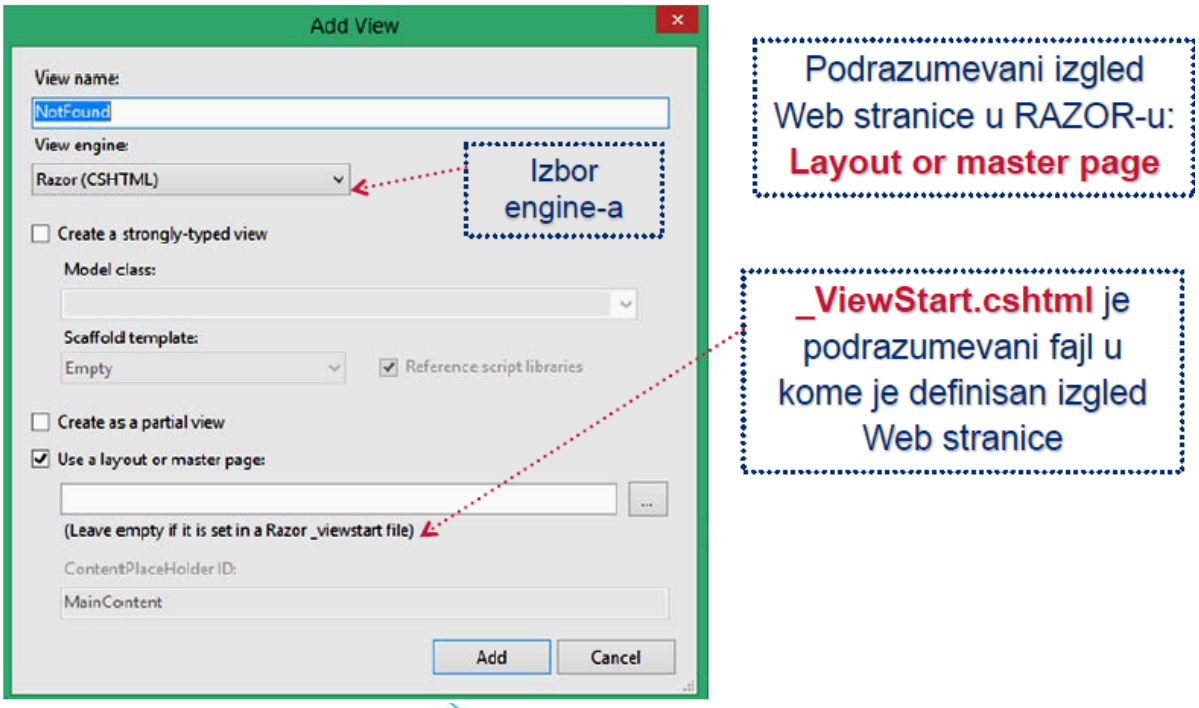
U zavisnosti od potreba, mogu se dodatno specificirati karakteristike View-a kroz izbor sledećih tipova View-a:

- **Partial view** (deo HTML koda koji se može ponovo koristiti),
- **Strongly-typed view** (prihvata model objekta kao parametar iz akc.met.),
- **Layout or master page** (definiše generalni izgled svih View-a). U **RAZOR**-u, podrazumevani **View** se nalazi u _ViewStart.cshtml fajlu.
- Ime View-a se bazira na **imenu** Akcione metode!

Na slici 4.1. prikazan je forma za dodavanje View-a i izbor View endžina u Visual Studi-u za Akcionu metodu NotFound



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Slika 4.1. Proces dodavanja View-a, RAZOR.*

**PROCES RENDEROVANJA VIEW-a (RAZOR)**

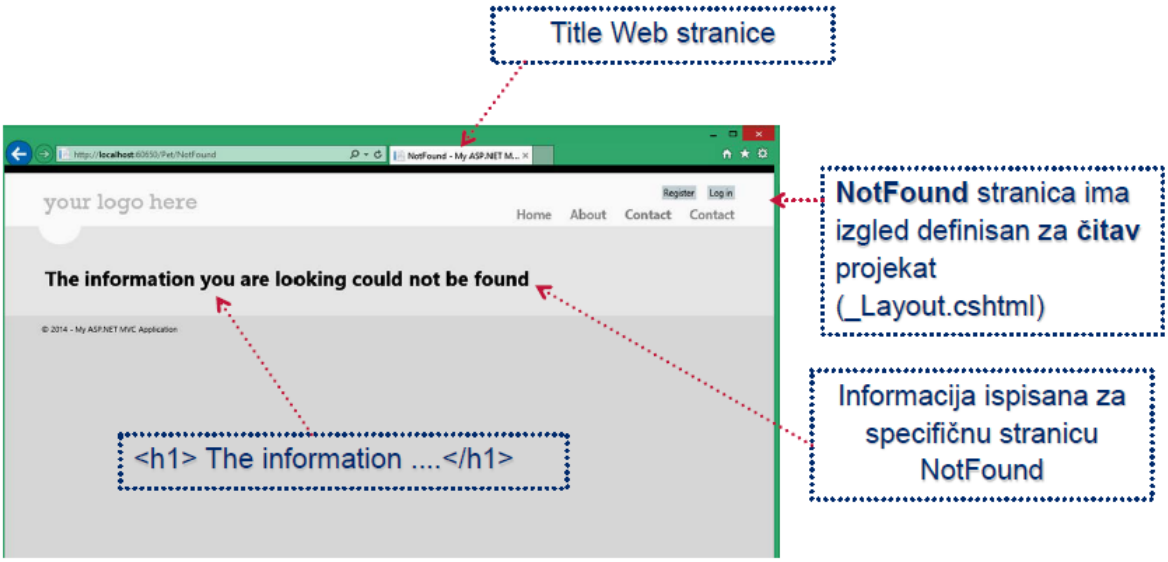
View kreiran za NotFound() akcionu metodu se nalazi u fajlu: *View/Pet/NotFound.cshtml*:

```
@{
        ViewBag.Title = "NotFound";
}
    <h1>The information you are looking could not be found</h1>
```

Posle kreiranja View-a za Akcionu metodu potrebno je obaviti kompajliranje koda kako bi aplikacija mogla da startuje sa prihvatanjem zahteva za prikazom.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Slika 4.2. Izgled renderovane stranice dobijene RAZOR-om.*

**PODRAZUMEVANI VIEW U INTERNET APP**

Prema konceptu **MVC-a 4** formira se podrazumevani View za Internet aplikaciju koji se nalazi u fajlu *_ViewStart.cshtml*. Kod koji se nalazi u ovom fajlu zapravo definiše svojstvo kojim se određuje lokacija fajla gde je konačno definisan izgled Web stranice. Dakle, **RAZOR** poseduje **Layout** svojstvo kojim se definiše fajl **šablona** koji određuje izgled stranice. Detalji podrazumevanog View-a za MVC 4 Internet aplikaciju su zapisani u **_Layout.cshtml** (u foldefu *Views/Shared*) i sadrži HTML kod. Sadržaj *_ViewStart.cshtm* je prikazan na slici 4.3.

```
@{
        Layout = "~/Views/Shared/_Layout.cshtml";
}
```

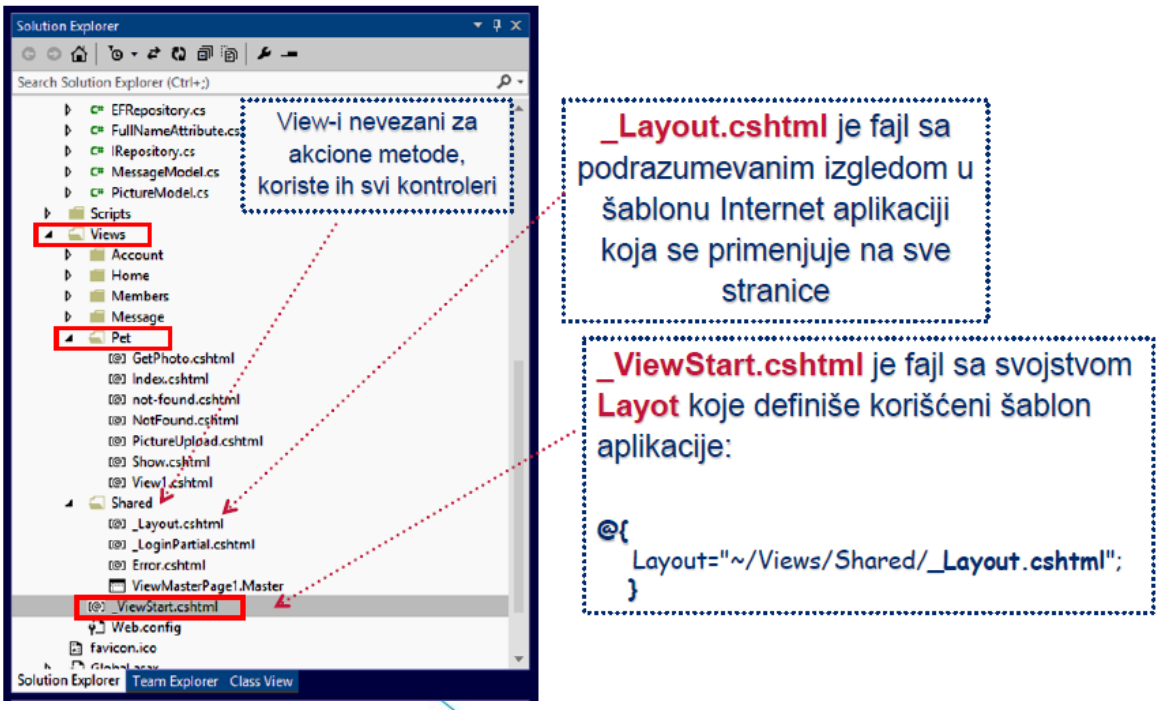
*Slika 4.3. Sadržaj stranice_ViewStart.cshtml.*

**STRUKTURA PROJEKTA POSLE DODAVANJA VIEW-a**

Na slici 4.4. prikazan je izgled Solution Explorer prozora posle dodavanja View-a u Internet App templejtu. Primetite da je **_Layout.cshtml** fajl sa **podrazumevanim** izgledom u šablonu Internet aplikaciji koja se primenjuje na sve stranice. **_ViewStart.cshtml** je fajl sa **svojstvom Layot** koje definiše korišćeni šablon aplikacije.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Slika 4.4. Struktura MVC projekta posle dodavanja View-a.*

**SEKCIJE U HOME PAGE INT. APP**

MVC 4 je koncept programiranja zasnovan na šablonima (templejtima). Već je pokazano da postoje više šablona Web aplikacija (ili Web stranica) koji dolaze sa instalacijom Visual Studia. U nastavku kursa biće razmatran templejt MVC 4 Internet aplikacije na kome će biti građene Vaše aplikacije na vežbama. Na slici 3.12. prikazan je izgled Web stranice koja se dobija primenom Šablona Internet Aplikacije u MVC 4. Na slici su jasno izdvojene sekcije koje se nalaze na u ovom šablonu. Ove sekcije su:

- HEADER,
- FOOTER,
- BODY i
- FEATURED.

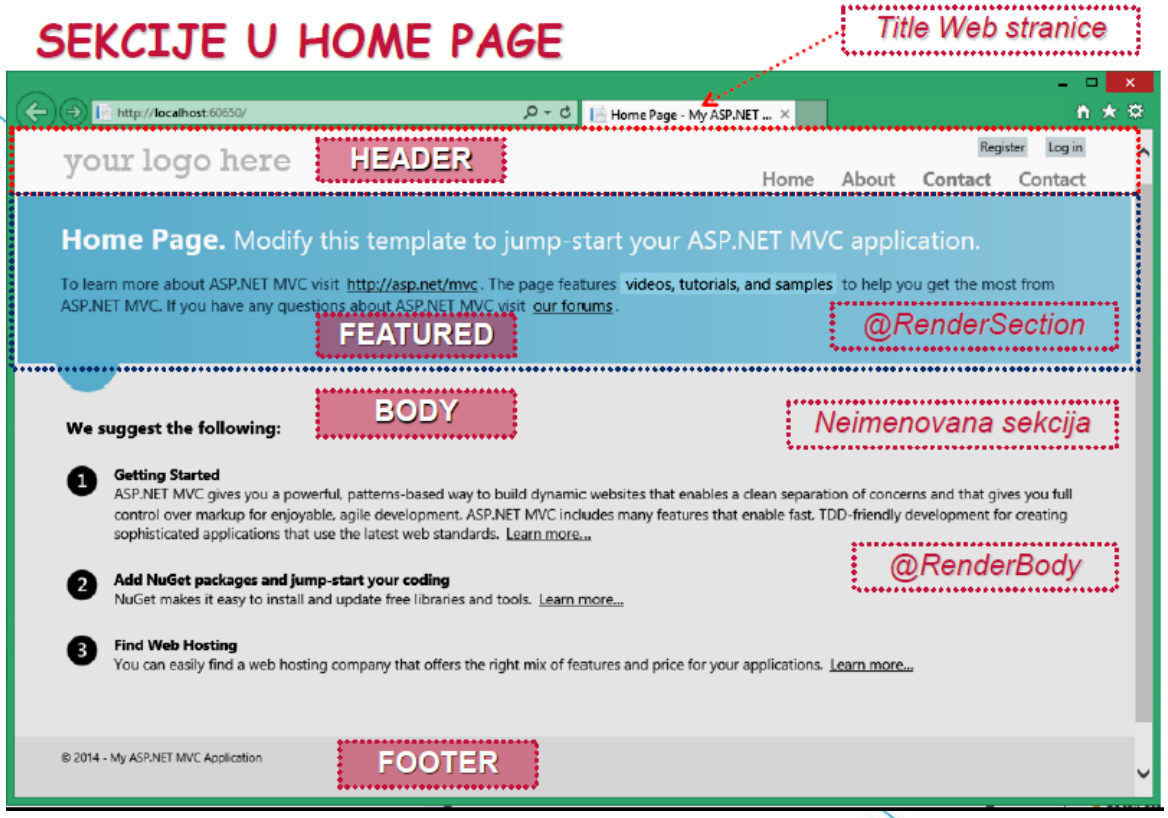
Takođe, prikazane su komande RAZOR-a koje formiraju ove sekcije. U nastavku će biti prikazan izvorni kod ove Web stranice sa detaljima neiphodnim za realizaciju.

Na osnovu MVC konvencije, podrazumevani izgled Internet APP je definisan RAZOR-om I smešten je u **_Layout.cshtml** fajlu**.** Izvorni kod podrazumevanog izgleda INTERNET APP je prikazan u



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

nastavku sa komentarima koji pojašnjavaju primenjeni koncept. Takođe, komentarisane su pojedine komande RAZOR-a kojim se zapravo definišu sekcije u samoj stranici. Pored imenovanih sekcija, RAZOR podržava i neimenovane sekcije i određuje način njihovog prikaza. San izvorni kod sadrži i primenu pojedinih pravila kojima se reguliše način renderovanja. Prikazana je upotreba nekoliko klasa HELPER metoda koje definiše RAZOR.




*Slika 4.5. Izgled stranice templejta za Internet App sa ugrađenim sekcijama.*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

## ZAGLAVLJE u _Layout.cshtml

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>@ViewBag.Title - My ASP.NET MVC Application</title>
        <link href="~/favicon.ico" rel="shortcut icon" type="image/x-icon" />
        <meta name="viewport" content="width=device-width" />
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
    </head>
```

- Dekleracija HTML 5 dokumenta
- RAZOR - izmena naslova prozora u Web čitaču
- Display Mode meta-tag
- RAZOR: reference na spoljne fajlove za **stilove**
- RAZOR: reference na spoljne fajlove za **skriptove**

*Slika 4.6. Izvorni kod zaglavlja Internet APP (1).*

## SEKCIJE u _Layout.cshtml (1)

```
<body>
    <header>
        <div class="content-wrapper">
            <div class="float-left">
                <p class="site-title">
                    @Html.ActionLink("your logo here", "Index", "Home")
                </p>
            </div>
            <div class="float-right">
                <section id="login">
                    @Html.Partial("_LoginPartial")
                </section>
```

- **Zaglavlje** stranice u HTML 5
- **CSS klase**: content-wrapper, float-left, site-title
- RAZOR: Helper metoda
- Sekcija u HTML5 sa **id** - login
- RAZOR:Sekcija sa parcijalnim View-om



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

*Slika 4.7. Izvorni kod zaglavlja Internet APP (2).*

**SEKCIJE u _Layout.cshtml (2)**

Sekcijanavigacije u HTML 5

```
<nav>
    <ul id="menu">
        <li>@Html.ActionLink("Home", "Index", "Home")</li>
        <li>@Html.ActionLink("About", "About", "Home")</li>
        <li>@Html.ActionLink("Contact Us", "Contact", "Home")</li>
    </ul>
</nav>
</div>
</div>
</header>
```

RAZOR: Uređena lista formirana **Helper** metodama

Kraj **zaglavlja** stranice u HTML 5

*Slika 4.8. Izvorni kod sekcije za navigaciju u zaglavlju stranice Internet APP (2).*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

## SEKCIJE u _Layout.cshtml (3)

```
<div id="body">
   @RenderSection("featured", required: false)
      <section class="content-wrapper main-content clear-fix">
         @RenderBody()
   </section>
</div>
<footer>
   <div class="content-wrapper">
         <div class="float-left">
         <p>&copy; @DateTime.Now.Year - My ASP.NET MVC Application</p>
   </div>
</div>
</footer>
```

Sekcija **body** u HTML 5

RAZOR: Formiranje imenovane sekcije **featured**, kasnije se može adaptirati

RAZOR: Dekleracija za renderovanje **neimenovanih** sekcija

Sekcija **futer** u HTML 5

RAZOR

HTML kreiran na ovaj način ne podrazumeva <section> oznaku iz HTML 5

*Slika 4.9. Izvorni kod body sekcije stranice Internet APP (1).*

## SEKCIJE u _Layout.cshtml (4)

RAZOR: Reference na spoljne fajlove za skriptove

```
      @Scripts.Render("~/bundles/jquery")
      @RenderSection("scripts", required: false)
   </body>
</html>
```

RAZOR: Sekcija **scripts** u HTML 5

*Slika 4.10. Izvorni kod body sekcije stranice Internet APP (2).*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

## PRIMER: SEKCIJA featured ZA HOME PAGE

```
@ { ViewBag.title="Home Page"; }
@section featured {
    <div class="featured">
        <div class="content-wrapper">
            <hgroup class="title">
                <h1>@ViewBag.Title.</h1>
                <h2>@ViewBag.Message</h2>
            </hgroup>
            <p> . . . </p>
        </div>
    </div>
} . . .
```

Home Page stranica

Imenovana sekcija **featured** biće zamenjena sa ovim HTML sadržajem u podrazumevanom Layout-u

Helper metode

Neimenovane sekcije

*Slika 4.11. Izvorni kod imenovane sekcije Internet APP (2).*

Primer imenovane sekcije sa pridruženim pravilima za renderovanje dat je na slici 3.17.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

## PRIMER: CSS featured SEKCIJE

```
/* featured */
.featured {
   background-color: #fff;
}
    .featured .content-wrapper {
      background-color: #7ac0da;
      background-image: -ms-linear-gradient(left, #7ac0da 0%, #a4d4e6 100%);
      background-image: -o-linear-gradient(left, #7ac0da 0%, #a4d4e6 100%);
      background-image: -webkit-gradient(linear, left top, right top, color-stop(0, #7ac0da),
                                                                  color-stop(1, #a4d4e6)),
      background-image: -webkit-linear-gradient(left, #7ac0da 0%, #a4d4e6 100%);
      background-image: linear-gradient(left, #7ac0da 0%, #a4d4e6 100%);
      color: #3e5667;
      padding: 20px 40px 30px 40px;
   }
        .featured hgroup.title h1, .featured hgroup.title h2 { color: #fff; }
    . . .
```

Stilovi koji se primenjuju na featured sekciju se nalaze u: App_Start/Content/Uploads/Site.css

*Slika 4.12. Deo CSS koda za imenovanu sekciu featured u Internet APP.*

**RAZOR SINTAKSA I HTML-HELPER METODE**

Obzirom da RAZOR sintaksa zahteva označavanje Helper metoda oznakom @, lako se kogu identifikovane Helper metode. Na slici 3.18. prikazan je izvorni kod stranice kod koga se koriste Helper metode. U ovom kodu se koriste sledeće Helper metode:

- **HTML Helper metoda @Html.Partial**("_LoginPartial") koristi se za parcijalni View unutar main View-a.
- **HTML Helper metoda @Html.ActionLink**("Home", "Index", "Home")se koristi za kreiranje HTML hyperlinka.
- **@Html.BeginForm()** renederuje <form> element, a @Html.AntiForgeryToken() se koristi za nevidljiva polja u obrascu.

```
<div class="float-right">
        <section id="login">
```

```
            @Html.Partial("_LoginPartial")
        </section>
        <nav>
            <ul id="menu">
                <li>@Html.ActionLink("Home", "Index", "Home")</li>
                <li>@Html.ActionLink("About", "About", "Home")</li>
                <li>@Html.ActionLink("Cont. Us", "Contact","Home")</li>
            </ul>
        </nav>
</div>
```

*Slika 4.13. Korišćenje Helper metode u delu za meni  Internet APP.*

**ViewData HELPER METODA**

Pored RAZOR metoda za formiranje hipoerlinkova, za prenos podataka iz Akcionih metoda u View-e koristi se nekoliko novih Helper metoda. Najjednostavnija Helper metoda je **ViewData**. Na slici 3.19. prikazani je izvorni kod kojim se pripremaju podaci za prenos u View, dok je na slici 3.20 prikazan način korišćenja tako pripremljenih podataka.

```
public ActionResult ShowError()
{
        ViewData["ErrorCode"] = 12345;
        ViewData["ErrorDescription"] = "Something bad happened";
        ViewData["ErrorDate"] = DateTime.Now;
        return View();
}
```

*Slika 4.14. Izvorni kod zapripremu podataka za prenosiz Akcione metode  u View korišćenjemHelper metode* **ViewData** *.*

```
@* View *@
<h1>An error was found</h1>
<h2>Error Code: @ViewData["ErrorCode"]</h2>
<h2>Error Date: @ViewData["ErrorDate"]</h2>
```

*Slika 4.15.Korišćenje podataka definisanih korišćenjem Helper metode* **ViewData**.

Još jedan način za prenos podataka iz Akcionih metoda u View-a je korišćenje **ViewBag HELPER METODE.**

**ViewBag HELPER METODE**

Definisanje podataka i korišćenje ViewBag HELPER METODE je prikazano na slici 3.21.

```
public ActionResult ShowError()
{
        ViewBag.ErrorCode = 12345;
        ViewBag.ErrorDescription = "Something bad happened";
        ViewBag.Exception = new Exception();
        return View();
}

@* View *@
<h1>An error was found</h1>
<h2>Error Code: @ViewBag.ErrorCode</h2>
<h2>@ViewBag.ErrorDescription</h2>
```

*Slika 4.16.Korišćenje podataka definisanih korišćenjem Helper metode* **ViewBag**.

**STRONGLY TIP VIEW-a**

Za kompleksnije potrebe prenosa podataka iz Akcionih metoda u View, koriste se View model objekti. Kada se View-u pridruži View model, govori se o strongly typed View-u.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

View model je klasa sa svojstvima koja predstavljaju informacije koje treba proslediti iz Akcijske metode kontrolera u View tako da ih on može renderovati. Ove klase (i metode) svoja svojstva označavaju atributima. Obaveštavanje View-a da mu je pridružen View objekt se obavlja direktivom @model koja ukazuje na pridruženu klasu.

Primer ovog načina prenosa podataka iz Akcione metode u View je realizovan u Login akcionoj metodi. Ovoj klasi modela se dodaje nastavak *<ime_modela>Model*.

**LoginModel METODA SA PRIDRUŽENIM VIEW-om**

Dozvoljeno je korišćenje svojstava View model objekta u **HTML Helper** metodama **implicitnim pozivom Modela objekta** (bez potrebe za direktnim referenciranjem Modela objekta):

```
@Html.LabelFor(m => m.UserName)
@Html.TextBoxFor(m => m.UserName)
```

- Ove dve **Helper metode** kreiraju **label** i **text box**, respektivno, na osnovu postavljenih atributa za svojstvo **UserName** u **View model** klasi.
- Ovako će biti renderovani ovi iskazi:

```
<label for="UserName">User name</label>
<input data-val="true" data-val-required="The UserName is required"
id="UserName" name="UserName" type="text" value="" />
```

**LoginModel METODA SA PRIDRUŽENIM VIEW-om**

Na slici 3.22. je prikazan model podataka za LoginModel.

```
public class LoginModel
{
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
        [Required(ErrorMessage = "The UserName is required")]
        [Display(Name = "User name")]
        public string UserName { get; set; }

        [Required(ErrorMessage = "The Password is required")]
        [StringLength(20, MinimumLength = 6, ErrorMessage = "The Password is invalid")]
        [DataType(DataType.Password)]
        [Display(Name = "Password")]
        public string Password { get; set; }

        [Display(Name = "Remember me?")]
        public bool RememberMe { get; set; }
}
```

*Slika 4.17. Pridruženi model u metodi LoginModel.*

U kodu na slici koriste se sledeća svojstva:

- svojstva metode *UserName* označena atributima.
- Svojstvo *Display* metode *RememberMe* označeno atributom.

```
@model HaveYouSeenMe.Models.LoginModel
@{
        ViewBag.Title = "Log in";
  }
<ol>
        <li>
                @Html.LabelFor(m => m.UserName)
                @Html.TextBoxFor(m => m.UserName) </li>
        <li>
                @Html.LabelFor(m => m.Password)
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
            @Html.PasswordFor(m => m.Password) </li>
        <li>
            @Html.CheckBoxFor(m => m.RememberMe)
            @Html.LabelFor(m => m.RememberMe, new { @class = "checkbox" })
</li> </ol> . . .
```

*Slika 4.18. Korišćenje pridruženog modela LoginModel u formi za prijavljivanje.*

**MOBILNE STRANICE I MVC 4**

Trend pregledavanja Web sadržaja **mobilnim platformama** je već postojeći **problem kompatibilnosti** između Web čitača samo uvećao. Dizajn Web aplikacija koje se **istovremeno** mogu efikasno pregledavati i na mobilnim i desktop uređajima stvorio je **novi izazov** Web dizajnerima i programerima. Dizajn Web aplikacija koje izgledaju **podjednako dobro** i poseduju odličnu funkcionalnost na svim platformama nije jednostavan zadatak.

Prilikom dizajna mobilnih aplikacija treba voditi računa o specifičnostima **mobilnih platformi** i nešto drugačiju upotrebu samih uređaja. Često mobilni Web čitači imaju **redukovani** skup funkcija i nešto skromnije računarske mogućnosti. **Form faktor** displeja je takođe značajan faktor izgleda Web stranice.

**VIEWPORT I @media**

Mobilni Web čitači poseduju virtuelni prozor **viewport**, koji definiše **dimenzije** aktuelnog mobilnog uređaja. Ovako definisani viruelni prozor može proizvesti niz problema tako da se on se može definisati korišćenjem sledećeg **meta-taga** u hederu:

```
<meta name="viewport" content="width=device-width">
```

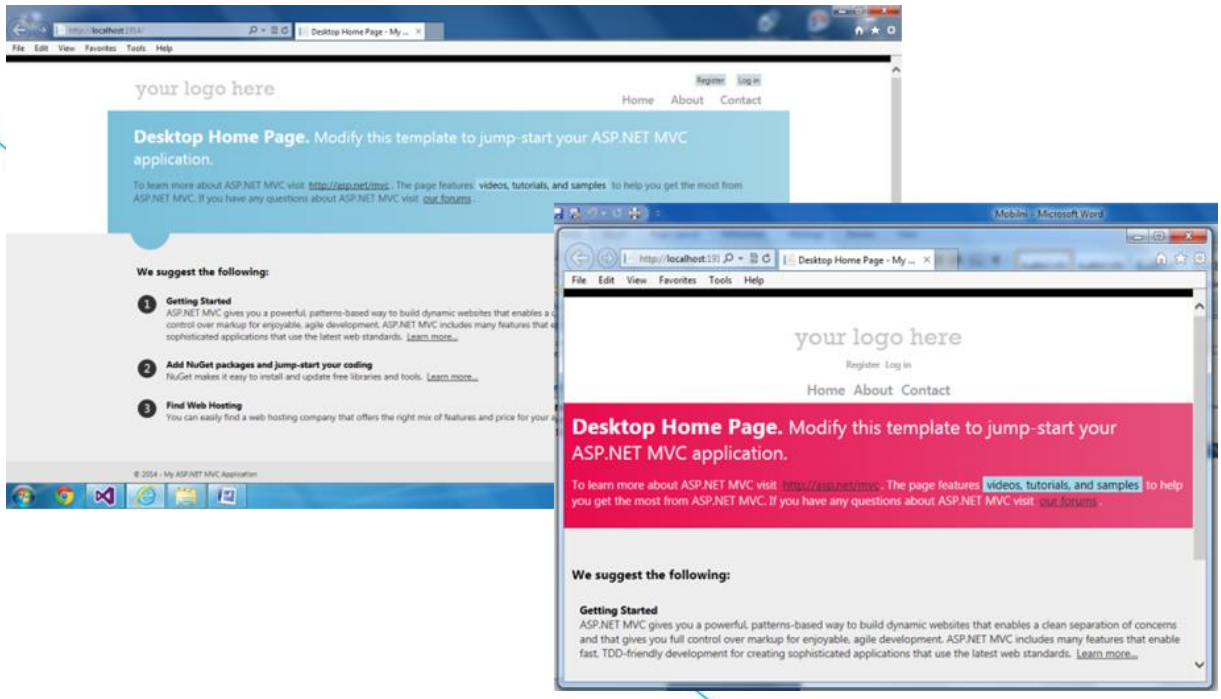
Da bi se mogli primeniti stilovi za specifične **form faktore** neophodno je implementirati **CSS3 Media query**. Pomoću **Media query** se definiše tip medija i skup njegovih karakteristika.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Tip medija zavisi u prvom redu od uređaja kojim se Web stranica čita: ekran, štampač, projektor, ...
- Internet aplikacija u MVC 4 poseduje sledeći CSS iskaz:

**@media only screen and** (**max-width**: 850px){ **...** }




*Slika 4.19. Desktop stranice različite širine.*

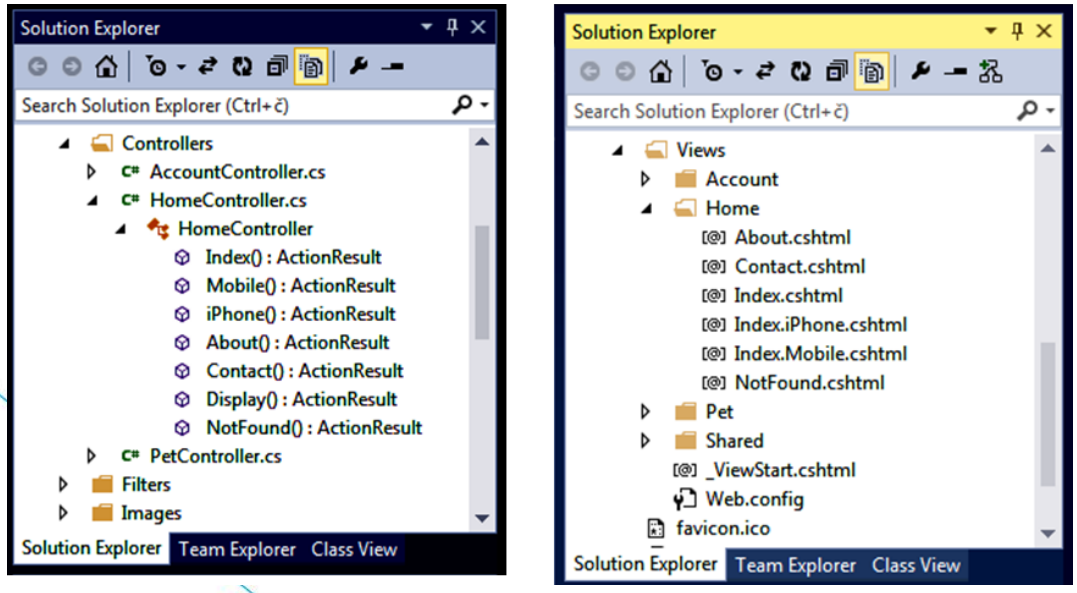
**MOBILNE STRANICE I VIEW OVERRIDE**

Za preklapanje načina za prikazivanje Web stranice u **ASP MVC 4** se koristi mehanizam pod nazivom **view override**. Pomoću ovog mehanizma je moguće prepoznati (otkriti) pristup Web stranici putem **mobilnog čitača**. Za primenu ovog mehanizma neophodno je kreirati **View** čije ime se završava sa **.Mobile.cshtml** (desktop View ima ekstenziju .cshtml).

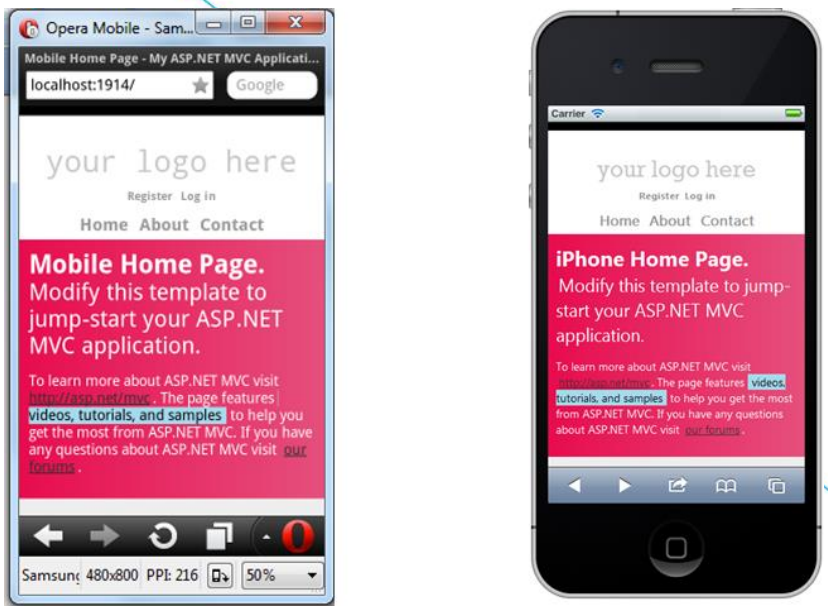
- **Mobilna verzija** Web stranice se može potpuno **izmeniti** u odnosu na desktop verziju (kako po sadržini tako i stilu).
- MVC 4 omogućava prepoznavanje korišćenog browsera, odnosno dodavanje novog displej moda.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Slika 4.20.* Izgled prozora Solution explorer za mobilne stranice.




*Slika 4.21. Prikaz mobilne stranice na mob. Telefonu.*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 5. MVC modeli

Termin model u ASP .NET MVC-u se odnosi na skup objekata koji implementiraju funkcionalnost Web aplikacije. Najznačajniji tipovi modela u MVC-u su: data model, business model i View model. Data model je značajan za interakciju sa bazom podataka.

Business model se koristi da obavi generalna izračunavanja, i u većini slučajeva on sarađuje sa data modelom da bi preuzeo podatke iz baze i kasnije ih smestio u nju. View modeli su klase koje prosleđuju podatke iz kontrolera na prikaz. Data i business modeli se nazivaju domain modelima. View modeli nisu deo domain modela i njihova uloga je da proslede podatke iz kontrolera na prikaz i obratno.

**DATA MODELI I ORM U ASP .NET-u**

U ASP.NET aplikacijama (MVC ili Web Forms), rad sa **klasama** je mnogo jednostavniji nego sa rad sa **bazama** podataka (ova konstatacija generalno važi za sve Objektno Orijentisane - **OO** jezike C# ili VB.NET). Za potrebe rada sa **bazama podataka** OO jezici uvode jedan **apstrakcioni sloj** koji služi za komuniciranje sa bazom podataka.

- Nivo apstrakcije može biti veoma **nizak**, kada se koristi ADO.NET tehnologija.
- **Viši** nivo apstrakcije se može ostvariti softverskim alatima kao što je **M.S**. **Entity Framework**.
- **ORM** (**O**bject-**R**elational **M**apping*)* je tehnologija koja podržava ovu ideju.
- Rezultat rada **ORM**-a su **klase** koje omogućavaju kreiranje, čitanje i brisanje podataka iz tabela baze podataka (Vežba Internet App).
- Veoma važan aspekt **ORM**-a je da on obezbeđuje i klase koje sarađuju sa zapamćenim procedurama i omogućavaju njihovo izvršavanje iz **C#** koda.

**BUSINESS MODEL**

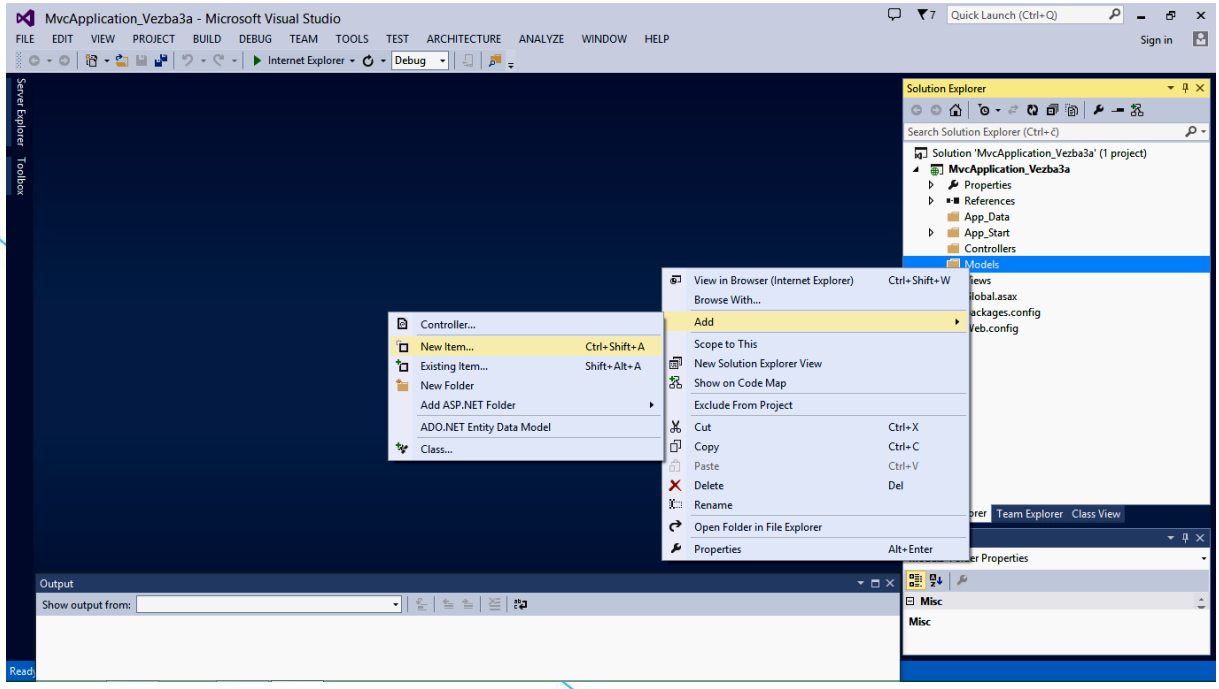
- **Business model** se sastoji od klasa koje obavljaju glavno procesiranje.
- Objekti Business modela **ne rade** direktno kao odgovor na zahtev korisnika.
- Zapravo, metode Business modela rade na zahtev Akcionih metoda iz kontrolera koje primaju korisničke zahteve.
- Tako na primer, klase iz **Business modela** HaveYouSeenMe su zadužene za slanje e-mail poruke vlasnicima ljubimaca (Vežba Internet App).
- Druga klasa **Business modela** implemntira pravljenje malih sličica - minijatura (engl. thumbnails) ljubimaca od uploadovanih slika (Vežba Internet App).



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Zapravo, **bilo koje** procesiranje na **serveru** se realizuje klasama iz Business modela.

Na slici 5.1. prikazan je postupak formiranja Data modela.




*Slika 5.1. Formirabnje modela u Visual Studi-u.*

Na slici 5.2. prikazan je izvorni kod formiranja data modela i njegovo korišćenje u View-u slika 5.3.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
namespace MvcApplication_Vezba3a.Models
{
   public class Korisnik
   {
      public int Id { set; get; }
      public string KorisnickiKod { set; get; }
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
        public double IznosRacuna { set; get; }
    }
}
```

*Slka 5.2. Izvorni kod formiranja data modela.*

```
using System; using System.Collections.Generic; using System.Linq;
using System.Web; using System.Web.Mvc;
using MvcApplication_Vezba3a.Models;
namespace MvcApplication_Vezba3a.Controllers
{
    public class PozoviKorisnikaIPrikaziController : Controller
    {
            public ActionResult Index()
        {
            Korisnik objKorisnik = new Korisnik();
            objKorisnik.Id = 264110;
            objKorisnik.KorisnickiKod = "zorvel";
            objKorisnik.IznosRacuna = 100000.99;
            return View();
        }
        public string zorvel { get; set; }
    }
}
```

*Slika 5.3. Korišćenje Data modela u View-u.*

**MODEL BINDING**

- Kada korisnik šalje obrazac, infomacije iz polja se šalju u **kontroler**.
- ASP.NET koristi **HttpRequest** objekt da opsluži te zahteve.



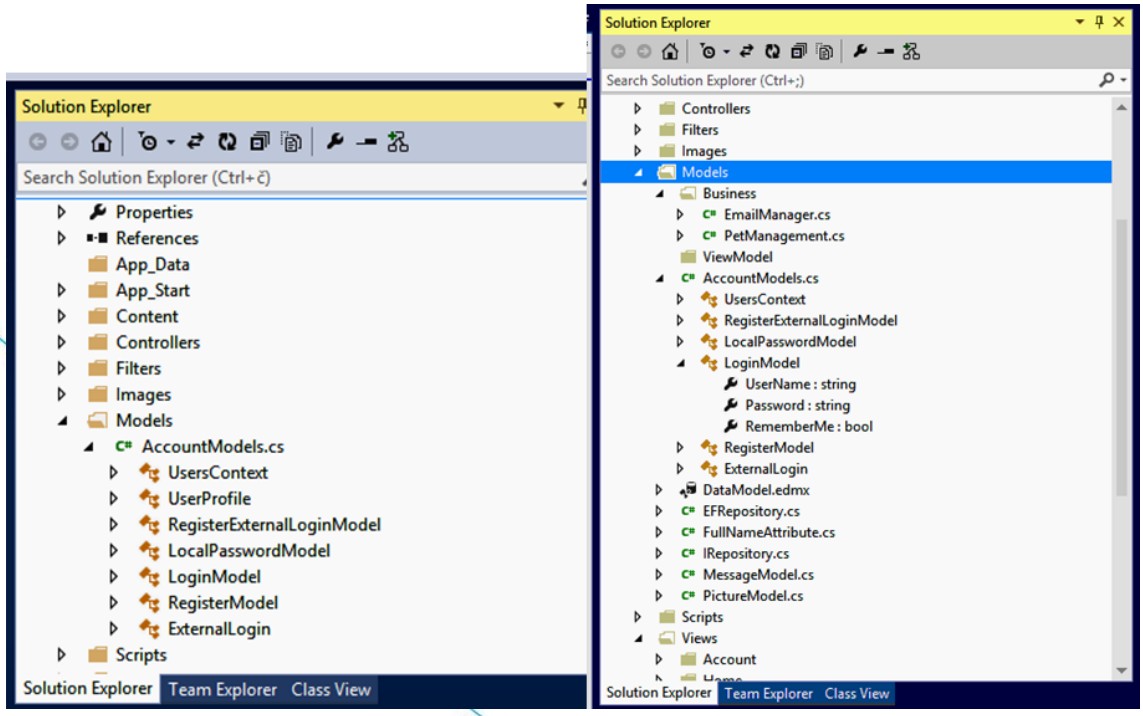
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Informacije iz obrazaca se smeštaju u različite **kolekcije objekata** zavisno od **načina** na koji su zahtevi poslati i samog **tipa** informacije.
- Ako se zahtev pošalje pomoću **HTTP GET** metode (Primer: http://mydomain.com/page**?t**=1&**s**=true), tada se vrednost za parametar **t** i **s** smeštaju u **HttpRequest.*QueryString*** kolekciju.
- Ako se zahtev pošalje pomoću **HTTP POST** metode, tada se poslate vrednosti smeštaju u **HttpRequest.*Form*** kolekciju.
- Upload-ovani fajlovi se smeštaju u **HttpRequest.*Files*** kolekciju.
- Ako se zahtev bazira na **rutama**, vrednosti se smešaju u **RouteData.Values** rečnik.
- Kada korisnik pošalje obrazac u strongly typed View, ASP.NET MVC automatski ispituje **HttpRequest** objekt i mapira poslate inform. u polja.
- Proces mapiranja informacija **iz HttpRequest** objekta **u model objekta** se naziva **model binding**.
- Benefiti model binding-a su značajni:
- Ne piše se kod za ekstrahovanje podataka iz objekta HttpRequest čime se sprečavaju kodne greške;
- Konverzija između tipova podataka se odvija automatski;
- Validacija podataka je jednostavna.
- Model binding je omogućen ugrađenom binder klasom DefaultModelBinder.
- Ova klasa se koristi u ASP.NET MVC kada nema korisničke binder klase.
- Važan aspekt DefaultModelBinder klase je da **imene vrednosti** u HttpRequest objektu moraju biti **iste** kao **imena svojstava** u objektu modela.
- Ovo je već spomenuto (i obrađeno) u delu o View modelima.

Izled Solution Explorer-a pre i posle pridruživanja modela je prikazan na slici 5.4.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Slika 5.4. Izgled Solution Explorer-a sa Data modelima (pre i posle).*



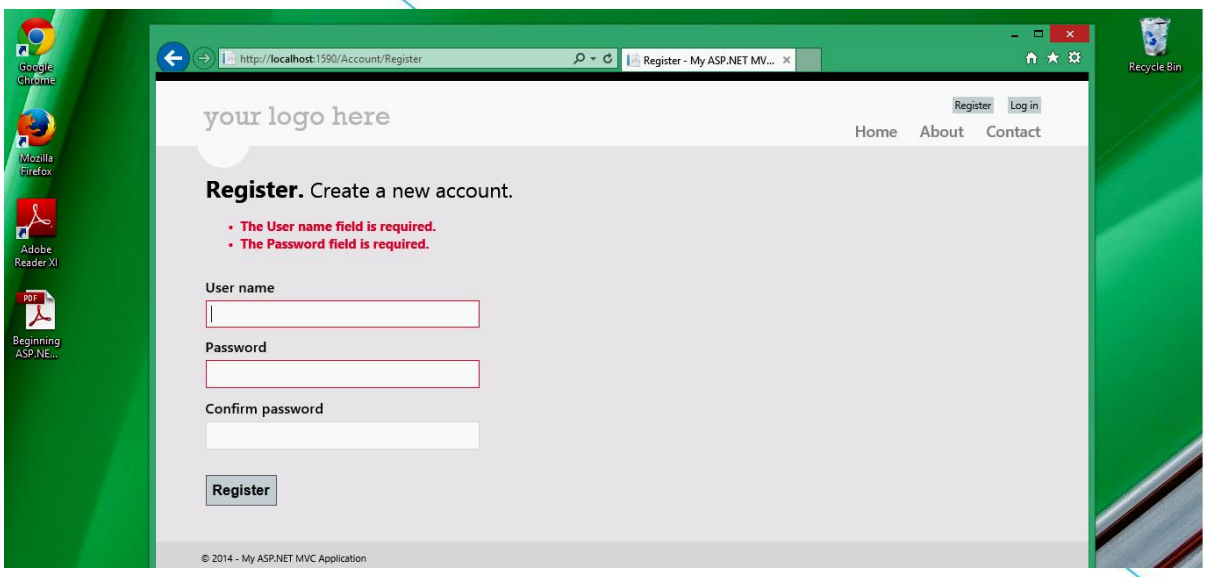
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 6. Validacija podataka

Neophodna komponenta svakog softvera je validacija podataka, zapravo provera da li su uneti (i primljeni) podaci iz obrasca validni. Analiza validnosti podataka mora biti veoma efikasna jer se u slučaju greške o tome mora obavestiti klijenta (u razumno kratkom vremenu) kako bi on korigovao svoj unos. Validaciona logika se može implementirati i na klijentskoj (Web čitač - JavaScript) i na serverskoj strani Web aplikacije (serverski skript). Validacija na klijentu neposredno obaveštava korisnika, dok validacija na serverskoj strani obavlja posredno i ima mnogo veći značaj jer se na serveru obavlja glavno procesiranje podataka.

ASP.NET MVC ima ugrađene mehanizme koji potpomažu validaciju (i odgovor klijentu) i na klijentskoj i na serverskoj strani. Zahvaljujući **data binding** procesu, proces validacije podataka se naziva **model validation**.

Primer validacione forme i unos kojinije korektan u MVC-u 4 jeprikazan na slici 6.1




*Slika 6.1. Primer validaconog obrasca u MVC- u 4.*

**VALIDACIJA PODATAKA NA KLIJENTU I SERVERU**

1. U klijentskom Web čitaču kada korisnik pošalje podatke obavlja se:
   - Verifikacija da su **svi** zahtevani podaci prisutni.
   - Verifikacija da su **formati podatak** korektni (da li u e-mail-u postoji adresno polje, da li postoji znak @, da lo iza njega sledi ime, ...).



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Kada je moguće verifikuje se **tip podataka** (podatak o datumu ili vremenu).
- Ako je nešto neispravno u tom trenutku od korisniku se odmah traži da **koriguje grešku** i podaci se ne šalju serveru. Po ispravci, korisnik ponovo šalje zahtev.

2. Kada je validacija na klijentu uspešna, podaci se šalju na server.
3. Na serveru se obavlja **drugi nivo** validacije, ako nema grešaka obavlja se procesiranje podataka.
4. Međutim, ako ima grešaka, zahtev se ne procesira i šalje se informacija o grešci klijentu kako bi se servisirala.
5. Ako je server-side validacija uspešna, zahtev se procesira i rezultat se prezentira klijentu.

**"RUČNA" VALIDACIJA PODATAKA**

Podaci u poljima obrazaca se koriste za kreiranje View model objekta (korišćenjem data binding-a), a onda se podaci šalju u kontroler na procesiranje. Neophodno je validirati svako svojstvo i formirati validacionu logiku u akcionim metodama. Akcione metode se smeštaju u kontroler za rad sa ovim modelom.

Primer View model objekta sa raspoloživim svojstvima:

```
public class MessageModel
    {
        public string From { get; set; }
        public string Email { get; set; }
        public string Subject { get; set; }
        public string Message { get; set; }
    }
```

Na slici 6.2 prikazana je sekvenca izvornog koda pri validaciji podataka unosa u formu.

```
@model HaveYouSeenMe.Models.MessageModel
. . .
<div class="editor-field">
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
        @Html.EditorFor(model => model.Email)
        @Html.ValidationMessageFor(model => model.Email)
</div>
<div class="editor-label">
        @Html.LabelFor(model => model.Subject)
</div>
<div class="editor-field">
        @Html.EditorFor(model => model.Subject)
        @Html.ValidationMessageFor(model => model.Subject)
</div>
. . .
if (string.IsNullOrEmpty(model.From))
 {
        ModelState.AddModelError("From", "The From field is required.");
 }
if (string.IsNullOrEmpty(model.Email))
 {
        ModelState.AddModelError("Email", "The Email field is required.");
 }
if (string.IsNullOrEmpty(model.Message))
 {
        ModelState.AddModelError("Message", "The Message field is required.");
 }  . . .
```

*Slika 6.2. Sekvenca izvornog kodaza validaciju podataka forme.*

**VALIDACIJA PODATAKA ATRIBUTIMA**

Validacija korišćenjem **data annotations** se zasnova na dodavanju metapodataka o ograničenjima u svojstvima u View modelu. **Atributi** zapravo predstavljaju mehanizme za dodavanje metapodataka.

- Atributi se koriste od strane podrazumevanog binder-a koji izvršava model-binding proces koji uključuje dodatne validacije u poljima obrasca.
- Benefiti ovog koncepta su:
  - Validacija se odvija na čitavoj aplikaciji.
  - Nema pozivanja metoda ili formiranje objekata ili bilo čega drugog što se odnosi na validaciju.
  - Kod koji se zahteva za validaciju je značajno manji od ručne validacije.
- Dodati: ograničenja max. dužina 150 (From), odnosno 1500 (Message) karaktera, korektna forma e-mail-a, korisnička poruka (ErrorMessage).

Na slici 6.3. prikazan je izvorni kod validacije podataka anotacijama.

```
public class MessageModel
 {
   [Required(ErrorMessage="Unesite svoje ime")]
   [StringLength(150, ErrorMessage="Može se dodati samo 150 karaktera")]
   public string From { get; set; }

   [Required(ErrorMessage="Unesite svoju email adresu ")]
   [StringLength(150, ErrorMessage = " Može se dodati samo 150 karaktera ")]
   [EmailAddress(ErrorMessage="Ne prepoznaje se validna e-mail adresa")]
   public string Email { get; set; }

   [StringLength(150, ErrorMessage = " Može se dodati samo 150 karaktera ")]
   public string Subject { get; set; }
 . . .
}
```

*Slika 6.3. Sekvenca izvornog kodaza validaciju podataka forme.*

Raspoloživi atributi u .NET-u su prikazani na slici 6.4.

## RASPOLOŽIVI ATRIBUTI

| Annotation | Description |
|---|---|
| [CreditCard] | Defines the value as a credit card number (15 or 16 digits without spaces or dashes). |
| [EmailAddress] | Indicates that the value must be a well-formatted e-mail address. |
| [EnumDataType(typeof(name of enum))] | Enables a .NET Framework enumeration to be mapped to a data column. The value must be one of those defined in the enumeration. |
| [FileExtension(Extensions="")] | Validates that the file extension of the uploaded file is in the acceptable extensions defined; for example, [FileExtensions(Extensions=".jpg,.png,.gif")]. |
| [Range(min, max)] | Indicates that the value must be between the specified minimum and maximum values. |
| [RegularExpression(pattern)] | Indicates that the value must match the specified regular expression pattern. |
| [Required] | Indicates that the property must contain a value. A validation exception is raised if the property is null, contains an empty string (""), or contains only whitespace characters. If you want to treat whitepace characters as valid characters, use [Required(AllowEmptyStrings=true)]. |
| [StringLength(max)] | Indicates the maximum character length the value can have and, optionally, the minimum character length. This attribute is for properties of type string. |
| [Url] | Indicates that the value is a well-formatted URL. |

José Rolando Guay Paz, **Beginning ASP.NET MVC 4**, Apress, 2013.

*Slika 6.4. Raspoloživi atributi za proveru podataka u MVC –u 4.*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 7. Ajax & jQERY

**Ajax** (engl. Asynchronous JavaScript and XML) je programski model koji omogućava da se pošalje zahtev serveru u pozadini klijentskog koda. Ovaj koncept omogućava osvežavanje izgleda Web stranice bez njenog kompletnog iscrtavanja (osvežavanja). Ovo ima za posledicu značajno **povećanje performansi** Web stranice i uvećano zadovoljstvo korisnika. Za ove potrebe razvijena je **WebAPI** tehnologija koja omogućava transport informacija preko **HTTP** protokola. Najvažnija osobina Ajax-a je da je on **asinhron** u odnosu na korisničke interakcije na Web stranici. Zahtev serveru i obrada njegovog odgovora se može obaviti **odmah** po njegovom prijemu na osnovu generisanih događaja na Web stranici.

U **ASP.NET MVC** je implementiran koncept **unobtrusive** JavaScript koji sledi ideju **separacije** prezentacionih poslova (čitaj HTML koda) od procesiranja (čitaj JavaScript kod).

- Implementacija obrade događaja i druge funkcionalnosti su postaljene u zasebnim fajlovima.
- **Unobtrusive** JavaScript se bazira na **jQuery** JavaScript biblioteci koja je besplatna i veoma korisno *open source* rešenje.
- jQuery poseduje jednostavan mehanizam **selekcije** (nalaženje) HTML elemenata (kao što su , <button>, i form polja) na Web stranici.
- Dodatno, **jQuery** uključuje izuzetan **API** za manipulaciju i animaciju **HTML** elemenata.
- Funkcionalnost jQuery–a je implementirana pomoću sloja koji sakriva način rada Web čitača sa **DOM**-om (engl. Document Object Model).
- Ovao je značajna karakteristika jer eliminiše razlike između različitih Web čitača.
- Programeri za Web jedino treba da znaju **kako** da rade sa JavaScript i HTML elementima, a **jQuery** će se pobrinuti za korektnu implemenaciju u svim Web čitačima.
- **jQuery** je možda najpopularnija JavaScript biblioteka koja je sada uključena u Visual Studio i sve šablone projekata Web aplikacija.
- **ASP .NET MVC** koristi jQuery za implemntaciju **unobtrusive** JavaScript koncepta.
- U **Solution** prozoru Visual Studija se može proveriti aktuelna verzija jQuey-a koja se koristi u šablonima projekata.

**V.S. i jQuery BIBLIOTEKA**

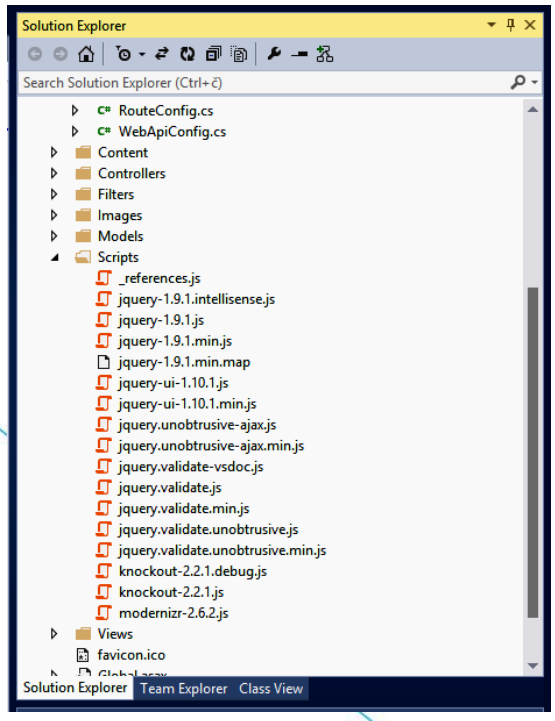
- jQuery biblioteka je locirana u ***Scripts/jquery-[version].js*** fajlu.
- IntelliSense za V.S. Help se nalazi u ***jquery-[version].intellisense.js***



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Minified verzija ***jquery-[version].min.js***.
- Source map fajl se koristi za translaciju *minified* verzije u nonminified verziju.

Na slici 7.1 prikazan je izgled Solution Explorera i lokacija biblioteke skriptova.




*Slika 7.1. Raspoloživi atributi za proveru podataka u MVC –u 4.*

**jQuery i BUNDLE**

- Za pristup jQuery biblioteci treba uključiti ***jquery-[version].js*** fajl u Web stranice koje ga koriste.
- Ovo se može uraditi jednostavnim dodavanjem reference na jQuery fajl u Web stranici pomoću **<script>** oznake.
- U <head> odeljku dodaje se skript **<script>** oznaka na stranici:

**<script** src="~/Scripts/jquery-1.9.1.min.js"**></script>**

**Bundle** (svežanj, snop, breme) je **drugi način** da se ova biblioteka uključi u projekat i zahteva nešto veše koda (posla). **Bundle** je sposobnost ASP.NET-a da dozvoljava kombinovanje više spsecijalizovanih fajlova u jedan zajednički.

- Tako, JavaScript i CSS fajlovi mogu biti u **minified** formi da bi se redukovala veličina rezultujućeg fajla.
- Dovlačenje jednog fajla sa servera zahteva samo jedan **HTTP** zahtev, čime se **skraćuje vreme** obezbeđivanja svih resursa Web stranice.
- Za kreiranje bundle koncepta, neophodno je koristiti klase *ScriptBundle* i *StyleBundle*.
- Ove klase poseduju metod *Include()* za dodavanje fajlova u bundle.
- Dodavanje fajlova u **bundle** se obavlja u *BundleConfig* klasi lociranoj u **App_Start/BundleConfig.cs** fajlu.
- Ova klasa ima statičnu metodu *RegisterBundles()* koja prihvata parameter tipa **BundleCollection** koju dodajete Vaš bundle.
- **BundleConfig** klasa uključuje sledeći bundle za jQuery biblioteku.
- Za korišćenje formiranih bundle-ova koriste se sledće klase:

**@Scripts.Render("~/bundles/jqueryval")**

**@Styles.Render("~/Content/css")**

- Imenski prostor jQuery-a se skraćeno označava sa **$** .
- Za rad jQuery-a, Web stranica mora biti učitana u celosti.

Formiranje budle-a za jquery biblioteku prikazan je na slici 7.2.

```
. . .
public class BundleConfig
{
        public static void RegisterBundles(BundleCollection bundles)
        {
                bundles.Add(new ScriptBundle("~/bundles/jquery")
                .Include("~/Scripts/jquery-{version}.js")
                );
                bundles.Add(new StyleBundle("~/Content/css")
```

```
                    .Include("~/Content/site.css")
                );
. . .
}
```

*Slika 7.2. Bundle za jquery biblioteku.*

**jQuery SELEKTORI**

Da bis se manipulisalo sa HTML elementima, oni se prvo moraju locirati u HTML kodu. Za pretraživanje i lociranje traženih HTML elemenata koriste se tzv. **selektori**. Selektor predstavlja string koji se pretražuje, a najznačajniji su prikazani na slici 7.3.

| Selector | Description |
|---|---|
| `$("#menu")` | Finds the element with an `id` of `menu` |
| `$(".selected")` | Finds the elements with the class name of `selected` |
| `$("h1")` | Finds all the `<h1>` elements |
| `$("#menu ul")` | Finds all the `<ul>` elements that are descendants of the element with `id menu` |

*Slika 7.3. Raspoloživi jQuery selektori-.*

Izvorni kod koji podržava događaje i jQuery prikazan je na slici 7.4.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

DOGAĐAJI I jQuery

```
@using (Html.BeginForm()){
        @Html.Label("email", "Email:");
        @Html.TextBox("email");
                <input type="submit" id="submitButton" value="Send" />
}
<script>
        $(document).ready(function () {
        $("#submitButton").on("click", function (e) {
                var email = $("#email").val();
                if (email == "") {
                        e.preventDefault();
                        alert("Please enter your email address first.");
                }
        });    });
</script>
```

Forma

Čekaj učitavanje stranice

Čekaj događaj click

Anonimna funkcija

Funkcija koja se poziva posle pritiska dirke

*Slika 7.4.Izvorni kod koji podržava događaje i jQuery.*

**Ajax**

Kada Web stranica zahteva obnovu podataka, to se obično realizuje tako što neka od oznaka to zatraži od servera (kao što je recimo taster).

- Kada server završi zahtevano procesiranje, nova stranica se vraća čitaču.
- Ovaj proces se naziva **full page refresh**, koji za sobom nosi mnoge negativne posledice.
- Jedna važna posledica ovog koncepta je veoma sporo refreširanje Web stranice.
- Zatim, uvećanje opterećenja servera, jer se mora mora procesirati stranica na svaki zahtev čak iako su izmene minorne.
- **Ajax** nudi rešenje kako će unaprediti refreširanje stranice i otkloniti pomenute nedostatke.

**Ajax** nudi rešenje kako unaprđuju refreširanje stranice.

- Neki od benefita koji se dobijaju primenom **Ajax**-a su:
    - Zahtev se može poslati **asinhrono**;

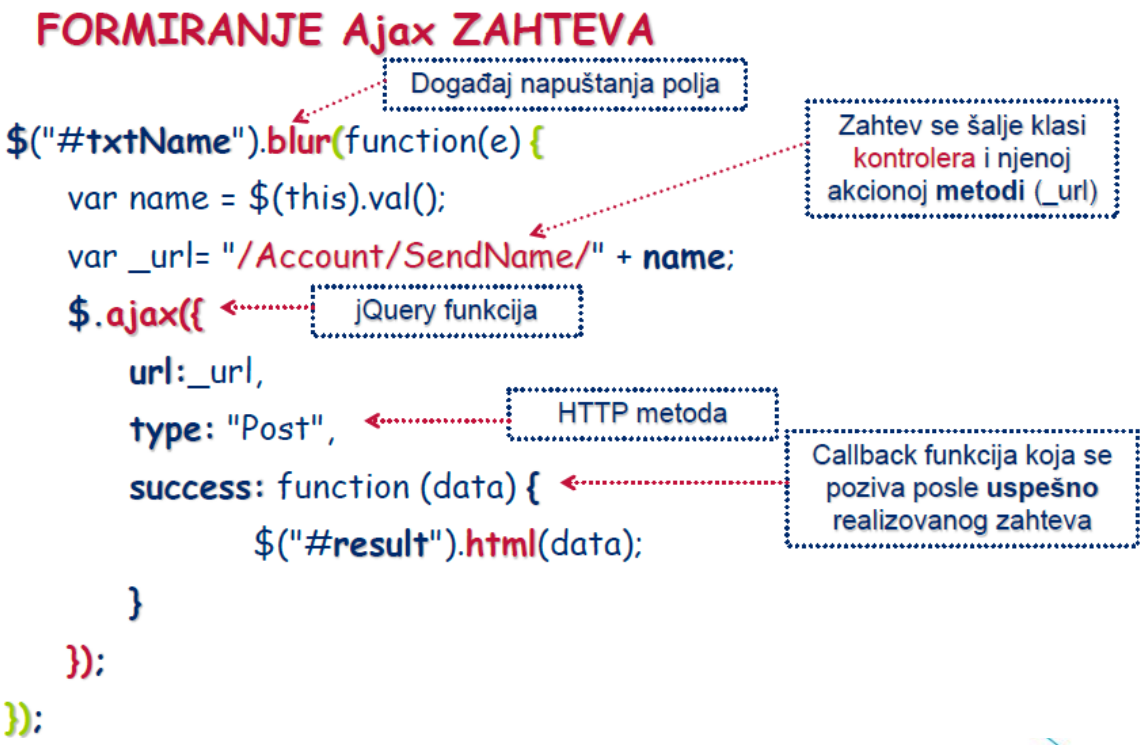


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Odgovor servera je **minimalan**;
- Odgovor servera je **brži**;
- Refreširanje stranica se obavlja samo na **delu** Web stranice koje to zahtevaju.

- **Ajax** zahtev može biti iniciran korisničkom interakcijom (pritisak na dugme, napuštanje stranice, promena fokusa, ... )
- Kada je **Ajax** zahtev kompletiran, **HTML** sa određenim **id** biće osvežen odgovorom koji je poslao server.

Na slici 7.5. prikazan je način formiranja Ajax zahteva.




*Slika 7.5. Formiranje Ajax zahteva.*

**@Ajax SVOJSTVA**

ASP.NET MVC 4 realizuje korišćenje Ajax-a kroz @Ajax svojstvo u RAZOR View objektima. Svojstvo @Ajax je tipa System.Web.Mvc.AjaxHelper i obezbeđuje implementaciju HTML Helper metoda za formiranje hipeveza.

- HTML Helper metode u @Ajax svojstvu su slične @Html svojstvima.
- Razlika je da helper metode u @Ajax svojstvu rade asinhrono.
- Evo primera formiranja hipertekstualnih veza na ova dva načina:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

@**Html.ActionLink**(“Ovo je reg. veza","actionName","controllerName")

@**Ajax.ActionLink**(“Ovo je Ajax veza", "actionName", "controllerName", ajaxOptions)

**@Ajax.ActionLink SVOJSTVA**

```
<div id="resultDiv"> </div>
@Ajax.ActionLink(“Ovo je Ajax veza", “ImeAkcije", “ImeKontrolera",
  new AjaxOptions  // Objekt tipa AjaxOptions
        {
                Confirm = “Da li ste sigurni?",
                HttpMethod = "Post",
                UpdateTargetId = "resultDiv",
                InsertionMode = InsertionMode.Replace
        });
```

**@Ajax.ActionLink SVOJSTVA**

- **Confirm**: String koji će biti prikazana u dijalogu da li želite da nastavite ili ne. Ako korisnik pritisne dugme OK zahtev se procesira, inače se briše.
- **HttpMethod**: Ovo svojstvo indicira tip HTTP metoda koji se koristi u zahtevu.
- **UpdateTargetId**: Ovo svojstvo definiše id vrednost HTML elementa koji će se koristiti za prikaz rezultata Ajax zahteva.
- **InsertionMode**: Određuje kako će vraćeni odgovor biti dodat HTML elementu definisanom u UpdateTargetId (moguće su vrednosti InsertAfter, InsertBefore i Replace).
- **OnBegin**, **OnComplete**, **OnSuccess**, **OnFailure**: Se koriste za opsluživanje događaja (početak i kraj), odnosno zahtev je uspešno ili neuspešno procesiran.
- **LoadingElementId** i **LoadingElementDuration**: ova svojstva definišu HTML element koji će biti prikazani dok se zahtev procesira i njegovo trajanje u [ms] (može se kreirati  element koji animira sliku).

Primer obrasca i sa primenom Call-back funkcija prikazan je na slikama 7.6., 7.7., i 7.8.

## PRIMER: Ajax OBRAZAC

```
@using (Ajax.BeginForm (new AjaxOptions
{
    Confirm = "Da li ste sigurni da želite da pošaljete ovu poruku?",
    HttpMethod = "Post",
    InsertionMode = InsertionMode.Replace,
    LoadingElementId = "loading",
    UpdateTargetId = "messageForm"
}))
{
@Html.AntiForgeryToken()
@Html.ValidationSummary(true)
...
<div id="loading" style="display:none">
Sending Message... </div>
```

*Slika 7.6. Formiranje Ajax zahteva.*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

## PRIMER; Ajax OBRAZAC – Callback funkcije

```
@using (Ajax.BeginForm(new AjaxOptions
{
    Confirm = "Are you sure you want to send this message?",
    HttpMethod = "Post",
    InsertionMode = InsertionMode.Replace,
    LoadingElementId = "loading",
    UpdateTargetId = "messageForm",
    OnBegin = "beginRequest",
    OnComplete = "endRequest"
}))
```

Callback funkcije koje se pozivaju **pre/posle** zahteva

*Slika 7.7. Primena Call back funkcija u Ajax obrascu (1).*

## Callback FUNKCIJE Ajax-a

```
@section Scripts {
@Scripts.Render("~/bundles/jqueryval")
<script>
        function beginRequest() {
        $("#messageForm").hide();
    }
        function endRequest(request, status) {
        $("#messageForm").show();
    }
</script>
}
```

JavaScript Callback funkcije koje se pozivaju pre/posle zahteva

*Slika 7.8. Primena Call back funkcija u Ajax obrascu (2).*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 8. Bezbednost web aplikacija i MVC 4

Bezbednost Web aplikacije je značajan aspekt Web programiranja. Neophodno je implemntirati bezbedonosne mere u cilju očuvanja smeštenih podataka na serveru. Pristup osetljivim delovima aplikacije treba omogućiti samo pojedinim korisnicima (administratori). U ASP .NET MVC 4 je Implemntirano nekoliko metoda autentifikacije i zaštita od cross-site scripting-a (XSS) i cross-site request forgery-a (CSRF).

U Web applikacijama se procesom autentifikacije određuje ko je korisnik, dok se autorizacijom određuje šta je korisniku dozvoljeno da radi. Umesto formiranja individualnih dozvola, zgodno je kreirati grupne dozvole (role) čime se definišu ista prava za sve pripadnike grupe. ASP.NET MVC ima podršku za nekoliko šema za autentifikacije: **WindowsAuthentication** i **FormsAuthentication**.

**WINDOWS / FORM AUTENTIFIKACIJA**

Windows Authentication šema se koristi tamo gde svi korisnici imaju naloge realizovane u Windows Active Directory. U tom slučaju sam proces autentifikacije obavlja IIS, bez učešća Web aplikacije. Ako je korisnički zahtev validan, onda IIS prosleđuje zahtev ASP.NET-u na dalje procesiranje.

- Windows Authentication je veoma zahvalan za developere jer ih oslobađa upravljanja korisničkim nalozima.
- Forms autentifikacija je namenjena za Internet bazirane aplikacije gde se korisnici identifikuju pomoću korisničkog imena i lozinke.
- Alternativno, korisnici mogu koristiti OAuth protokol za povezivanje na third-party naloge (Facebook, Twitter, Google, ...).

**WINDOWS AUTHENTICATION**

- Ova vrsta autentifikacije se postavlja u Web.config fajlu na sledeći način:
- U sekciji <authentication> načiniti sledeće izmene:

```
. . .
<configuration>
        <system.web>
                <authentication mode="Windows" />
        </system.web>
</configuration
. . .
```

- Takođe, treba i na **IIS**-u (serveru) omogućiti ovu vrstu autentifikacije.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**FORMS AUTHENTICATION**

- **Forms Authentication** se zasniva na kreiranju jednostavne login Web stranice koja ima zadatak validacije korisnika putem provere korisničkog imena (engl. username) i lozinke (engl. password) korisnika.
- Ako je korisnik validan, kreira se **HTTP kolačić** (engl. **cookie**) u koji se smeštanja informacija o korisniku.
- Već znate da se svi kolačići prosleđuju prilikom svakog HTTP zahteva od strane Web čitača, tako da server na ovaj nači može prepoznati korisnika.
- Ime kolačića je **.ASPXAUTH**.
- Informacije u kolačiću se kriptuju.
- Forms Authentication brine o kolačićima i njihovom transportu, tako da se programer time ne bavi.
- Da bi se Forms Authentication aktivirala, u Web.config fajlu treba identifikovati **gde** se login stranica nalazi (loginUrl).
- Šablon ASP.NET MVC 4 Internet Application podrazumeva Forms Authentication (radi se na vežbama).

```
<authentication mode="Forms">
    <forms loginUrl="~/Account/Login" timeout="2880" />
</authentication>
```

- loginUrl: gde će se korisnik usmeriti kada zatraži zaštićenu stranicu.
- Trebalo bi postaviti stranicu za logovanje u folderu koji zahteva Secure Sockets Layer (**SSL**).
- timeout: specificira ograničeno vreme za Forms Authentication sesiju.

Dodatna svojstva za Forms Autentication su:

```
<authentication mode="Forms">
    <forms loginUrl="/Account/Login"
    protection="All"
```

```
        timeout="2880"
        name="AppNameCookie"
        path="/FormsAuth"
        requireSSL="false"
        slidingExpiration="true"
        defaultUrl="/"
        cookieless="UseCookies"
        enableCrossAppRedirects="false" />
</authentication>
```

**ZAŠTITA KONTROLERA I AKCIONIH METODA**

Često je potrebno omogućiti pristup osetljivim delovima Web aplikacije samo specifičnim klasama korisnika (autorizaovanim korisnicima). Pristup ovim delovima aplikacije je moguće posle autentifikacije korisnika. Osetljivi delovi aplikacija u ASP.NET MVC se identifikuju akcionim metodama i kontrolerima (a ne URL-om kao kod klasičnog pristupa).

- Ovo omogućava da jedino autentifikovani korisnici mogu pristupiti osetljivim delovima Web aplikacije (ove sekcije se označavaju atributom **[Authorize]** ili još specifičnije **[Authorize(Users="Zoran")]**.
- Ovaj atribut se može primenit na pojedine akcione metode ili kontrolere.
- Ako se doda atribut [Authorize] kontroleru, **sve** akcione metode u kontroleru biće dostupne samo autorizovanim korisnicima.
- Ako neke od metoda ne zahtevaju autorizovane korisnike dekorišu se (obeležavaju) atributom **[AllowAnonymous]**.
- Ako su kontroleri i akcione metode postavljeni da zahtevaju autorizaciju, **framework** će redirektovati (uputiti) korisnike na login page.
- Posle autentifikacije korisnici će biti vraćeni na originalnu poziciju pomoću korišćenja parametra ReturnUrl koji obaveštava Login Page **gde** da pošalje autorizovane korisnike.
- Primer definisanja dva kontrolera MembersController i AdminController koji se koriste za sekcije za članove, odnosno, za administratore.

. . .



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
[Authorize]
public class AdminController : Controller
        {
                public ActionResult Index()
        {
return View();  . . .
. . .
[Authorize]
public class MembersController : Controller
{
        //
        // GET: /Members/
        public ActionResult Index()
        {
                return View();
        }
}. . .
```

*Slika 8.1. Zaštita pristupu kontroleru.*

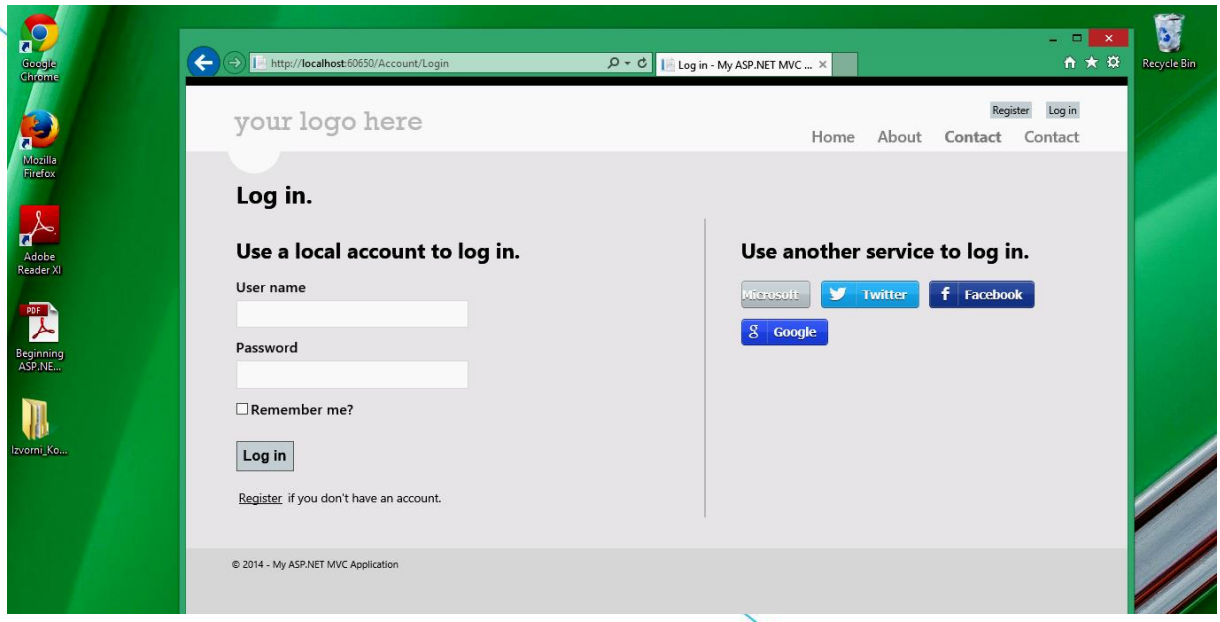
**AUTORIZACIJA EKSTERNIM IZVORIMA**

- Šablon Internet Application ASP.NET MVC 4 podržava autorizaciju protokolom **Open Authentication** (OAuth).
- Šablon podržava sve što je potrebno za autorizaciju korisnika putem Twitter, Facebook, Google, ili Microsoft naloga.
- Ovi servisi su veoma pogodni za Internet korisnike jer se **ne moraju** posebno autorizovati za Vašu Web aplikaciju.
- Korisnici ne pamte posebne **username/password** za svaku aplikaciju.
- Programer ne vodi računa o bezbednosti ovih podataka.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Da bi se dozvolila ova vrsta autorizacije, treba ukazati na te spoljnje izvore koji se mogu koristiti.
- U App_Start/AuthConfig.cs file treba dozvoliti željene spoljnje izvore autorizacije.
- Za Microsoft, Facebook, i Twitter, neophodno je obezbediti identifikacione ključeve (vežba) na razvojnim stranama ovih socijalnih mreža.




**MEMBERTSHIP I ASP .NET**

- ASP.NET **Membership System** je svojstvo ugrađeno u ASP.NET koje obezbeđuje bezbedno pamćenje šifri za aplikacione korisnike.
- Takođe, obezbeđen je **API** koji pojednostavljuje zadatak validacije korisnika kada se koristi Forms Authentication.
- ASP.NET uključuje **dva** membership provajdera, ActiveDirectoryMembershipProvider i SqlMembershipProvider.
- Jednostavniji Membership System je SimpleMembership koji je prethodno već korišćen (za lokalnu i eksternu autorizaciju).
- Za korišćenje ASP.NET Membership-a, prvo se formiraju profili korisnika a zatim i pamte (već rađeno sa LocalDB).

- Korisnici su smešteni u UserProfile i webpages_Membership tabelama, ili webpages_OAuthMembership tabeli za eksternu autorizaciju.
- Uloge (roles) su smeštene u webpages_Roles tabeli.
- Za korišćenje SimpleMembership-a, sve što treba je **connection string** koji ukazuje na bazu podataka gde su smešteni korisnički nalozi.
- Konfiguracija SimpleMembership-a se dešava **automatski** - kada se prvi put zatraži AccountController klasa.
- Ovo se dešava zbog toga što je AccountController klasa označena atributom **[InitializeSimpleMembership]**.
- **[InitializeSimpleMembership]** atribut se poziva naredbom:

WebSecurity.**InitializeDatabaseConnection**("DefaultConnection", "UserProfile", "UserId", "UserName", autoCreateTables: true);

- InitializeDatabaseConnection() metod obaveštava Web aplikaciju da koristi bazu podataka definisanu u DefaultConnection stringu koja se nalazi u Web.config fajlu.
- Na vežbama će ovaj connection string pokazivati na LocalDB bazu podataka.
- **Drugi i treći** patrametar u InitializeDatabaseConnection() pozivaju tabelu i **primarni ključ** kolona gde je smešten korisnički nalog.
- **Četvrti** parametar je je kolona u tabeli koja pamti username koje identifikuje svaki korisnički nalog.
- **Poslednji** parametar definiše da li će SimpleMembership tabele biti kreirane ako ih nema u bazi.

**ROLE U ASP .NET-u**

- **Role** definišu grupe korisnika, jer se njima lakše upravlja.
- U ASP.NET, zadatak upravljanja rolama je delegiran osobini nazvanoj **Role Manager**.
- Role Manager podrazumeva API za kreiranje, brisanje i dodeljivanje uloga korisnicima.
- U SimpleMembership-u, uloge se pamte u webpages_Roles i webpages_UsersInRoles tabelama.
- Sa SimpleMembership-om, se takođe mogu lakše upravljati sa ulogama korisnika koji su se eksterno autorizovali putem **OAuth** i **OpenID**.
- Konfiguracija Role Manager je specificirana u Web.config fajlu i omogućava tri provajdera: SqlRoleProvider, WindowsTokenRoleProvider, AuthorizationRoleStoreProvider.

**ZAŠTITA ASP .NET MVC APP OD SPOLJNIH NAPADA**

- Kako korisnici počnu sa korišćenjem Web aplikacija postaju ranjivi na bezbedonosne napade.
- Maliciozni korisnici kompromituju Web čitače čime Web aplikacije čine još ranjivijim.
- Radi pomoći zaštite Web aplikacija u ASP.NET MVC 4 je uključeno mnogo tehnika koje pomažu u sprečavanj najčešćih tipova napada (cross-site scripting (**XSS**) i cross-site request forgery (**CSRF**)).
- **XSS** je ranjivost Web aplikacije koji dozvoljavaju napadaču da unese JavaScript kod u polja obrasca.
- Posle toga, ovaj kod se smešta u bazu podataka i kasnije se poziva na izvršenje.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 9. Testiranje i instalacija MVC 4 aplikacije

**UNIT TESTING**

Testiranje Web aplikacije je veoma važan zadatak programera.

Pravilno testiranje Web aplikacije pomaže u otkrivanju grešaka koje mogu dovesti do gubljenjem podataka.

Unit testing je tehnika koja pomaže u kreiranju klasa i metoda za testiranje specifičnih funkcija Web aplikacija.

Na rasplaganju je mnogo (komercijalnih i besplatnih) framework-ova koji omogućavaju Unit testiranje.

U ASP.NET MVC je moguće izvršiti metode kontrolera u potpuno izolovanom okruženju.

Osnovna ideja u unit testiranju je kreiranje programa (obično metoda u klasi) koji će načiniti određene pretpostavke o nekom kodu i odrediti očekivani rezultat rada tog koda.

Unit test je automatizovani kod koji poziva metode ili klase koje će biti testirane, a zatim proverava pretpostavke o logičnom ponašanju metoda ili klasa.

Ako recimo metod vraća objekt baziran na id-u, metod se može ponašati na jedan od dva načina:

Naći će objekat sa specificiranim id-om (i vratiti ga), ili

Neće naći traženi objekt (vratiće null).

Zbog toga što postoje dva različita ponašanja metode, treba napisati dva unit testa.

Može se imati više test jedinica za jedan isti kod.

Sa Microsoft Test Framework-om, kreiraju se unit testovi kao metode koje su označene atributom [TestMethod].

Metode moraju biti u klasama koje su označene atributom [TestClass].

**STRUKTURA UNIT TESTINA**

Najpoznatiji šablon za testiranje je AAA, koji testira tri dela plikacije:

Arrange: Postavka unut testa, inicijalizuju se promenljive i objekti koji su spremni za prosleđivanje i testiraju funkcionalnost.

Act: Poiziva se kod koji se želi testirati.

Assert: Analiza dobijenih rezultata. Verifikuje se vraćena vrednost i svi vraćeni parametri koji se očekuju.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Visual Studio omogućava korišćenje različitih framewok-ova za potrebe testiranja unutar njegovog IDE-a (mogu se downloadovati sa NuGet).

Pored osnovnih, postoje i tzv. mocking framework-ovi (Moq, NUnit.Mocks, RhinoMocks, M.S. Fakes) koji formiraju mock objekte.

Mock objekti simuliraju funkcionalnost realnih objekata, sa kojim se mogu simulirati i spoljni resursi.

```
. . .
using Microsoft.VisualStudio.TestTools.UnitTesting;
using HaveYouSeenMe;
using HaveYouSeenMe.Controllers;
namespace HaveYouSeenMe.Tests.Controllers
{
        [TestClass]
        public class HomeControllerTest
                {
                    [TestMethod]
                    public void Index()
                     { . . .
```

*Slika 9.1. Struktura UNIT TEST sistema.*

**ŠTA POSLE TESTA?**

Testiranjem se obuhvata funkcionalnost aplikacije: business modela, testiranje kontrolera i testiranje ruta. Ovi zadaci nisu kompleksni, ali se zahteva pisanje testa za sva ponašanja objekata. Testiranje business modela obezbeđuje test osnovne funkcionalnosti aplikacije. Kod treba da bude tako napisan da uticaj eksternih izvora može jasno biti identifikovan. Ako test ne bude uspešan, treba locirati problem.

Testiranje kontrolera i ruta je takođe značajna aktivnost jer oni opslužuju korisničke HTTP zahteve.

**INSTALACIJA WEB APLIKACIJE**

Web aplikacija se može instalirati na Web server na razne načine. Visual Studio poseduje sopstveni alat za instalaciju Web aplikacije (podržava in-house i cloud-baziran server). Najčešće je



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

potrebno izvršiti migraciju sa razvojne baze podataka (LocalDB) koja ima ograničenja, na realnu bazu koja se koristi na sereru domaćinu (SQL Server).

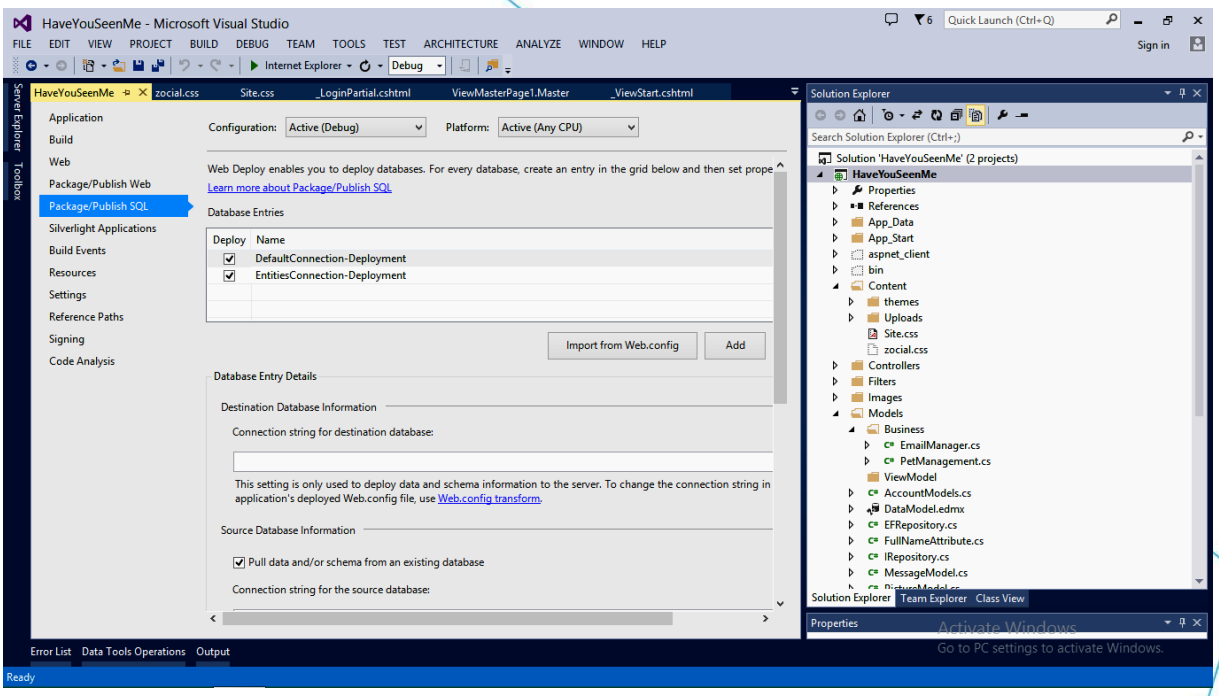
Pre postavljanja Web aplikacije na server treba voditi računa o sledećem:

- Dozvoliti proveru na greške u vreme kompajliranja (compile-time error) za View-e;
- Zabraniti debugging i verbose errors zbog bezbedonosnih razloga;
- Korigovati postavke u Web.config fajlu za realno korišćenje (ne za proces razvoja).

**INSTALACIJA BAZE PODATAKA I APP**

Ako se ima pristup serveru sa administrativnim privilegijama, prvo je potrebno formirati bazu podataka sa odgovarajućim tabelama koje zahteva aplikacija. Tabele vezane za SimpleMembeship se formiraju iz V.S. generisanjem skriptova koji se pamte u fajlovima.

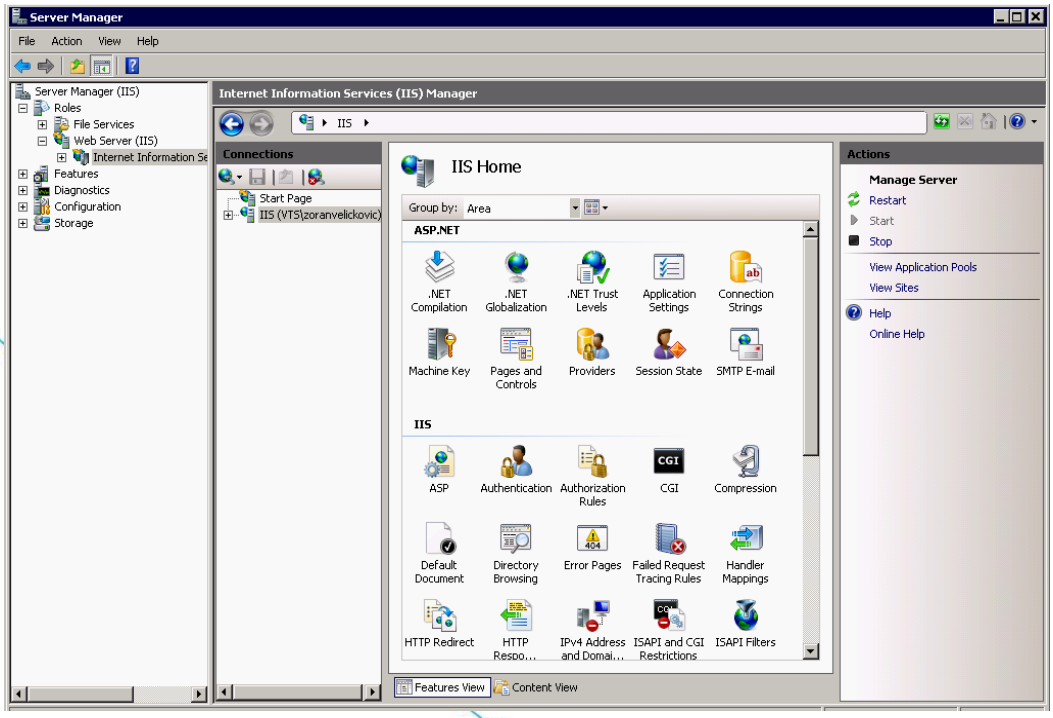
Skriptovi dati na vežbama formiraju tabele neophodne za rad aplikacije. Za povezivanje na DB server je potpomognuto čarobnjacima. Ovde je značajno postaviti korektne pristupne stringove i lociranje baze. Što se tiče instalacije aplikacije, u IIS menađženu se definiše direktorijum u kome se nalazi aplikacija i prava pristupa. Kroz proces testiranja proverava se dostupnost aplikacije.



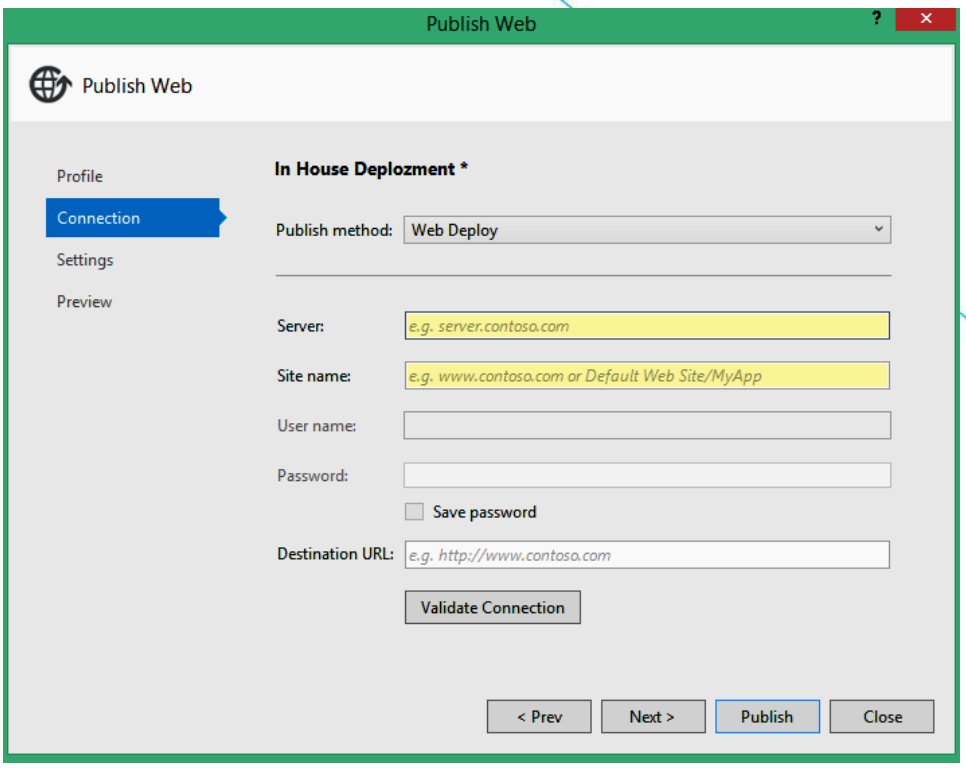

*Slika 9.2. Struktura UNIT TEST sistema.*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




*Slika 9.3. Struktura UNIT TEST sistema.*




*Slika 9.4. Struktura UNIT TEST sistema.*



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 10. Literatura


[1] José Rolando Guay Paz, Beginning ASP.NET MVC 4, Apress, 2013.

[2] Jess Chadwick, Programming Razor, O'Reilly Media Inc., 2011.

[3] Jess Chadwick, Jess Chadwick, Todd Snyder, and Hrusikesh Panda, Programming ASP.NET MVC 4, O'Reilly Media Inc., 2012.

[4] Adam Freeman, Pro ASP.NET MVC 4, Apres, 2012.

[5] Zoran Veličković, Milojko Jevtović, „Web programiranje i responsive Web dizajn u ASP .NET MVC 4", Infoteh '14, pp. 939-944, Jahorina, 2014.

[6] Zoran Veličković, Milojko Jevtović, „Adaptacija izgleda Web stranice uslovljena klijenskim specifičnostima u ASP .NET MVC 4 okruženju", IT '14, pp. 193-196, Žabljak 2014.

[7] Zoran Veličković, Milojko Jevtović, Zoran Milivojević „Podrška razvoju multiplatformske Web aplikacije u ASP .NET MVC 4, YUInfo '14, pp. 447-452, Kopaonik 2014.

[8] Z. Veličković, Predavanja: Web programiranje 2, ICT Cluster Academy 2014.

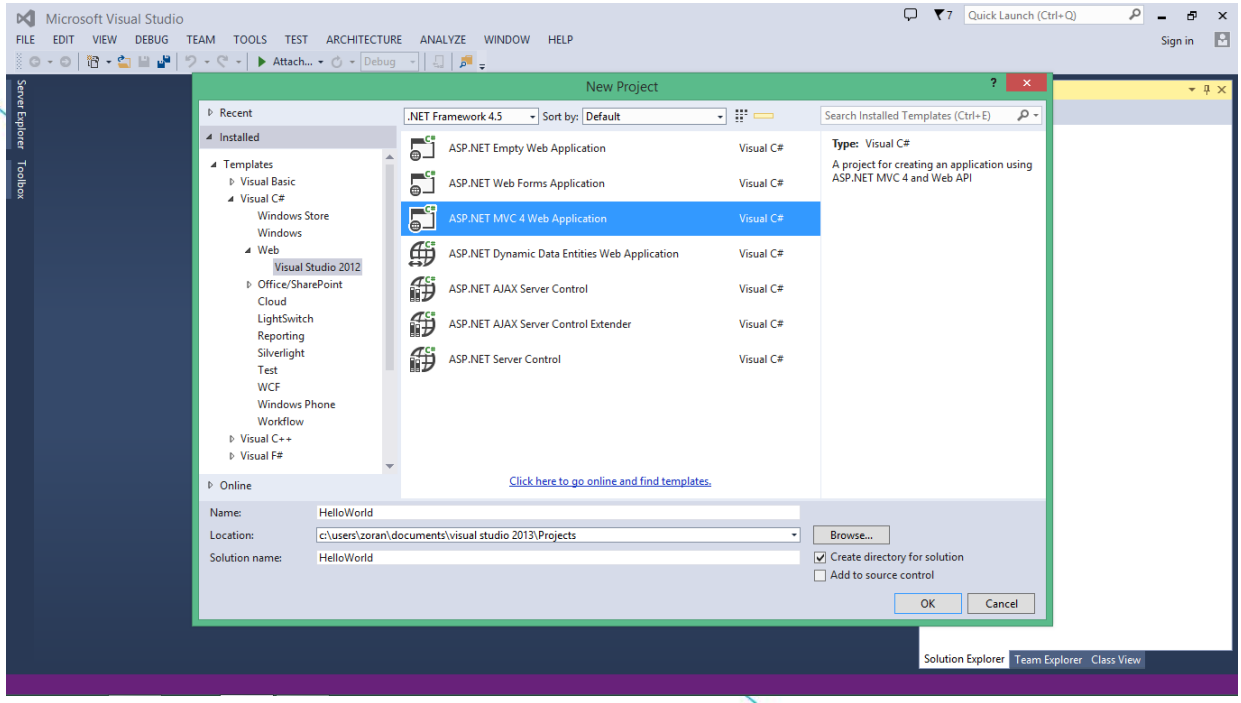
[9] Z. Veličković, Praktikum: Web programiranje 2, ICT Cluster Academy 2014.

[10] Z. Veličković, Internet app template, ICT Cluster Academy 2014.

**Prilog 1: Vežba br. 1**

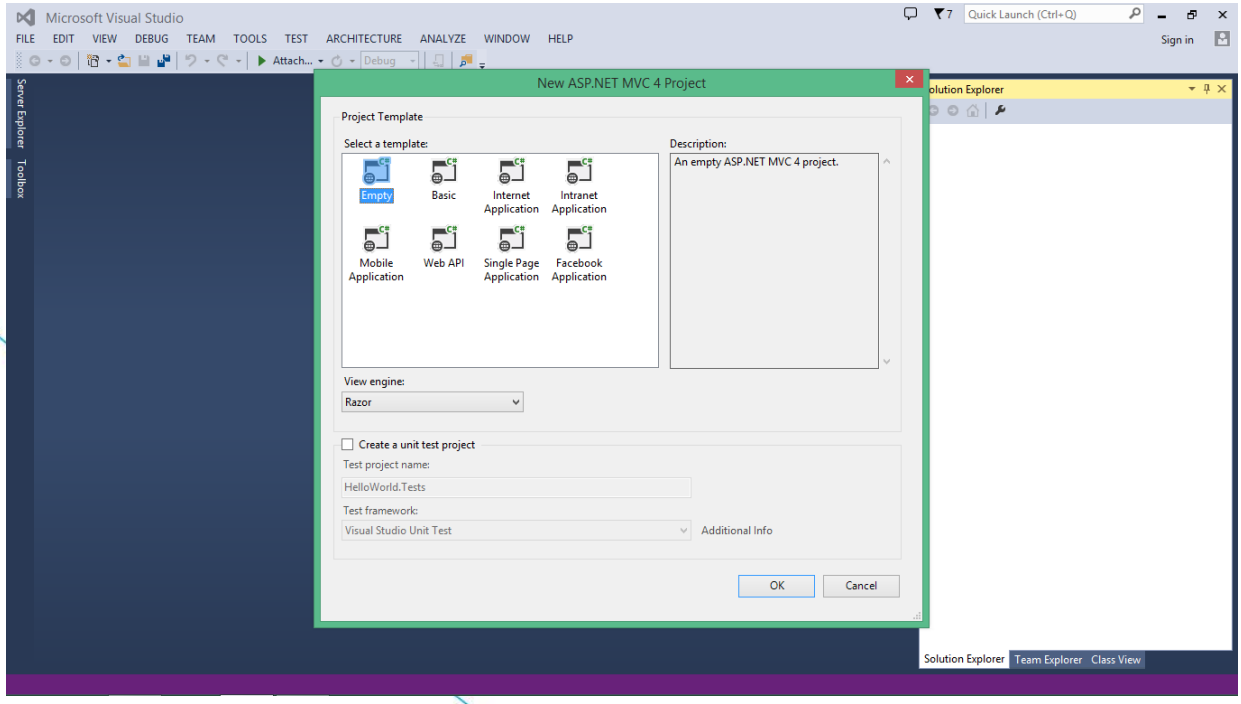
HELLO WORLD APLIKACIJA

HelloWorld je uobičajeno prva aplikacija prilikom proučavanja programskih jezika. Slično ćemo uraditi i ASP .NET MVC-u. Biće formirana jednostavna tzv. Empty aplikacija korišćenjem Visual Studi-a.

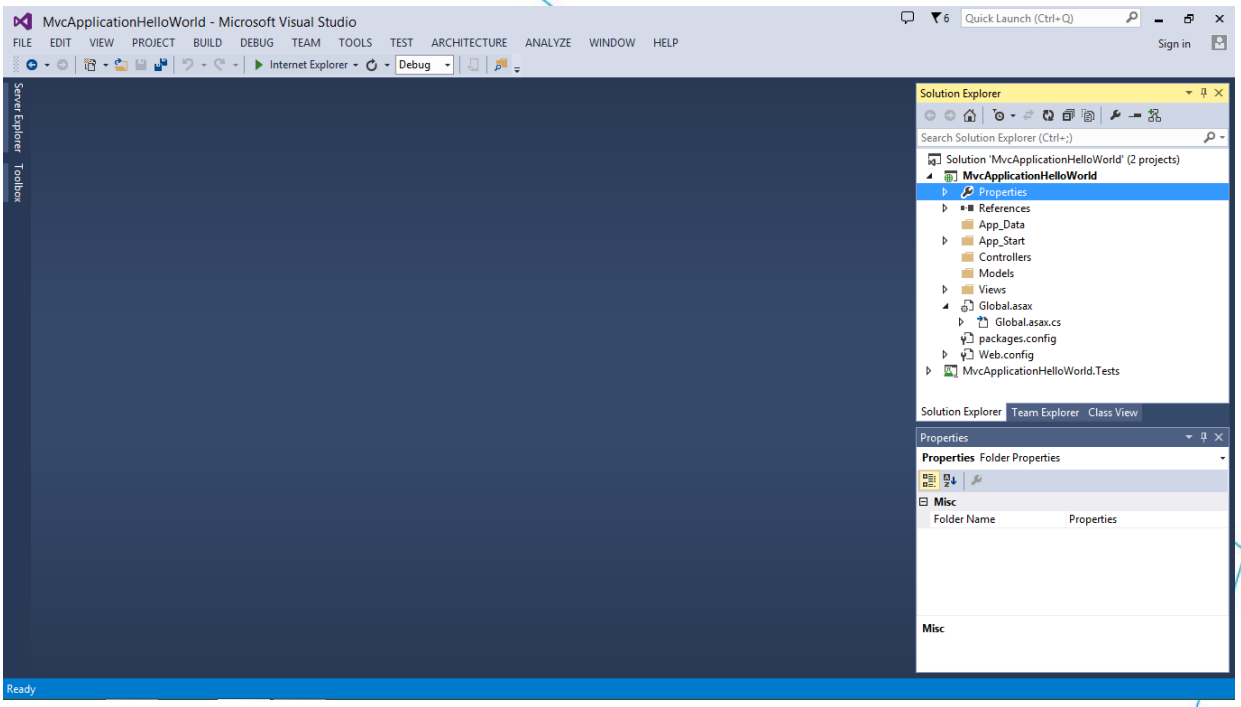





Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES



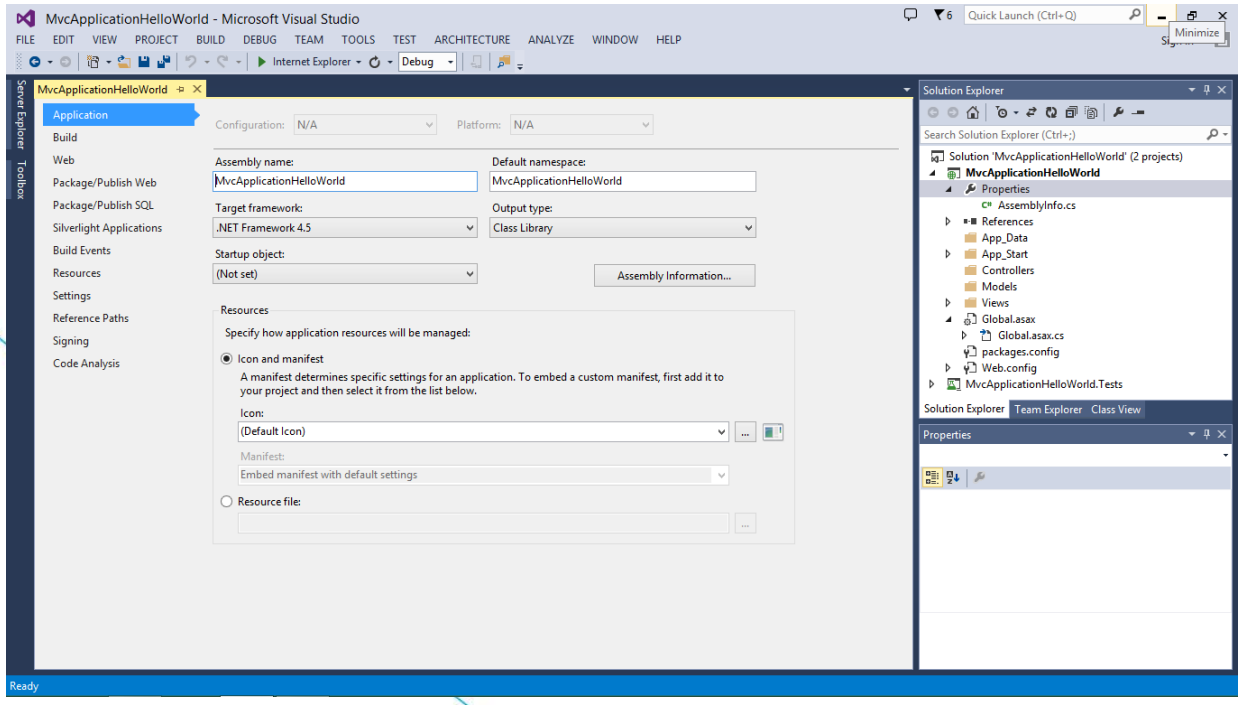

Struktura Empty MVC aplikacije.




Properties prozor Struktura **Empty MVC aplikacij**e, određuje način pravljenja (build) aplikacije. U solution Exporeru klikni desnim tasterom na MvcApplicationHelloWorld.



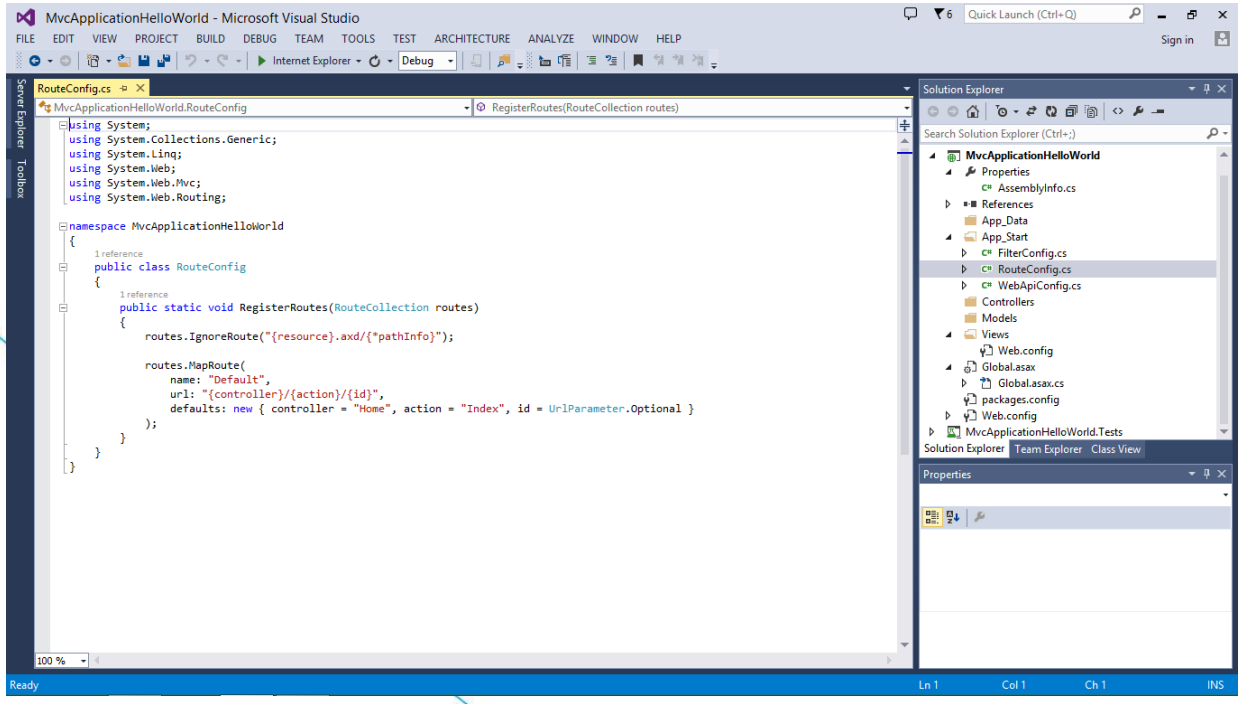
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES



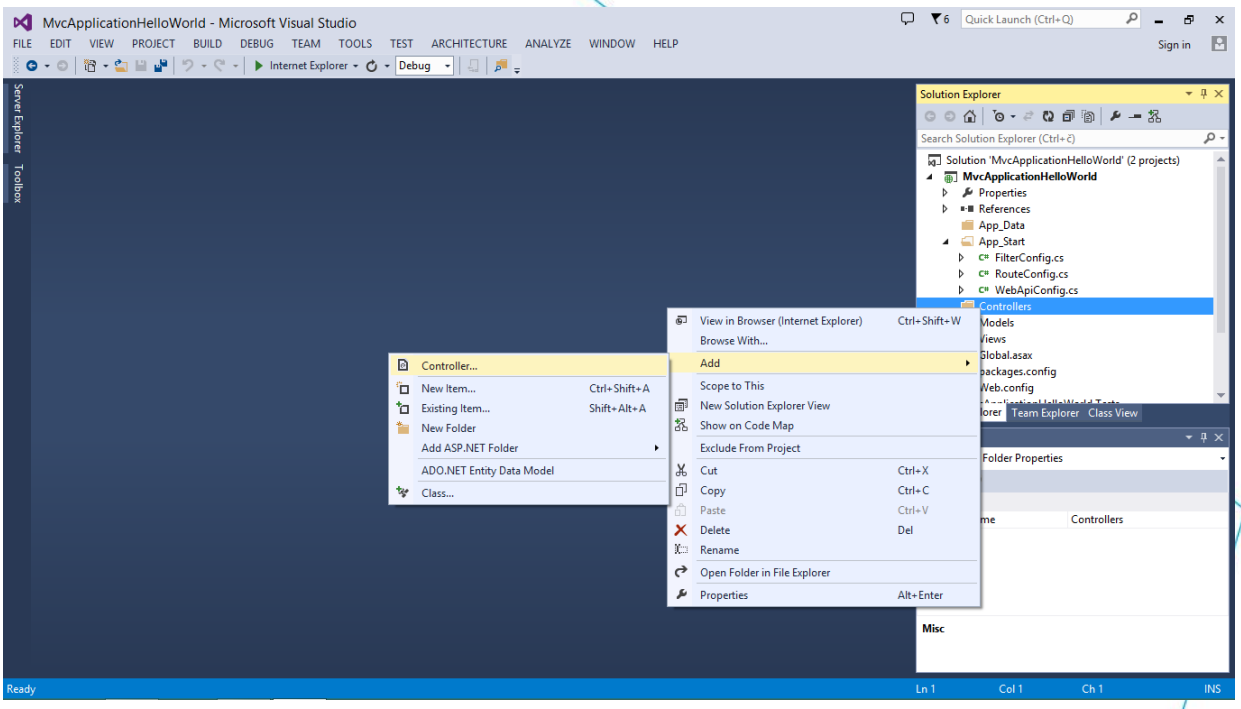

Konfiguracija ruta se obavlja u .cs fajlu **RouteConfig.cs** koji se nalazi u **MvcApplicationHelloWorld/App_Start**. Metoda **RegisterRoutes** koristi **routes.MapRoute** za dodavanje novih ruta. Rute se koriste za pristup aplikaciji, tako da za se za svaki kontroler treba odrediti ruta putem ove klase.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES



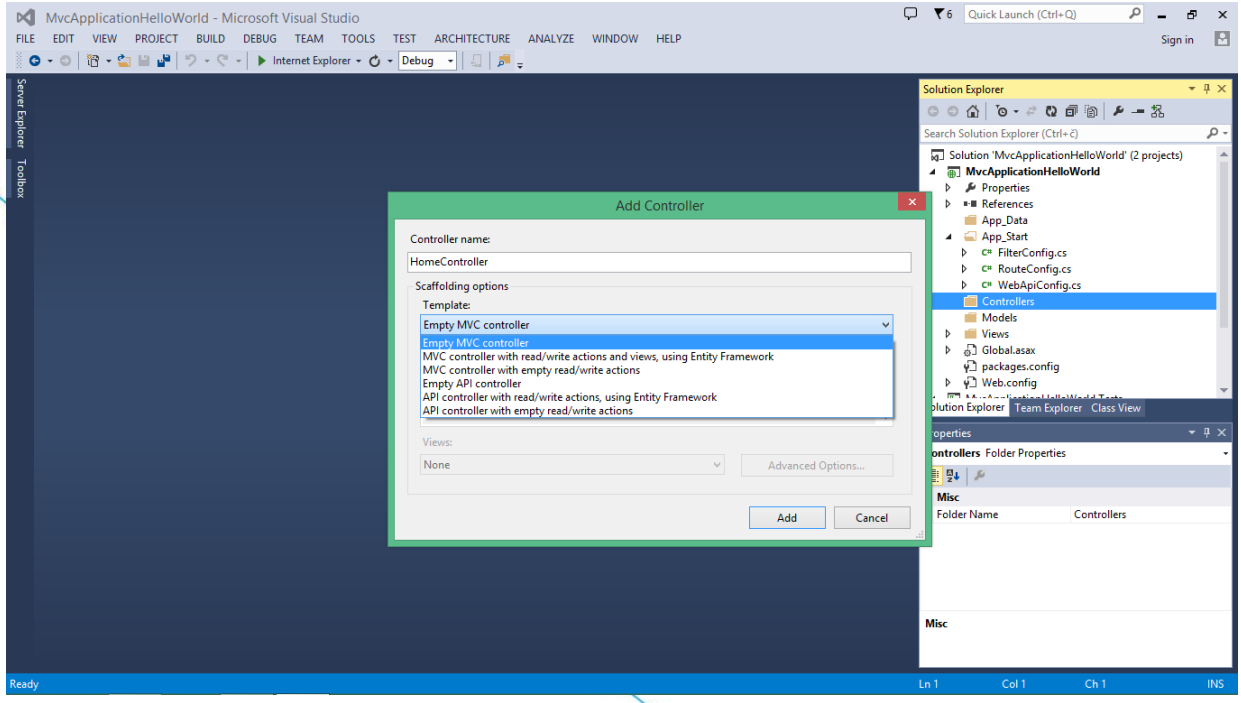

Dodavanje **Controller**-a MvcApplicationHelloWorld aplikaciji. Desni klik na Controllers i izbor Add/Controller.






Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

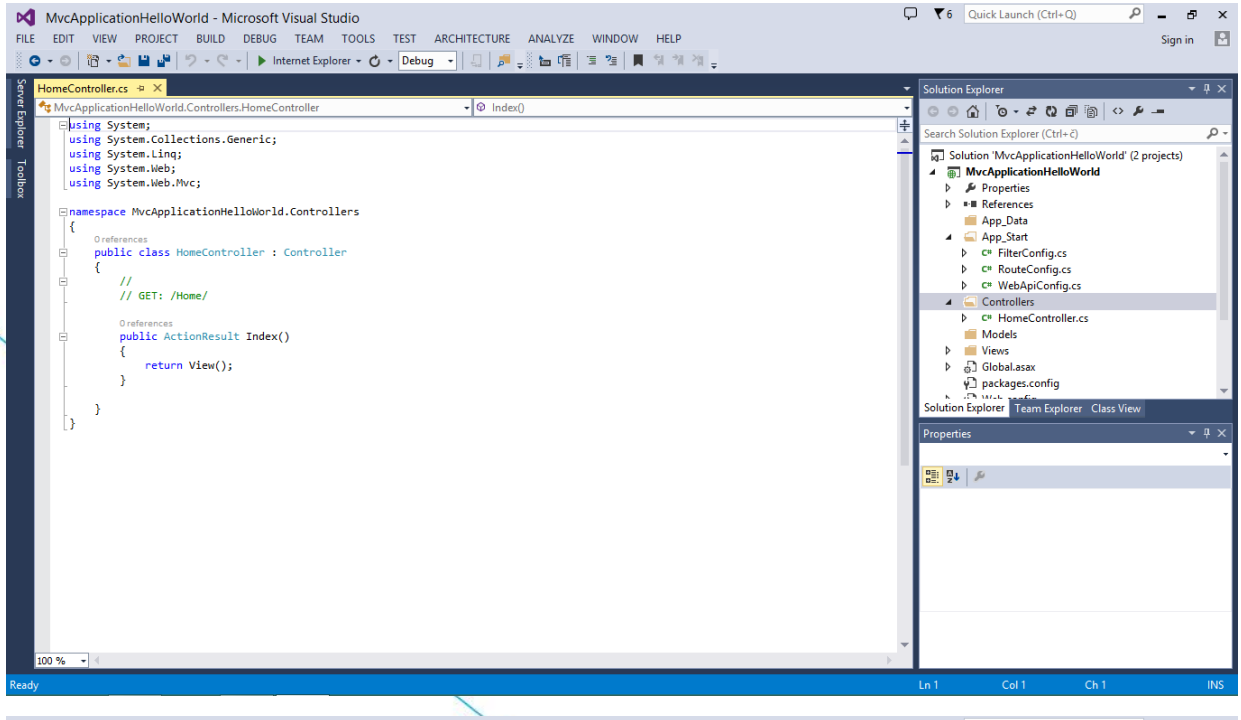
Izaberite imena kontrolera. Kontroleri su zapravo klase koje sadrže C# kod. Unutar njih se nalaze **Akcione metode**. Akcione metode mogu vraćati različat HTML sadržaj, rezultete upuita i rezultati nekog proračuna. Metoda u prikazanom primeru vraća **string**.

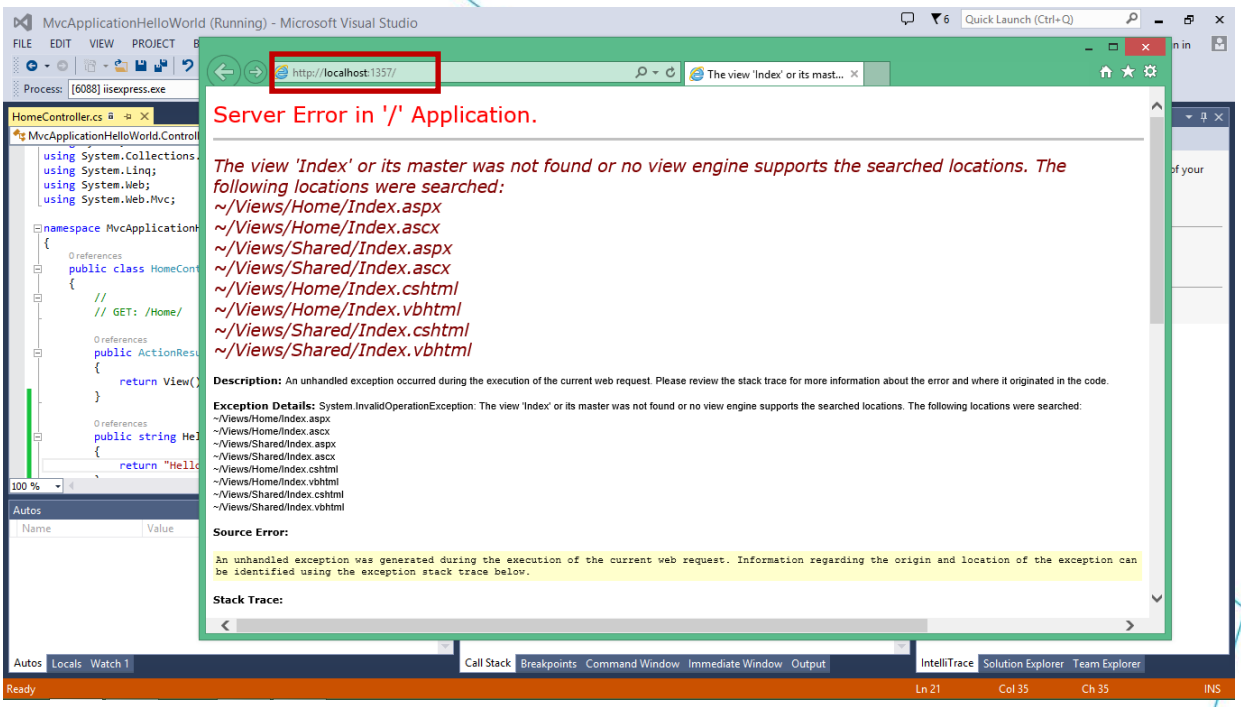



**POKRETANJE AKCIJE** Kako pokrenuti akciju? Objasniti pojam Domena i ruta. Uvek kada se browseru pogodi ruta - dešava se neka akcija. Primer: http:vtsnis.edu.rs/primer/565/1



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

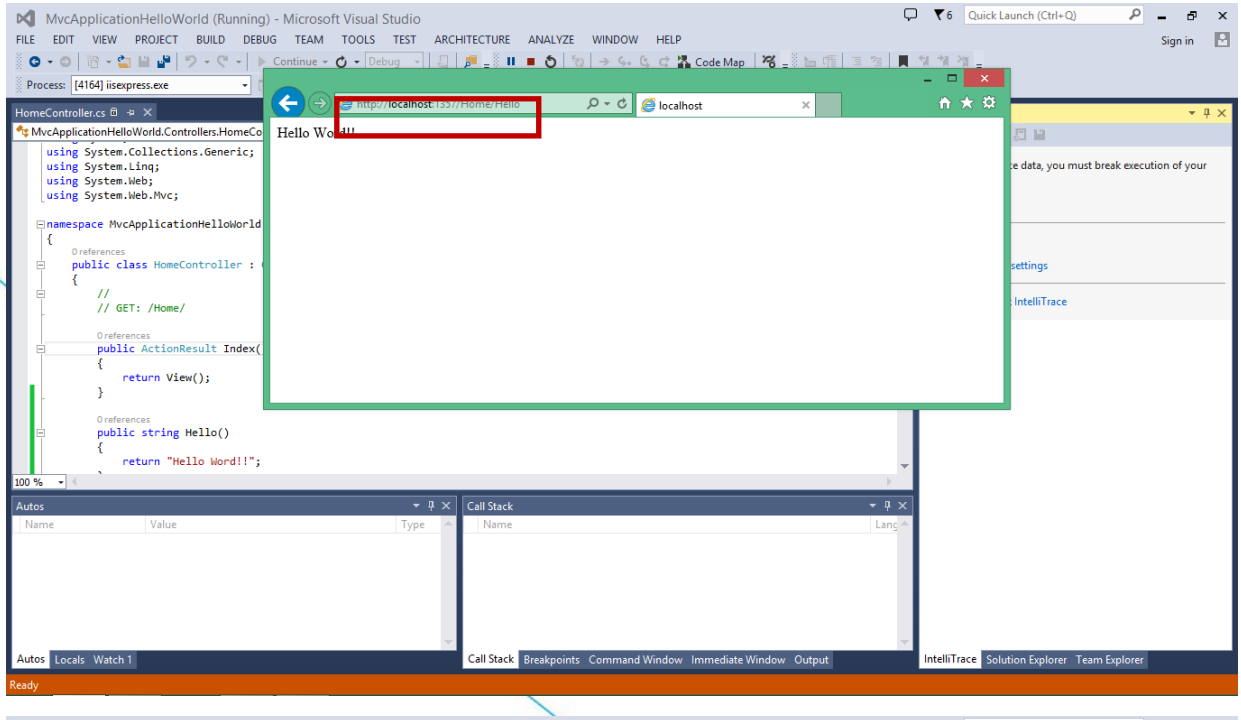


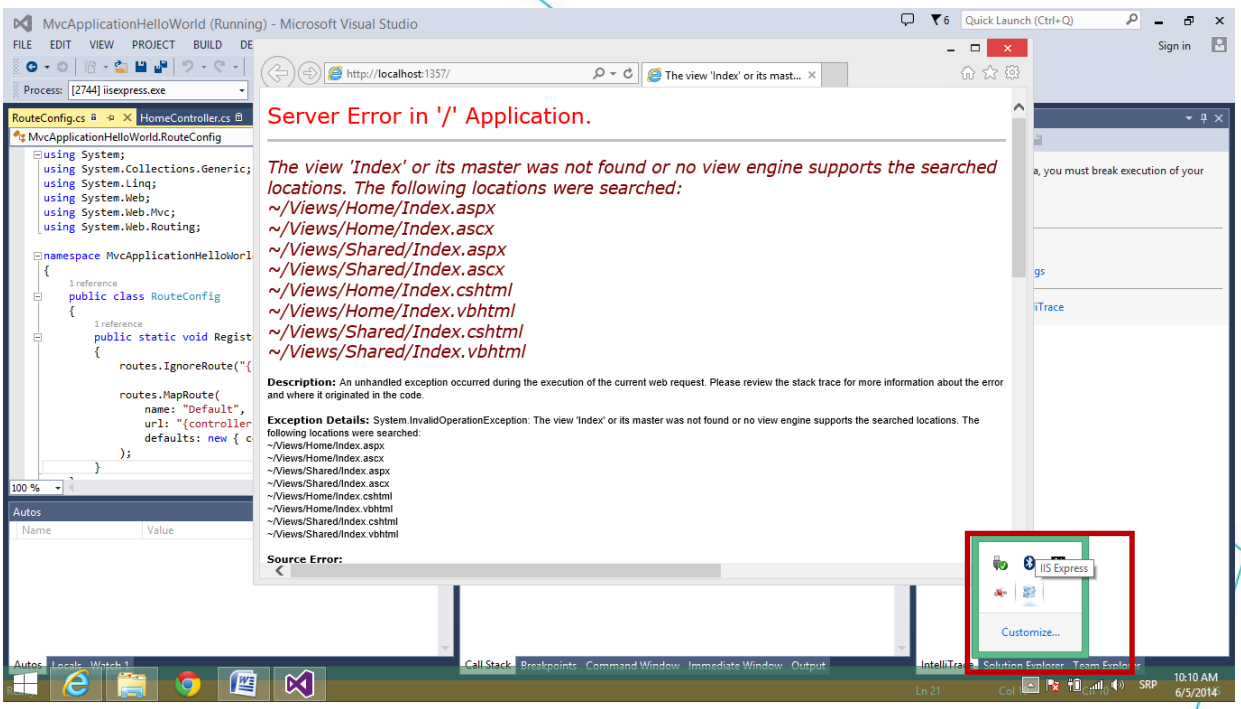




Ako se ne pogodi ruta slično će izgledati poruka framework-a. Nije pronađena odgovarajuća Akciona metoda, dakle zahtev korisnika nije dobro mapiran u klasu. Izvorni kod kontrolera, pojavljuje se u folderu Controllers i ima ekstenziju .cs. Nasleđuje se klasa Controller. U Controller-u treba dodati Akcione metode. Za našu aplikaciju Hello Word napisaćemo Akcionu metodu Helo().



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES



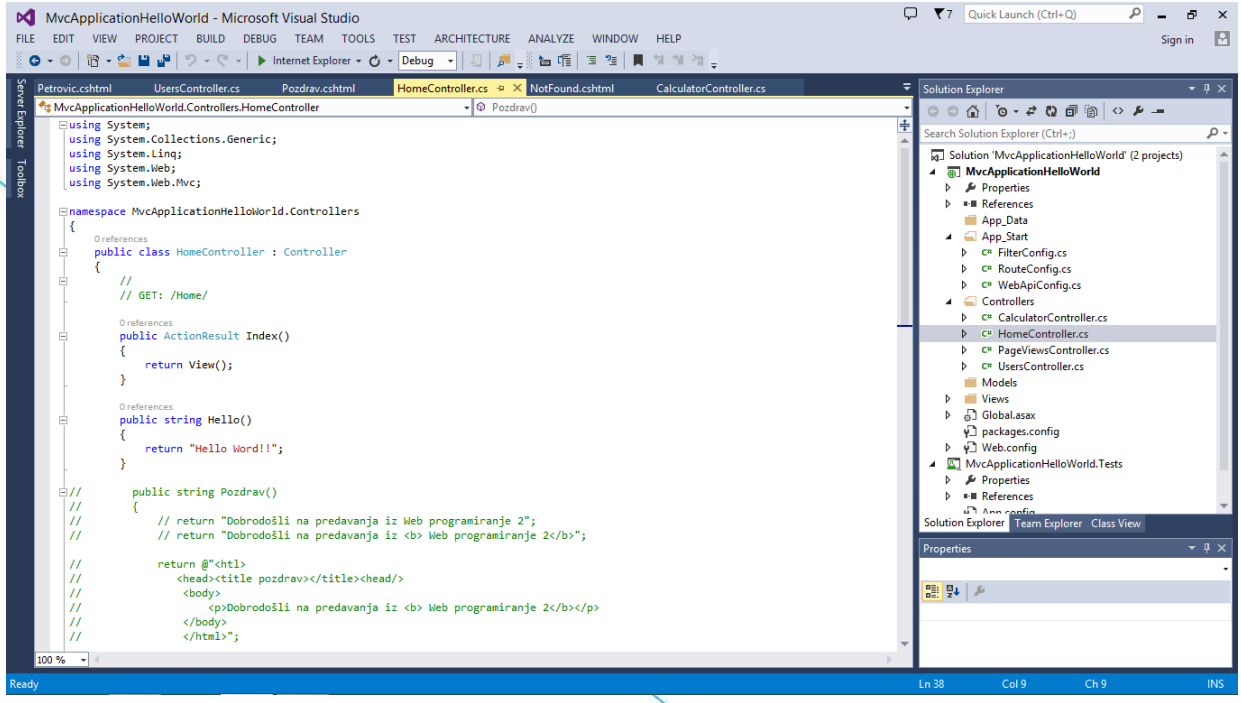




**PORT WEB APLIKACIJE**. Port sa kojim se povezujemo na server. Koji je standardni port za WWW? Port je neobičan i ne odgovara standardnom 80! Evo je način da se obezbedi normalan rad



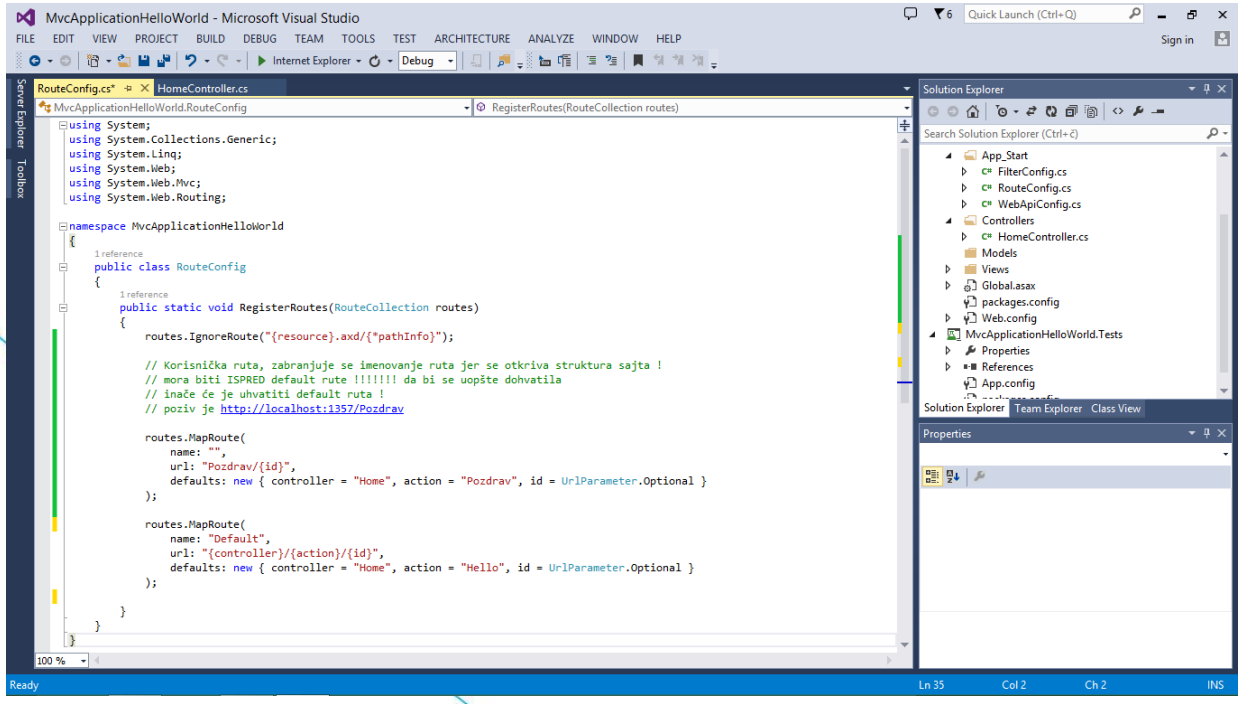
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

browswera bez konflikta. Primetite da se radi sa **localhost**-om jer se podiže lokalni server za potrebe razvoja Web aplikacije. Može se birati koji Web browser ćete koristiti iz menija.




**RUTE I AKCIJE**

Više ruta na istu akciju, ovo više nije popularno a i nije dozvoljeno u MVC 4 jer otkriva strukturu sajta. Dodati rutu na novu Akcionu Metodu u istom kontroleru.




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Routing;

namespace MvcApplicationHelloWorld
{
  public class RouteConfig
  {
    public static void RegisterRoutes(RouteCollection routes)
    {
      routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

      // Korisnička ruta, zabranjuje se imenovanje ruta jer se otkriva struktura sajta !
      // mora biti ISPRED default rute !!!!!!! da bi se uopšte dohvatila
```

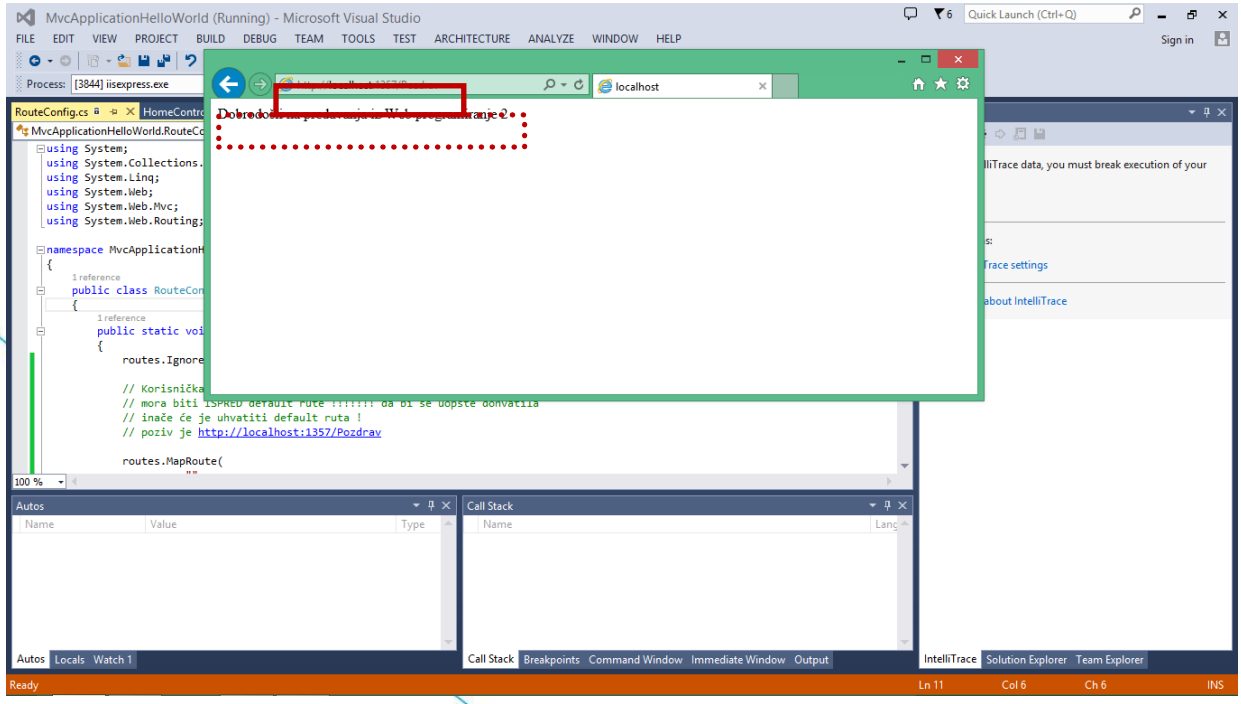


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
            // inače će je uhvatiti default ruta !
            // poziv je http://localhost:1357/Pozdrav

            routes.MapRoute(
                name: "",
                url: "Pozdrav/{id}",
                defaults: new { controller = "Home", action = "Pozdrav", id = UrlParameter.Optional }
            );

            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Hello", id = UrlParameter.Optional }
            );

        }
    }
}
```



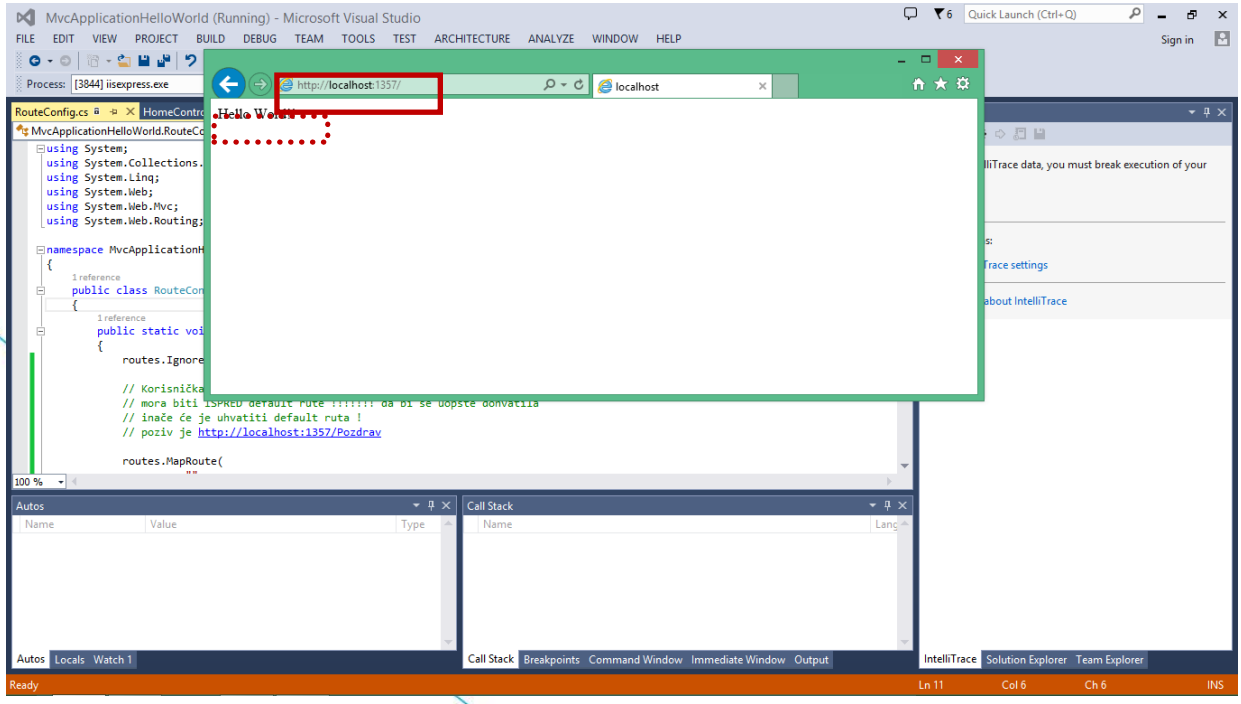
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




**VIŠE AKCIJA**



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

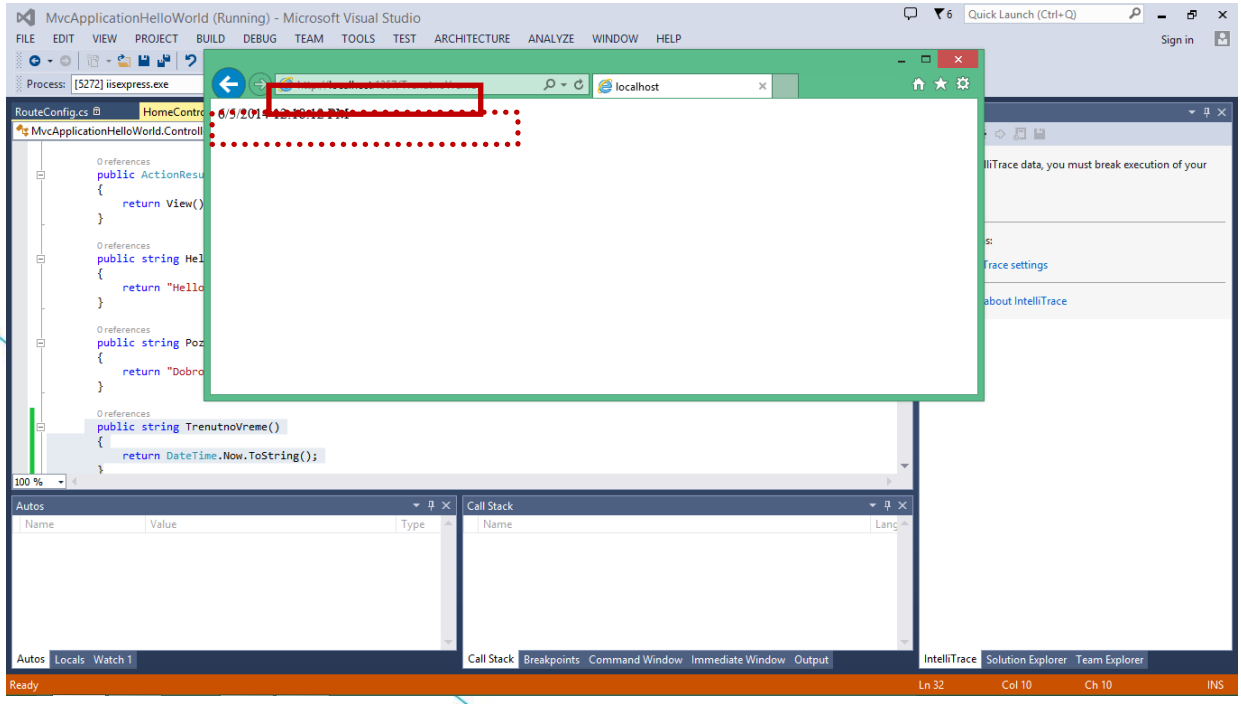



Dodavanje akcione metode u Home kontroleru za ispis tačnog vremena:

```
public string TrenutnoVreme()
    {
        return DateTime.Now.ToString();
    }
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES



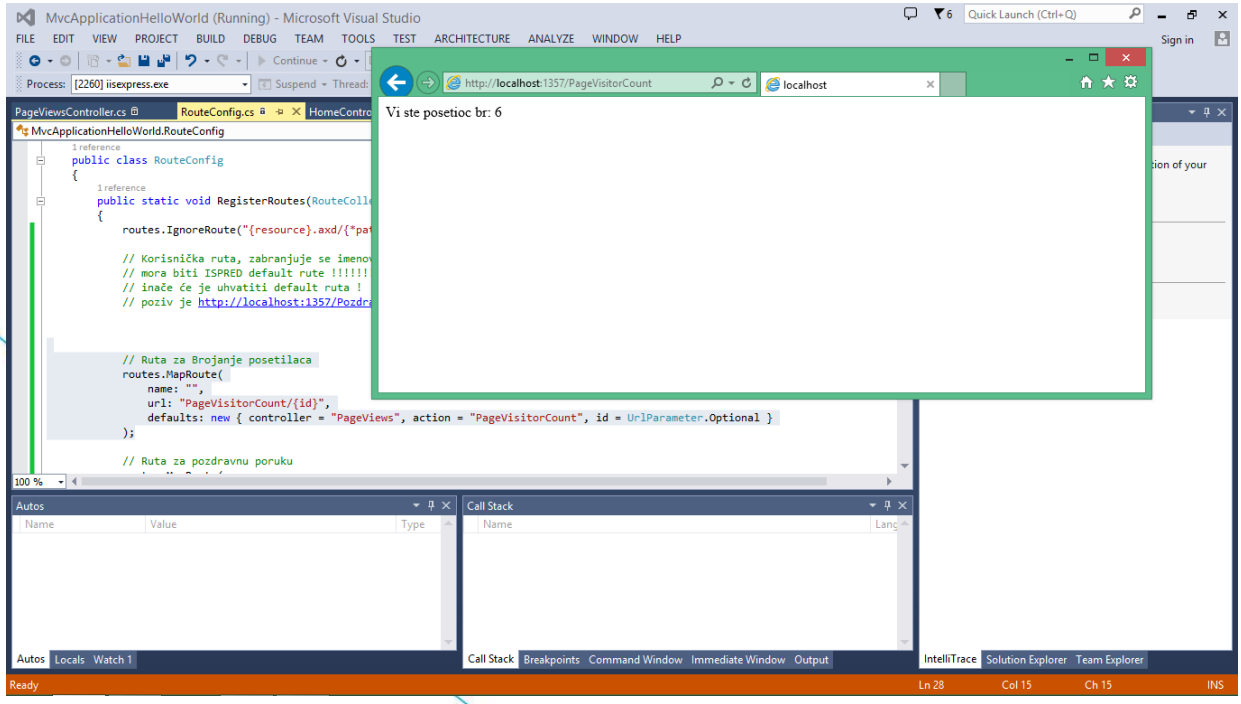

**BROJANJE POSETILACA STRANICE**

Dodaje se novi kontroler PageViews koji poseduje PageVisitorCount metodu:

```
public string PageVisitorCount()
    {
        count++;
        return "Vi ste posetioc br: " + count;

    }
```

Treba dodati rutu na ovu akciju iz novog kontrolera

```
        // Ruta za Brojanje posetilaca
        routes.MapRoute(
            name: "",
            url: "PageVisitorCount/{id}",
            defaults: new { controller = "PageViews", action = "PageVisitorCount", id = UrlParameter.Optional }
        );
```




Primetite da je promenljiva **count** statična, inače se nebi uvećavala. Treba smestiti u bazu podataka.

**METODE SA PARAMETRIMA**

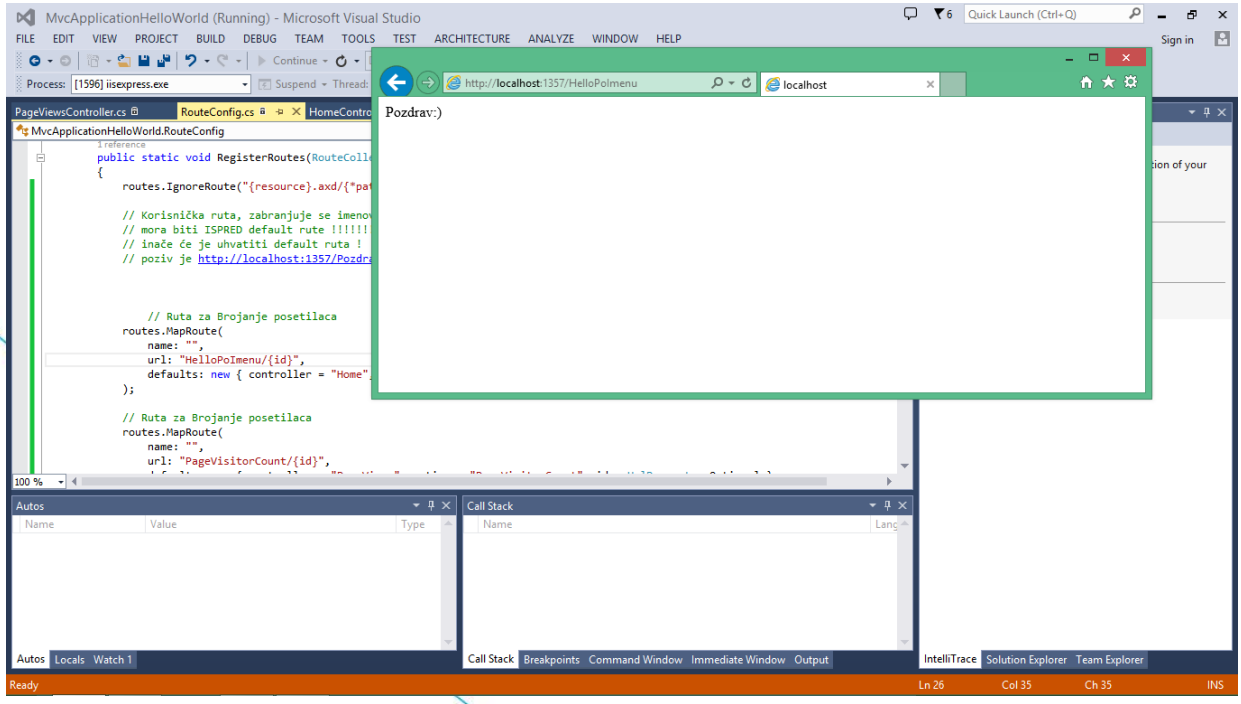
Pozdrav odgovarajućem korisiku. Kako dodati parametre sa rutama. Koristi se **url** parametri, pokušava se izbaciti. Lista parametara može veoma narasti i tako zakomplikovati.

url?key=value&key=value

Bez parametara:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




Dodati **akcionu metodu** u kontroler Home:

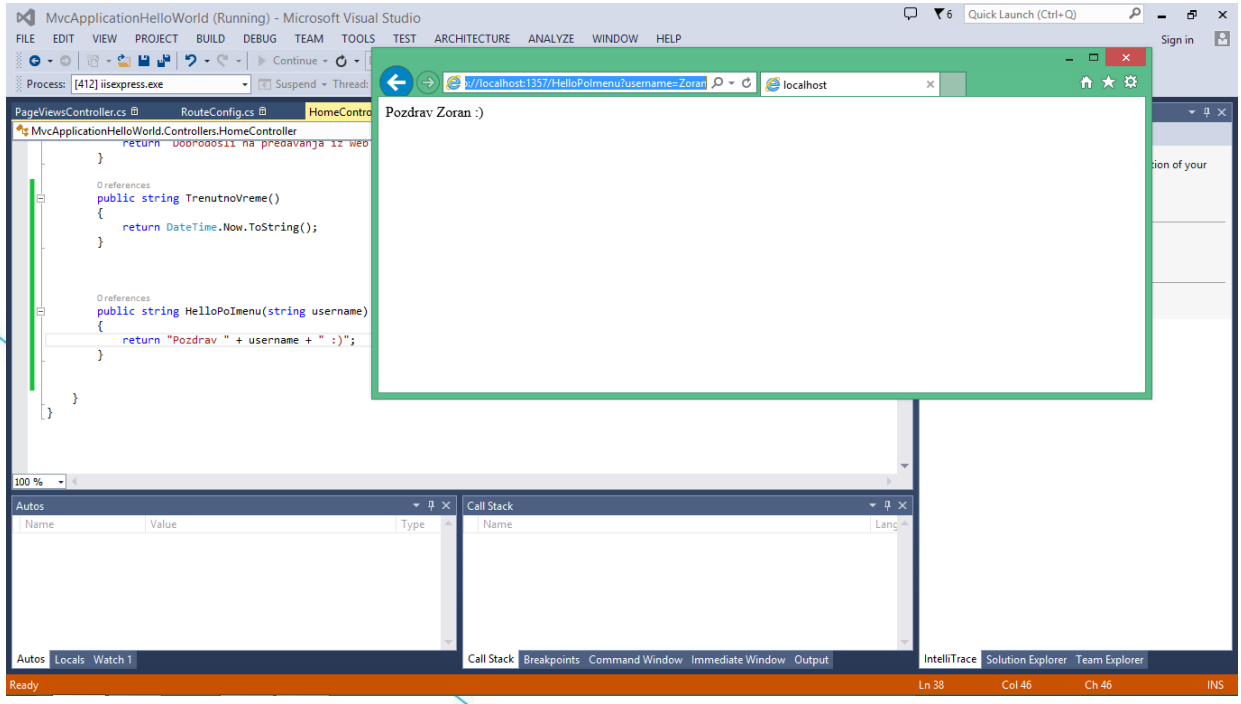
```
public string HelloPoImenu(string username)
{
    return "Pozdrav " + username + " :)";
}
```

Dodati **rutu** za Pozdrav po imenu:

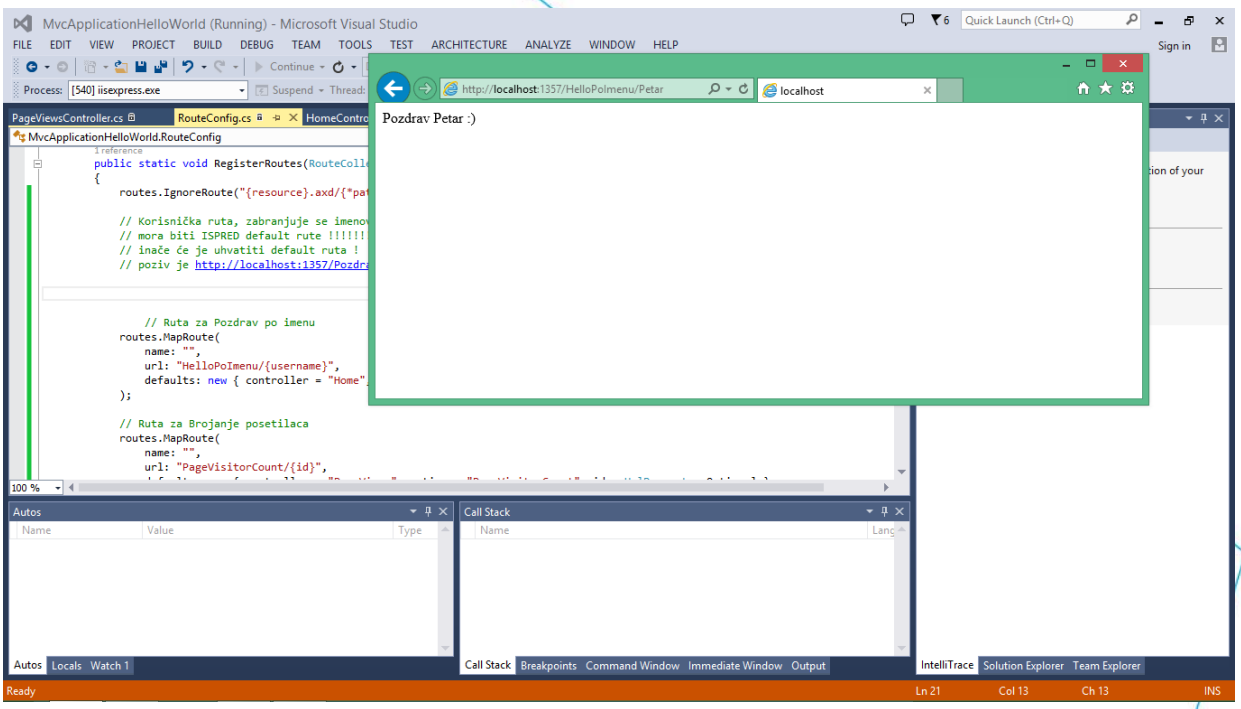
```
        // Ruta za Pozdrav po imenu
    routes.MapRoute(
        name: "",
        url: "HelloPoImenu/{id}",
        defaults: new { controller = "Home", action = "HelloPoImenu", id = UrlParameter.Optional }
    );
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




Korišćenje parametara u ruti.

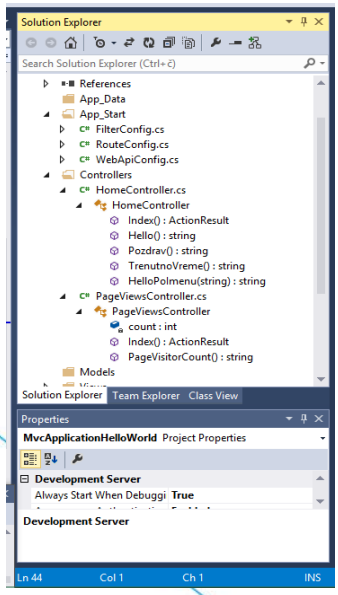



```
// Ruta za Pozdrav po imenu sa parametrima
routes.MapRoute(
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
        name: "",
        url: "HelloPoImenu/{username}",
        defaults: new { controller = "Home", action = "HelloPoImenu", username = UrlParameter.Optional }
    );
```

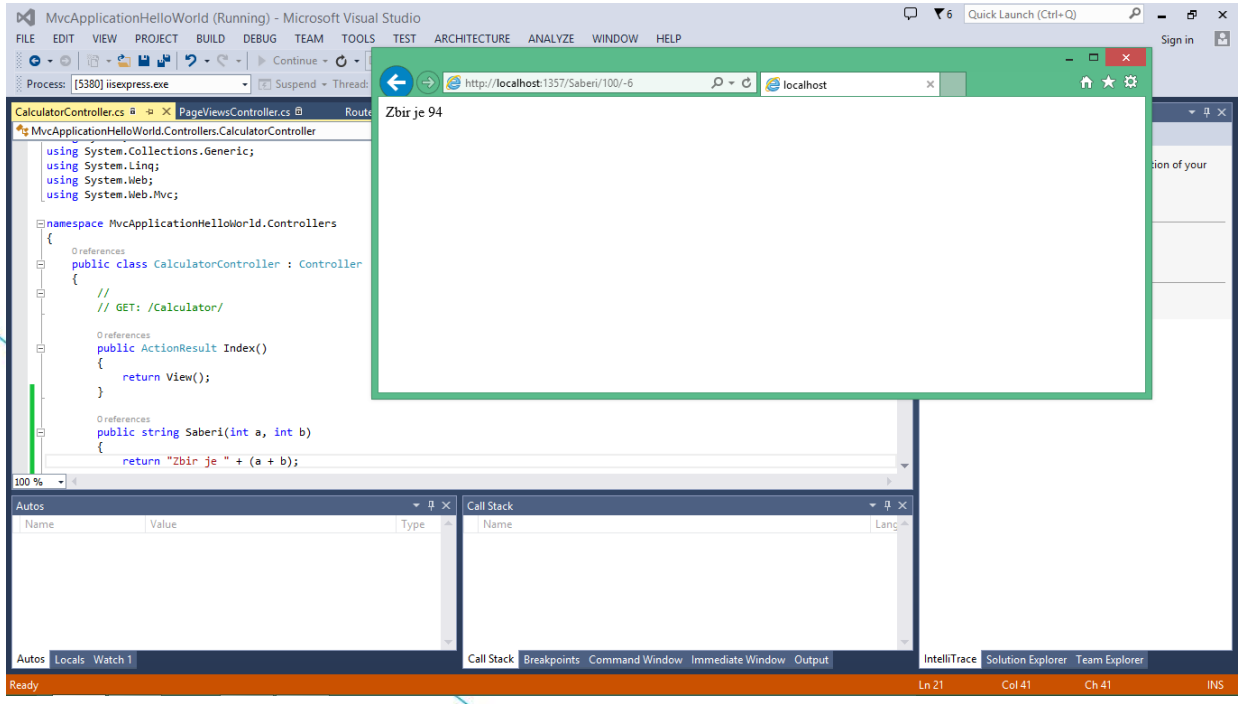



**KALKULATOR**

Formira se novi **kontroler** Calculator.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




Dodajte **Akcionu metodu** Saberi (int a, int b) u Calculator kontroleru.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MvcApplicationHelloWorld.Controllers
{
   public class CalculatorController : Controller
   {
      //
      // GET: /Calculator/

      public ActionResult Index()
      {
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

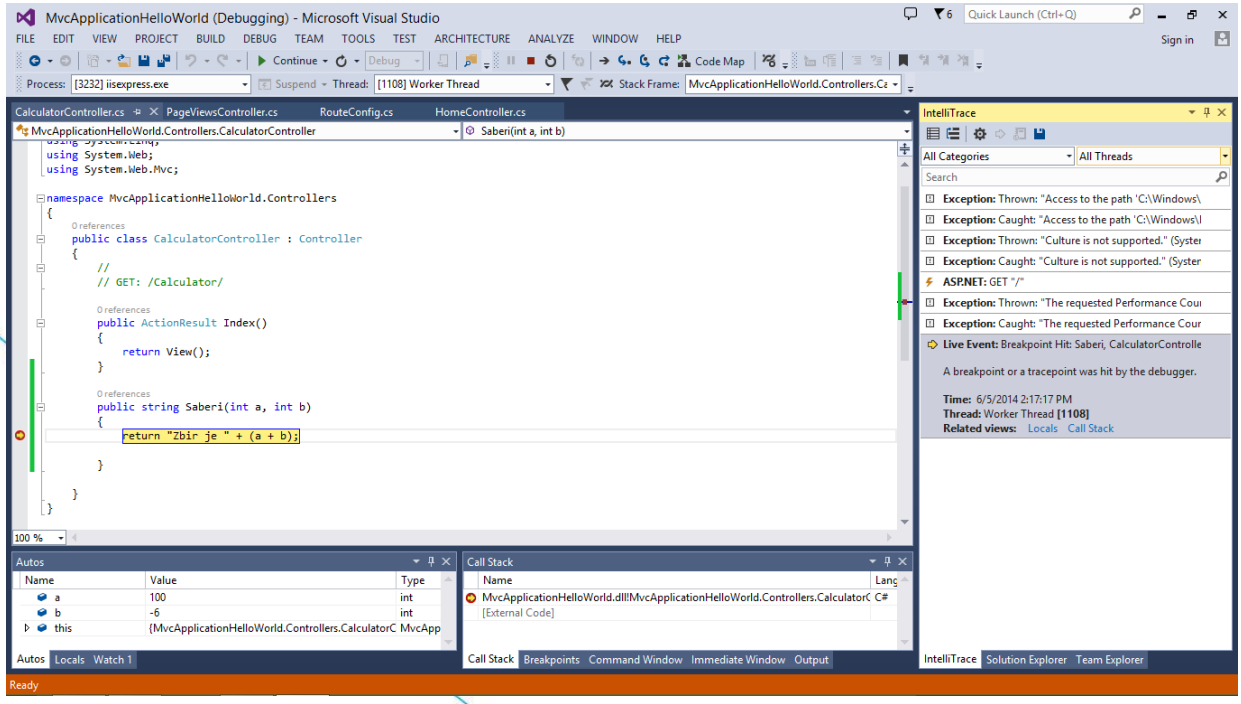
```
        return View();
    }

    public string Saberi(int a, int b)
    {
        return "Zbir je " + (a + b);

    }

  }
}
```

**Dodaj rutu** za Kalkulator rutu:

```
// Ruta za Kalkulator
        routes.MapRoute(
            name: "",
            url: "Saberi/{a}/{b}",
            defaults: new { controller = "Calculator", action = "Saberi", username = UrlParameter.Optional }
        );
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




Bitno je ime i tip promenljive u **url** pozivu. Promenljive se u debagu mogu lepo pregledati. Primer rute za forume.

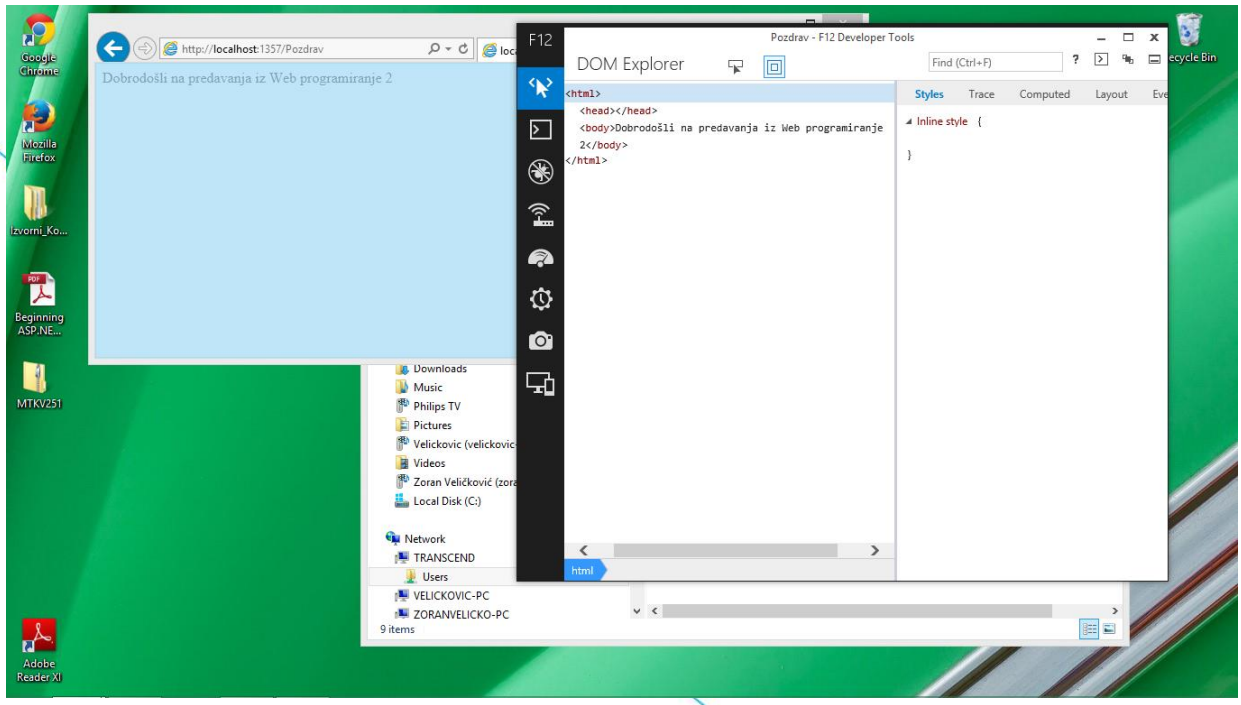


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**Prilog 2: Vežba br. 2**

VIEW, HTML KOD, RAZOR, HELPER METODE, MVC FORME, PRIJAVNA STRANICA

GDE SE TO KRIJE **HTML**? Pogledajte u razvojnom alatu **IE**-a **DOM** strukturu prikazane stranice.




Može se vratiti I HTML kod, evo primera kako se Pozdrav() može formatirati HTML-om. Sve se isto vraća kao I bez njega.

```
public string Pozdrav()
{
  // return "Dobrodošli na predavanja iz Web programiranje 2";
  // return "Dobrodošli na predavanja iz <b> Web programiranje 2</b>";

  return @"<htl>
    <head><title pozdrav></title><head/>
     <body>
       <p>Dobrodošli na predavanja iz <b> Web programiranje 2</b></p>
     </body>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
        </html>";

    }
```

Nije zgodno jer pri pisanju HTML-a ne radi Intelisense I sve se "crveni"!! Ne treba u C# pisati HTML kod, teško se održava kod.

Ne treba pisati HTML kod u akcijama!!!

**ŠTA JE REŠENJE ?---- VIEW-i**

Često se iz metoda vraćaju različiti podaci. Textovi ponajmanje. Uglavnom se vraća HTML kod. Taj HTML kod je izdvojen u View-ima. Sada se nazivaju .CSHTML. Svaki kontroler ima svoj View.

Da bi akcija vratila View, mora se vratiti neki objekt. Koristi se return View("Ime_view-a"). Treba promeniti return values i vraća se ActionResult tip objekta.Iz imena View-a ukloniti nastavak View!!!

**Pozdrav.cshtml:**

```
<html>
  <head>
    <title>pozdrav</title> </head>
    <body>
      <p>Dobrodošli na predavanja iz <b> Web programiranje 2</b></p>
  </body>
</html>
```

```
    public ActionResult Pozdrav()
    {
      // return "Dobrodošli na predavanja iz Web programiranje 2";
      // return "Dobrodošli na predavanja iz <b> Web programiranje 2</b>";

      return View("Pozdrav");
    }
```



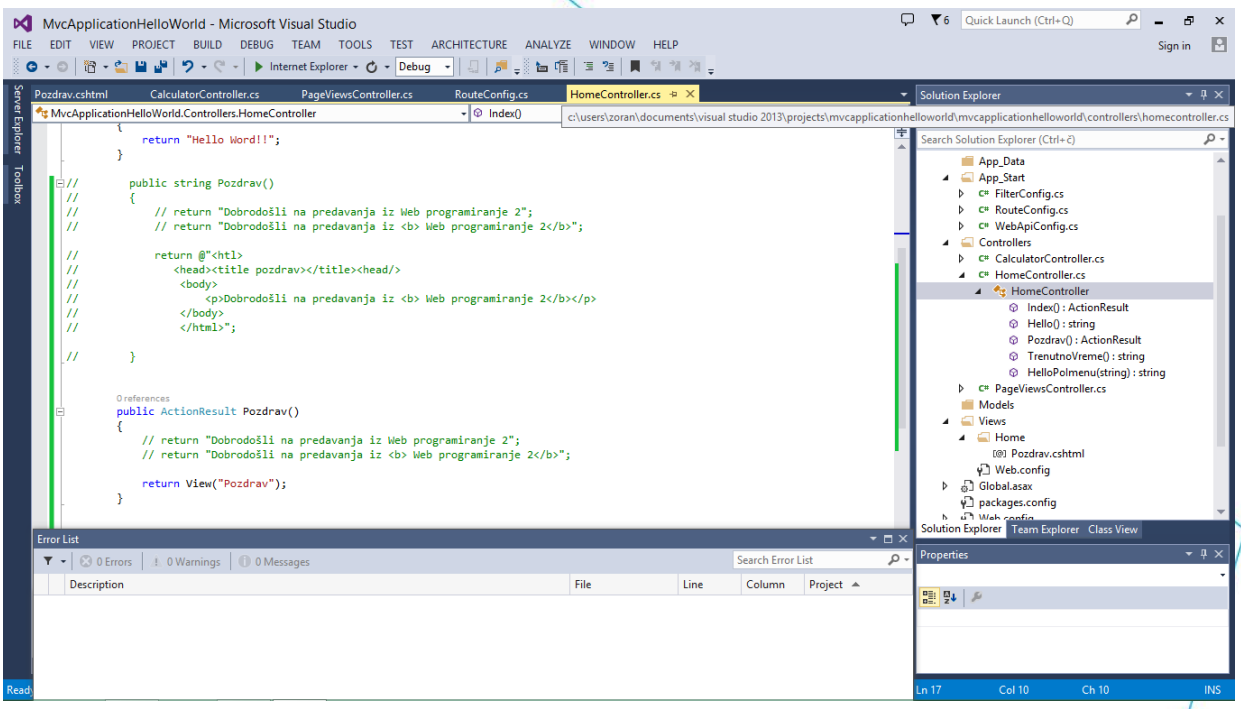
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Pogledaj sada strukturu aplikacije. Otvoreno je u View novi direktorijum za svaki kontroler, a u okviru njega za svaku ActionResult metodu odgovarajući View.

Izmena u kontroleru Home i metode Pozdrav(). Promeniti povratnu vrednost metode i povratni tip!!

MOŽE I OVAKO:

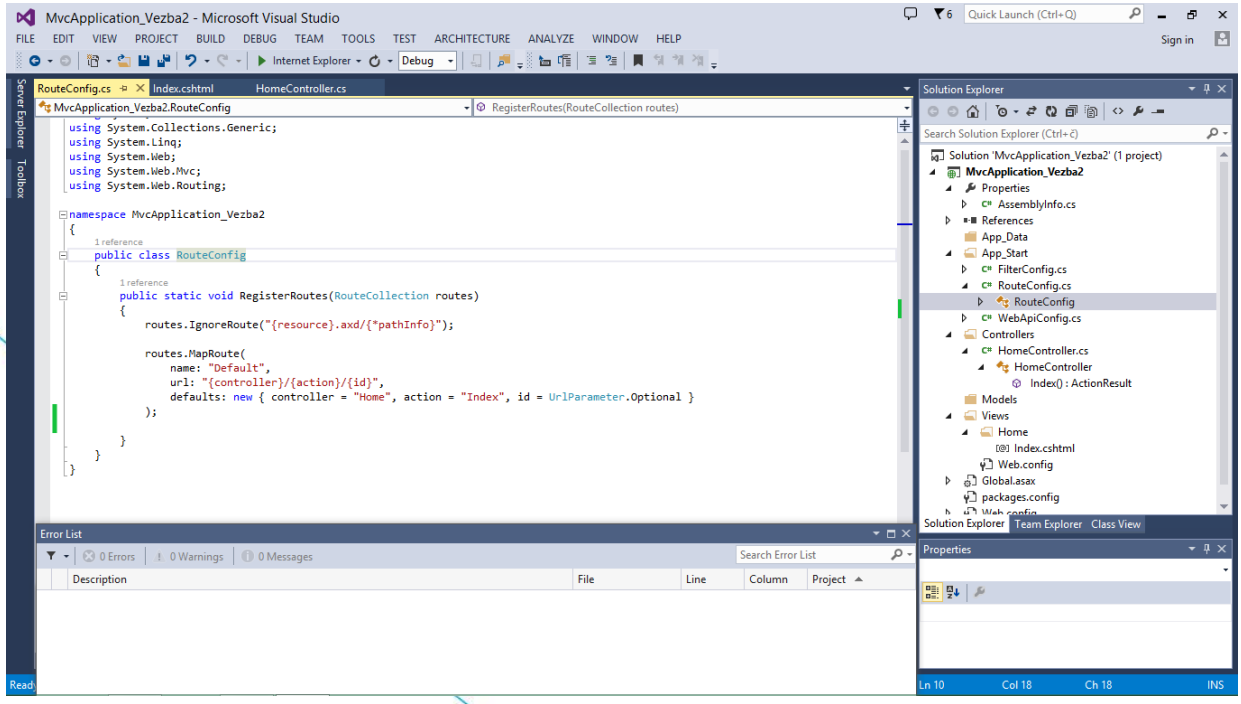
```
public ActionResult Pozdrav()
    {
        // return "Dobrodošli na predavanja iz Web programiranje 2";
        // return "Dobrodošli na predavanja iz <b> Web programiranje 2</b>";

        //return View("Pozdrav");
        // može i ovako jer View odgovara ovoj akciji, tako da se ime može izostaviti
        return View();
    }
```

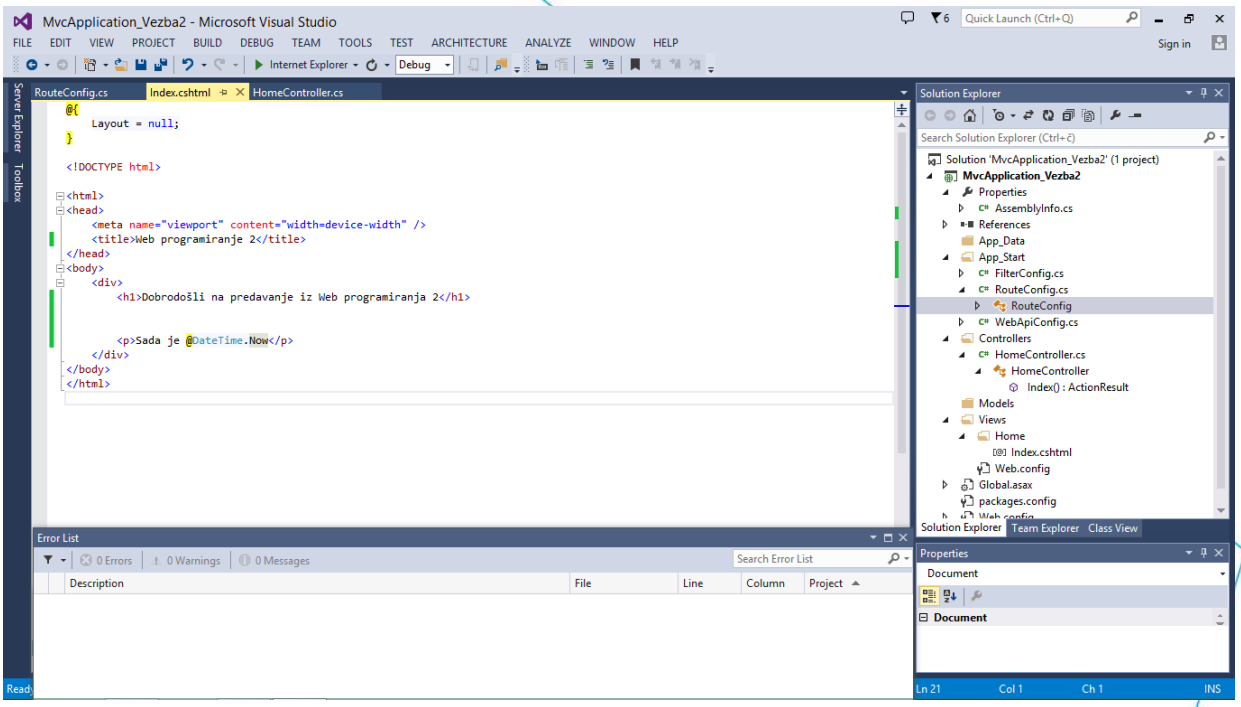



Formira se jednostavna Web aplikacija sa Home kontroler i Index akcijom



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




RAZOR ne zahteva veliku pažnju.

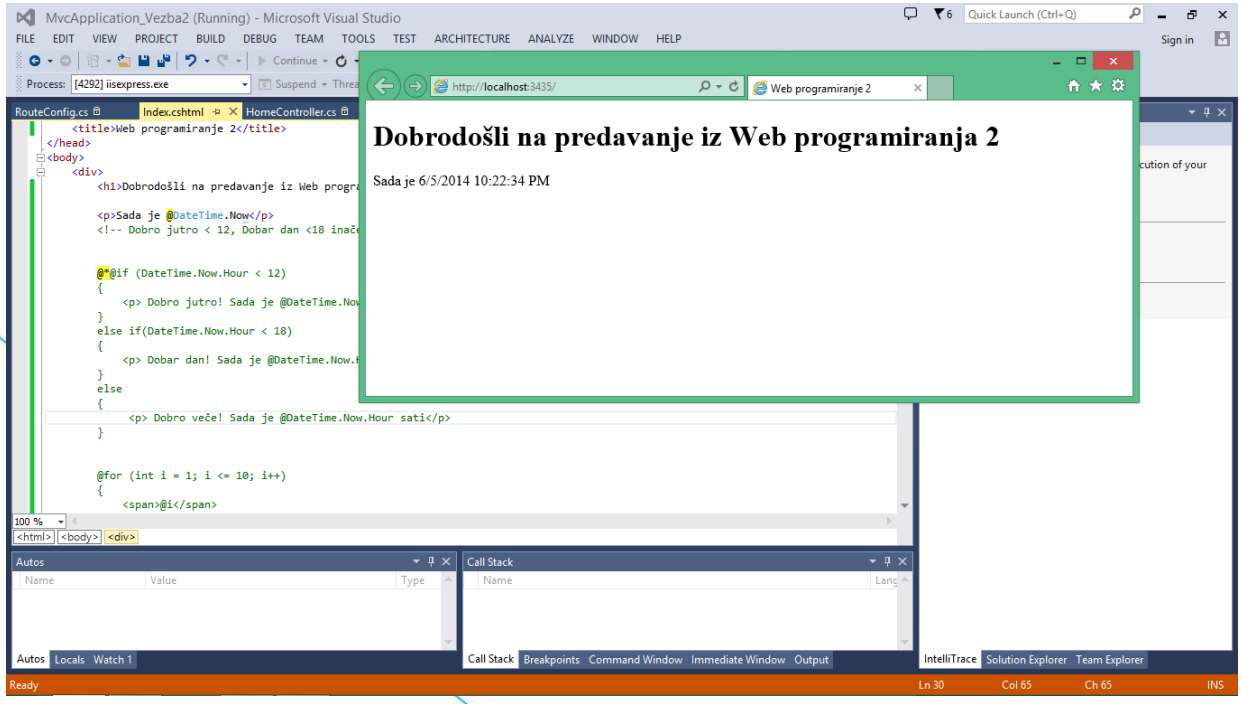


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

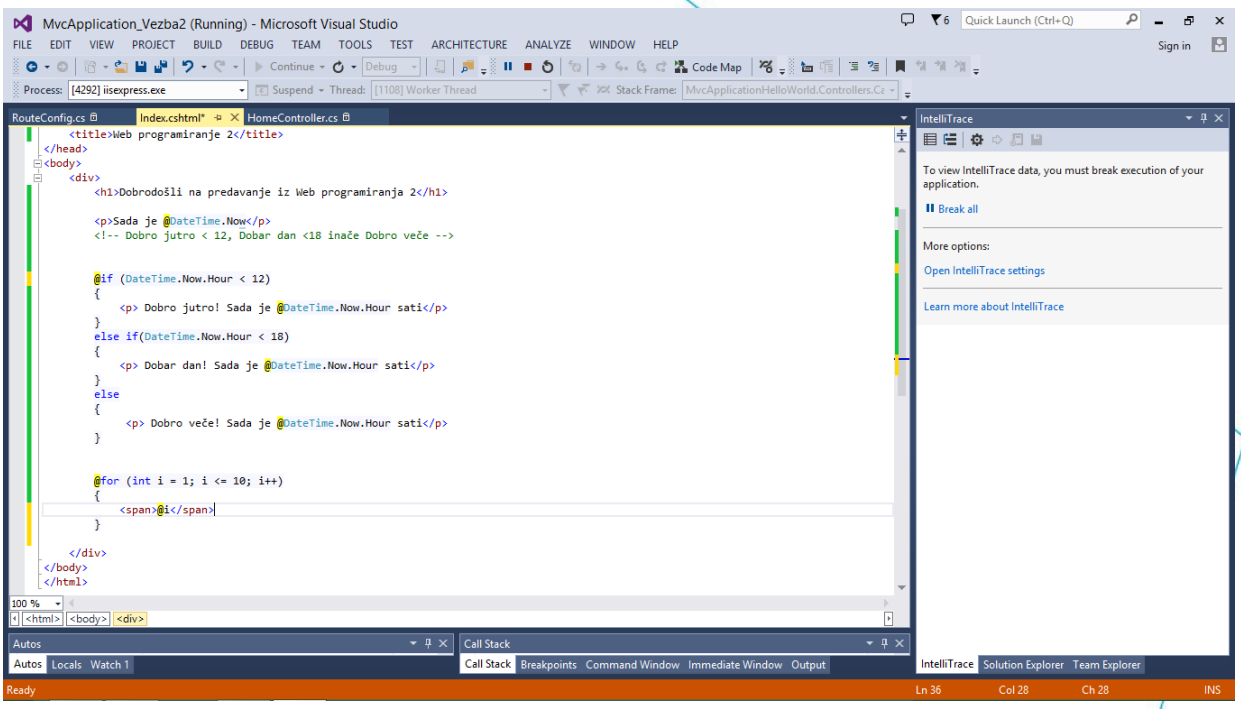
```
@{
   Layout = null;
}

<!DOCTYPE html>
<html>
<head>
   <meta name="viewport" content="width=device-width" />
   <title>Web programiranje 2</title>
</head>
<body>
   <div>
      <h1>Dobrodošli na predavanje iz Web programiranja 2</h1>



      <p>Sada je @DateTime.Now</p>
   </div>
</body>
</html>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




Dodaje se **logika** i View-u uz RAZOR koji izračunava i prikazuje izraze.





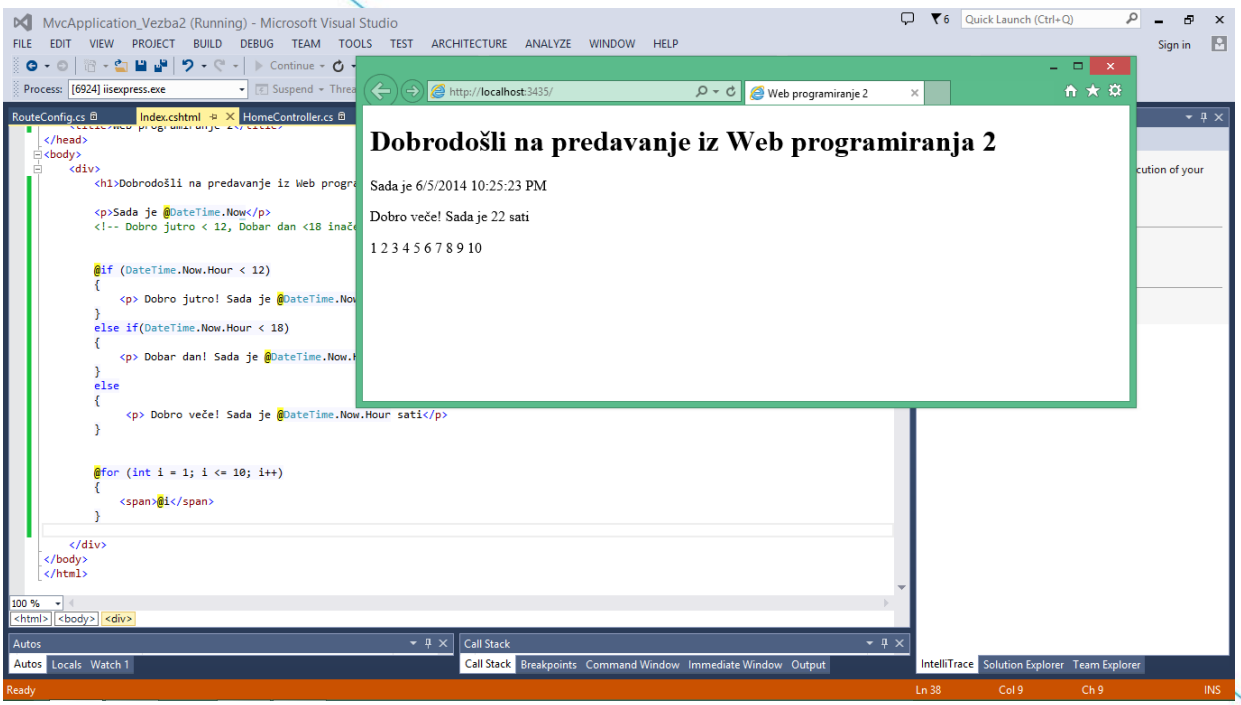

Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
@{
   Layout = null;
}

<!DOCTYPE html>

<html>
<head>
   <meta name="viewport" content="width=device-width" />
   <title>Web programiranje 2</title>
</head>
<body>
   <div>
      <h1>Dobrodošli na predavanje iz Web programiranja 2</h1>

      <p>Sada je @DateTime.Now</p>
      <!-- Dobro jutro < 12, Dobar dan <18 inače Dobro veče -->


      @if (DateTime.Now.Hour < 12)
      {
         <p> Dobro jutro! Sada je @DateTime.Now.Hour sati</p>
      }
      else if(DateTime.Now.Hour < 18)
      {
         <p> Dobar dan! Sada je @DateTime.Now.Hour sati</p>
      }
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
    else
    {
        <p> Dobro veče! Sada je @DateTime.Now.Hour sati</p>
    }
    @for (int i = 1; i <= 10; i++)
    {
       <span>@i</span>
    }

  </div>
</body>
</html>
```




U RAZOR-u se može koristiti C# kod vrlo jednostavno. Pogledajte kako se formira promenljiva sada.

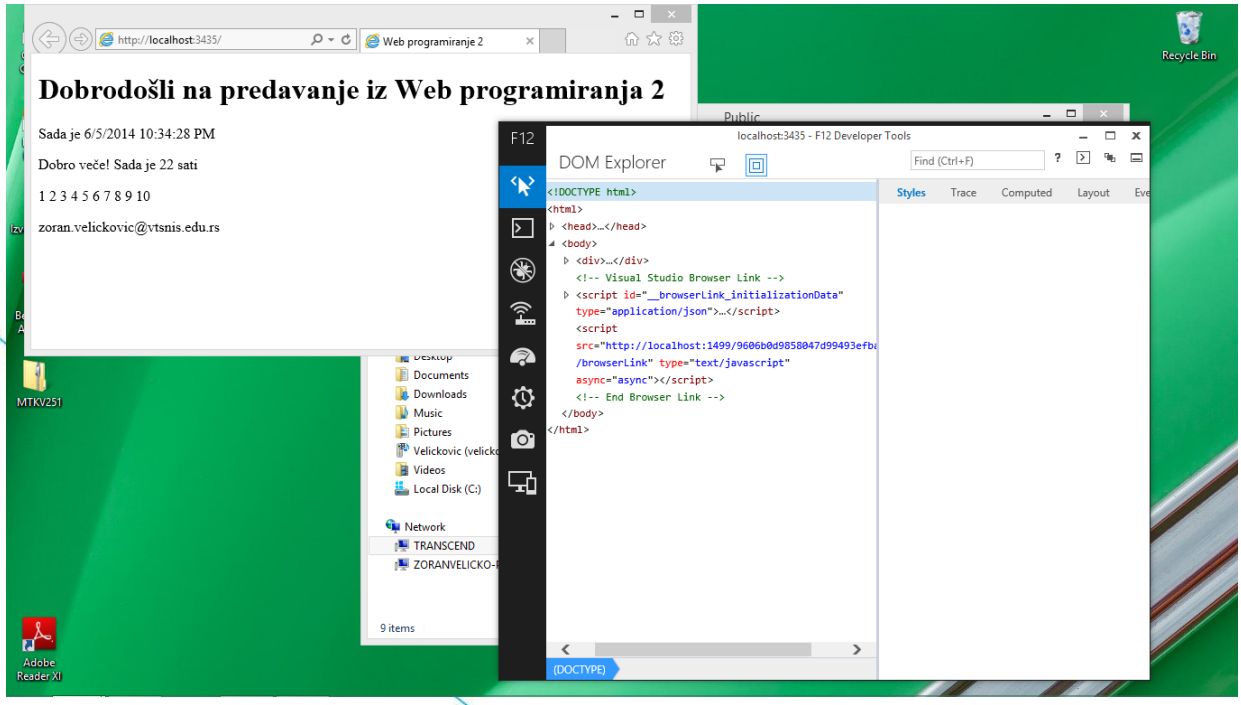
```
@{
  var sada = DateTime.Now;
}
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<!DOCTYPE html>

<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>Web programiranje 2</title>
</head>
<body>
  <div>
    <h1>Dobrodošli na predavanje iz Web programiranja 2</h1>

    <p>Sada je @sada</p>
    <!-- Dobro jutro < 12, Dobar dan <18 inače Dobro veče -->

. . .
```

RAZOR sve C# I HTML kod prevodi u HTML kod.




Podaci se prenose iz kontrolera u View pomoći **ViewBag.ime_promenljive**.

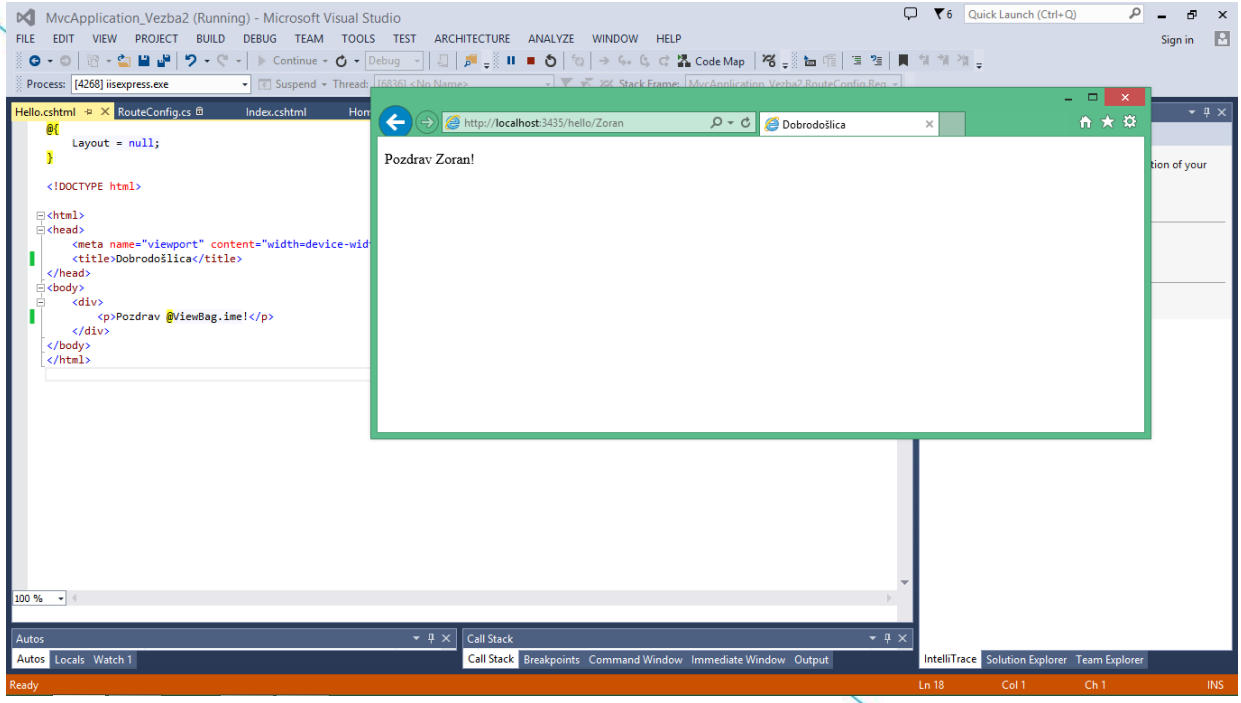
```
public ActionResult Hello(string name)
{
    ViewBag.ime = name;
    return View();
}
```

```
@{
  Layout = null;
}
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>Dobrodošlica</title>
</head>
```



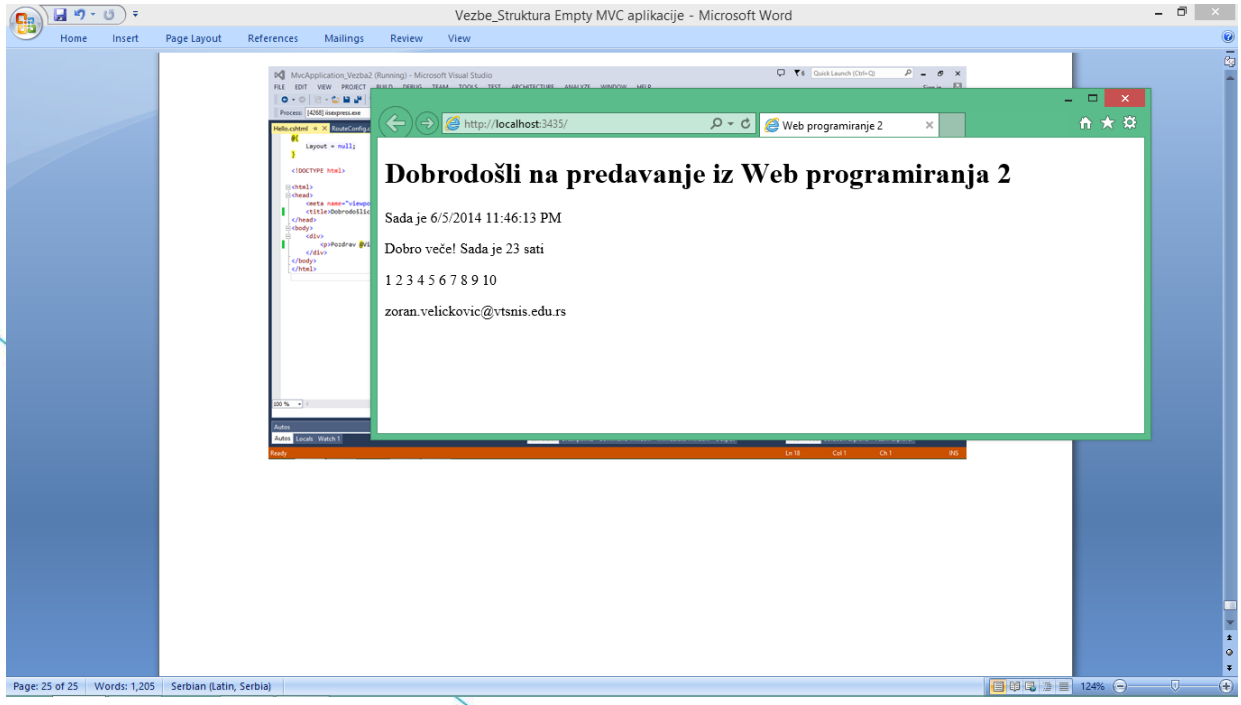
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<body>
  <div>
    <p>Pozdrav @ViewBag.ime!</p>
  </div>
</body>
</html>
```






Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES



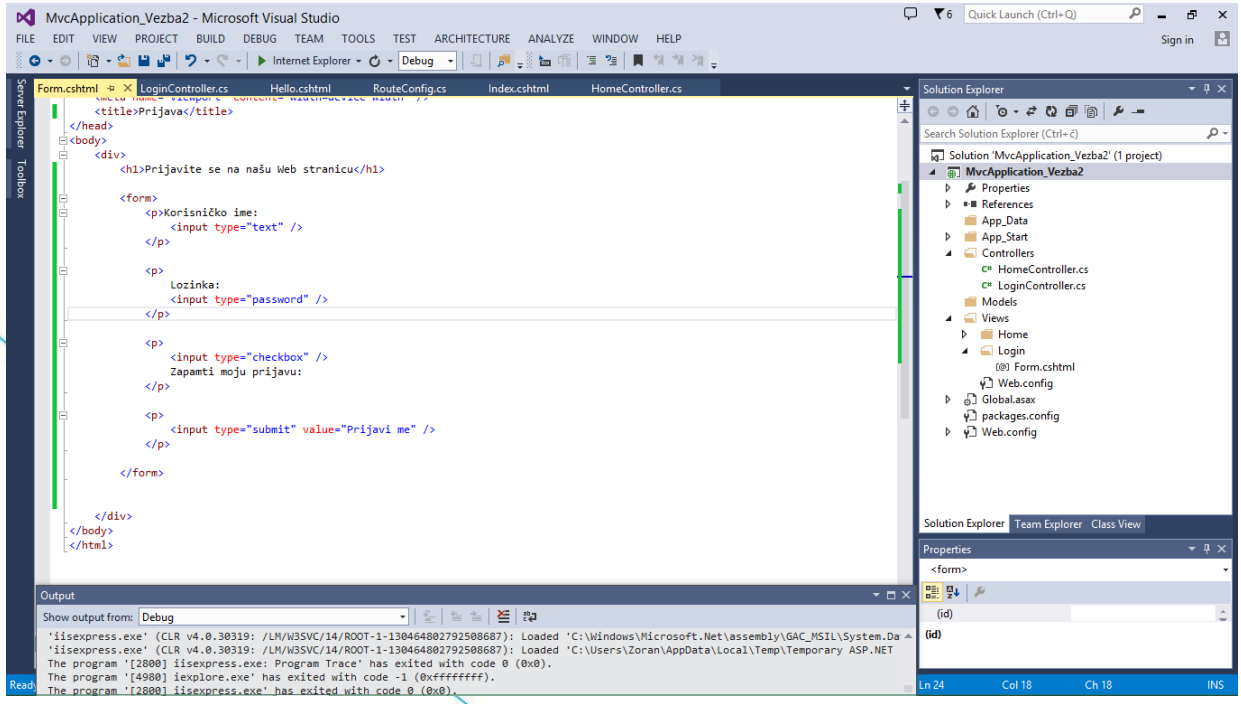

Ispisuje se isto, ali sada podaci dolaze iz kontrolera! U kontroleru se smešta vrednostu pomoću ViewBag.ime_promenljive koja se koristi u View-u. U **ViewBag**-u se mogu stavljati bilo koji tipovi podataka (čak i objekti kao što je **DateTime**).

## SLANJE PODATAKA IZ FORME NA SERVER

Pravimo novi kontriler **Login**, sa Akcionom metodom Form().



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MvcApplication_Vezba2.Controllers
{
    public class LoginController : Controller
    {
        //
        // GET: /Login/

        public ActionResult Form()
        {
            return View();
        }
```

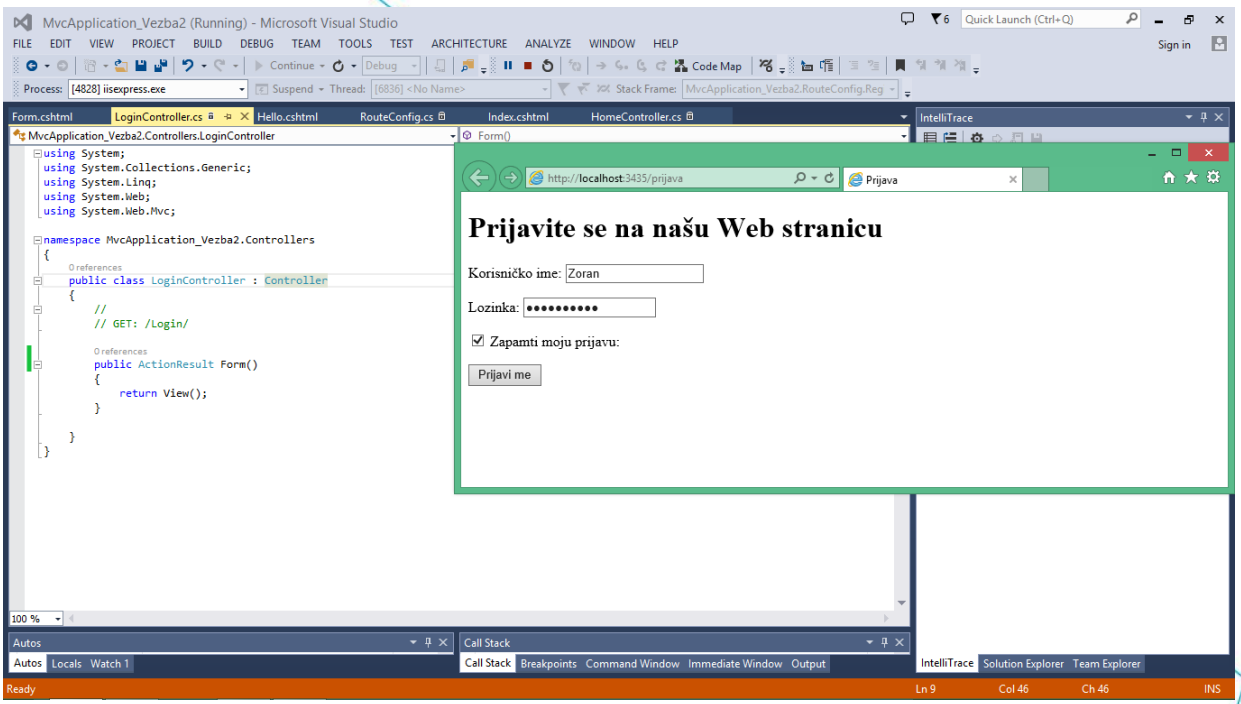


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
    }
}
```

**Dodata ruta:**

```
routes.MapRoute(
    name: "",
    url: "prijava",
    defaults: new { controller = "Login", action = "Form", id = UrlParameter.Optional }
);
```




```
<!DOCTYPE html>

<html>
<head>
  <meta name="viewport" content="width=device-width" />
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

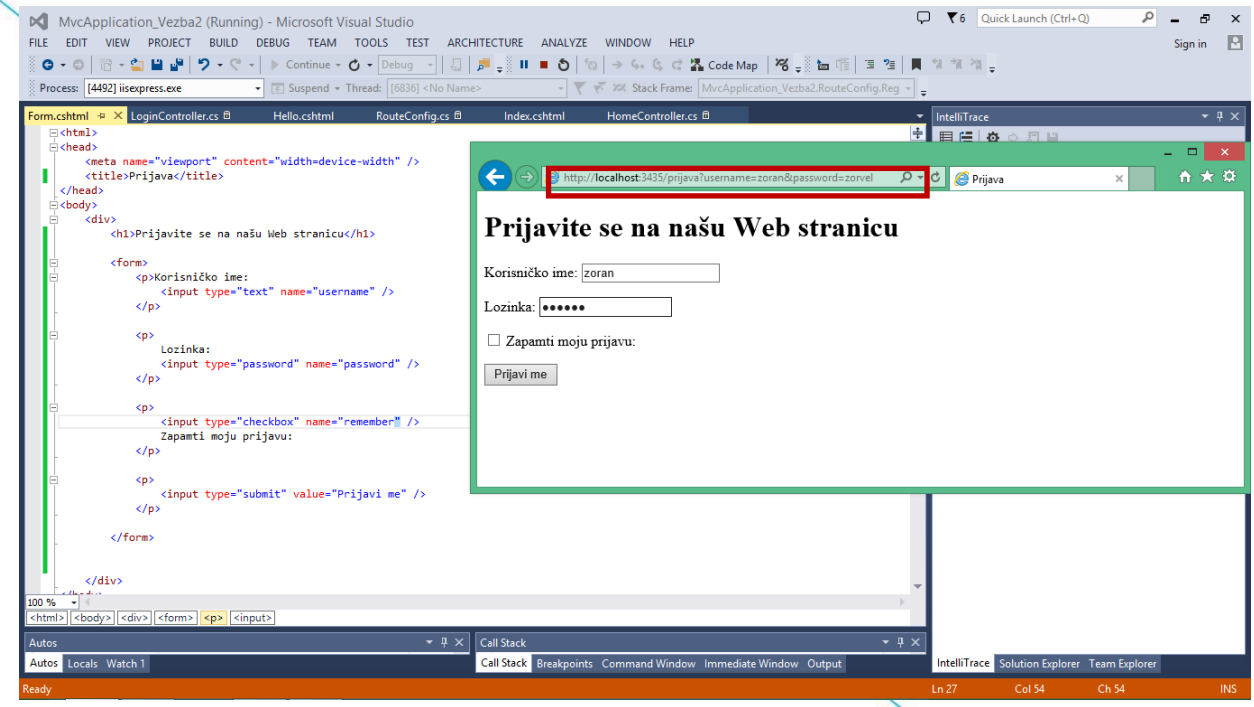
```
  <title>Prijava</title>
</head>
<body>
  <div>
    <h1>Prijavite se na našu Web stranicu</h1>

    <form>
      <p>Korisničko ime:
        <input type="text" />
      </p>

      <p>
        Lozinka:
        <input type="password" />
      </p>

      <p>
        <input type="checkbox" />
        Zapamti moju prijavu:
      </p>

      <p>
        <input type="submit" value="Prijavi me" />
      </p>

    </form>

  </div>
</body>
</html>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

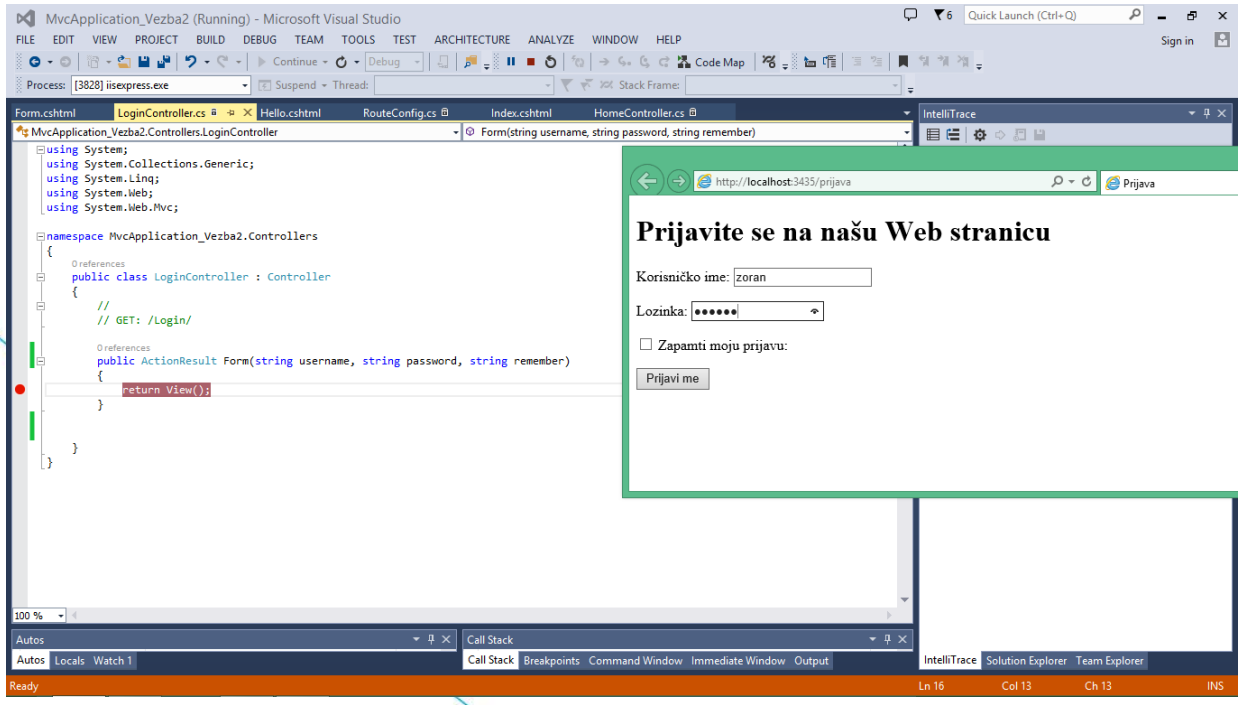
Šta se dešava kada se pritisne dugme? Kako će se to preneti na server? Treba dodati atribut name I on će omogućiti pristup poslatim podacima na serveru.

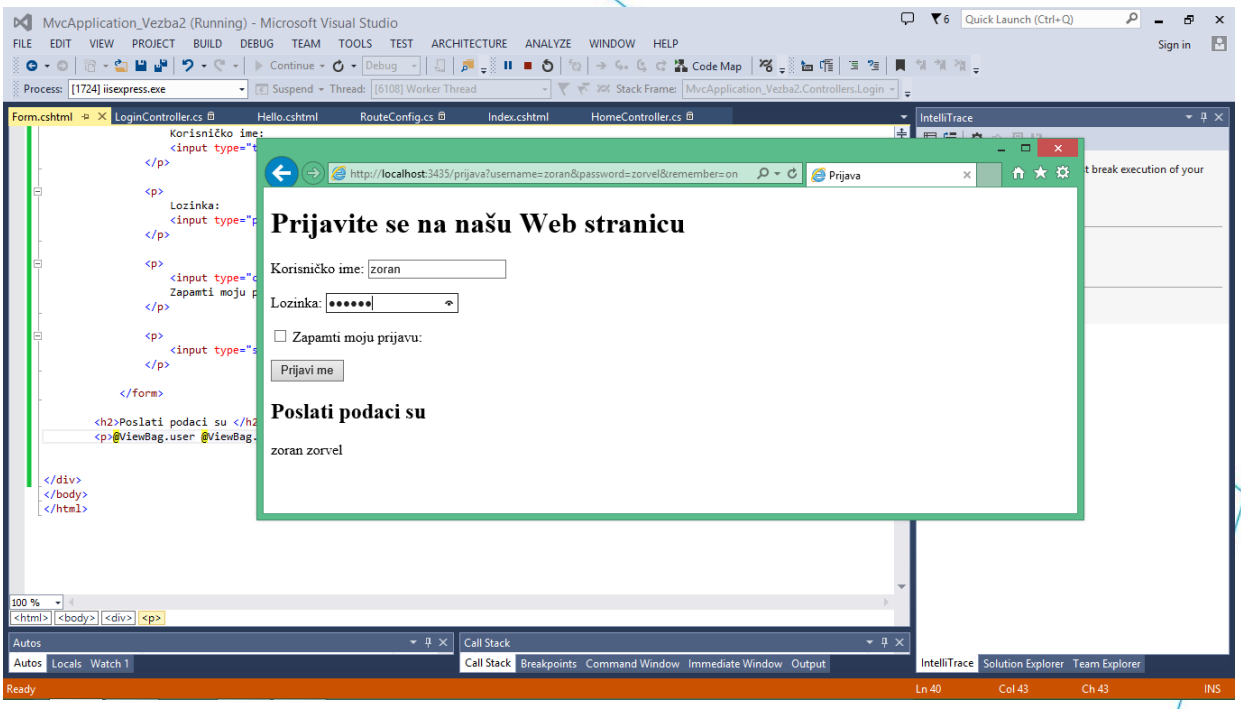
Podaci su se upisali u adresnom polju. Podaci se šalju u istu akciju sa koje ste došli. Zbog toga se ništa ne dešava.






Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES






Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
@{
   Layout = null;
}

<!DOCTYPE html>

<html>
<head>
   <meta name="viewport" content="width=device-width" />
   <title>Prijava</title>
</head>
<body>
   <div>
      <h1>Prijavite se na našu Web stranicu</h1>

<!--      <form method = "post">  za slanje bez vidljivih informacija u adresnom delu -->
         <form>
            <p>
               Korisničko ime:
               <input type="text" name="username" />
            </p>

            <p>
               Lozinka:
               <input type="password" name="password" />
            </p>

            <p>
               <input type="checkbox" name="remember" />
               Zapamti moju prijavu:
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
            </p>

            <p>
              <input type="submit" value="Prijavi me" />
            </p>

          </form>

        <h2>Poslati podaci su </h2>
        <p>@ViewBag.user @ViewBag.password</p>


    </div>
    </body>
    </html>

    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Web;
    using System.Web.Mvc;

    namespace MvcApplication_Vezba2.Controllers
    {
      public class LoginController : Controller
      {
        //
        // GET: /Login/

        public ActionResult Form(string username, string password, string remember)
        {
```

```
            ViewBag.user = username;
            ViewBag.password = password;
            return View();
        }
    }
}
```

Sada provera da li si uspešno prijavljen. Ako si prijavljen idi na neku drugu stranicu.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MvcApplication_Vezba2.Controllers
{
    public class LoginController : Controller
    {
        //
        // GET: /Login/

        public ActionResult Form(string username, string password, string remember)
        {
            //ViewBag.user = username;
            //ViewBag.password = password;
            return View();
        }

        public ActionResult FormCheck(string username, string password)
        {
            ViewBag.user = username;
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
            ViewBag.password = password;
            return View();
        }

    }
}

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Prijava</title>
</head>
<body>
    <div>
        <h1>Prijavite se na našu Web stranicu</h1>
<!--        <form method = "post">  za slanje bez vidljivih informacija u adresnom delu -->
            <form action="/prijava/provera">
                <p>
                    Korisničko ime:
                    <input type="text" name="username" />
                </p>
                <p>
                    Lozinka:
                    <input type="password" name="password" />
                </p>
```



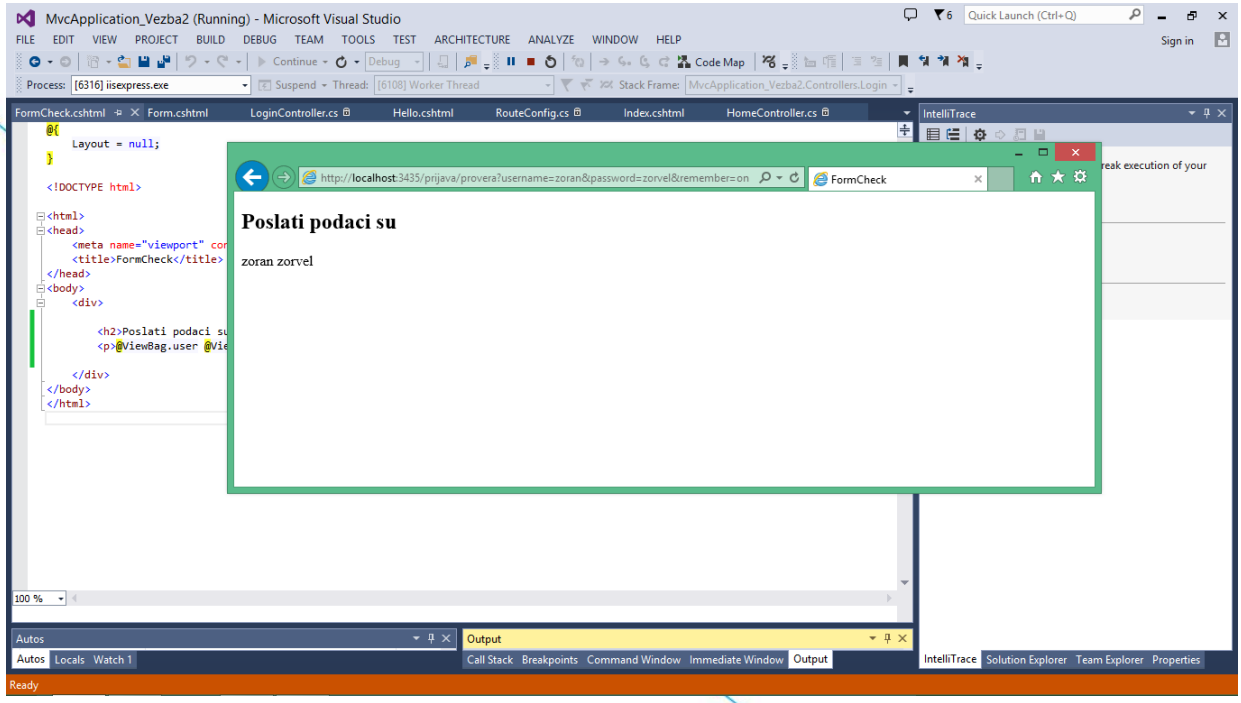
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
            <p>
              <input type="checkbox" name="remember" />
              Zapamti moju prijavu:
            </p>
            <p>
              <input type="submit" value="Prijavi me" />
            </p>
          </form>
        <h2>Poslati podaci su </h2>
        <p>@ViewBag.user @ViewBag.password</p>
</div>
</body>
</html>
```

**FormCheck**

```
@{
  Layout = null;
}

<!DOCTYPE html>

<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>FormCheck</title>
</head>
<body>
  <div>

    <h2>Poslati podaci su </h2>
    <p>@ViewBag.user @ViewBag.password</p>
```

```
    </div>
</body>
</html>
```




Provera da li je korisnik prijavljen. Obično se to radi u bazi, ali je ovde nešto jednostavnije.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MvcApplication_Vezba2.Controllers
{
    public class LoginController : Controller
    {
        //
```



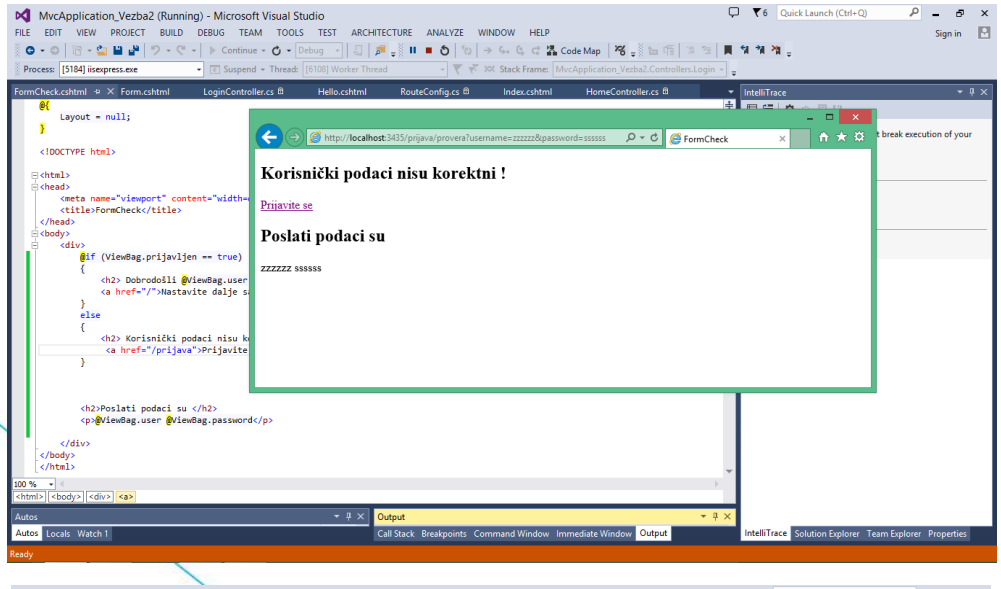
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

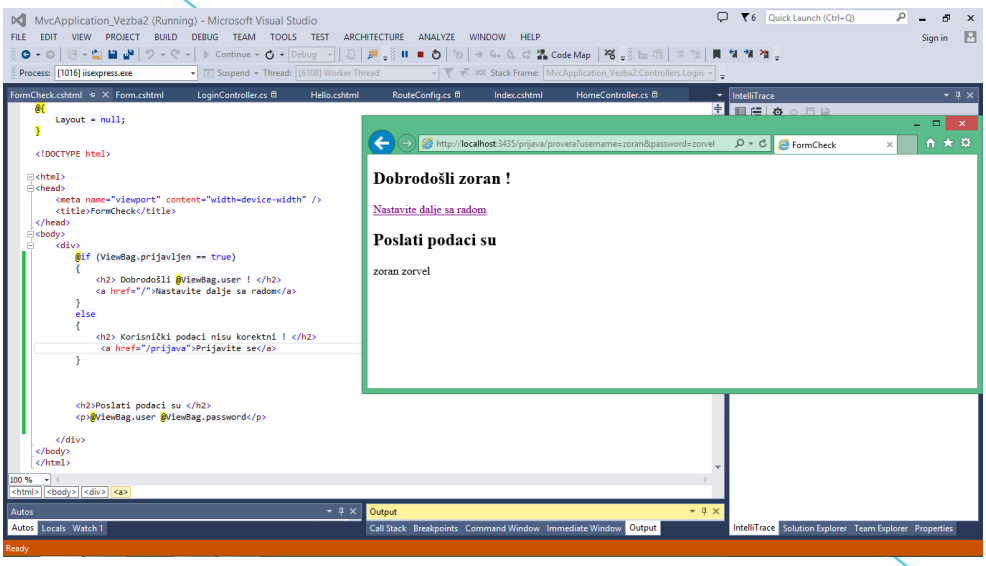
```
        // GET: /Login/

        // public ActionResult Form(string username, string password, string remember)
        public ActionResult Form()
        {
            //ViewBag.user = username;
            //ViewBag.password = password;
            return View();
        }
        public ActionResult FormCheck(string username, string password)
        {
            if (username == "zoran" && password == "zorvel")
            {
                ViewBag.prijavljen = true;
            }
            else
            {
                ViewBag.prijavljen = false;
            }
            ViewBag.user = username;
            ViewBag.password = password;
            return View();
        }

    }
}

@{
    Layout = null;
}
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

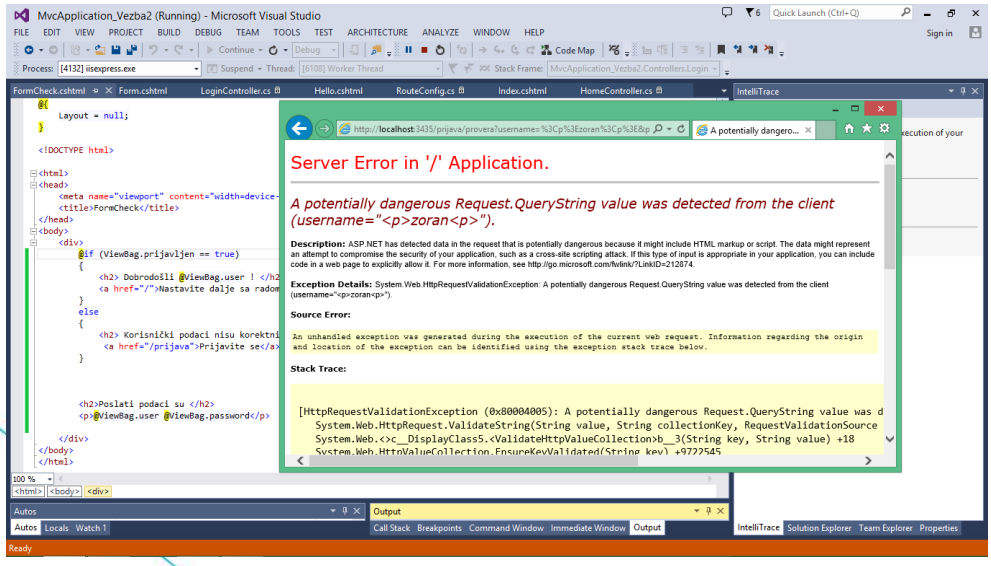
```
<!DOCTYPE html>

<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>FormCheck</title>
</head>
<body>
  <div>
    @if (ViewBag.prijavljen == true)
    {
      <h2> Dobrodošli @ViewBag.user ! </h2>
      <a href="/">Nastavite dalje sa radom</a>
    }
    else
    {
      <h2> Korisnički podaci nisu korektni ! </h2>
       <a href="/prijava">Prijavite se</a>
    }
    <h2>Poslati podaci su </h2>
    <p>@ViewBag.user @ViewBag.password</p>

  </div>
</body>
</html>
```






Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES



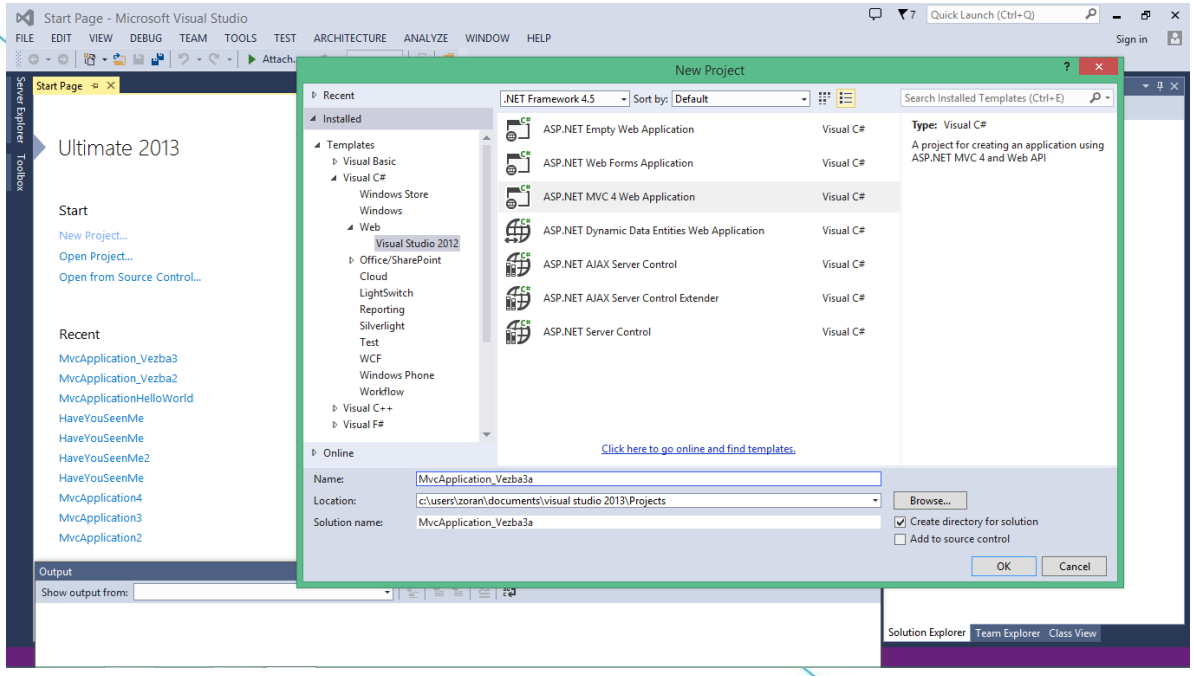



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

**Prilog 3: Vežba br. 3**

MODELI u MVC-u

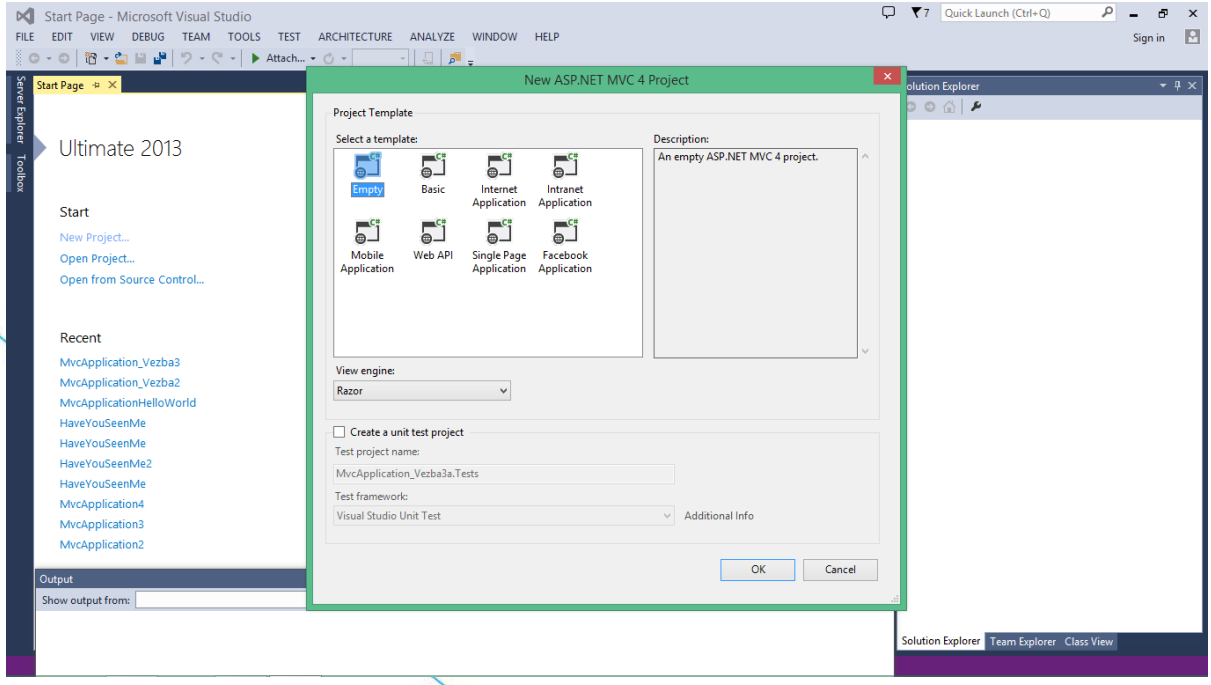
Prvo se formira model podataka, a zatim se taj model koristi u Controller-u. U kontroleru se formira objekt na osnovu klase napravljene u Model-u. Tako formirani objekat se šalje na View-u radi prikaza. Pokazaćemo kako se može pratiti Debag-u dodeljivanje vrednosti promenljivama I objektima. Formiranje PROJEKTA, MvcApplication_Vezba3a



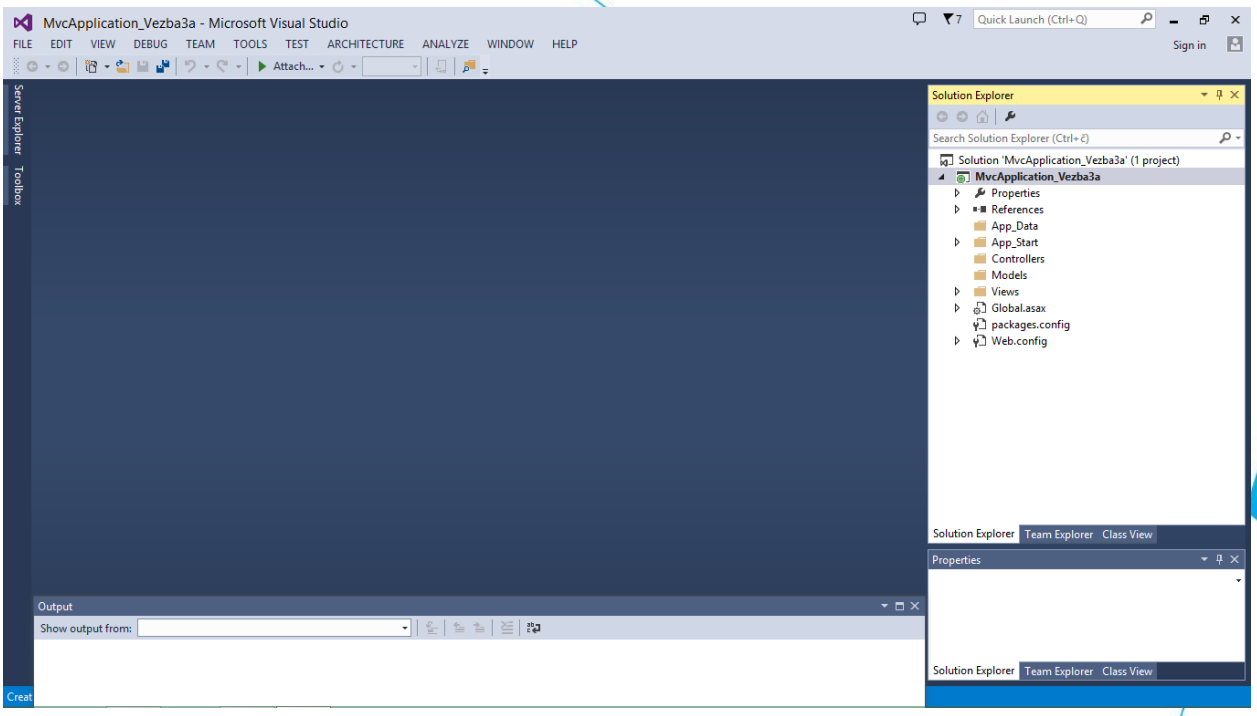

Izbor tipa Web aplikacije



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

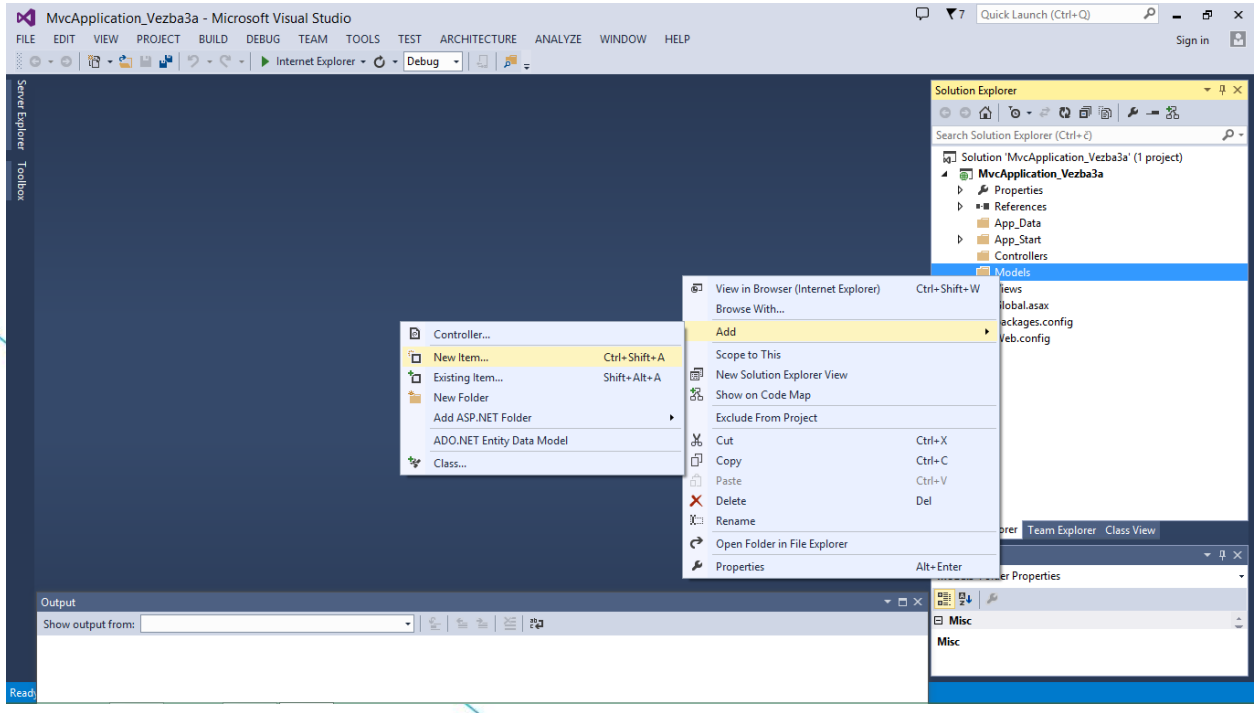



Empty model Web applikacije





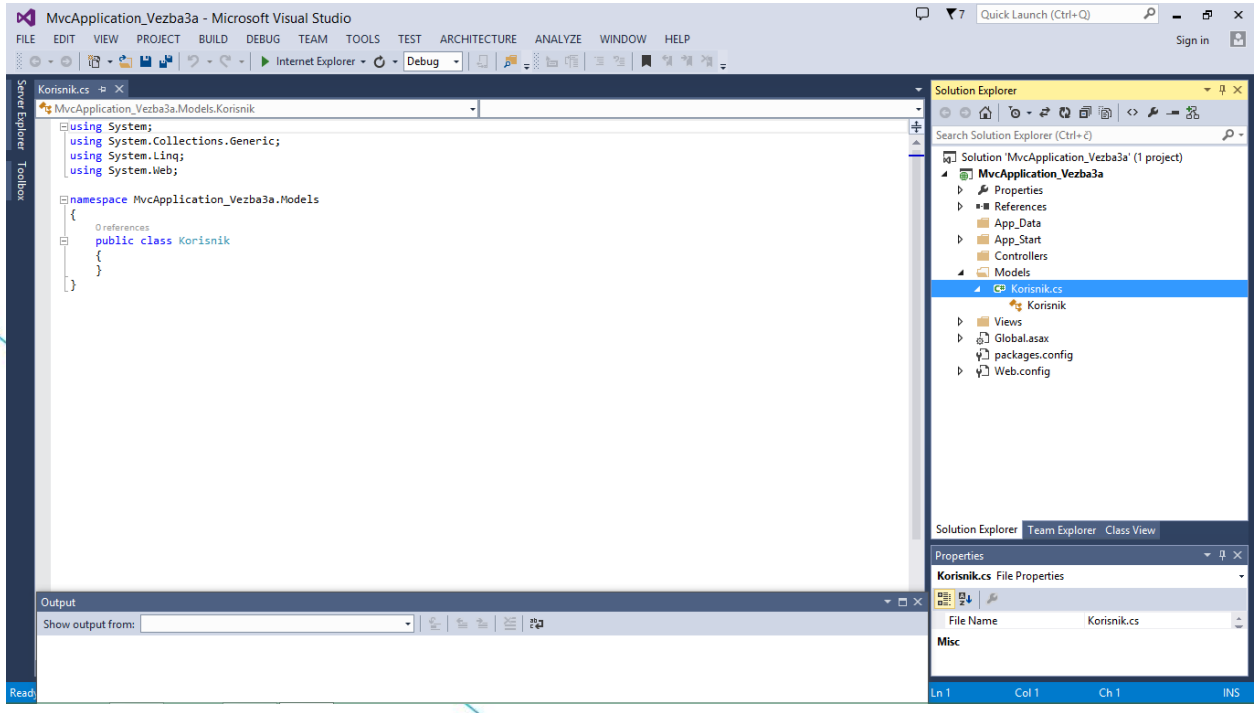

Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




Formiranje MODEL-a.



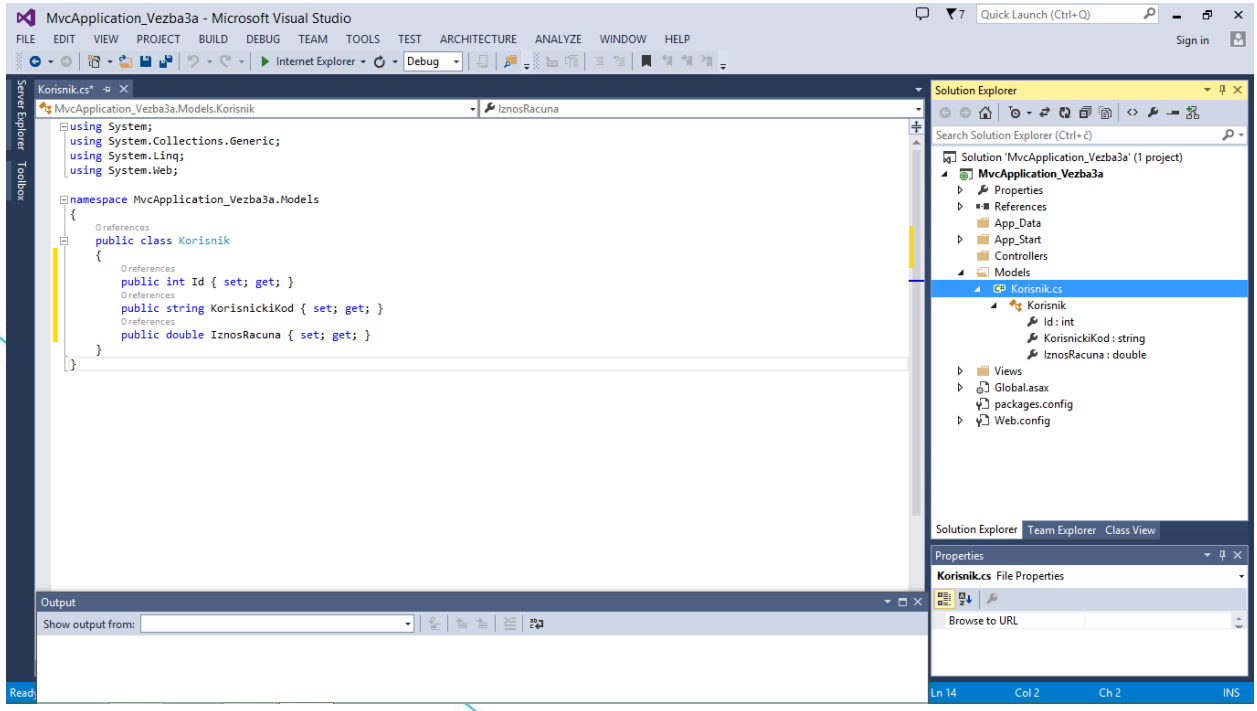
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




Klasa Korisnik.cs. poseduje samo tri svojstva (definisana sa set I get naredbama) a to su Id, KorisnickiKod i IznosRacuna.



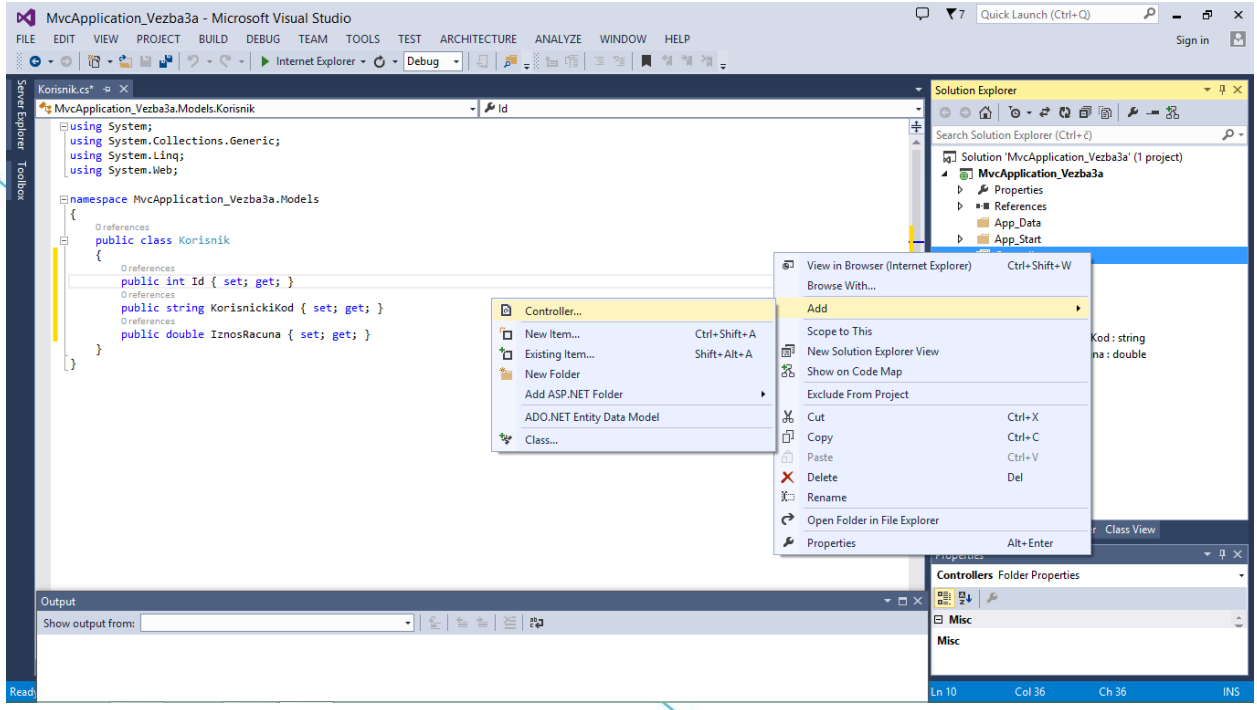
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




MODEL, Customer.cs

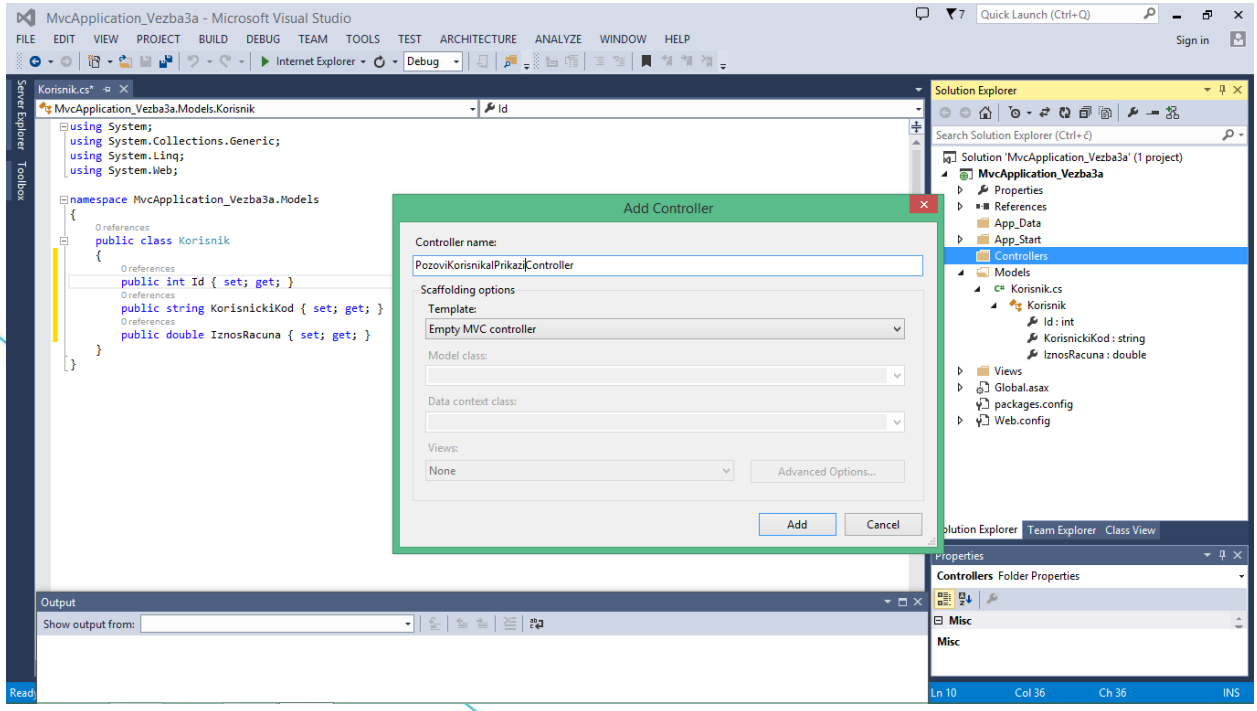
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace MvcApplication_Vezba3a.Models
{
   public class Korisnik
   {
      public int Id { set; get; }
      public string KorisnickiKod { set; get; }
      public double IznosRacuna { set; get; }
   }
}
```

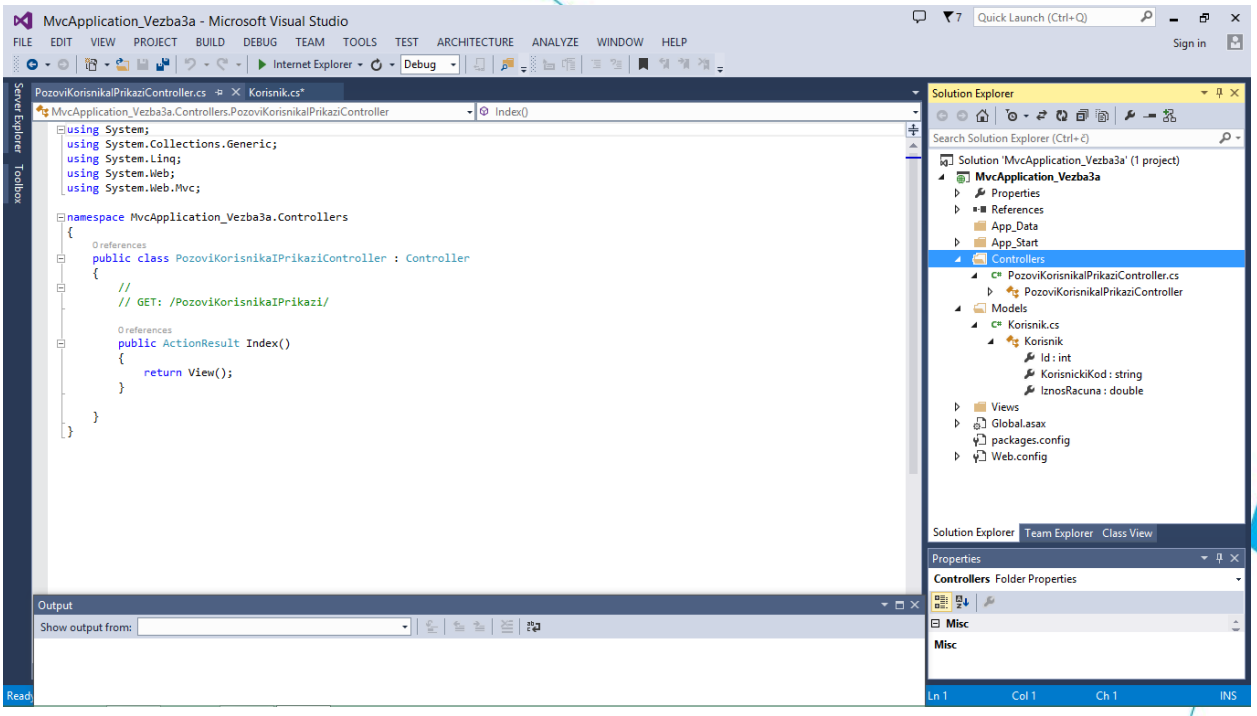
Sada treba napraviti Controller koji će napraviti objekat tipa Korisnik I dodeliti mu početne vrednosti. Kontriler će se zvati PozoviKorisnikaIPrikazi.






Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

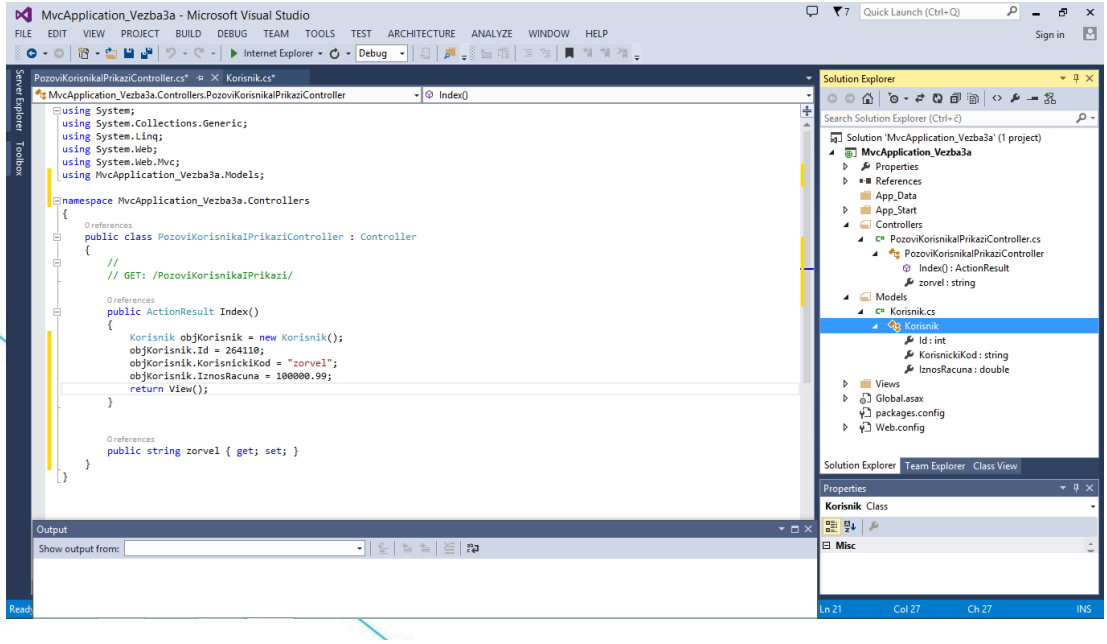



Dodaj naredbom using formirani model (using MvcApplication_Vezba3a.Models;).



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

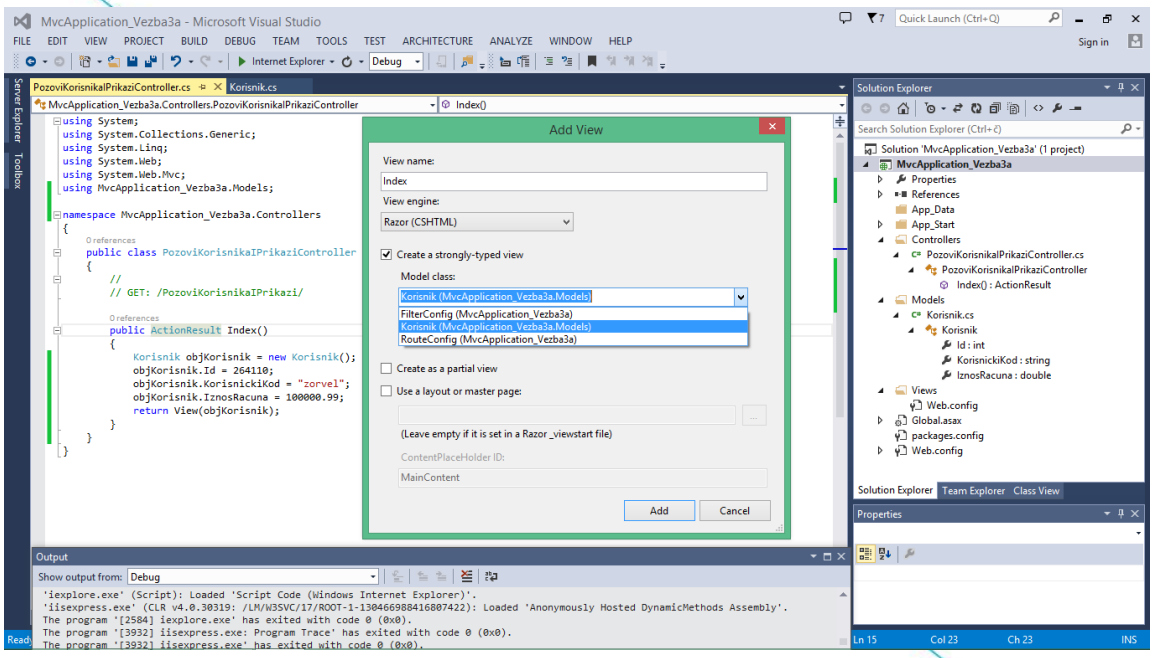



CONTROLLER, LoadCustomerAndDisplay

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using MvcApplication_Vezba3a.Models;

namespace MvcApplication_Vezba3a.Controllers
{
  public class PozoviKorisnikaIPrikaziController : Controller
  {
    //
    // GET: /PozoviKorisnikaIPrikazi/
    public ActionResult Index()
    {
      Korisnik objKorisnik = new Korisnik();
      objKorisnik.Id = 264110;
      objKorisnik.KorisnickiKod = "zorvel";
```

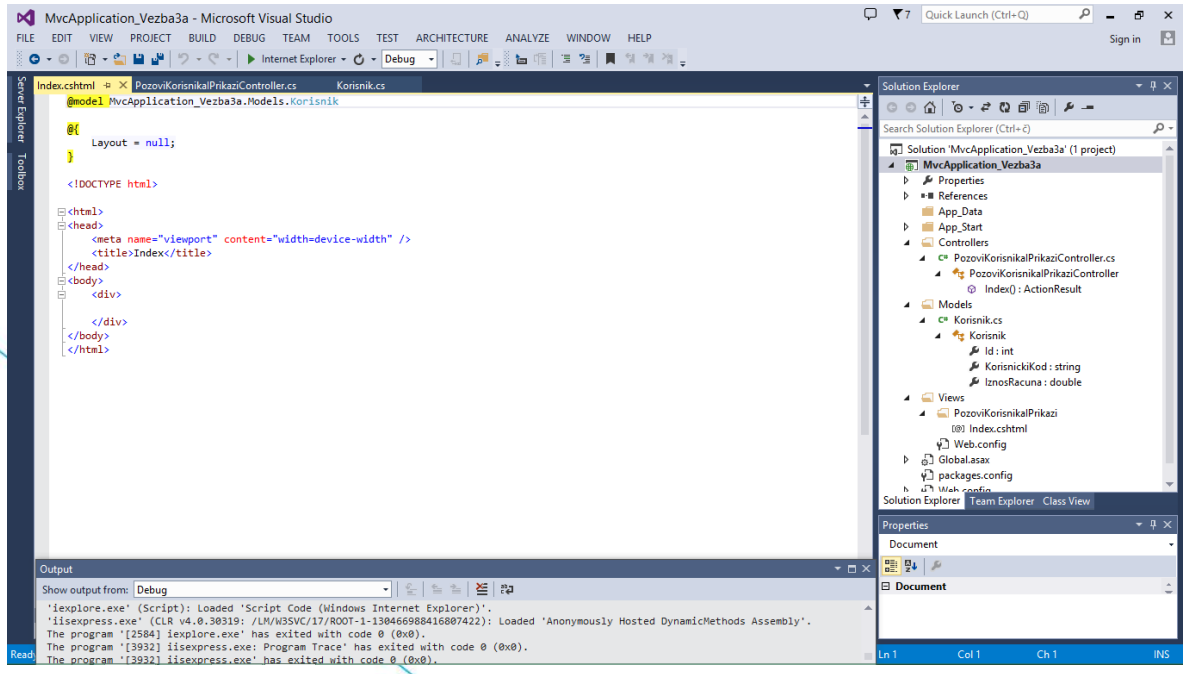
```
            objKorisnik.IznosRacuna = 100000.99;
            return View();
        }
        public string zorvel { get; set; }
    }
}
```

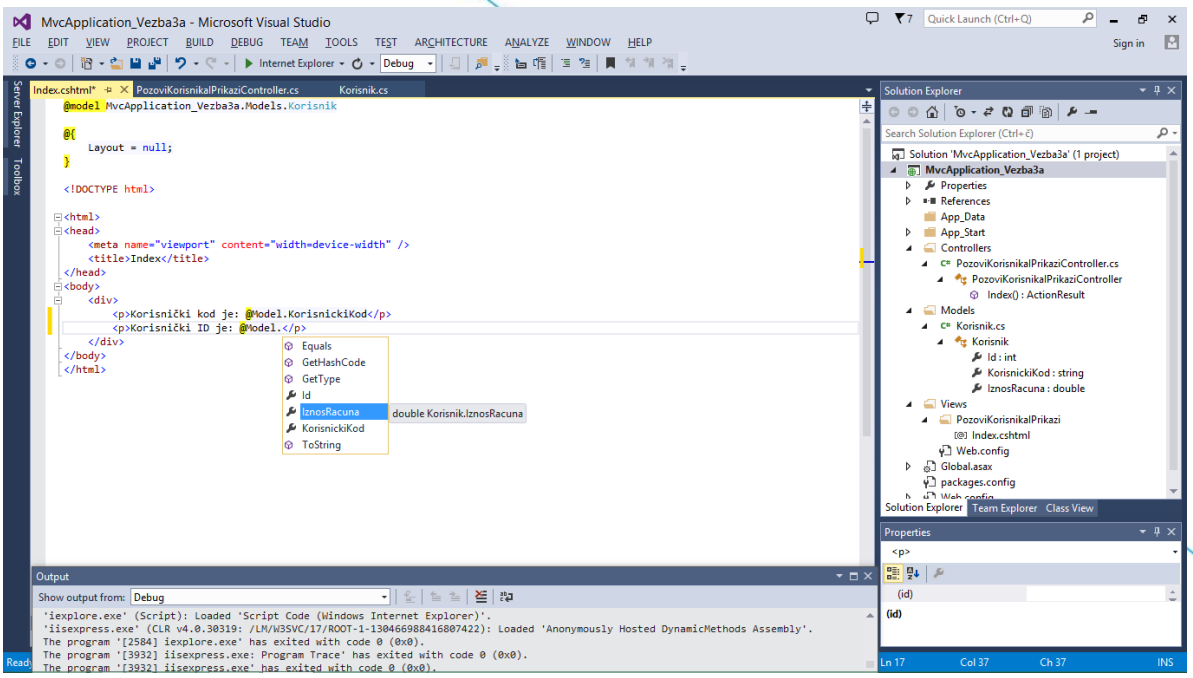
Sada treba dodati View za ovaj Controller. Prethodno promeniti generalnu povratnu vrednost sa return View(); na odgovarajući objekt return View(objKorisnik);. Sada se View-u prosleđuje odgovarajući objekt objKorisnik tipa Korisnik.






Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES



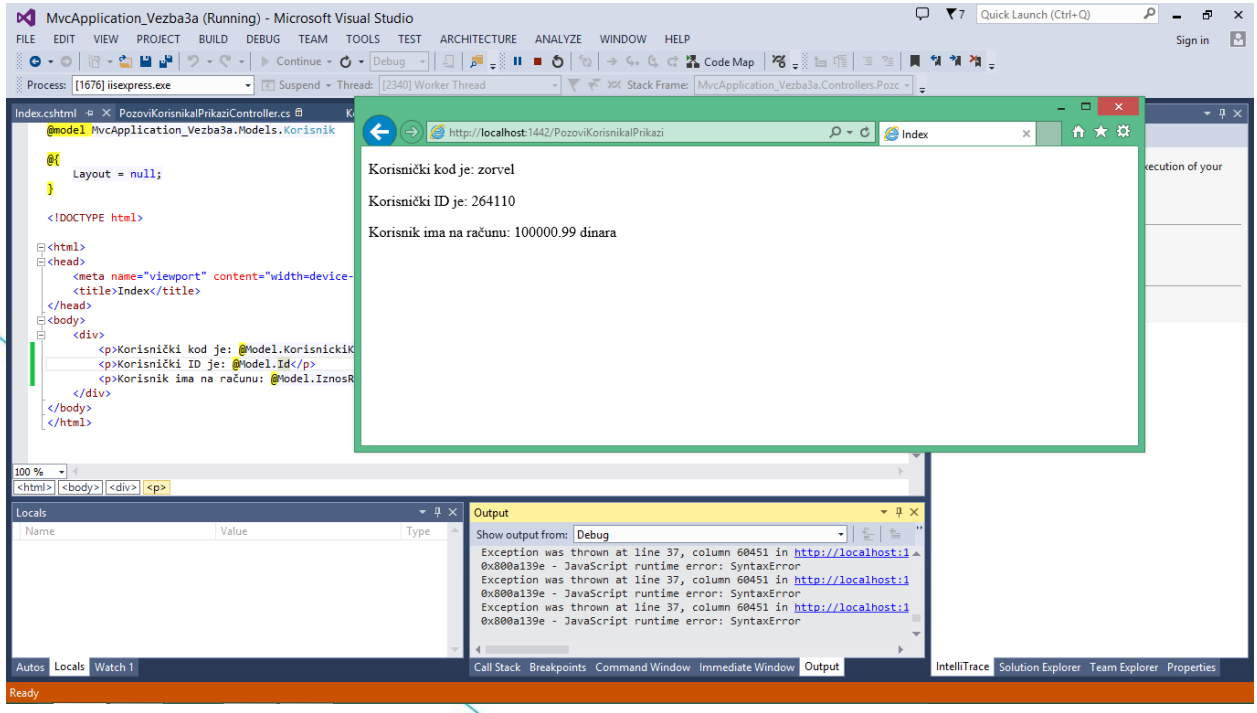




```
@model MvcApplication_Vezba3a.Models.Korisnik

@{
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

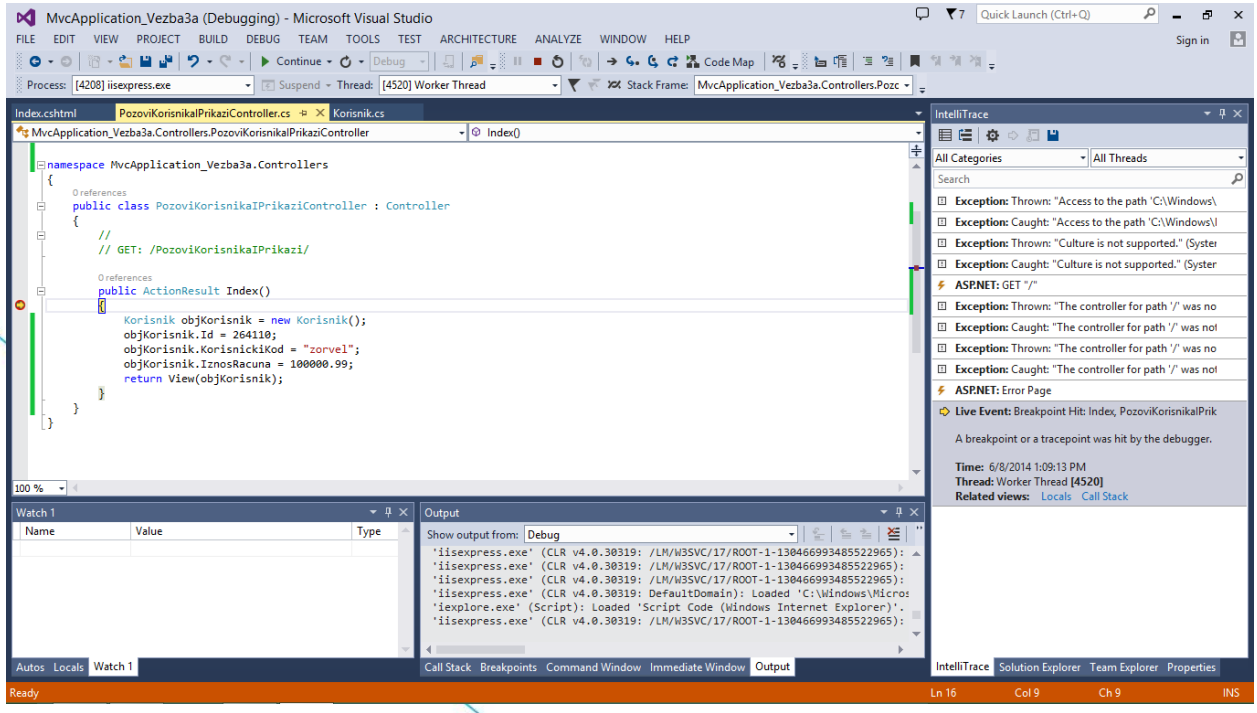
```
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
        <p>Korisnički kod je: @Model.KorisnickiKod</p>
        <p>Korisnički ID je: @Model.IznosRacuna</p>
        <p>Korisnik ima na računu: @Model.IznosRacuna dinara</p>
    </div>
</body>
</html>
```



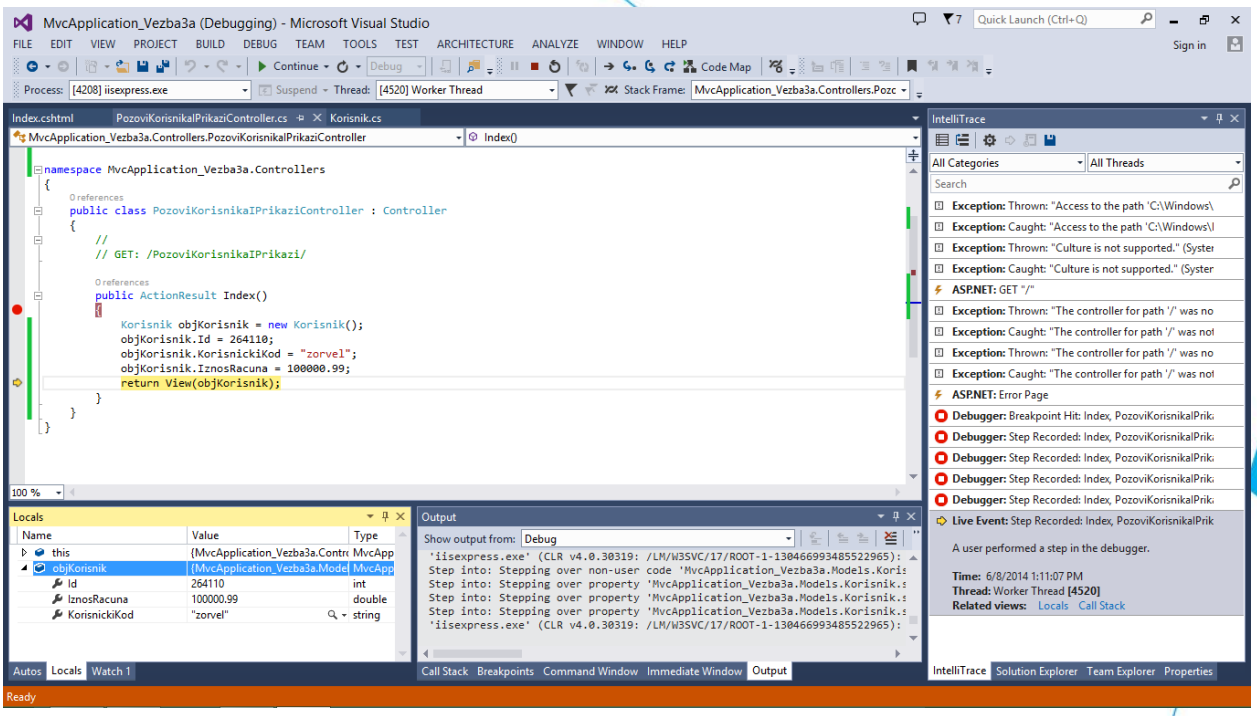

Debaging Korak po Korak.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

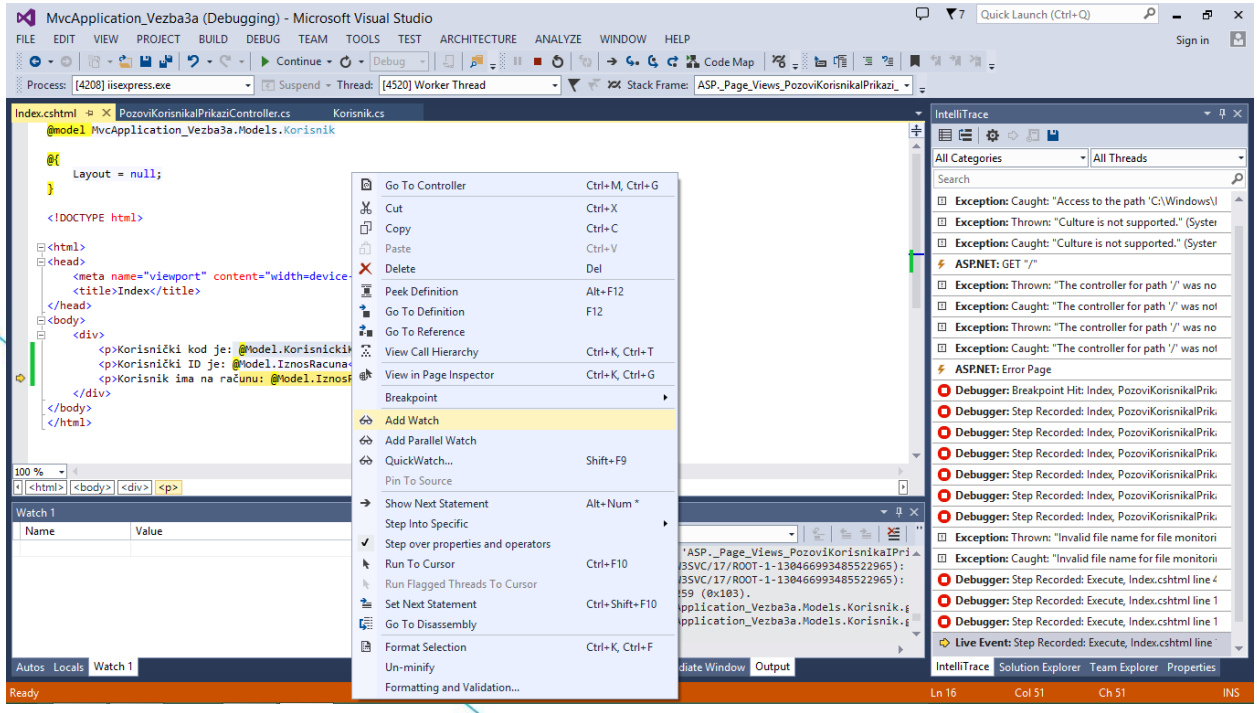



Prikazati u dibageru kroz Watch (locals) željenu promenljivu. Sa F11 se prelazi na sledeći korak.

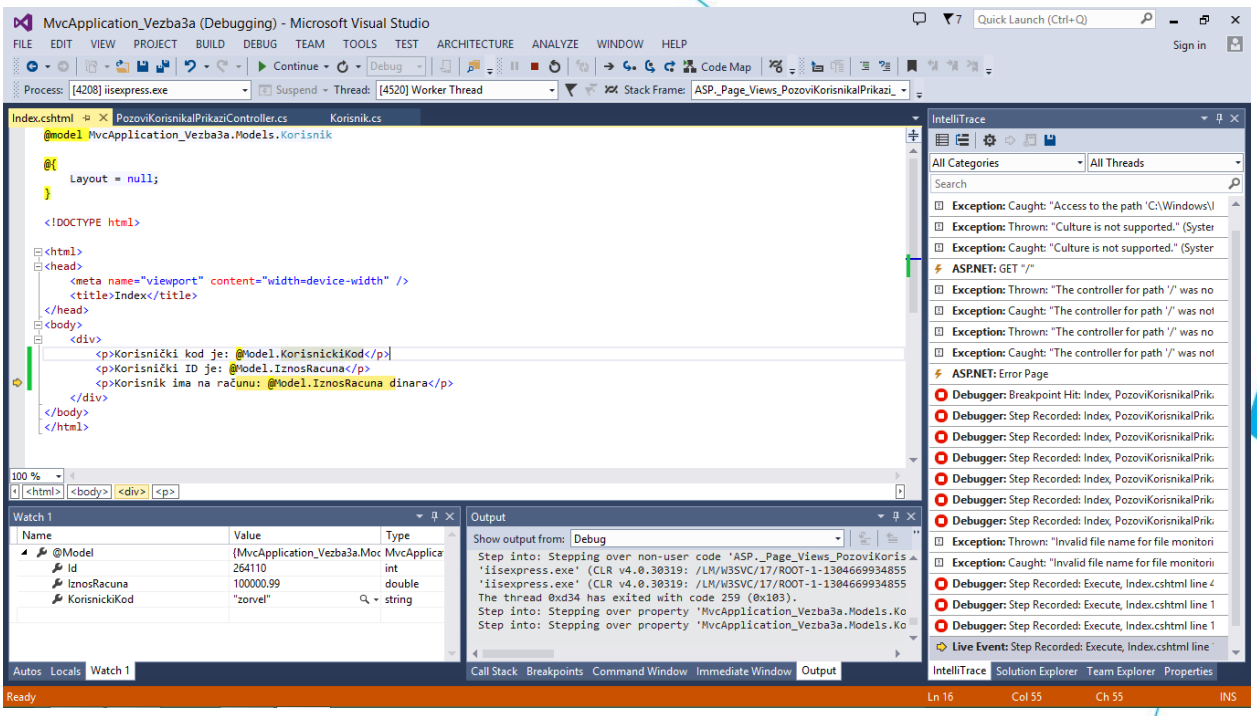





Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES



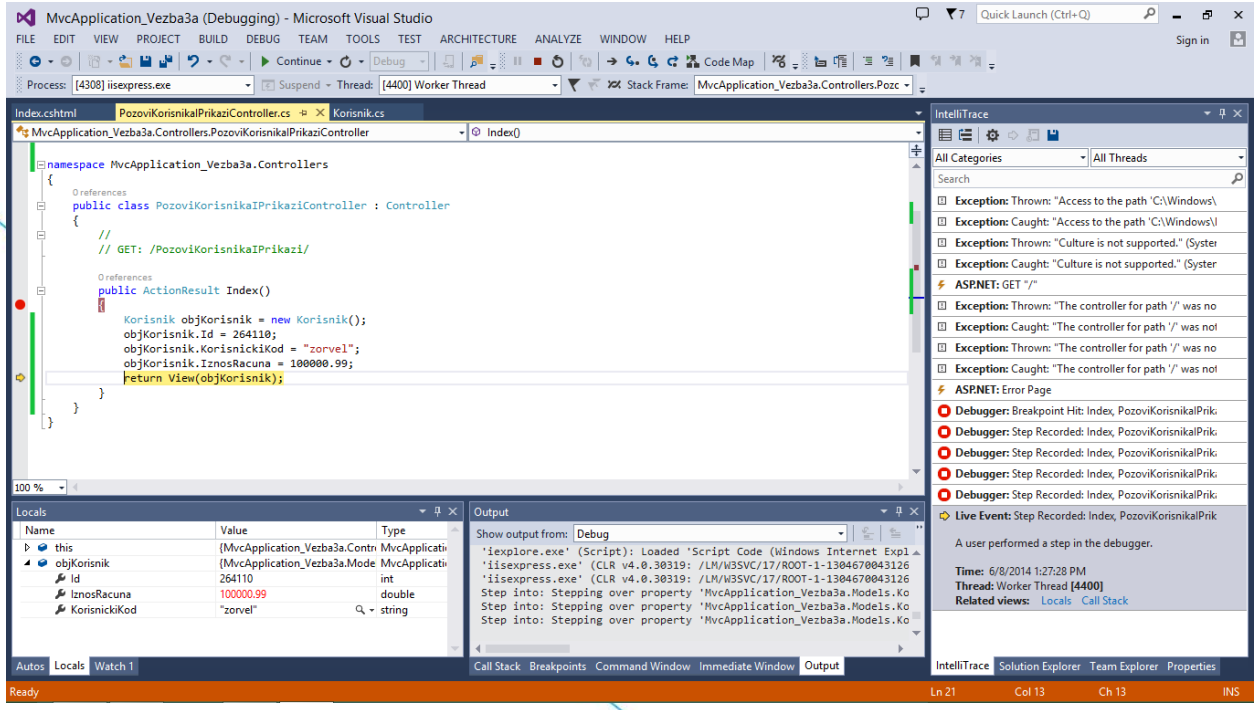

Prikaz celog objekta, samo se napiše Model.

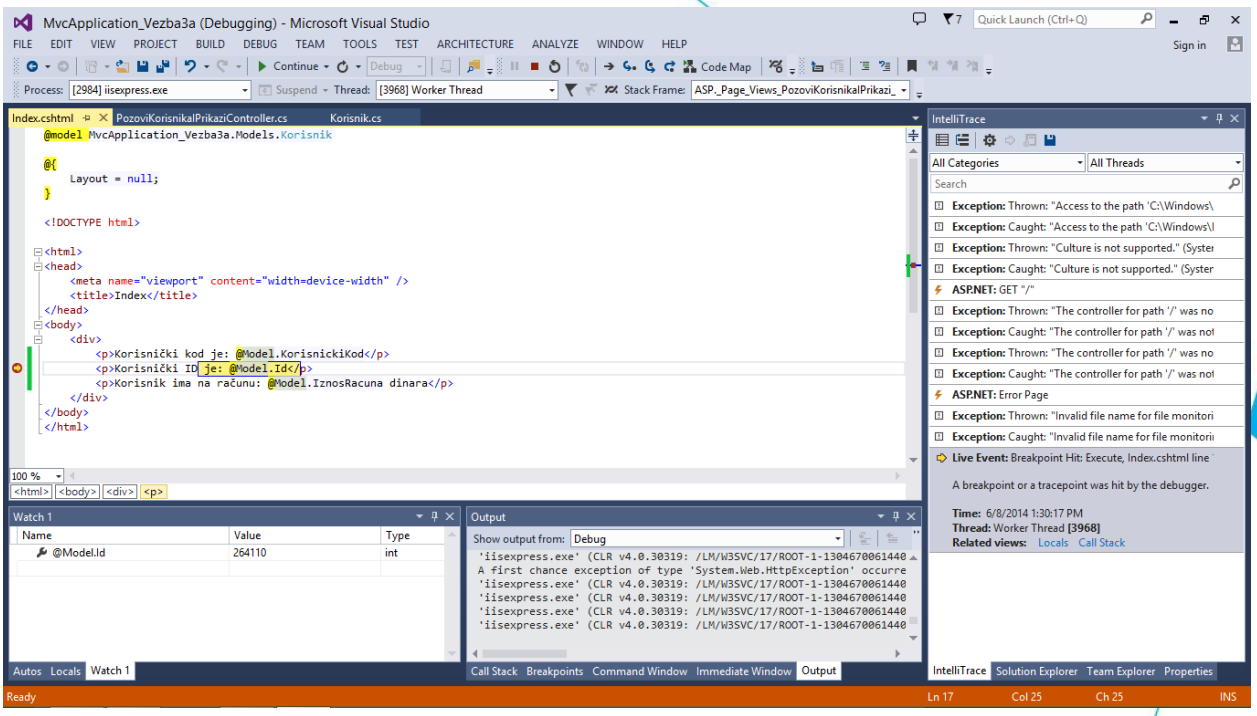





Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Prikaz samo jednog dela objekta. Može se ući I prepraviti tekst u Model!






Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# Razvoj mobilnih aplikacija

**Predavači: Dragan Stojanović, Bratislav Predić**

**Ciljevi predmeta:**
Ovaj kurs nudi učesnicima uvod u metodologiju razvoja mobilnih aplikacija i pregled osnovnih koncepata mobilnog računarstva kao i tehničke informacije o različitim aspektima mobilnih aplikacija.

**Ishod predmeta:**
Učesnici će steći znanje o arhitekturi mobilnih aplikacija, procesu razvoja softvera za mobilne platforme, uvid u karakteristike mobilnih operativnih sistema kao i znanje potrebno za razvoj osnovnih mobilnih aplikacija.

**Teorijska nastava:**
1. Uvod u razvoj mobilnih aplikacija, mobilni operativni sistemi i platforme
2. Android mobilna platforma, Java programski jezik za Android platformu i Eclipse ADT IDE
3. Arhitektura i životni ciklus Android mobilnih aplikacija
4. Dizajn i implementacija mobilnih grafičkih korisničkih interfejsa i principi interakcija sa mobilnim aplikacijama
5. Perzistencija podataka i mobilne baze podataka
6. Lociranje mobilnih korisnika u prostoru, korišćenje mapa i lokaciono zasnovani servisi (LBS)
7. Mrežna komunikacija u mobilnim aplikacijama i korišćenje tipičnih mrežnih servisa
8. Razmena poruka (SMS i MMS) u mobilnim aplikacijama i notifikacije korisnika

**Praktične vežbe:** 20% do 50% ukupnog broja sati. Rad sa Eclipse ADT razvojnim okruženjem, demonstracija životnog ciklusa Android aplikacije, dizajn i implementacija savremenih mobilnih grafičkih korisničkih interfejsa (GUI): layout, widget itd. Rad sa lokalnim podacima na mobilnom uređaju, SQLite baza podataka, različite metode lociranja mobilnog korisnika u prostoru, korišćenje Google Maps API v2, HTTP mrežna komunikacija, broadcast receiver-i i notifikacije.

**Sadržaj kursa:** Predavači će upotpuniti kurs, svojim nastavnim planom.

**Trajanje kursa:** Kurs traje 30 sati (24 za časove i 6 za testove). Predavanja će se održavati po blok principu tokom jednog meseca (popodne petkom i prepodne subotom).

**Kurikulum:**

Kurs je podeljen na 7 tema:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

1. Uvod u razvoj mobilnih aplikacija i Android platformu obuhvata pregled osnova Android operativnog sistema, upoznavanje sa Eclipse ADT razvojnim okruženjem, kao i alatom ADB - Android Debug Bridge. Polaznici će se upoznati sa dostupnim Android emulatorima (Google i Genymotion), kao i konceptom virtuelnih mašina
2. Životni ciklus Android aplikacija i implementacija korisničkih interfejsa daje pregled arhitekture Android aplikacija, analizira komponente Android aplikacija (Intent, Activity, Service, BroadcastReceiver, ContentProvider itd.). Polaznici će se upoznati sa svrhom i sadržajem manifest datoteke Android aplikacije. U ovoj temi se uvodi i koncept layout-a pri izgradnji grafičkog korisničkog interfejsa Android aplikacije. U praktičnom delu polaznici će implementirati bazičnu Android aplikaciju, implementirati metode životnog ciklusa aplikacije i upoznati se sa korišćenjem ListActivity komponente i interakcijom sa GUI komponentama korišćenjem listener komponenti.
3. Dizajn mobilnih grafičkih korisničkih interfejsa i korišćenje GUI widget-a upoznaje polaznike sa konceptima efikasnim mobilnih grafičkih korisničkih interfejsa, tipovima i načinima korišćenja menija, kao i različitih tipova dialog box-ova (Toast i AlertDialog). U ovoj temi se obrađuje i prilagođavanje dijaloga aplikacije dodavanjem dugmića, preuzimanje podataka iz dijaloga. Obrađuju se i Intent-i kao mehanizam koji povezuje različite komponente Android aplikacije kroz: prosleđivanje podataka kroz Intent korišćenjem Bundle i Parcelable klasa, preuzimanje podataka iz Intent-a. U praktičnoj vežbi će polaznici implementirati aplikaciju za rad sa objektima od interesa (POI - Point of Interest) i isprobati Model-View koncept GUI widget-a kroz korišćenje ArrayAdapter klase.
4. Perzistencija podataka korišćenjem SQLite baze podataka obrađuje temu trajnog snimanja podataka na mobilnim uređajima i detaljno analizira SQLite bazu podataka korišćenu na Android platformi kroz analizu binarne kompatibilnosti, specifičnostima u odnosu na druge relacione baze podataka. Polaznici će se upoznati sa tipovima upita: Action i Retrieval, rad sa kursorima. U ovoj temi se demonstriraju i mogućnosti deljenja baze podataka između više Android aplikacija. Polaznici se upoznaju i sa konceptima višenitnog programiranja uključujući i specifične Android klase: Handler, Message, AsyncTask. U praktičnoj vežbi polaznici dodaju u mobilnu aplikaciju sa prethodne teme mogućnost trajnog snimanja podataka o objektima od interesa (POI) korišćenjem SQLite baze podataka.
5. Google Maps API v2 tema obrađuje različite metode lociranja korisnika u prostoru koji su dostupni na Android platformi, korišćenje servisa geokodiranja i reverznog geokodiranja kao i proceduru korišćenja Google Map-a u Android aplikacijama. Tema se završava demonstracijom metoda za dodavanje oznaka na mapu integrisanu u mobilnu aplikaciju. U praktičnoj vežbi polaznici će dodati mapu u implementiranu aplikaciju i na toj mapi prikazati lokacije prethodno unetih POI.

6. Mrezna komunikacija korišćenjem HTTP protokola tema daje pregled specifičnosti HTTP mrežne komunikacije na Android platformi. To se demonstrira na primeru razmene XML kodiranih podataka. Polaznici se takođe upoznaju sa Android servisima kao bitnom komponentom Android aplikacija: kako se servisi startuju, servisi u zasebnom i istom procesu kao i aplikacija itd. U okviru praktične vežbe polaznici će modifikovati implementiranu aplikaciju tako da unete podatke o POI može da snimi na serveru i da preuzme podatke sa servera. Kao server će biti korišćena jednostavna C# aplikacija.
7. BroadcastReceiver i ProximityAlert prezentuje polaznicima mogućnost Android aplikacija da međusobno komuniciraju korišćenjem SMS i MMS poruka. Tema demonstrira i koncept broadcast-a u Android operativnom sistemu kao i načine slanja i prijema broadcast-a korišćenjem BroadcastReceiver klase. Ovaj koncept će biti demonstriran prijemom obaveštenja o dolaznoj SMS poruci. Tema se završava osnovnim informacijma o preuzimanju podataka sa različitih tipova senzora koje Android platforma može da ima integrisane (akceleracioni i magnetni senzor, senzori vazdušnog pritiska, vlažnosti i temperature itd.). u praktičnom delu polaznici će u implementiranu aplikaciju dodati ProximityAlert komponenta kako bi korisnik bio obavešten kada se nađe u blizini nekog registrovanog POI-a. Takođe će isprobati kreiranje BroadcastReceiver komponente koja će po prijemu notifikacije startovati aplikaciju.

**Ocenjivanje:**

1. Tokom trajanja kursa polaznici će polagati dva parcijalna testa sa ABCD pitanjima i ponuđenim odgovorima iz do tada pređenog gradiva.
2. Tokom trajanja kursa polaznici na praktičnim vežbama i u samostalnom radu implementiraju mobilnu aplikaciju koja omogućava unos podataka o objektima od interesa (POI): naziv, opis, slika, komentari itd. Unete POI je moguće locirati na mapi koja je sastavni deo aplikacije i koja prikazuje lokacije svih unetih POI. Uneti POI se trajno snimaju na mobilnom uređaju korišćenjem SQLite baze podataka. U cilju razmene unetih POi između različitih korisnika polaznici će implementirati i HTTP XML komunikaciju sa serverom i razmenu podataka o POI. U samostalnom radu polaznici će prilagoditi ovako implementiranu aplikaciju nekom domenu (bioskopi, restorani, kafići, prodavnice i sl.). U okviru finalnog testa polaznici demonstriraju mobilnu aplikaciju koju su implementirali i odgovaraju na postavljena pitanja (eventualno finalni test čine ABCD pitanja). Polaznici će položiti ovaj finalni test ukoliko osvoje 50% ili više bodova.

**Testovi:** Testovi neće biti objavljivani iz razloga poverljivosti

# 1. Uvod u razvoj Android mobilnih aplikacija

## 1.1 Mobilne tehnologije i standardi

U današnje vreme, na tržištu postoje mobilni uređaji: pametni telefoni, tableti, pametni satovi, sve naprednijih računarsko/komunikacionih karakteristika i funkcija, tako da imaju veću procesorsku snagu nego PC pre 5-10 godina i više RAM-a. Sve je više senzora integrisanih u mobilne uređaje: GPS, kamere, mikrofon, akcelerometar, proximity, kompas, žiroskop, itd. Značajan je i razvoj bežičnih komunikacione mreža koje omogućavaju mobilni pristup podacima i servisima:

- WWAN (GSM, GPRS, EDGE, UMTS, LTE, ...),
- WLAN (WiFi, WiMAX, ZigBee) i
- WPAN (Bluetooth, RFID)

Uporedo s tim razvija se softverska infrastruktura, platforme i okruženja za razvoj mobilnih aplikacija i servisa, poput Android SDK, iOS SDK, Windows Phone SDK, J2ME, HTML5, itd., kao i standardi, HTML5, SyncML, Bluetooth LE, itd.

Kratka istorija razvoja mobilni uređaja, platformi aplikacija:

- 2007: Windows Mobile 6.0
- 2008: iPhone, Google Android
- 2009: Nokia Symbian Foundation, Windows Mobile 6.5
- 2010: iPad - Tablet računari, Windows Phone 7
- 2011: iPad2, iPhone 4S, Android 3.x, 4.0, Nokia – WP7
- 2012: Android 4.2 Jelly Bean, Windows Phone 8, iOS6, Samsung Galaxy S3, iPhone 5, Nokia Lumia, HTC WP 8X, Nexus 4/7/10
- 2013: Android 4.4 Kit Kat, Samsung Galaxy S4, iPhone 5S/C – iOS 7, Google Glass, Windows Phone 8 uspon, BlackBerry 10, Firefox OS, 1080p na 5" ekranima

Noviteti u domenu mobilnih uređaja, platform i aplikacija u 2014 su:

- Samsung Galaxy S5, HTC One 2, Sony Xperia Z2, ...
- iPhone 6, iPhone phablet (iOS 8)
- Windows Phone 8.1, Tizen OS
- 4-jezgarni 64-bitni procesori (8 core)
- 2-3GB RAM/64 GB memorije
- 2Kp na 5“+ ekranima (2560x1440 piksela)
- Smart Watches: Google Watch, Apple iWatch, ...
- Prenosivi senzorski uređaji (Wearable Sensor Devices)
- Fleksibilni i savitljivi displeji



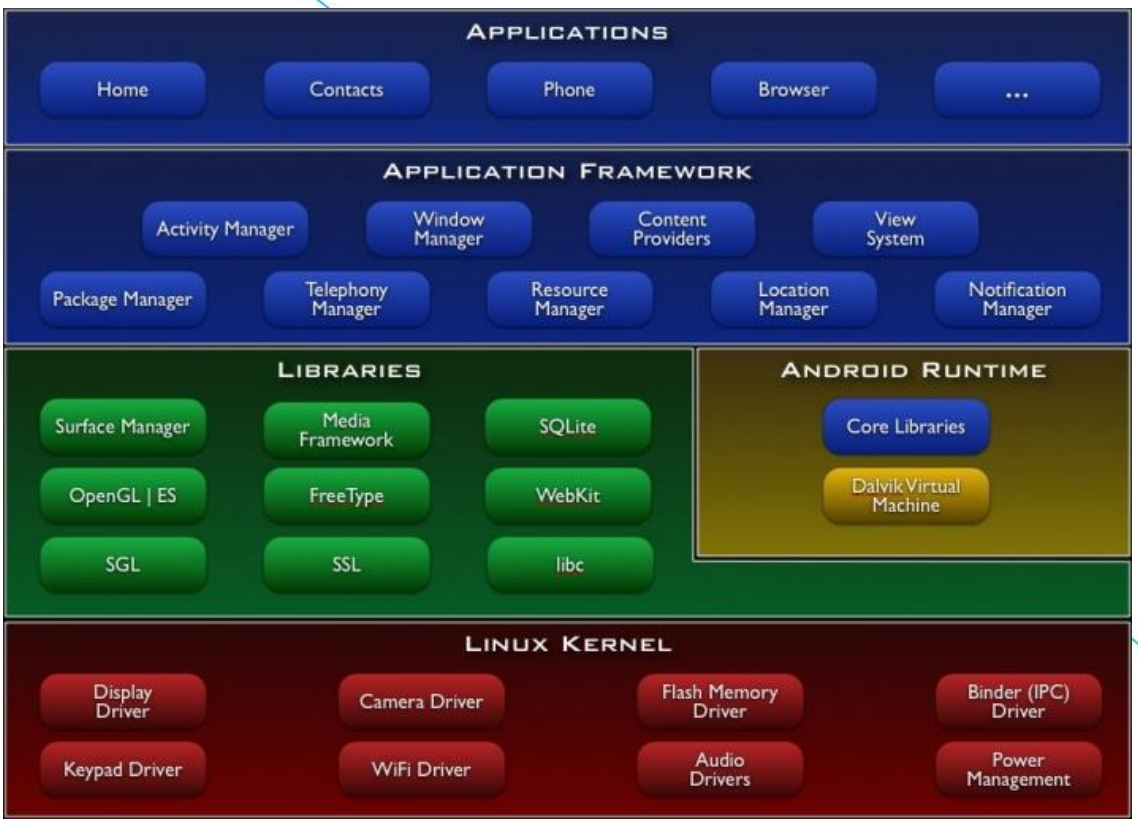
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Trajnije baterije

### 1.2 Android

Android je softversko okruženje, mobilna platforma koja uključuje operativni system i komponente za razvoj mobilnih aplikacija. Android uključuje (slika 1.1):

2 Kernel OS zasnovan na Linux-u
3 Bogat i atraktivan korisnički interfejs
4 Podršku za funkcionalnost mobilnog telefona
5 Korisničke aplikacije
6 Biblioteke
7 Framework za razvoj aplikacija
8 Multimedijalnu podršku, ...




Slika 1.1. Android

Aplikacije za Android se razvijaju u Java programskom jeziku. Korisničke aplikacije kao i Android "sistemske" aplikacije se pišu u Javi i prevode u byte code (dex).



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Elementi Android aplikacije:

- Intent – opisuje šta želimo da uradimo, predstavlja mehanizam komunikacije između komponenti jedne ili različitih Android aplikacija
- IntentFilter – aplikacija ovim mehanizmom registruje svoju sposobnost da obradi neki Intent

Android aplikacija se može sastojati od:

- Activity – jedan ekran aplikacije na kome korisnik obavlja neku aktivnost
- Service – funkcionalnost aplikacije koja se obavlja u pozadini bez grafičkog korisničkog interfejsa i direktne interakcije sa korisnikom
- BroadcastReceiver – komponenta koja obrađuje objave (poruke) poslate na nivou operativnog sistema
- ContentProvider – mehanizam preko koga različite mobilne aplikacije mogu razmenjivati podatke

Android emulator je prikazan na slici 1.2

Slika 1.2 Izgled Android emulatora

Kontrolisanje Android emulatora se obavlja različitim komandama. Emulator simulira

- ARM CPU
- LCD displej
- Tastaturu / DPad,
- Phone dugmiće



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Zvuk sa ulazom i izlazom
- Flash memoriju
- GSM model sa simuliranom SIM karticom
- GPS prijemnik

Na development računaru možemo kreirati proizvoljan broj Android Virtual Device (AVD) emulatora različitih karakteristika

Prikaz liste kreiranih AVD se obavlja komadnom:

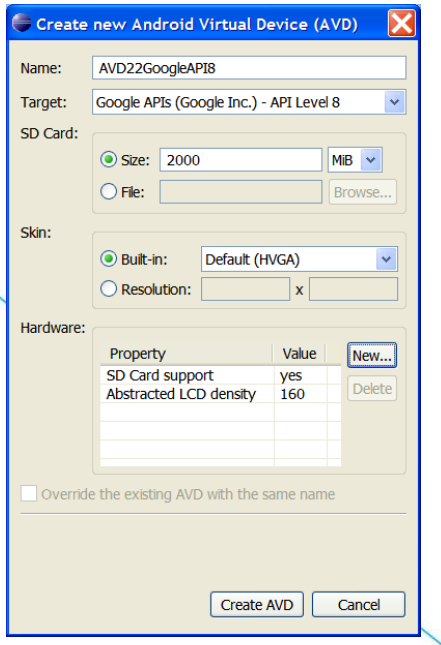
```
android list avd
```

Startovanje AVD emulatora se obavlja sledećom komandom:

```
emulator -avd <AVDName>
```

AVD se sastoji od sledećih elemenata:

- Hardverskog profila (kamera, QWERTY tastatura, DPAD ...)
- Image-a sistema
- Ostalih opcija (rezolucija ekrana, SD kartice ...)

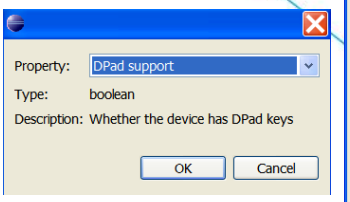



AVD se može kreirati i iz Eclipse okruženja:

Main Menu (AVD Manager ) → Virtual Device → New

Osnovni parametri AVD-a su:

- Naziv
- API (verzija Android-a)
- Veličina SD kartice
- Rezolucija ekrana
- Hardverske karakteristike
  - DPad podrška
  - Podrška za senzore (akcelerometar)
  - LCD density
  - ...




Za rad sa simuliranom SD karticom koristimo DDMS (Dalvik Debug Monitor Service) (slika 1.3)



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

ICT CLUSTER ACADEMY

TVOJA KARIJERA U TVOJIM RUKAMA

Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA

Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
adb devices

adb -s HT845GZ45737 pull data/app/theInstalled.apk
c:/theInstalled.apk
```

```
adb -s emulator-5554 install c:\theInstalledApp.apk

adb -s emulator-5554 uninstall data/app/theInstalled.apk
```

Android shell podržava neke od standardnih Linux shell komandi:

```
ls................ show directory (alphabetical order)
mkdir............. make a directory
rmdir............. remove directory
rm-r ............. to delete folders with files
rm................ remove files
mv................ moving and renaming files
cat ............... displaying short files
cd................ change current directory
pwd............... find out what directory you are in
df................ shows available disk space
chmod............. changes permissions on a file
date ............... display date
exit ............... terminate session
```

Svi eksterni događaji (SMS poruke, glasovne pozive, GPS koordinate itd.) mogu se simulirati iz konzole ili korišćenjem DDMS GUI alata (slika 1.3).

Na primer, da bi simulirali slanje SMS iz konzole, koristi se komanda:

```
adb devices
telnet localhost 5554
sms send <broj_telefona_posiljaoca> <tekst_poruke>
```

Na skoro identičan način kao i simulacija SMS poruka simuliraju se govorni pozivi iz konzole:

```
adb devices
telnet localhost 5554
gsm call <broj_telefona_sa_koji_zove>
```

Emulator omogućava zadavanje koordinata i testiranje lokaciono svesnih aplikacija, sledećom komandom iz konzole:

```
telnet localhost 5554
geo fix -121.45356 46.51119 4392
ili
geo nmea $GPRMC,081836,A,3751.65,S,14507.36,E,000.0,360.0,130998,011.3,E*62
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

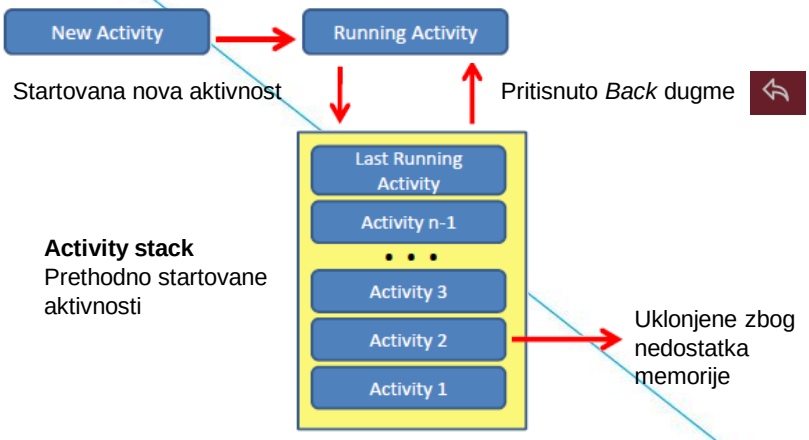
Ova komanda prihvata GPRMC i GPGGA NMEA rečenice za zadavanje simuliranih koordinata. Postoje dve mogućnosti:

- Ručno unošenje koordinata
- Playback GPX (GPS eXchange) ili KML (Keyhole Markup Language) snimljenih ruta

### 1.3 Životni ciklus Android aplikacije

Svaka Android aplikacija se izvršava u svom procesu na svojoj instanci Dalvik virtuelne mašine. Aplikacija ne kontroliše direktno svoj životni ciklus, već životni ciklus svake aplikacije kontroliše Android sistem na osnovu startovanih komponenti aplikacije, prioriteta aplikacije i raspoložive glavne memorije.

*Activity stack* vodi evidenciju o startovanim aktivnostima. Kada se startuje nova aktivnost smešta se na vrh stack-a i ona je aktivna, dok prethodno aktivna aktivnost ostaje ispod na stack-u. Kada se pritisne Back dugme aktivnost ispod na stack-u se pomera na vrh i postaje aktivna (slika 1.4)




Slika 1.4 Stek aktivnosti (*Activity stack*)

Aktivnost može biti u tri stanja:

- Aktivna (izvršava se)
- Pauzirana
- Zaustavljena

Stanje aktivan (izvršava se) označava da je aktivnost je na vrhu activity stack-a za trenutni task i ima fokus korisničke interakcije.

Stanje pauziran (paused) označava da aktivnost nema više fokus ali je i dalje vidljiva korisniku (potpuno ili delimično). Aktivnost čuva sva stanja i vrednosti promenljivih članica klase. Sistem će ovakvu aktivnost uništiti samo pri ekstremnim uslovima nedostatka memorije.

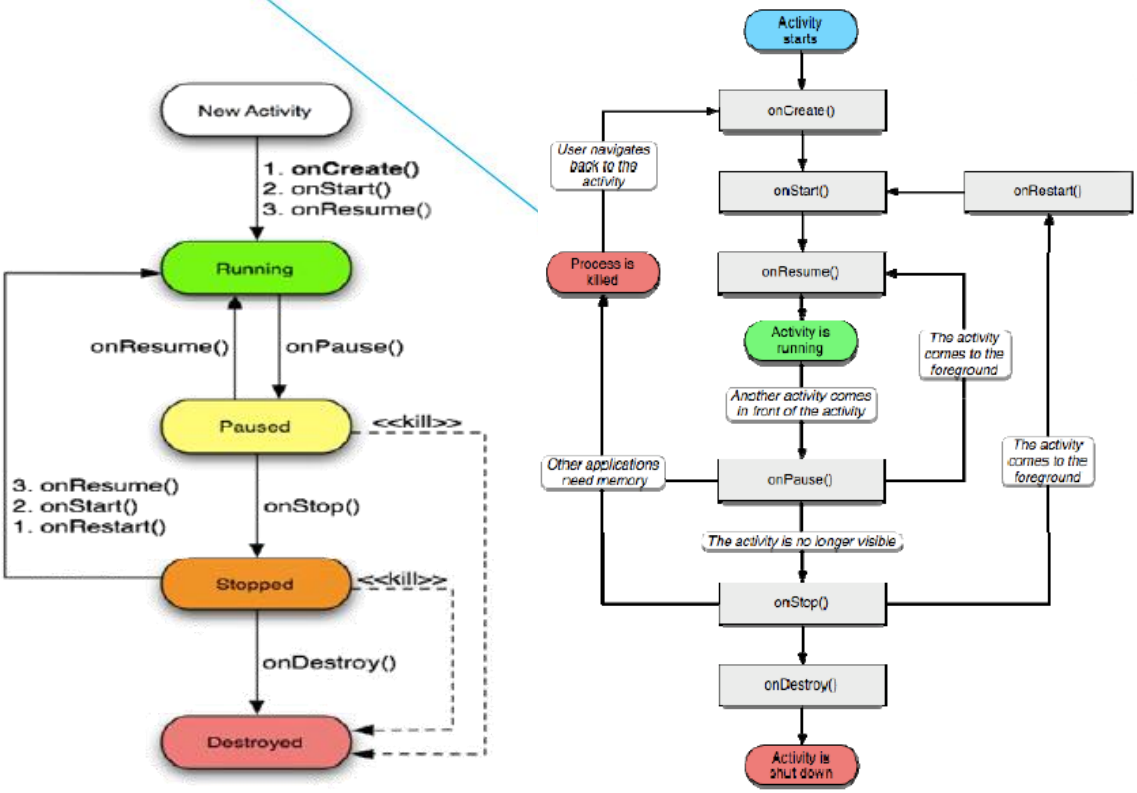


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Stanje zaustavljen (stopped) označava da aktivnost više nije vidljiva korisniku, ali i dalje čuva sva stanja i vrednosti promenljivih članice klase.

Prilikom prelaza aktivnost iz jednog u drugo stanje system poziva *callback* metode faza životnog ciklusa (slika 1.5):

- onCreate
- onStart
- onResume
- onPause
- onStop
- onRestart
- onDestroy




Slika 1.5. Stanja aktivnosti i clallback metode koje se pozivaju pri ovim prelazima



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

ICT CLUSTER ACADEMY

TVOJA KARIJERA U TVOJIM RUKAMA

Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA

Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

iii. Npr. Create Activity = HelloWorld, ovo opciono polje automatski kreira "Main Activity" klasu projekta.
iv. Npr. Min SDK Version = 4, ovo definiše minimali API nivo projekta i treba da se poklapa sa Build Target Platform izborom napravljenim ranije. Projekat neće moći da se izvršava na na uređajima sa nižim API nivoom.

f. Kliknite "Finish".

3. Detaljno proučite kreiranu aplikaciju:
   a. src: sadrži hijerarhiju paketa i .java klase koje se dodaju u projekat.
      i. HelloWorld.java: Aktivity koji je kreiran na osnovu unosa u prethodnim koracima.
   b. Android 1.6: Verzija biblioteke koja izabrana za projekat u prethodnim koracima. Aplikacija će biti kreirana na osnovu ove verzije android.jar fajla.
   c. res: ovaj folder će sadržati sve resurse koje će aplikacija koristiti. U startu sadrži tri osnovna foldera koje ADT podrazumevano kreira ali ih može imati veliki broj sa različitim opcijama (multimedija, orijentacija ekrana, regionalna podešavanja jezika itd.)
      i. drawable: sadrži slike i animacije koje se koriste u aplikaciji (moguće je da umesto jednog inicijalno sadrži tri foldera koji počinju sa drawable a koji sadrže slike za ekrane različite gustine piksela) .
         1. U njemu se nalazi icon.png kao ikonica aplikacije (ili Npr. Ic_launcher.png).
      ii. layout: sadrži xml fajlove u kojima se opisuje izgled svakog Activity-a. Kod Android programiranja preferira se korišćenje xml lazout fajlova u odnosu na kreiranje interfejsa u kodu.
         1. Inicijalno sadrži main.xml u kome je definisan UI kreiranog Activity-a.
      iii. values: sadrži value type resurse poput stringova, celih brojeva, nizova (sve konstante koje se koriste u kodu, preferira se njihovo navodjenje u resursima u odnosu na kod)
         1. Inicijalno sadrži strings.xml u kome su definisani stringovi potrebni HelloWorld aplikaciji.
   d. gen: sadrži fajlove koje ADT automatski generiše.
      i. R.java: autogenerisana klasa koja služi za pristup resursima.
   e. assets: sadrži asset fajlove, slično kao kod resursa, ali im se pristupa kao fajlovima korišćenjem AssetManager klase preko koje se učitavaju u vidu strimova.
   f. AndroidManifest.xml: svaki projekat mora da sadrži ovaj fajl istog imena i sadrži sve podatke neophodne za izvršavanje aplikacije ( privilegije, dozvole, listu Activity, Service, Broadcast Reciever i Content Provider klasa itd.)

   g. default.properties: ovaj fajl sadrži podešavanja projekta.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

ICT CLUSTER ACADEMY

TVOJA KARIJERA U TVOJIM RUKAMA

Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA

Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/instructions_text" />
    <EditText
        android:id="@+id/txtMsg"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
    <Button
        android:id="@+id/btnFinish"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/finish_button" />/

</LinearLayout>
```

d. U LifeCycleActivity.java klasi dodajte odgovarajuće importe i dodajte onClick metod kreiranom dugmetu.

```
import android.app.Activity;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import android.view.*;


public class LifeCycleActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Toast.makeText(this, "onCreate", 1).show();

        final Button button = (Button) findViewById(R.id.btnFinish);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                finish();
            }
        });
    }
```

e. Za svaku ključnu metodu vezanu za životni ciklus android aplikacije napravite Override

(onRestart(), onStart(), onResume(), onPause(), onStop(), onDestroy()).

f. Proverite kako se aplikacija ponaša.

g. Uskladite kod kao što je prikazano na slici. Svaki metod treba da ima poziv Toast.makeText(this, "*ime_funkcije_u_kojoj_se_nalazi*", 1).show(); i odgovarajući poziv metodi nadklase (Npr. super.onStart()).



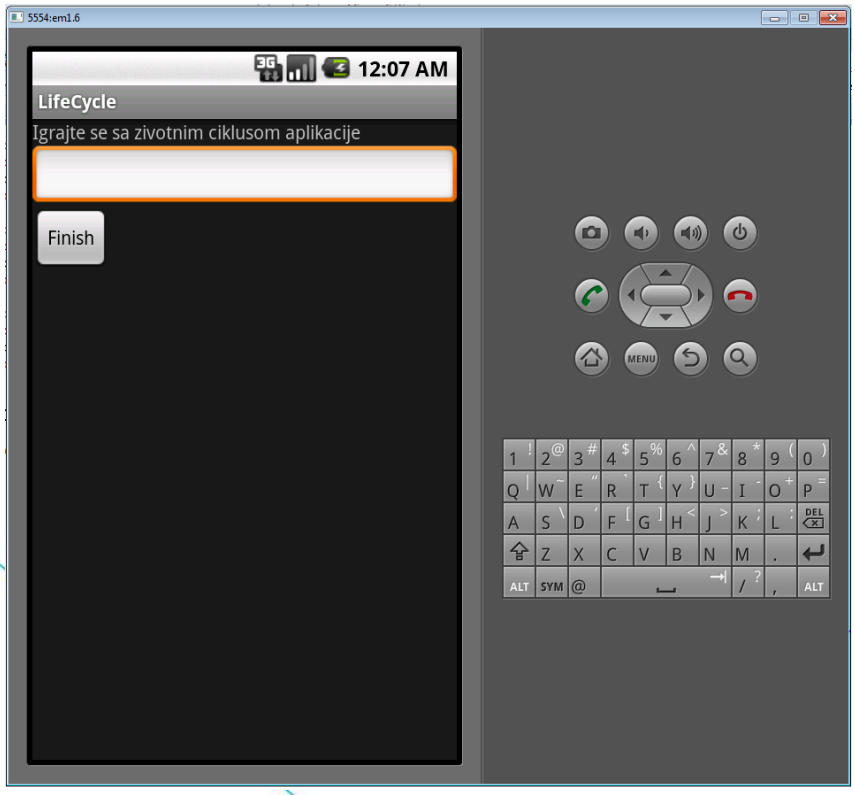
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
@Override
public void onRestart(){
    super.onRestart();
    Toast.makeText(this, "onRestart", 1).show();
}

@Override
public void onStart(){
    super.onStart();
    Toast.makeText(this, "onStart", 1).show();
}

@Override
public void onResume(){
    super.onResume();
    Toast.makeText(this, "onResume", 1).show();
}

@Override
public void onPause(){
    Toast.makeText(this, "onPause", 1).show();
    super.onPause();
}
@Override
public void onStop(){
    Toast.makeText(this, "onStop", 1).show();
    super.onStop();
}
@Override
public void onDestroy(){
    Toast.makeText(this, "onDestroy", 1).show();
    super.onDestroy();
}
```

h. Pratite dešavanje aplikacije u emulatoru nakon pozivanja telefonskog broja, pritiska na Back dugme itd.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES






Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 2. Android grafički korisnički interfejs i GUI layout-i

## 2.1 Mobilno korisničko iskustvo (Mobile user experience - UX)

Mobilni (pametni) telefoni su personalizovani uređaji koji se upotrebljavaju u različitom kontekstu, pri čemu je estetika mobilnih aplikacija važna koliko i funkcionalnost. Korisničko iskustvo (UX- User eXperience) se postiže grafičkim korisničkim interfejsom (GUI). Dizajn korisničkog interefjsa čini: informaciona arhitektura, načini interakcije, upotrebljivost i vizuelni dizajn (layout).

Dizajn korisničkog interfejsa se obavlja korišćenjem:

- Žičanog modela (Wireframe) - raspoređivanje elemenata GUI na prikazu ekrana - predstavlja skicu interfejsa mobilne aplikacije
- Prototipa – razvoj prototipa služi za vizuelizaciju i evaluaciju interfejsa i interakcije sa mobilnom aplikacijom.
    - Papirni prototip – statički prikaz, ali ne i interakcija,
    - HTML prototip – polufunkcionalni statički prototip razvijen korišćenjem HTML, CSS i JavaScript-a,
    - Interaktivni prototipovi - koriščenjem specijalizovanih alata.

Wireframe and prototip alati koji se mogu koristiti u dizjanu Androiod GUI-a:

- Prototyper - http://www.justinmind.com/prototyper
- Fluid - http://www.fluidui.com/
- Balsamiq - http://www.balsamiq.com/
- Pencil - http://pencil.evolus.vn/
- WireframeSketcher - http://wireframesketcher.com
- Moqups - https://moqups.com/home/ https://moqups.com/

Razvoj interaktivnih prototipova je moguć i korišćenjem App Inventor alata. App Inventor je framework za kreiranje mobilnih aplikacija za Android. Obezbeđuje Korisnički interfejs koji omogućava korisnicima da kreiraju aplikaciju korišćenjem *drag-and-drop* mehanizma sa vizuelnim elementima GUI koja može da se izvršava na Androidu. Moguće je kreirati jednostavne aplikacije u cilju razmatranja dizajan GUI elemenata i načina na koji je organizovana programska logika. Razvijen je od strane Google-a, a trenutno ga unapređuje MIT Center for Mobile Learning (http://appinventor.mit.edu/ ). Razvoj Android aplikacija se obavlja korišćenjem Web browsera i App Inventor Web aplikacije. Aplikacije se mogu testirati na emulator-u i povezanom mobilnom telefonu. Komponente App Inventor:

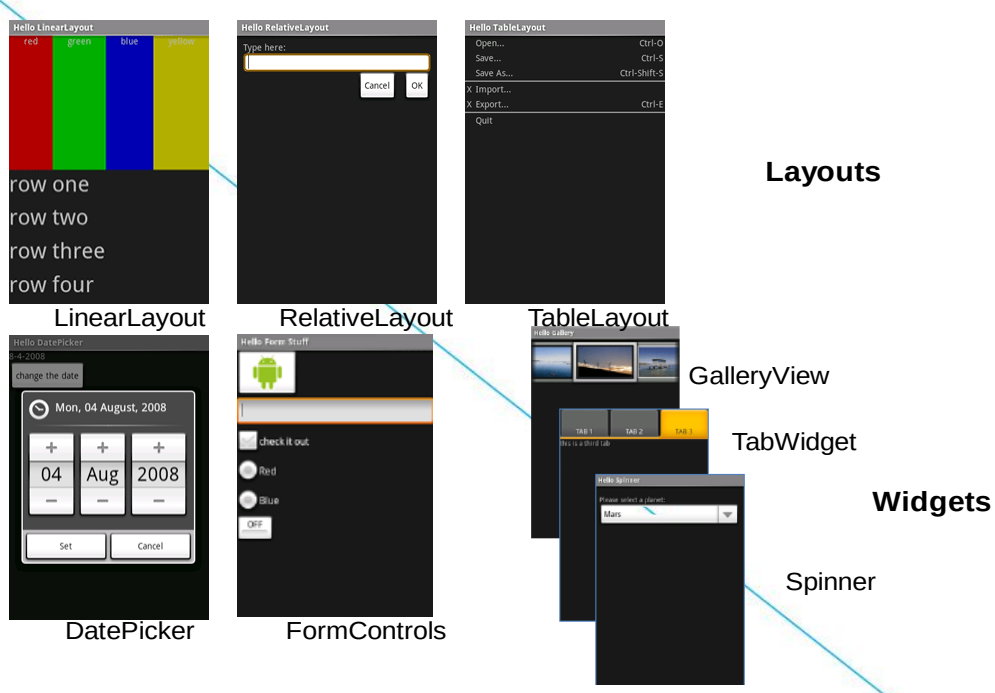
- The App Inventor Designer
- The App Inventor Blocks Editor

Android aplikacija se razvija vizuelno bez programiranja, a nedostaci su što podržava samo Android (nisu podržane druge platform), dizajn korišćenjem vizuelnih tehnika može biti težak i nije moguć export u Java programski kod, tako da nije moguće kreiranje koda na osnovu dizajna.

## 2.2 Android korisnički interfejs

Korisnički interfejs Android aplikacije se sastoji od (slika 2.1):

- Stila organizacije – Layout
- Elemenata korisničkog interfejsa – Widgeta

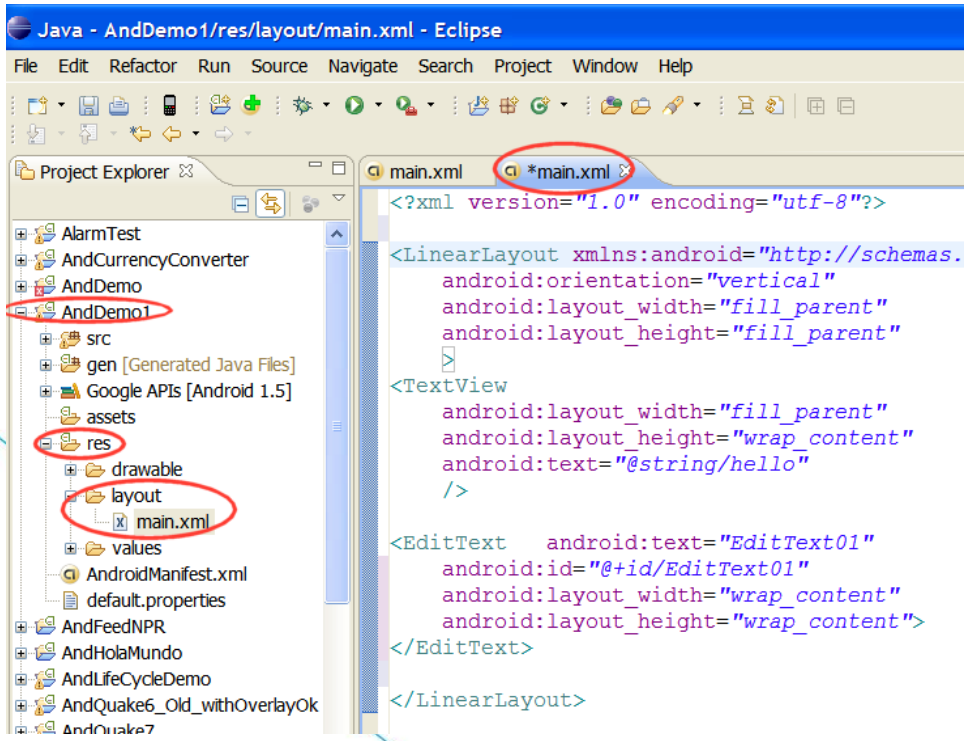



Slika 2.1. Elementi korisničkog interfejsa Android aplikacije

GUI Layout se specificira XML dokumentom i predstavlja resurs Android aplikacije (slika 2.2). XML layout predstavlja stablo UI elemenata. Atributi XML elemenata su svojstva (*properties*) i određuju kako izgledaju i kako se ponašaju UI elementi. Svaki element UI-a u XML layout-u kome se pristupa iz koda mora da ima dodeljen ID atribut (android:id). Ukolikotreba prikazati tekst na dugmetu koristi se android:text. Dimenzije dugmeta i način na koji "popunjava" roditeljski kontejner specificiraju se atributima: android:layout_width i android:layout_height.

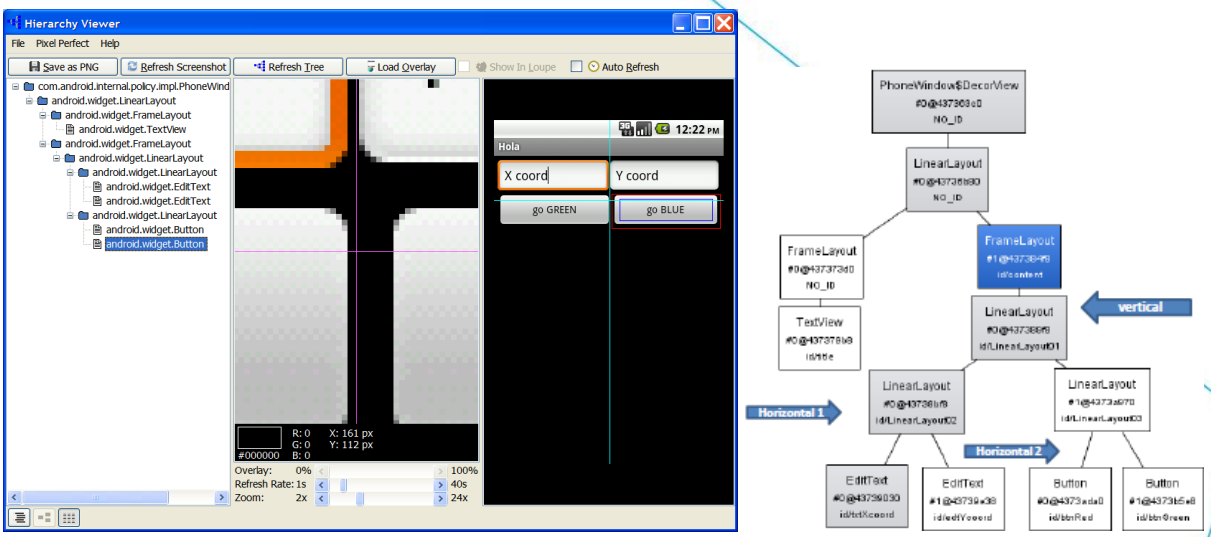


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




Slika 2.2. XML specficikacija layout-a Android aplikacije

Alat **HierarchyViewer** iz Tools podfoldera u SDK (hierarchyviewer.bat) prikazuje UI stablo trenutno aktivne Android aplikacije (na emulatoru ili povezanom uređaju) (slika 2.3).




Slika 2.3. Prikaz UI stabla Android aplikacije u HierarchyViewer-u

Primer layout-a dat je na slici 2.4.



Program finansira: USAID SRBIJA OD AMERIČKOG NARODA Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

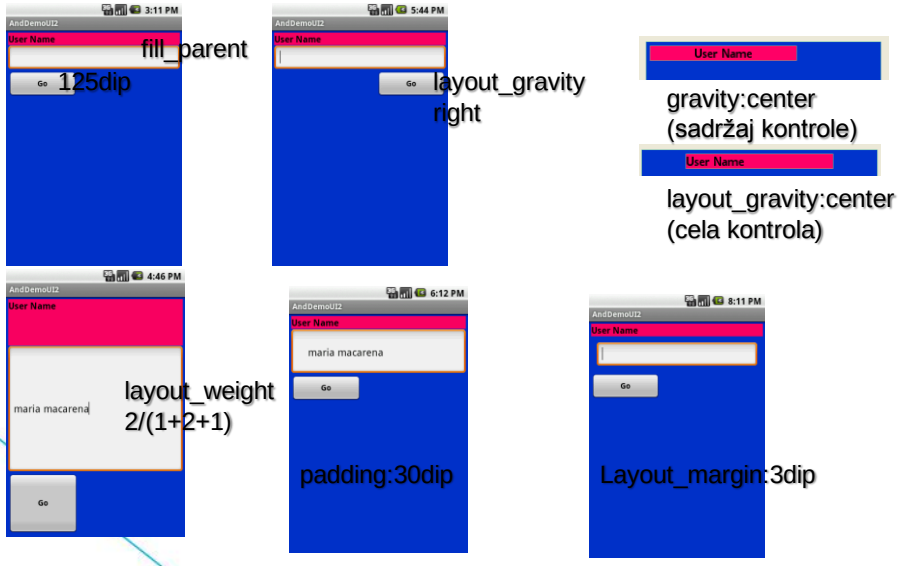
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id="@+id/LinearLayout01"
android:layout_width="fill_parent" android:layout_height="fill_parent"
android:orientation="vertical" xmlns:android="http://schemas.android.com/apk/res/android">
      <LinearLayout android:id="@+id/LinearLayout02"
      android:layout_width="fill_parent" android:layout_height="wrap_content">
            <EditText android:id="@+id/txtXcoord" android:layout_width="wrap_content"
                  android:layout_height="wrap_content" android:text="X Coord"
                  android:layout_weight="1">
            </EditText>
            <EditText android:id="@+id/edtYcoord" android:layout_width="wrap_content"
                  android:layout_height="wrap_content" android:text="Y Coord"
                  android:layout_weight="1">
            </EditText>
      </LinearLayout>
      <LinearLayout android:id="@+id/LinearLayout03"
      android:layout_width="fill_parent" android:layout_height="wrap_content">
            <Button android:id="@+id/btnRed" android:layout_width="wrap_content"
                  android:layout_height="wrap_content" android:text="red"
                  android:layout_weight="1">
            </Button>
            <Button android:id="@+id/btnGreen" android:layout_width="wrap_content"
                  android:layout_height="wrap_content" android:text="green"
                  android:layout_weight="1">
            </Button>
      </LinearLayout>
</LinearLayout>
```

Slika 2.4 Primer layout-a

Najčešće korišćeni layout-i:

- LinearLayout
- RelativeLayout
- TableLayout
- FrameLayout
- ScrollView
- ListView, GridView, WebView, MapView...

FrameLayout je najjednostavniji i postavlja sve GUI kontrole (widget-e) u gornji levi ugao jednu preko druge. Različiti položaji GUI kontrola na Linear Layout-u dati su na slici 2.5.



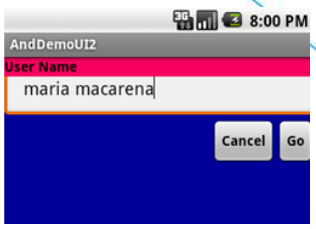

Slika 2.5 Položaji GUI kontrola na Linear Layout-u

Relative Layout smešta kontrole (widget-e) relativno u odnosu na ostale elemente UI. Kako bi koristili relativni layout svim kontrolama treba dodeliti id-jeve android:id="@+id/ediUserName"

Primer referenciranja ovakve kontrole je android:layout_below="@+id/ediUserName“. Primer Relative Layout-a sa raspoređenim kontrolama dat je na slici 2.6.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
android:id="@+id/myRelativeLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#ff000099"
xmlns:android="http://schemas.android.
oid">

<TextView
android:id="@+id/lblUserName"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:background="#ffff0066"
android:text="User Name"
android:textStyle="bold"
android:textColor="#ff000000"
android:layout_alignParentTop="true"
android:layout_alignParentLeft="true">
</TextView>
```

```
<EditText
android:id="@+id/ediUserName"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_below="@+id/lblUserName"
android:layout_alignParentLeft="true"
android:layout_alignLeft="@+id/myRelativeLayout"
android:padding="20dip">
</EditText>

<Button
android:id="@+id/btnGo"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_below="@+id/ediUserName"
android:layout_alignRight="@+id/ediUserName"
android:text="Go"
android:textStyle="bold">
</Button>

<Button
android:id="@+id/btnCancel"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_toLeftOf="@+id/btnGo"
android:layout_below="@+id/ediUserName"
android:text="Cancel"
android:textStyle="bold">
</Button>
</RelativeLayout>
```

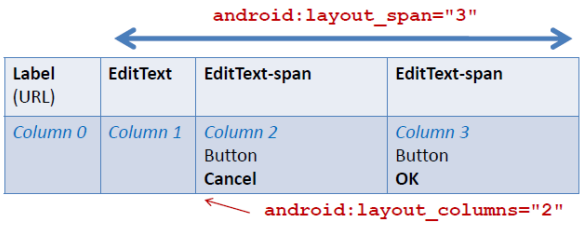



Slika 2.6. Relative layout primer



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Table Layout definiše grid kolona i redova. Kolone se šire i skupljaju u skladu sa dimenzijama widgeta koje sadrže. Table Layout definiše generalno ponašanje tabele, a widget-i se dodaju u TableRow. Broj kolona je automatski određen od strane sistema i broja dodatih widget-a. Jedan widget može da zauzme više kolona, na primer android:layout_span="3“ (slika 2.7)



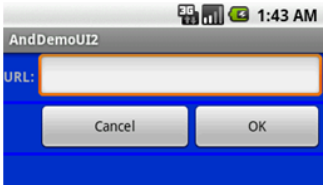

Slika 2.7 Table Layout

Primer Table Layout-a dat je na slici 2.8.

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
android:id="@+id/myTableLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#ff0033cc"
android:orientation="vertical"
xmlns:android="http://schemas.android.com/apk/res/android"
>
<TableRow>
<TextView
android:text="URL:" />
<EditText android:id="@+id/ediUrl"
android:layout_span="3"/>
</TableRow>
<View
android:layout_height="3dip"
android:background="#0000FF" />
<TableRow>
<Button android:id="@+id/cancel"
android:layout_column="2"
android:text="Cancel" />
<Button android:id="@+id/ok"
android:text="OK" />
</TableRow>
<View
android:layout_height="3dip"
android:background="#0000FF" />

</TableLayout>
```

Strech up to column 3

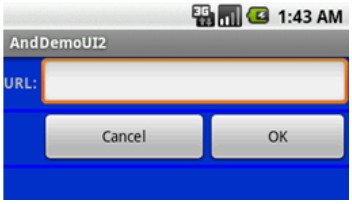
Skip columns: 0, 1




Slika 2.8 Primer Table Layout-a

Podrazumevano ponašanje kolone je da prilagodi svoju veličinu prirodnoj veličini widget-a koji sadrži. Ukoliko je sadržaj manji od dostupnog prostora kolona može da popuni preostali proctor korišćenjem atributa: android:stretchColumns=“…” i navođenjem spiska kolona koje će se proširiti da zauzmu preostali proctor (slika 2.9)

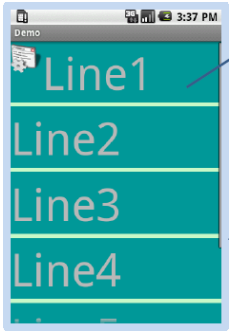
```
<TableLayout
android:id="@+id/myTableLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#ff0033cc"
android:orientation="vertical"
android:stretchColumns ="2,3,4"
xmlns:android="http://schemas.android
>
```




Slika 2.9 Podešavanje širine kolona u Table Layout-u

Ukoliko layout sadrži više widget-a nego što može da se prikaže na ekranu koristi se ScrollView layout u kome korisnik može da skroluje layout. Primer ScrollView layout-a dat je na slici 2.10.

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView
android:id="@+id/myScrollView1"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#ff009999"
xmlns:android="http://schemas.android.com/apk/res/android"
>
<LinearLayout
android:id="@+id/myLinearLayoutVertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical"
>

<LinearLayout
android:id="@+id/myLinearLayoutHorizontal1"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="horizontal"
>
<ImageView
        android:id="@+id/myPicture"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/icon" />
<TextView
        android:id="@+id/textView1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Line1"
        android:textSize="70dip" />
</LinearLayout>

<View
        android:layout_width="fill_parent"
        android:layout_height="6dip"
        android:background="#ffccffcc" />
<TextView
        android:id="@+id/textView2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Line2"
        android:textSize="70dip" />
<View
        android:layout_width="fill_parent"
        android:layout_height="6dip"
        android:background="#ffccffcc" />
<TextView
        android:id="@+id/textView3"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Line3"
        android:textSize="70dip" />
<View
        android:layout_width="fill_parent"
        android:layout_height="6dip"
        android:background="#ffccffcc" />
<TextView
        android:id="@+id/textView4"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Line4"
        android:textSize="70dip" />
<View
        android:layout_width="fill_parent"
        android:layout_height="6dip"
        android:background="#ffccffcc" />
<TextView
        android:id="@+id/textView5"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Line5"
        android:textSize="70dip" />
</LinearLayout>

</ScrollView>
```

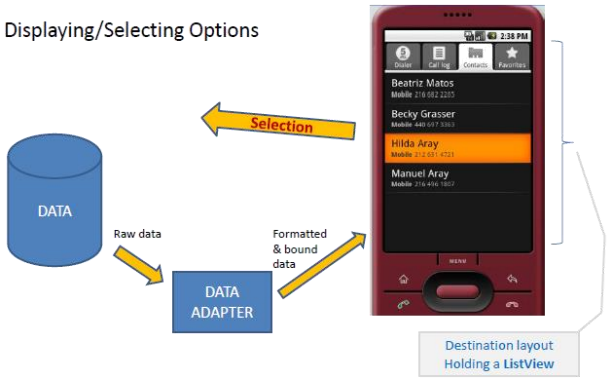



Slika 2.10 Primer ScrollView layout-a

U okviru GUI Android aplikacije mogu se koristiti različiti widget-i (UI kontrole). Vrlo često u korisničkom interfejsu treba izabrati neku od ponuđenih stavki. Za manji broj opcija koriste se RadioButton i CheckButton. Za veći broj opcija koriste se listbox, combobox, drop-down liste, galerije slika, itd. U pozadini svih ovih kontrola je model podataka povezan kroz koncept data adapter-a (slika 2.11).



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




Slika 2.11 Koncept Data Adapter-a

Za popunjavanje widget-a podacima se koriste adapter. Najjednostavniji primer je ArrayAdapter. *Wrap*-uje se oko niza ili java.util.List.

```
String[] items={"this", "is", "a","really", "silly", "list"};

new ArrayAdapter<String>(this,
                         android.R.layout.simple_list_item_1,
                         items);
```

Parametri konstruktora ArrayAdapter-a su:

- Context – tipično Activity
- Resource ID view-a koji se koristi za red - android.R.layout.simple_list_item_1 je jedan od predefinisanih
- Array ili List koji se povezuje.

Koristimo ListActivity za jednostavne liste. Layout može i da se prilagodi. Vrednost atributa @android:id/list je view za prikaz liste, dok se @android:id/empty prikazuje kada nema podataka (slika 2.12).



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <TextView
        android:id="@+id/selection"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#ff0000cc"
        android:textStyle="bold" />
    <!-- Here is the list. Since we are using a ListActivity, we hav
    <ListView
        android:id="@android:id/list"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:drawSelectorOnTop="false" />

     <TextView android:id="@android:id/empty"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Empty set"  />
</LinearLayout>
```

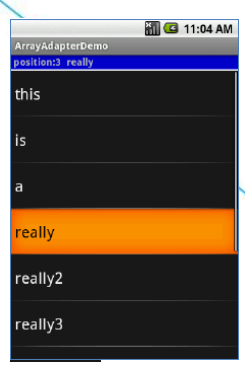
Slika 2.12 Layout sa specificiranim ListView-om

U ovom slučaju koristi se ListActivity umesto Activity.

```
public class ArrayAdapterDemo extends ListActivity {

   TextView selection;
   String[] items = { "this", "is", "a", "really",
                      "really2", "really3","really4",
                      "really5", "silly", "list" };
```

ListView povezujemo sa podacima korišćenjem adaptera ArrayAdapter.

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    setListAdapter(new ArrayAdapter<String>(
                   this,
                   android.R.layout.simple_list_item_1,
                   items));

    selection=(TextView)findViewById(R.id.selection);
}
```

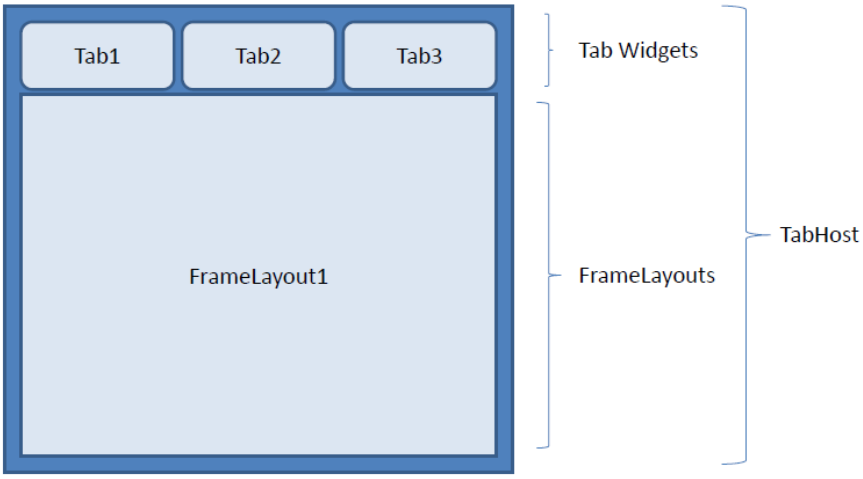



Preporuka je da Android UI bude što je moguće jednostavniji. Ako imamo mnogo informacija za prikaz koristi se tab widget da se te informacije logički podele (slika 2.13). U kreiranju tab widget-a učestvuju:

- TabHost - Glavni kontejner
- TabWidget - Dugmići za tabove
- FrameLayout - Sadržaj tab-a.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




Slika 2.13 Tab Widget-i, TabHost i Tab Widget / Frame Layout

Primer za Tab Widget-e i Frame Layout je prikazan na slici 2.14.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=http://schemas.android.com/apk/res/android
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
  <TabHost android:id="@+id/tabhost"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
    <TabWidget android:id="@android:id/tabs"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"/>
    <FrameLayout android:id="@android:id/tabcontent"
          android:layout_width="fill_parent"
          android:layout_height="fill_parent"
          android:paddingTop="62px">

          <!--PUT HERE FrameLayout1 -->
          <!--PUT HERE FrameLayout2 -->
    </FrameLayout>
  </TabHost>
</LinearLayout>
```

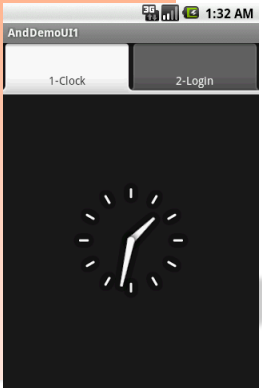
Slika 2.14 Primer TabWidget-a i FrameLayout-a

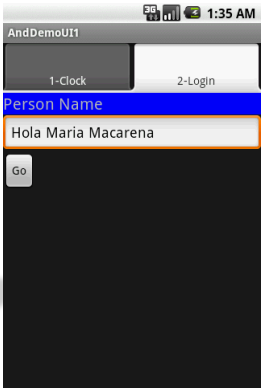
Deo koda demo aplikacije koja GUI organizuje korišćenjem Tab Widget-a dat je na sledeći način:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
public class AndDemoUI1 extends Activity {
  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
    TabHost tabs=(TabHost)findViewById(R.id.tabhost);
    tabs.setup();

    TabHost.TabSpec spec;
    spec =tabs.newTabSpec("tag1");
    spec.setContent(R.id.tab1);
    spec.setIndicator("1-Clock");
    tabs.addTab(spec);

    spec=tabs.newTabSpec("tag2");
    spec.setContent(R.id.tab2);
    spec.setIndicator("2-Login");
    tabs.addTab(spec);

    tabs.setCurrentTab(0);
```




Na Tab Widget može da se postavi i ikona, na sledeći način:

```
spec.setIndicator("2-Login",
        getResources().getDrawable(R.drawable.ic_menu_info_details));
```

Ukoliko je neophodno detektovanje promene tab-a registruje se odgovarajući listener:

```
tabs.setOnTabChangedListener(
  new OnTabChangeListener() {
    @Override
    public void onTabChanged(String tagId) {
      String text = "Imcurrently in: "+ tagId+ "\nindex: "+
              tabs.getCurrentTab();
      Toast.makeText(getApplicationContext(), text, 1).show();
    }
  });
```

### 2.3 Praktična vežba 2 - Android grafički korisnički interfejs i layout-i

Nakon upoznavanja sa organizacijom Android projekta i životnim ciklusom Android aplikacija, neophodno je upoznati se i sa osnovnim konceptima vezanim za korisnički interfejs. Vežba 2 daje uvod u osnovne elemente korisničkog interfejsa (TextView, Button, ListActivity itd.) pri čemu se poseban akcenat stavlja na korišćenje resursa i modularnost koju pružaju programerima. Potrebno je napraviti kostur aplikacije koja služi za pregledavanje i dodavanje omiljenih lokacija (My places). Ovaj projekat će kroz vežbe biti nadograđivan tako da je neophodno čuvati prethodne verzije.

Ciljevi ove vežbe su:

1. Kreiranje Android projekta My Places.
2. Definisanje glavnog ekrana aplikacije.
3. Dodavanje About Activity-a i dodavanje Click Listener-a.
4. Dodavanje jednostavnog ListActivity-a koji prikazuje elemente statičkog niza.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

1. Kreiranje Android projekta My Places
    a. Pokrenuti Eclipse razvojno okruženje
    b. Kreirati novi projekat (File->New Project->Android project).
    c. Dodeliti ime novom projektu (Project Name: MyPlaces)
    d. Izabrati željenu verziju Androida. Za sada je dovoljna i verzija 1.6.
    e. Popunite stavke:
        i. Application Name = My Places
        ii. Package Name = edu.mad.myplaces
        iii. Create Activity = MyPlacesActivity
        iv. Min SDK Version = 4
    f. Kliknite "Finish".
2. Definisanje glavnog ekrana aplikacije:
    a. U string.xml dodati string resurse za prikaz na elementima korisničkog interfejsa koji će biti dodati u layout-u.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="main_title">My Places</string>
    <string name="app_name">MyPlaces</string>
    <string name="my_places_label">My places</string>
    <string name="new_place_label">Add new place</string>
    <string name="show_map_label">Show map</string>
    <string name="about_label">About</string>
    <string name="exit_label">Exit</string>
</resources>
```

    b. U res/layout/main.xml dodati novi LinearLayout I u njega TextView i pet Button-a. TextView treba da prikaže ime aplikacije. Button-i će biti korišćeni za pregledavanje omiljenih lokacija, dodavanje nove lokacije, prikaz lokacija na mapama, prikaz teksta o aplikaciji i dugme za izlaz. Na sledećoj slici je prikazan layout.xml.



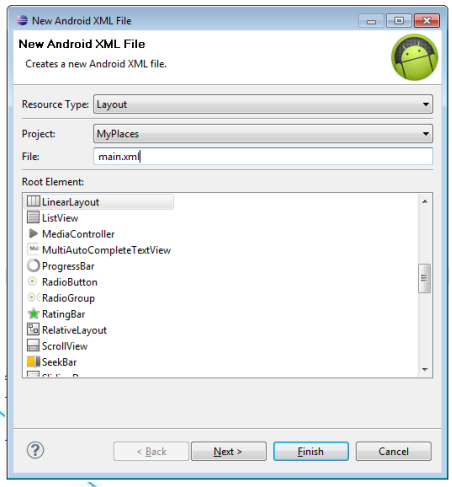
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

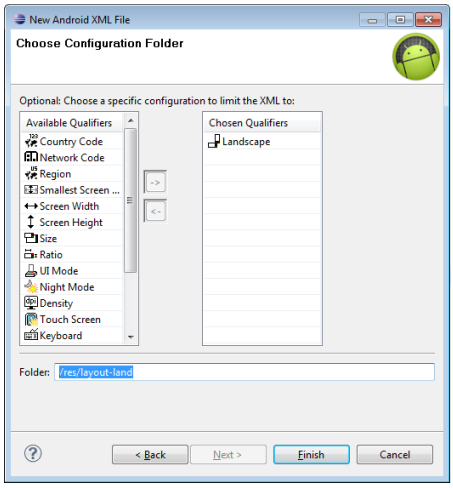
ICT CLUSTER ACADEMY

TVOJA KARIJERA U TVOJIM RUKAMA

Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA

Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




f. U layout.xml fajlu definisati layout kao na slici. Uočiti novi element TableLayout i TableRow. TableLayout omogućava tabelarno uređenje korisničkog interfejsa poput korišćenja tabela u HTML-u.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="fill_parent"
    android:layout_width="fill_parent"
    android:padding="15dip"
    android:orientation="horizontal" >
    <LinearLayout
        android:orientation="vertical"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:layout_gravity="center"
        android:paddingLeft="20dip"
        android:paddingRight="20dip" >
        <TextView
        android:text="@string/main_title"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:layout_gravity="center"
        android:layout_marginBottom="20dip"
        android:textSize="24.5sp" />
        <TableLayout
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:layout_gravity="center"
            android:stretchColumns="*" >
            <TableRow>
                <Button
                    android:id="@+id/my_places_button"
                    android:text="@string/my_places_label" />
                <Button
                    android:id="@+id/new_place_button"
                    android:text="@string/new_place_label" />
            </TableRow>
            <TableRow>
                <Button
                    android:id="@+id/show_map_button"
                    android:text="@string/show_map_label" />
                <Button
                    android:id="@+id/about_button"
                    android:text="@string/about_label" />
            </TableRow>
            <TableRow>
                <Button
                    android:id="@+id/exit_button"
                    android:text="@string/exit_label"
                    android:layout_span="2" />
            </TableRow>
        </TableLayout>
    </LinearLayout>
</LinearLayout>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

ICT CLUSTER ACADEMY

TVOJA KARIJERA U TVOJIM RUKAMA

Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA

Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dip" >
    <TextView
        android:id="@+id/about_content"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/about_text" />
</ScrollView>
```

c. Zatim dodati odgovarajuće stringove u strings.xml fajl.

```
    <string name="about_label">About</string>
    <string name="exit_label">Exit</string>
    <string name="about_title">About My Places</string>
    <string name="about_text">\
    <i>My places</i> is application that will help you remember characteristics and locations of your favorite places.
    It will be able to show them on map and share them with your friends. Each week this application will be better and better
    and you will collect <i>bonus stars</i> from teaching assistant.
    </string>
</resources>
```

d. Da bi mogli da prikažemo About Activity, potrebno je dodati Listener-e svakom od dugmića MyPlacesActivity.java.

i. Prvo treba dodati odgovarajuće import-e.

```
import android.app.Activity;
import android.os.Bundle;
import android.content.Intent;
import android.view.View;
import android.view.View.OnClickListener;
```

ii. Zatim prepraviti klasu tako da implementira OnClickListener.

```
public class MyPlacesActivity extends Activity implements OnClickListener {
```

iii. Primetićete grešku. MyPlacesActivity.java mora da implementira i metod OnClick(View). Dovoljno je dva puta duplim klikom naterati Eclipse da nam pomogne u tome.

iv. Postavimo Listener-e svakom od dugmadi.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    View myPlacesButton = findViewById(R.id.my_places_button);
    myPlacesButton.setOnClickListener(this);
    View newPlaceButton = findViewById(R.id.new_place_button);
    newPlaceButton.setOnClickListener(this);
    View showMapButton = findViewById(R.id.show_map_button);
    showMapButton.setOnClickListener(this);
    View aboutButton = findViewById(R.id.about_button);
    showMapButton.setOnClickListener(this);
    View exitButton = findViewById(R.id.exit_button);
    exitButton.setOnClickListener(this);
}
```

v. Dodati u onClick metod proveru koje je dugme kliknuto i za about_button dodati kreiranje Intent-a i poziv startActivity(). Ukoliko je kliknuto Exit dugme, pozvati finish() metod za izlazak iz aplikacije (upoznali smo ga na prošloj vežbi).

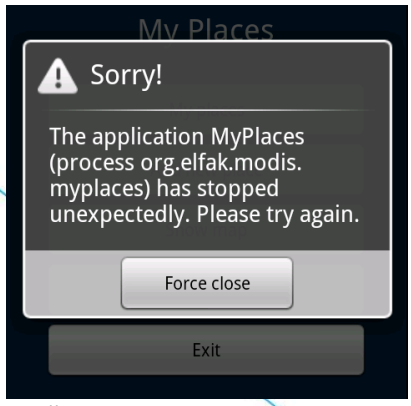


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
@Override
public void onClick(View v) {
    switch(v.getId())
    {
        case R.id.my_places_button:
            //poziv odgovarajuceg metoda
            break;
        case R.id.new_place_button:
            //poziv odgovarajuceg metoda
            break;
        case R.id.show_map_button:
            //poziv odgovarajuceg metoda
            break;
        case R.id.about_button:
            Intent i = new Intent(this, About.class);
            startActivity(i);
            break;
        case R.id.exit_button:
            finish();
            break;
    }
}
```

e. Probajte aplikaciju. Šta se dešava ako kliknemo na dugme About?

f. Aplikacija neobjašnjivo prestaje sa radom uz sledeće obaveštenje:




g. Razlog za ovo je što novi Activity nije registrovan u AndroidManifest fajl-u. U AndroidManifest.xml dodajte:

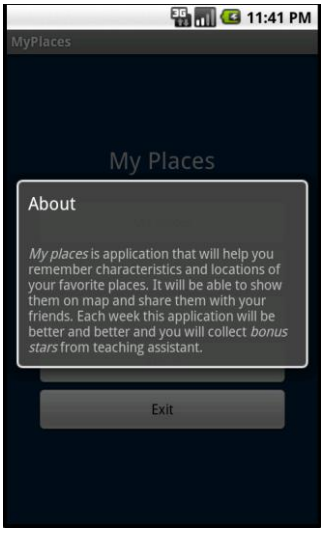
```
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity android:name=".About"
        android:label="@string/about_label">
</activity>
```

h. Probajte aplikaciju sada.

i. Izgled About-a bi više trebalo da liči na dijalog a ne da popunjava čitav ekran. To je moguće definisati pomoću tema. Teme (theme) su kolekcije stilova koji određuju izgled Android widget-a slično kao CSS kod Web tehnologija. Android poseduje nekoliko predefinisanih tema a moguće ih je definisati nasleđivanjem postojećih. Koristićemo postojeću temu Dialog. U AndroidManifest.xml dodajte temu About Activiti-ju kako je prikazano na slici.

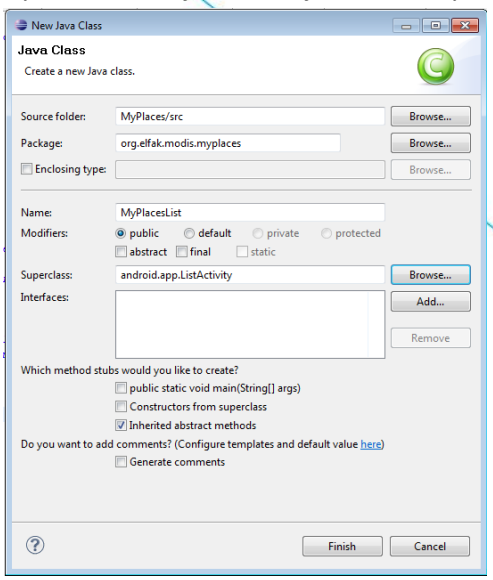


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




```
<activity android:name=".About"
        android:label="@string/about_label"
        android:theme="@android:style/Theme.Dialog" >
</activity>
```

4. Dodavanje ListActivity-a sa predefinisanim stringovima za lokacije.
   a. Potrebno je dodati jednostavan ListActivity koji prikazuje vrednosti iz statičkog niza stringova. Ovo predstavlja samo osnovno upoznavanje sa ovim widget-om dok će se na narednim vežbama ova kontrola značajnije koristiti.
   b. Dodati MyPlacesList koja nasleđuje ListActivity.




   c. Dodati odgovarajuće stringove u resurse i uskladiti AndroidManifest.xml.
   d. Dodati statički niz stringova u klasu MyPlacesList.java.
   e. U onCreate metodi popuniti listu korišćenjem ArrayAdaptera.

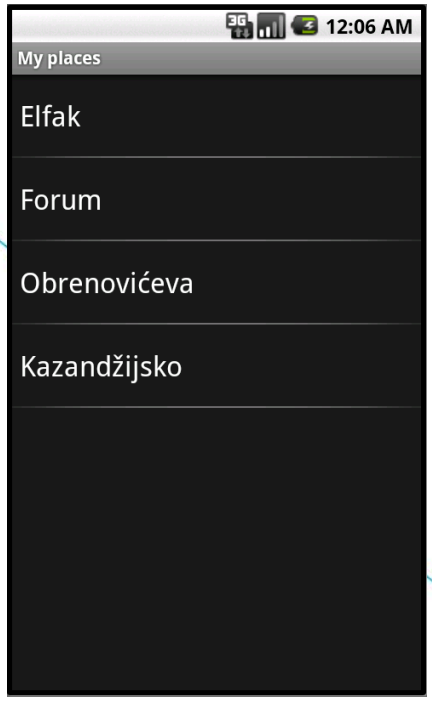


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
public class MyPlacesList extends ListActivity {


    String[] places = {"Elfak","Forum", "Obrenovićeva", "Kazandžijsko"};
     public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            //ne treba nam layout jer koristimo predefinisani layout za ListActivity
            //setContentView(R.layout.main);
            setListAdapter(new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, places));
        }
}
```

f. Iz MyPlacesActivity-ja pozvati MyPlacesList Activity na odgovarajući klik dugmeta.

g. Izgled aplikacije nakon klika na dugme My places.






Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 3. Interakcija sa korisnikom - Meniji, diajalozi, intent-i

## 3.1 Meni

Meni proširuje funkcionalnost aplikacije tako što prikazuje dodatne operacije na malom preklapajućem panelu. Na Androidu postoje dva tipa menija (slika 3.1):

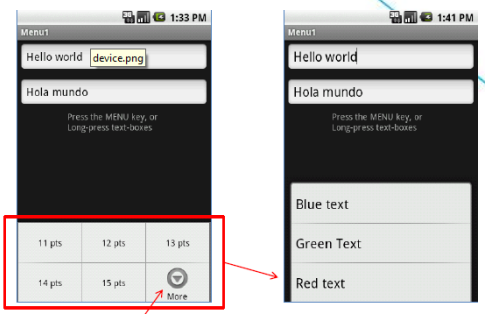
- Options menu - aktivira se pritiskom na hardversko Menu dugme na telefonu ili na menu virtuelno dugme na Navigatioon bar-u na novijim uređajima. Options menu ima ograničen broj stavki koje može da prikaže.
- Context menu - aktivira se *tap-and-hold* akcijom nad widget-om za koji je meni vezan.

a) b)

Slika 3.1 Meniji u Android aplikacijama a) options, b) context

Obe vrste Android menija mogu da sadrže

- Tekst
- Ikonu
- Radio dugme
- Check polje
- Podmeni
- Prečice (shortcuts)

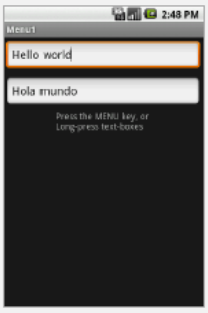


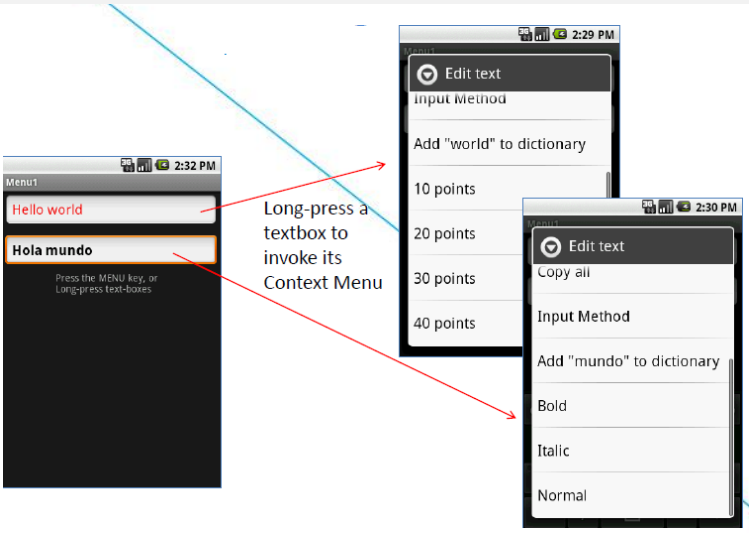

Options meni prikazuje maksimalno 6 opcija. Ukoliko ima više od 6 opcija automatski se dodaje *More* polje. Primer obe vrste menija u android aplikaciji imaće GUI sa dva EditText widget-a sa kontekstnim menijima (slika 3.2).



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"  >

<EditText
     android:id="@+id/etMessage1"
     android:text="Hello world"
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:layout_margin = "5dp" />

<EditText
     android:id="@+id/etMessage2"
     android:text="Hola mundo"
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:layout_margin = "5dp" />

<TextView
     android:text="Press the MENU key, or \nLong-press text-boxes"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:layout_gravity="center" />
</LinearLayout>
```




Slika 3.2 Primer context menija

Vezivanje kontekstnog menija za widget obavlja se sledećim programskim kodom.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NICAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
import android.app.Activity;
…
Public class Menu1Act1 extends Activity {
  EditText etMessage1;
  EditText etMessage2;
  Integer[] arrayPointSize = {10, 20, 30, 40, 50};

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    etMessage1= (EditText)findViewById(R.id.etMessage1);
    etMessage2= (EditText)findViewById(R.id.etMessage2);

    // you may register an individual context menu for each view
    registerForContextMenu(etMessage1);
    registerForContextMenu(etMessage2);
  } //onCreate
...
```

Popunjavanje menija:

```
  // set the option menu for the current activity
  @Override
  publi cboolean onCreateOptionsMenu(Menu menu) {
    // only one Option menu per activity
    populateMyFirstMenu(menu);
    return super.onCreateOptionsMenu(menu);
  }

  // detect what view is calling and create its context menu
  @Override
  public void onCreateContextMenu(ContextMenu menu, View v,
        ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    // decide what context menu needs to be made
    if(v.getId() == etMessage1.getId()){
      // create a menu for etMessage1 box
      populateMyFirstMenu(menu);
    }
    if(v.getId() == etMessage2.getId()){
      // create a menu for etMessage2 box
      populateMySecondMenu(menu);
    }
  } //onCreateContextMenu
```

Dodavanje stavki u meni:

```
  private void populateMyFirstMenu(Menu menu){
    int groupId= 0; int order= 0;
    //arguments: groupId, optionId, order, title
    menu.add(groupId, 1, 1, "10 points");
    menu.add(groupId, 2, 2, "20 points");
    menu.add(groupId, 3, 3, "30 points");
    menu.add(groupId, 4, 4, "40 points");
    menu.add(groupId, 5, 5, "50 points");
    menu.add(groupId, 6, 8, "Red text");
    menu.add(groupId, 7, 7, "Green Text");
    menu.add(groupId, 8, 6, "Blue text");
  } //populateMyMenu

  private void populateMySecondMenu(Menu menu){
    int groupId= 0; int order= 0;
    //arguments: groupId, optionId, order, title
    menu.add(groupId, 9, 1, "Bold");
    menu.add(groupId, 10, 2, "Italic");
    menu.add(groupId, 11, 3, "Normal");
  }//populateMySecondMenu
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Reakcija na izbor stavke iz menija se obavlja korišćenjem Event observer-a koji vraćaju *boolean* i to vraćaju *true* ako je event obrađen i *false* ako se event prosleđuje dalje:

```
  // called whenever an item in your context menu is selected
  @Override
  public boolean onContextItemSelected(MenuItem item) {
    return(applyMenuOption(item) ||
           super.onContextItemSelected(item) );
  }

  // called whenever an item in your options menu is selected
  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
    return(applyMenuOption(item) ||
           super.onOptionsItemSelected(item) );
  }
```

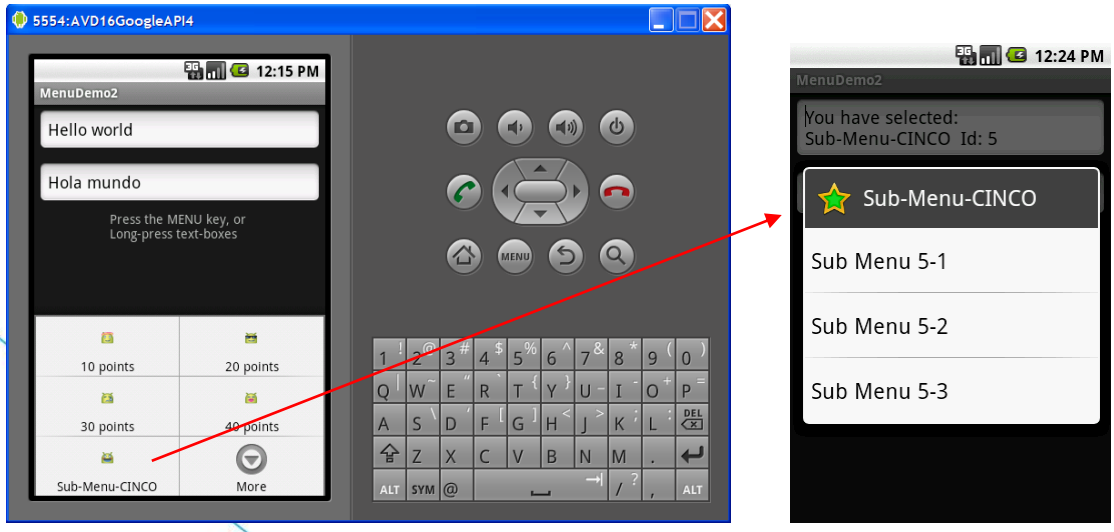
```
  // apply the action associated to selected item
  private boolean applyMenuOption(MenuItem item){
    int menuItemId = item.getItemId(); // 1, 2, 3, ...11
    String strMsg2 = etMessage2.getText().toString();
    if(menuItemId<= 5) {
      // first five option are for setting text size
      int newPointSize= arrayPointSize[menuItemId-1];
      etMessage1.setTextSize(newPointSize);
      etMessage2.setTextSize(newPointSize);
    }else{
      // either change color on box text1 or style on text2
      if(menuItemId== 6){
        etMessage1.setTextColor(color.background_dark| Color.RED);
        etMessage1.setTextColor(0xffff0000); // red
      } else if(menuItemId== 7)
        etMessage1.setTextColor(0xff00ff00); // green
      else if(menuItemId== 8)
        etMessage1.setTextColor(0xff0000ff); // blue
      else if(menuItemId== 9)
        etMessage2.setText(beautify(strMsg2, "BOLD")); //bold
      else if(menuItemId== 10)
        etMessage2.setText(beautify(strMsg2, "ITALIC")); //italic
      else if(menuItemId== 11)
        etMessage2.setText(beautify(strMsg2, "NORMAL")); //normal
    }
    return false;
  } //applyMenuOption
```

```
  // changing text style using HTML formatting
  // Spanned is text to which you could add formatting features

  private Spanned beautify (String originalText, String selectedStyle){
    Spanned answer = null;
    if(selectedStyle.equals("BOLD"))
      answer = Html.fromHtml("<b>"+ originalText+"</b");
    else if(selectedStyle.equals("ITALIC"))
      answer = Html.fromHtml("<i>"+ originalText+"</i>");
    else if(selectedStyle.equals("NORMAL"))
      answer = Html.fromHtml("<normal>"+ originalText+"</normal");
    return answer;
  } //beautify
} //Menu1Act1
```

Proširenje primera uvodi podmeni, ikone i shortcut-ove (slika 3.3). Treba dodati ikone u resurse (za svaku gustinu piksela). Izmene u meniju se rade u metodi populateMyFirstMenu(). Izbacujemo originalnu stavku 5, dodajemo ikone i shortcut-ove.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




Slika 3.3 Podmeni

```
private void populateMyFirstMenu(Menu menu){
    int groupId = 0;
    //arguments: groupId, optionId, order, title
    MenuItem item1 = menu.add(groupId, 1, 1, "10 points");
    MenuItem item2 = menu.add(groupId, 2, 2, "20 points");
    MenuItem item3 = menu.add(groupId, 3, 3, "30 points");
    MenuItem item4 = menu.add(groupId, 4, 4, "40 points");
    //MenuItem item5 = menu.add(groupId, 5, 5, "50 points");

    MenuItem item6 = menu.add(groupId, 6, 8, "Red text");
    MenuItem item7 = menu.add(groupId, 7, 7, "Green Text");
    MenuItem item8 = menu.add(groupId, 8, 6, "Blue text");

    //set icons
    item1.setIcon(R.drawable.uno);
    item2.setIcon(R.drawable.dos);
    item3.setIcon(R.drawable.tres);
    item4.setIcon(R.drawable.cuatro);

    // shortcuts using device's keyboard-keypad
    // on a G1 open slide open the keyboard and
    // type letter u (same as pressing menu UNO)
    item1.setShortcut('1', '1');
    item2.setShortcut('2', '2');
    item3.setShortcut('3', '3');
    item4.setShortcut('4', '4');
```

Remove this line from previous version

Figures used in this example were taken from:
C:\android-sdk-windows\platforms\android-4\data\res\drawable

Neophodno je i posebno obraditi podmeni:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
private boolean applyMenuOption(MenuItem item){

    int menuItemId = item.getItemId(); //1, 2, 3, ...11

    String strMsg2 = etMessage2.getText().toString();

    if (menuItemId < 5) {
        // first four options are for setting text size
        int newPointSize = arrayPointSize[menuItemId - 1];
        etMessage1.setTextSize(newPointSize);
        etMessage2.setTextSize(newPointSize);
    }
    else if (menuItemId == 5) {
        // the sub-menu (attached to 5th item) is processed here
        etMessage1.setText (
                    "You have selected: \n" +item.getTitle()
                  + "\nId: " + menuItemId
                  + " order: " + item.getOrder() );
}
    // either change color on text1 or style on text2
    else if (menuItemId == 6)
        etMessage1.setTextColor(0xffff0000); // red
```

Same as before

Take care of sub-menu here

Izmene u metodi populateFirstMenu

```
    // adding a sub-menu as fifth entry of this menu
    // .addSubMenu(int groupId, int itemId, int order, CharSequence title)
    int smGroupId = 0; // don't care, same as Menu.NONE
    int smItemId = 5;  // fifth element
    int smOrder = 5;   // don't care, same as Menu.NONE

    SubMenu mySubMenu5 =  menu.addSubMenu(smGroupId, smItemId, smOrder, "Sub-Menu-CINCO");
    mySubMenu5.setHeaderIcon(R.drawable.btn_rating_star_on_pressed);
    mySubMenu5.setIcon(R.drawable.cinco);
    // .add(int groupId, int itemId, int order, CharSequence title)

    MenuItem sub51 = mySubMenu5.add(smGroupId,5,1,"Sub Menu 5-1");
    MenuItem sub52 = mySubMenu5.add(smGroupId,5,2,"Sub Menu 5-2");
    MenuItem sub53 = mySubMenu5.add(smGroupId,5,3,"Sub Menu 5-3");


} //populateMyFirstMenu
```

### 3.2 Dialog box

Android podržava dva tipa primitivnih dialog box-ova:

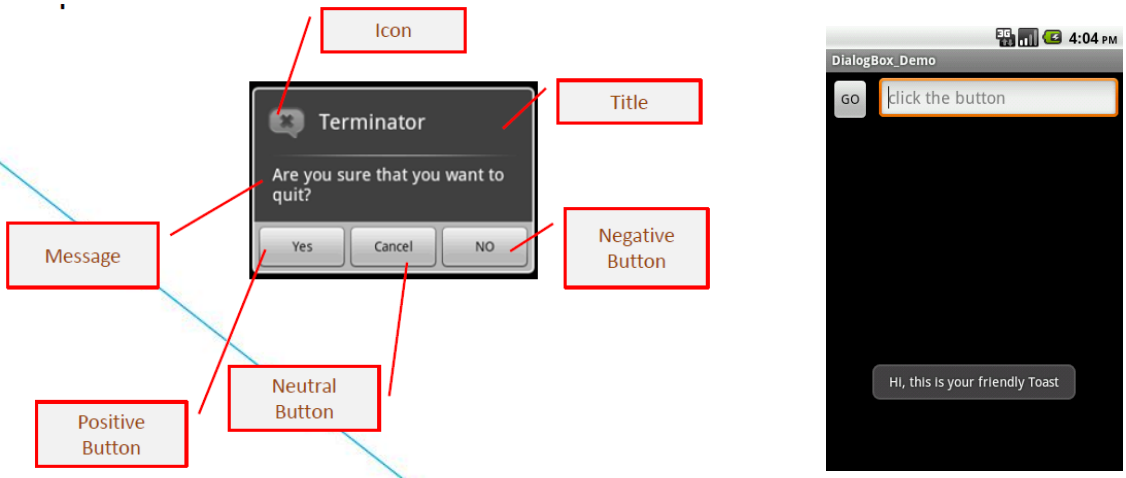
- AlertDialog
- Toast

AlertDialog je skoro modalan i prikazuje kratku poruku koja delimično zaklanja pozadinu. Prihvata kratak odgovor – kao klik na jedno od ponuđenih dugmića. AlertDialog je modalan zato što je



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

neophodna intervencija korisnika da bi se zatvorio (ne zatvara se automatski na timeout). Nije tipičan modalan dijalog zato što ne zaustavlja glavni thread - ostatak koda posle poziva za prikaz dijaloga normalno nastavlja da se izvršava i pre nego korisnik zatvori dijalog. Primer Alert dijaloga je prikazan na slici 3.4.




Slika 3.4 Primer Alert i Toast dijaloga

Programski kod za kreiranje Alert Dialoga sa slike 3.4:

```
...
import android.content.DialogInterface;
import android.os.Bundle;
...

public class AndDemoUI1 extends Activity {
  Button btnGo;
  EditText txtMsg;
  String msg;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    txtMsg= (EditText)findViewById(R.id.txtMsg);
    btnGo= (Button) findViewById(R.id.btnGo);
    btnGo.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View arg0) {
        AlertDialog dialBox= createDialogBox();
        dialBox.show();
        txtMsg.setText("I am here!");
      }
    });
  }//onCreate
```

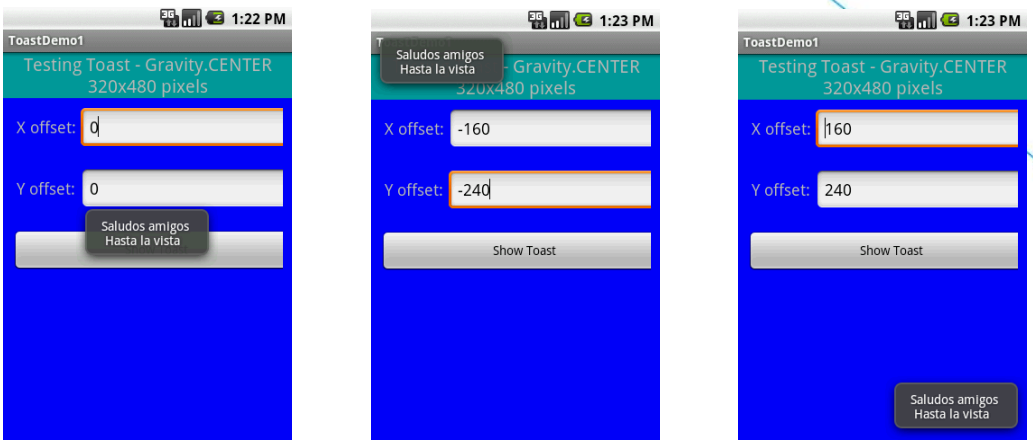
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
  private AlertDialog createDialogBox(){
    AlertDialog myQuittingDialogBox = new AlertDialog.Builder(this)
      //set message, title, and icon
    .setTitle("Terminator")
    .setMessage("Are you sure that you want to quit?")
    .setIcon(R.drawable.ic_menu_end_conversation)
    //set three option buttons
    .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int whichButton) {
        //whatever should be done when answering "YES" goes here
        msg= "YES "+ Integer.toString(whichButton);
        txtMsg.setText(msg);
      }
    })//setPositiveButton
    .setNeutralButton("Cancel",newDialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int whichButton) {
        //whatever should be done when answering "CANCEL" goes here
        msg= "CANCEL "+ Integer.toString(whichButton);
        txtMsg.setText(msg);
      }//OnClick
    })//setNeutralButton

    .setNegativeButton("NO", new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int whichButton) {
        //whatever should be done when answering "NO" goes here
        msg= "NO "+ Integer.toString(whichButton);
        txtMsg.setText(msg);
      }
    })//setNegativeButton
    .create();
    .return myQuittingDialogBox;
  }// createDialogBox
}// class
```

Toast je privremen box sa porukom. Automatski se gasi posle timeout-a bez neophodne korisnikove akcije. Prikazuju obično kratak tekst i nikada ne dobijaju fokus. Izgledaju kao da lebde iznad aplikacije. Primer Toast dijaloga je dat na slici 3.4, a programski kod za kreiranje toast-a je:

Toast.makeText( context, message, duration ).show();

Primeri pozicioniranja toast-a dati su na slici 3.5.




Slika 3.5 Različite pozicije Toast dijaloga

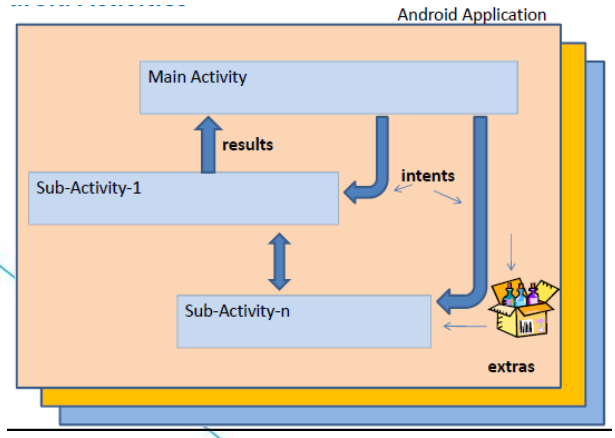


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 3.3 Intent

Tipična Android aplikacija ima više aktivnosti. Aktivnost sa setContentView() prikazuje UI.Aktivnosti su nezavisne ali mogu razmenjivati podatke. Tipično je jedna aktivnost glavna (main) i ona se startuje po startovanju aplikacije. Aktiviranje drugih aktivnosti aplikacije se obavlja korišćenjem intent-a. Aktivnosti međusobno komuniciraju u asinhronom režimu.

Namena Intent-a je ilustrovana na slici 3.6.



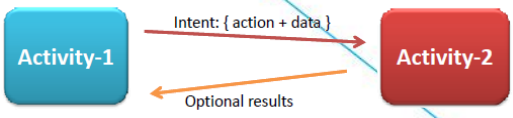

Slika 3.6 Ilustracija primene Intent-a

Intent-i se aktiviraju pozivima:

- startActivity(intent) - Startuje drugu aktivnost
- sendBroadcast(intent) - Šalje intent svim zainteresovanim BradcastReceiver komponentama
- startService(intent) i bindService(intent) – Koriste za komunikaciju sa servisima u pozadini

Argumenti Intent-a su:

- Action – predefinisane: ACTION_VIEW, ACTION_EDIT, ACTION_MAIN...
- Data – podaci sa kojima se radi




Primeri action/data parova su:

- ACTION_VIEW content://contacts/people/1
- ACTION_DIAL content://contacts/people/1
- ACTION_VIEW tel:123
- ACTION_DIAL tel:123
- ACTION_EDIT content://contacts/people/1
- ACTION_VIEW content://contacts/people/

Intent se tipično šalje na sledeći način:

```
Intent myActivity = new Intent(action, data);
startActivity(myActivity);
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Akcija može biti ugrađena ili korisnički definisana , dok su podaci vezani za akciju:

```
Intent myActivity = new Intent(Intent.ACTION_DIAL,
                                Uri.parse("tel:555-1234"));
startActivity(myActivity);
```

Prilikom kreiranja intent-a mogu se zadati i sekundarni atributi koji se mogu uključiti u Intent:

- Category
- Type
- Components
- Extras

```
Intent intent =
        new Intent(Intent.ACTION_WEB_SEARCH);
intent.putExtra(SearchManager.QUERY,
                "straight hitting golf club");
startActivity(intent);
```

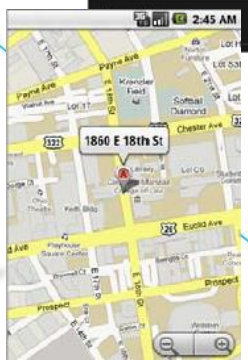
```
Intent intent =  new Intent(Intent.ACTION_SENDTO,
                        Uri.parse("sms:555-1234"));
intent.putExtra("sms_body",
                "are we playing golf next Saturday?");
startActivity(intent);
```

Startovanje aktivnosti za prikaz svih kontakata korišćenjem Intent-a:

```
String myData = "content://contacts/people/";
Intent myActivity = new Intent(Intent.ACTION_VIEW,
                        Uri.parse(myData));
startActivity(myActivity);
```

Izmena konkretnog kontakta:

```
String myData = "content://contacts/people/8";
Intent myActivity = new Intent(Intent.ACTION_EDIT,
                        Uri.parse(myData));
startActivity(myActivity);
```




Prikaz adrese u GoogleMaps:

```
String myData = "geo:0,0?1860+east+18th+street+clevelend+oh";
Intent myActivity = new Intent(Intent.ACTION_VIEW,
                        Uri.parse(myData));
startActivity(myActivity);
```

U ovom slučaju obavezan dodatk u Manifest je:

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.INTERNET" />
```

Puštanje zvučnog fajla



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
Intent myActivity2 = new Intent(android.content.Intent.ACTION_VIEW);
Uri data = Uri.parse("file:///sdcard/amarcord.mp3");
String type = "audio/mp3";
myActivity2.setDataAndType(data, type);
startActivity(myActivity2);
```

Slanje slike MMS porukom

```
Uri uri = Uri.parse("content://media/external/images/media/1");
Intent myActivity2 = new Intent(Intent.ACTION_SEND);
myActivity2.putExtra("address", "555-1234");
myActivity2.putExtra("sms_body", "some text message goes here");
myActivity2.putExtra(Intent.EXTRA_STREAM, uri);
myActivity2.setType("image/png");
startActivity(myActivity2);
```

Slanje Email poruke:

```
Uri uri = Uri.parse("mailto:v.matos@csuohio.edu");
Intent myActivity2 = new Intent(Intent.ACTION_SEND);
myActivity2.putExtra(Intent.EXTRA_SUBJECT,"subject goes here");
myActivity2.putExtra(Intent.EXTRA_TEXT,"The email's body goes here");
startActivity(myActivity2);
```

Mehanizam Intent-a treba da obezbezdi dobijanje povratne informacija od aktivnosti koja je startovana, tačnije kada se startuje neka aktivnost da bi vratila rezultat kada se završi. Primer: ako startujemo aktivnost za prikaz kontakata želimo da dobijemo kao povratnu vrednost kontakt koji je korisnik izabrao.Poseban poziv za aktiviranje takve aktivnosti je:

startActivityForResult(Intent, requestCodeID)

requestCodeID – je jedinsteveni identifikator poziva

Rezultat se “hvata” listener-om:

onActivityResult(int requestCodeID int resultCode, Intent data)

Pre nego što podaktivnost izađe može roditeljskoj aktivnosti da pošalje signal

setResult(resultCode)

resultCode – može biti neki od standardnih ili custom:

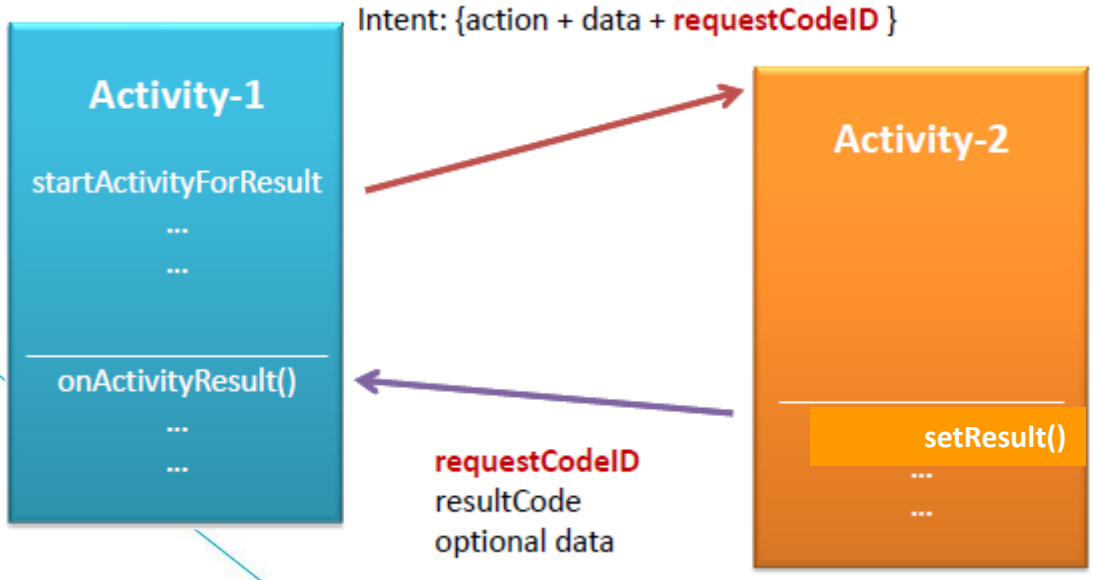
- Activity.RESULT_CANCELED
- Activity.RESULT_OK
- Custom

Roditeljska aktivnost rezultat hvata sa onActivityResult. Ilsutracija ovog mehanizma data je na slici 3.7.

U slučaju greške vraća se RESULT_CANCELED



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

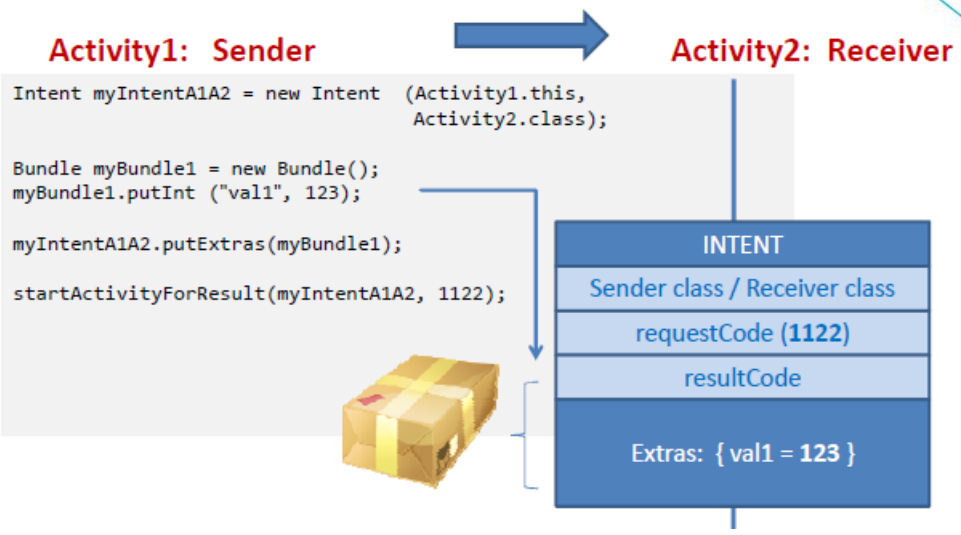



Slika 3.7 Startovanje aktivnosti intentom i dobijanje povratnog rezultata

Komunikacija između aktivnosti je asinhrona, koriste se listener-i (onActivityResult(...)). Intent opciono može da prenosi između aktivnosti imenovanu listu podataka (bundle). Bundle je kolekcija <type,value> parova sa odgovarajućim *get* i *set* metodama:

```
Bundle myBundle = new Bundle();
myBundle.putDouble ("var1", 3.1415);
...
Double v1 = myBundle.getDouble("var1");
```

Intent-i i bundle obezbeđuju Prenos podataka između aktivnosti (slika 3.8).



ICT CLUSTER ACADEMY

TVOJA KARIJERA U TVOJIM RUKAMA

Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA

Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

ICT CLUSTER ACADEMY

TVOJA KARIJERA U TVOJIM RUKAMA

Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA

Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

ICT CLUSTER ACADEMY

TVOJA KARIJERA U TVOJIM RUKAMA

Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA

Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
    MyPlace place = (MyPlace) getListAdapter().getItem(position);
    Toast.makeText(this, place.getName() + " selected", Toast.LENGTH_LONG).show();
}
```

4. Kreiranje EditMyPlaceActivity-a:
   a. Kreirati novi layout xml u resursima (editmyplace.xml). U layoutu definisati dva para kontrola TextView i EditText koje će služiti za unos imena i opisa novounete omiljene lokacije. U strings.xml dodati odgovarajuće nazive. Za dugme za dodavanje novog mesta nemojte definisati natpis, njega ćemo definisati programski u zavisnosti da li Activity koristimo za dodavanje nove lokacije ili za editovanje postojeće.

```
    It will be able to show your places on map and enable you to share the
    and you will collect <i>bonus stars</i> from your teaching assistant.
    </string>
    <string name="list_title">My places</string>
    <string name="editmyplace_name_label">Name:</string>
    <string name="editmyplace_desc_label">Description:</string>
    <string name="editmyplace_cancel_label">Cancel</string>
</resources>
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
     <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/editmyplace_name_label"
        />
     <EditText
        android:id="@+id/editmyplace_name_edit"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
         />
     <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/editmyplace_desc_label"
        />
     <EditText
        android:id="@+id/editmyplace_desc_edit"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
         />
     <TableRow
         android:layout_width="fill_parent"
         android:layout_height="wrap_content" >
         <Button
             android:id="@+id/editmyplace_finished_button"
             android:layout_weight="1" />
         <Button
             android:id="@+id/editmyplace_cancel_button"
             android:layout_weight="1"
             android:text="@string/editmyplace_cancel_label" />
     </TableRow>
</LinearLayout>
```

   b. Prva verzija ovog Activity-a će biti namenjena dodavanju nove lokacije. U kodu postaviti click listener-e za oba dugmeta, postaviti tekst “Add” finish dugmetu. Ukoliko je kliknuto na finish dugme, preuzmite stringove iz EditText View-a i na odnovu njih kreirajte novu lokaciju.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```java
package org.elrak.mobis.myplaces;

import android.app.Activity;

public class EditMyPlaceActivity extends Activity implements OnClickListener {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.editmyplace);
        Button finishedButton = (Button)findViewById(R.id.editmyplace_finished_button);
        finishedButton.setText("Add");
        finishedButton.setOnClickListener(this);
        Button cancelButton = (Button)findViewById(R.id.editmyplace_cancel_button);
        cancelButton.setOnClickListener(this);
    }
    @Override
    public void onClick(View v) {
        switch(v.getId())
        {
            case R.id.editmyplace_finished_button:
            {
                TextView twName = (TextView)findViewById(R.id.editmyplace_name_edit);
                String nme = twName.getText().toString();
                TextView twDesc = (TextView)findViewById(R.id.editmyplace_desc_edit);
                String desc = twDesc.getText().toString();
                MyPlace place = new MyPlace(nme, desc);
                MyPlacesData.getInstance().addNewPlace(place);
                setResult(Activity.RESULT_OK);
                finish();
                break;
            }
            case R.id.editmyplace_cancel_button:
            {
                setResult(Activity.RESULT_CANCELED);
                finish();
                break;
            }
        }
    }

}
```

c. Možete uočiti da se nakon klika na neko od dugmadi Activity završava. Dobro je vratiti rezultat u takvim situacijama jer će nam kasnije takva informacija može biti značajna. U tom slučaju u potivajućem Activity-ju (MyPlacesActivity.java) moramo da koristimo metod startActivityForResult();

d. Ukoliko želimo da obrađujemo vraćeni rezultat treba override-ovati i metod onActivityResult() u pozivajućem Activity-ju.

```java
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    super.onActivityResult(requestCode, resultCode, data);
    try
    {
        if (resultCode == Activity.RESULT_OK)
        {
            Toast.makeText(this, "New place added", Toast.LENGTH_LONG).show();
        }
    }
    catch (Exception e) {
        // TODO: handle exception
    }
}
```

e. Probajte aplikaciju.

f. Trenutno je moguće dodati novu lokaciju čak i ako su oba polja za unos prazna. Da bi ovo rešili moramo da onemogućimo korisniku klik na dugme za dodavanje ukoliko nikakav tekst nije dodat u polje za ime lokacije.

   i. Postavite u OnCreate metodi Activity-a da dugme finished bude neaktivno



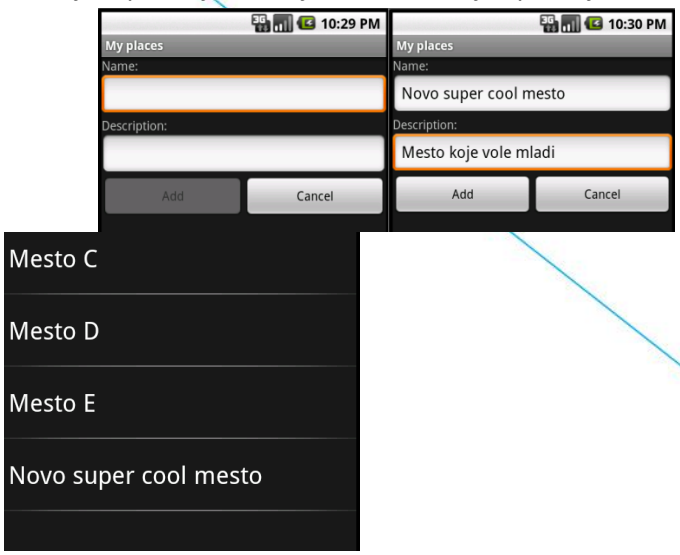
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
Button finishedButton = (Button)findViewById(R.i
finishedButton.setEnabled(false);
finishedButton.setText("Add");
```

ii. Implementirati TextWatcher interfejs i pratiti promene teksta u EditText polju za ime. Ukoliko postoji neki tekst postaviti da dugme bude aktivno.

```
cancelButton.setOnClickListener(this);
EditText etName = (EditText)findViewById(R.id.editmyplace_name_edit);
etName.addTextChangedListener(this);


@Override
public void afterTextChanged(Editable s) {
    EditText etName = (EditText)findViewById(R.id.editmyplace_name_edit);
    Button finishedButton = (Button)findViewById(R.id.editmyplace_finished_button);
    if(etName.getText().toString().length()>0)
        finishedButton.setEnabled(true);
    else
        finishedButton.setEnabled(false);

}
@Override
public void beforeTextChanged(CharSequence s, int start, int count,
        int after) {
    // TODO Auto-generated method stub

}
@Override
public void onTextChanged(CharSequence s, int start, int before, int count) {
    // TODO Auto-generated method stub

}
```

g. Probajte aplikaciju. Dodajte novu lokaciju i probajte listu.




5. Kreiranje ViewMyPlaceActivity-a.

a. Sa postojećim znanjem ovo možemo lako napraviti tako da neće biti detaljno opisivano kreiranje layout-a.

```
<string name="editmyplace_cancel_label">Cancel</string>
<string name="viewmyplace_finised_label">OK</string>
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
     <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/editmyplace_name_label"
        />
     <TextView
        android:id="@+id/viewmyplace_name_text"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
         />
     <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/editmyplace_desc_label"
        />
     <TextView
        android:id="@+id/viewmyplace_desc_text"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
         />
     <Button
        android:id="@+id/viewmyplace_finished_button"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/viewmyplace_finised_label"
         />
</LinearLayout>
```

b. Klik na neku od stavki iz liste treba da prikaže izabranu lokaciju u ovom Activiy-ju. Za to je potrebno proslediti podatak o poziciji kliknutog elementa kroz Bundle objekat koji dodeljujemo Intent-u prilikom poziva.

c. Taj podatak u metodi OnCreate ViewMyPlaceActivity-ja preuzimamo i koristimo za preuzimanje odgovarajuće lokacije iz klase sa podacima.

d. Sada MyPlaceListActivity izgleda ovako:

```
import android.app.ListActivity;

public class MyPlacesList extends ListActivity{

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setListAdapter(new ArrayAdapter<MyPlace>(this, android.R.layout.simple_list_item_1, MyPlacesData.getInstance().getMyPlaces()));
    }

    @Override
    protected void onListItemClick(ListView l, View v, int position, long id) {
        Bundle positionBundle = new Bundle();
        positionBundle.putInt("position", position);
        Intent i = new Intent(this, ViewMyPlaceActivity.class);
        i.putExtras(positionBundle);
        startActivity(i);
    }

}
```

e. Obratite pažnju na poziv Intent-a.

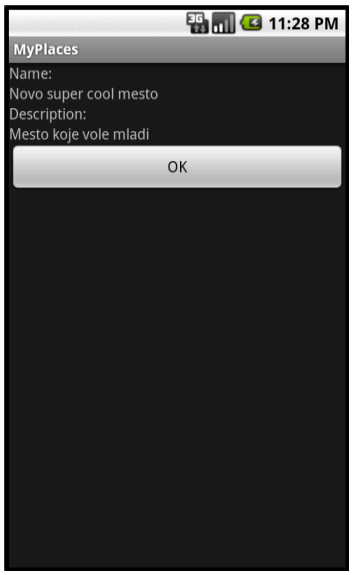
f. ViewMyPlaceActivity izgleda ovako:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```java
package org.elfak.mods.myplaces;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class ViewMyPlaceActivity extends Activity implements OnClickListener{
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.viewmyplace);
        int position = -1;
        try
        {
            Intent listIntent = getIntent();
            Bundle positionBundle = listIntent.getExtras();
            position = positionBundle.getInt("position");
        }
        catch(Exception e) {
            Toast.makeText(this, e.getMessage(), Toast.LENGTH_LONG).show();
            finish();
        }
        if(position >=0)
        {
            MyPlace place = MyPlacesData.getInstance().getPlace(position);
            TextView twName = (TextView)findViewById(R.id.viewmyplace_name_text);
            twName.setText(place.getName());
            TextView twDesc = (TextView)findViewById(R.id.viewmyplace_desc_text);
            twDesc.setText(place.getDesc());
        }
        Button finishedButton = (Button)findViewById(R.id.viewmyplace_finished_button);
        finishedButton.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        finish();
    }
}
```

g. Probajte aplikaciju.




6. Dodavanje Options menija u MyPlacesList Activity.
    a. Zgodno bi bilo i omogućiti dodavanje nove lokacije iz same liste lokacija. Za ovo možemo iskoristiti Options meni.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

b. U Activity-ju MyPlacesListActivity override-ovati metode onCreateOptionsMenu() i onOptionsItemSelected() metode.
c. U onCreateOptionsMenu() metodi dodati MenuItem sa tekstom “Add new place” a u onOptionsItemSelected() metodi dodati poziv EditMyPlaceAcivity-ja kako je to bilo urađeno ranije u MyPlacesActivity-ju.
d. Override-ovati i metod onActivityResult tako da se, ukoliko je izvršeno dodavanje novog place-a, osveži lista pozivom setListAdapter.

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
    menu.add(0, 1, 1, "Add new place");
    return super.onCreateOptionsMenu(menu);
}

@Override public boolean onOptionsItemSelected(MenuItem item)
{
    Intent i = new Intent(this, EditMyPlaceActivity.class);
    startActivityForResult(i, 1);
    return super.onOptionsItemSelected(item);
}
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    super.onActivityResult(requestCode, resultCode, data);
    try
    {
        if (resultCode == Activity.RESULT_OK)
        {
            setListAdapter(new ArrayAdapter<MyPlace>(this, android.R.layout.simple_list_item_1, MyPlacesData.getInstance().getMyPlaces()));
        }
    }
    catch (Exception e) {
        // TODO: handle exception
    }
}
```

e. Probati aplikaciju.

7. Omogućavanje editovanja postojeće lokacije korišćenjem EditMyPlaceActivity-ja i konteksnog menija.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

a. Registrovati konteksni meni listi. Pošto svaki ListActivity podarazumevano sadrži ListView, umestu korišćenjem resursa, možemo mu pristupiti korišćenjem metoda getListView().

```
public class MyPlacesList extends ListActivity{

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setListAdapter(new ArrayAdapter<MyPlace>(this, android.R.layout.simple_list_item_1, MyPlacesData.getInstance().getMyPlaces()));
        ListView lv = this.getListView();
        registerForContextMenu(lv);
    }
```

b. Dati naslov konteksnom meniju i dati dodati dve stavke, “View place” i “Edit place”.

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo)
{
    super.onCreateContextMenu(menu, v, menuInfo);
    AdapterView.AdapterContextMenuInfo info = (AdapterView.AdapterContextMenuInfo)menuInfo;
    menu.setHeaderTitle( MyPlacesData.getInstance().getPlace(info.position).getName());
    menu.add(0, 1, 1, "View place");
    menu.add(0, 2, 2, "Edit place");
}
```

c. Obratiti pažnju kako se preuzima pozicija kliknutog elementa liste.

d. Obraditi klik kreiranjem odgovarajućih Intent-a.

```
@Override
public boolean onContextItemSelected(MenuItem item)
{
    AdapterView.AdapterContextMenuInfo info = (AdapterView.AdapterContextMenuInfo)item.getMenuInfo();
    Bundle positionBundle = new Bundle();
    positionBundle.putInt("position", info.position);
    Intent i = null;
    if(item.getItemId() == 1)
    {
        i = new Intent(this, ViewMyPlaceActivity.class);
        i.putExtras(positionBundle);
        startActivity(i);
    }
    else
    {
        i = new Intent(this, EditMyPlaceActivity.class);
        i.putExtras(positionBundle);
        startActivityForResult(i, 1);
    }
    return super.onContextItemSelected(item);
}
```

e. Ostalo je još da u klasi EditMyPlaceActivity podržimo prosleđivanje parametra i u skladu sa tim omogućimo editovanje i snimanje. To možemo da omogućimo proverom da li je u Bundle objektu pozivajućeg Intent-a prosleđena pozicija ili nije. Sledi novi izgled klase EditMyPlaceActivity.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```java
package org.elfak.modis.myplaces;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
import android.widget.EditText;
import android.widget.Toast;

public class EditMyPlaceActivity extends Activity implements OnClickListener, TextWatcher {
    boolean editMode = true;
    int position = -1;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.editmyplace);
        try
        {
            Intent listIntent = getIntent();
            Bundle positionBundle = listIntent.getExtras();
            if(positionBundle != null)
                position = positionBundle.getInt("position");
            else
                editMode = false;
        }
        catch(Exception e) {
            editMode = false;
        }
        Button finishedButton = (Button)findViewById(R.id.editmyplace_finished_button);
        Button cancelButton = (Button)findViewById(R.id.editmyplace_cancel_button);
        EditText etName = (EditText)findViewById(R.id.editmyplace_name_edit);
        if(!editMode)
        {
            finishedButton.setEnabled(false);
            finishedButton.setText("Add");
        }
        else
        {
            if(position >=0)
            {
                finishedButton.setText("Save");
                MyPlace place = MyPlacesData.getInstance().getPlace(position);
                etName.setText(place.getName());
                EditText etDesc = (EditText)findViewById(R.id.editmyplace_desc_edit);
                etDesc.setText(place.getDesc());
            }
        }
```

Nastavak je na sledećoj strani.



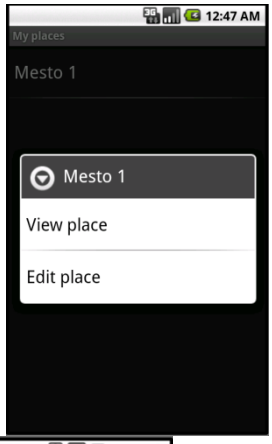
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

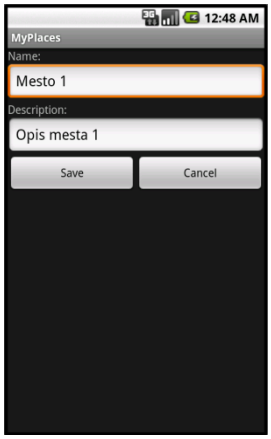
```
        finishedButton.setOnClickListener(this);
        cancelButton.setOnClickListener(this);
        etName.addTextChangedListener(this);
    }
    @Override
    public void onClick(View v) {
        switch(v.getId())
        {
            case R.id.editmyplace_finished_button:
            {
                EditText etName = (EditText)findViewById(R.id.editmyplace_name_edit);
                String nme = etName.getText().toString();
                EditText etDesc = (EditText)findViewById(R.id.editmyplace_desc_edit);
                String desc = etDesc.getText().toString();
                if(!editMode)
                {
                    MyPlace place = new MyPlace(nme, desc);
                    MyPlacesData.getInstance().addNewPlace(place);
                }
                else
                {
                    MyPlace place = MyPlacesData.getInstance().getPlace(position);
                    place.setName(nme);
                    place.setDesc(desc);
                }
                setResult(Activity.RESULT_OK);
                finish();
                break;
            }

            case R.id.editmyplace_cancel_button:
            {
                setResult(Activity.RESULT_CANCELED);
                finish();
                break;
            }
        }
    }
    @Override
    public void afterTextChanged(Editable s) {
        EditText etName = (EditText)findViewById(R.id.editmyplace_name_edit);
        Button finishedButton = (Button)findViewById(R.id.editmyplace_finished_button);
        if(etName.getText().toString().length()>0)
            finishedButton.setEnabled(true);
        else
            finishedButton.setEnabled(false);

    }
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count,
            int after) {

        // TODO Auto-generated method stub

    }
    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
        // TODO Auto-generated method stub

    }

}
```

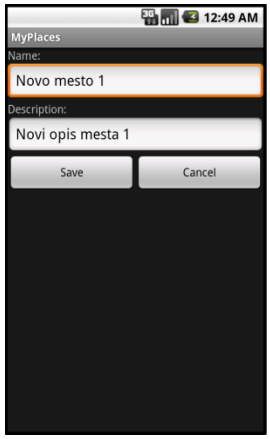
8. Obrisati statičko dodavanje lokacija iz konstruktora klase MyPlacesData i probati aplikaciju (Slike na sledećoj strani).

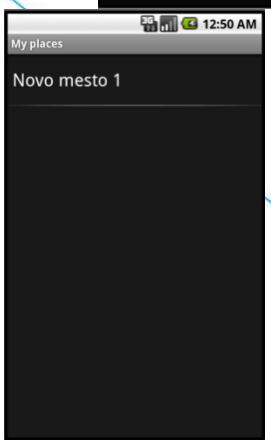


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




9. Za domaći zadatak ulepšati korisnički interfejs aplikacije.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 4. Mobilne baze podataka (SQLite) i višenitnost mobilnih aplikacija

## 4.1 Mobilne baze podataka

Baze podataka su osnovna komponenta *enterprise* aplikacija i aplikacija e-poslovanja. Sa razvojem aplikacija m-poslovanja, baze podataka postaju neizostavne i na mobilnim aplikacijama (pametnim klijentima) i predstavljaju način da enterprise podaci izađu iz okvira kompanije. Postavlja se pitanje: Perzistentni podaci na mobilnom telefonu ili pristup preko bežične mreže?

- Nedostupna ili nepouzdana bežična mreža,
- Nedovoljan propusni opseg,
- Podaci su uglavnom statički,
- Trajanje baterije je ograničeno.

Hibridno rešenje je koristiti lokalno skladište podataka za pristup podacima neophodnim u mobilnoj aplikaciji, putem bežičnog Interneta pristupati podacima koji su dinamički ili zavisni od drugih aplikacija.

Android fajl sistem čine 4 osnovna pristupa za smeštanje i upravljanje podacima:

- Shared preferences (deljena podešavanja) - Uglavnom služe za smeštanje UI podešavanja u obliku para ključ – vrednost.
- Fajlovi - Najlakši, ali ne i najbolji način za trajno čuvanje podataka
- SQLite baza podataka - Najlakši i najupotrebljiviji način za čuvanje struktuiranih podataka, svaka aplikacija može da kreira svoju bazu podataka nad kojom ima potpunu kontrolu.
- Provajderi sadržaja (Content Providers) - Omogućavaju pristup centralnim skladištima podataka u cilju deljenja podataka između više aplikacija i pristup ovim podacima predstavljenim u tabelarnom obliku.

Ključni elementi mobilnih enterprise aplikacija su sinhronizacija podataka i sigurnost. Mobilna aplikacija mora biti "osvežena" sa relevantnim i ažurnim podacima sa servera kompanije, a takođe i novi podaci sa mobilnog uređaja treba da budu preneti na server. Ograničenje u pristupu podacima je određeno tipom posla koji obavlja mobilni radnik, kao i politikom sigurnosti kompanije. Sinhronizacija podataka mora biti sigurna, pošto su podaci jedna od najvažnijih svojina kompanije.

Mobilne enterpise platforme

- Pega AMP (Application Mobility Platform) - Antenna (part of Pegasystems) - www.antennasoftware.com
- IBM MobileFirst Platform - www.ibm.com/mobilefirst/us/en/why-ibm-for-mobile/platform.html



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- IBM® Worklight® platform – http://www-03.ibm.com/software/products/en/worklight
- Kony KonyOne - www.kony.com/products
- SAP Mobile Platform - www.sap.com/mobileplatform
- Spring Mobile Solutions - www.springmobilesolutions.com

### 4.2 SQLite

Android (kao i iOS) koristi embedded, nezavistan program sqlite3 za rad sa lokalnim bazama podataka. SQLite DBMS implementira većinu SQL-92 standarda. Delimično podržava trigere i omogućava kompleksne upite. NE IMPLEMENTIRA referencijalni integritet kroz strane ključeve i koristi model slabih tipova. Tip podatka se dodeljuje ne koloni već svakoj ćeliji. Slično kao Variant tip u VB tako da možete upisati string u numeričku kolonu i sl. SQLite kao i većina DBMS-ova nije case sensitive

Ograničenja pristupa SQLite bazi:

- Ograničenja deljenja - Ne možete pristupati internim SQLite bazama drugih aplikacija (za to se koristi ContentProvider)
- Datoteka baze podataka koja je smeštena na SD kartici zahteva da Manifest uključuje

```
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Bilo koja aplikacija može pristupiti eksternoj datoteci SQLite baze koja je smeštena na SD kartici, samo je potrebno znati putanju do datoteke baze podataka. Međutim, za deljenje podataka između aplikacija preporučuje se ContentProvider mehanizam

Za rad sa SQLite bazom podataka koriste se sledeće metode:

openOrCreateDatabase u okviru Context klase:

```
SQLiteDatabase db = this.openOrCreateDatabase(
                                "myfriendsDB",
                                MODE_PRIVATE,
                                null);
```

Podrazumevani prefiks putanje do datoteke baze je: /data/data/<CURRENT_namespace>/databases/

Ovako kreiranu datoteku mogu da koriste druge aktivnosti ili se može eksportovati na desktop. Za rad sa bazom podataka (mogu se koristiti svi standardni SQLite alati.

MODE parametar: MODE_PRIVATE, MODE_WORLD_READABLE, MODE_WORLD_WRITEABLE

Definisana su dva tipa upita:

- Action query – kreiranje, brisanje, modifikovanje elemenata baze
- Retreival query – upiti za pribavljanje podataka



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Za izvršenje action query-a koristimo metodu: execSQL()

```
SQLiteDatabase db = this.openOrCreateDatabase(
                                "myfriendsDB",
                                MODE_PRIVATE,
                                null);
```

Retrieval query su praktično SELECT upiti. Rezultati ovih upita su uvek tabele. Rezultujuće tabele se obrađuju korišćenjem kursora (Cursor klasa). Kursor omogućava pristup podacima u rezultujućoj tabeli red-po-red.

Postoje dva načina izvršavanja SQL SELECT upita:

- Raw query – može da izvrši praktično bilo kakav SQL SELECT upit (outer join nije podržan).
- Simple query – jednostavan način zadavanja SQL SELECT parametrizovanih upita nad jednom tabelom bez detaljnog znanja SQL jezika – metod query.

Primer za RawQuery

```
Cursor c1 = db.rawQuery(
      "select count(*) as Total from tblAMIGO",
      null);
```

Upit prebrojava redove u tabeli tblAMIGO. Rezultat je tabela sa jednim redom i jednom kolonom. Kursor c1 se koristi za pristup rezultujućoj tabeli. Pribavljanje jednog reda iz rezultata korišćenjem kursora se obavlja prelaskom na sledeći red. Polje pribavljenog reda treba iskopirati u neku lokalnu promenljivu.

SQLite kursori se koriste za sekvencijalni pristup tabeli koja je rezultat retrieval query-a. Metode kursora su:

- Pozicija kursora: isFirst(), isLast(), isBeforeFirst(), isAfterLast()
- Navigacija po redovima: moveToFirst(), moveToLast(), moveToNext(), moveToPrevious(), move(n)
- Rad sa poljima reda: getInt(), getString(), getFloat(), getBlob(), getDate()...
- Rad sa šemom: getColumnName(), getColumnNames(), getColumnIndex(), getColumnCount(), getCount()

Kursori omogućavaju samo READ_ONLY pristup. Ranije verzije Android SDK su nudile metode kursora za modifikaciju podataka, ali takve metode, poput cursor.updateInt(...), cursor.deleteRow(...) su *deprecated* (zastarele). Preporuka je da se action komande izvršavaju metodom execSQL(...).

Postoje i metode za jednostavniju izmenu podataka koje sadrži tabela:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
public long insert(String table,
                   String nullColumnHack,
                   ContentValues values )
public int update(String table,
                   ContentValuesvalues,
                   String whereClause,
                   String[] whereArgs)
public int delete(String table,
                   String whereClause,
                   String[] whereArgs)
```

Drugi argument Insert naredbe je *nullable* kolona kada se ubacuje null values.

Android SDK poseduje *command line* alat za rad sa SQLite bazama. Prvo je potrebno povezati se na shell emulatora komandom adb shell, a zatim koristiti sqlite komande iz komandne linije.

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

E:\Android> adb shell

# sqlite3 /data/data/matos.sql1/databases/myfriendsDB

sqlite3 /data/data/matos.sql1/databases/myfriendsDB
SQLite version 3.5.9
Enter ".help" for instructions
```

```
sqlite> .tables
.tables
android_metadata  tblAMIGO

sqlite> select * from tblAMIGO;

1|AAAXXX|555
2|BBBXXX|777
3|Maria|999
4|Maria|000
5|Maria|001

sqlite> .exit
#
```

Sqlite3 alat podržava standardne DBMS komande:

- Prikaz tabela
- Izvršavanje upita
- Pomoći i lista komandi - .help

Mogu se koristiti i SQLite GUI alati. Datoteka SQLite baze podataka može da se prebaci na/sa emulatora standardnim komandama:

```
adb pull <full_path_to_database>
adb push <full_path_to_database>
```

Kada se SQLite datoteka baze prebaci na desktop jednostavnija je manipulacija nekim GUI alatom, poput:

- SQLite Administrator
- SQLite Database Browser (sada Database Browser for SQLite)



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 4.3 Višenitnost (Multithreading)

Na Android platformi svaka aplikacija se izvršava u svojoj virtuelnoj mašini (jednom procesu) i barem jednom thread-u. Proces može da startuje i dodatne thread-ove po potrebi. Na Android platformi thread-ovi mogu deliti objekte, a sinhronizuju se korišćenjem monitora koji su vezani za te objekte. Postoje dva načina da thread izvrši neki kod:

- Nova klasa nasledi Thread klasu i override-uje metodu run()
- Instancirati Thread klasu i konstruktoru proslediti Runnable objekat

U svakom slučaju se thread startuje pozivom start() metode.

Android niti (thread-ovi) imaju ista stanja kao i standardni Java thread-ovi. Glavni thread se obično naziva UI thread i ne sme dugo biti zauzet nekom operacijom kako bi UI imao odziv.Dva načina da se ovo obezbedi:

- Duge operacije mogu da se obavljaju u pozadinskom servisu
- Duge operacije mogu da se prebace u poseban thread

Postoji namenska Android klasa za obavljanje dugih operacija u posebnom thread-u: AsyncTask

### 4.4 Android Handler

Kada se kreira proces za aplikaciju u glavnom thread-u se izvršava message queue koji upravlja aktivnostima, intent receiver-ima itd. Svaki novi thread može da komunicira sa glavnim korišćenjem Handler-a. Kada se kreira Handler vezuje se za message queue thread-a koji ga je kreirao i isporučuje i preuzima poruke i runnables tom redu. Handler se koristi:

- Da se zakaže izvršenje poruke ili runnable u budućnosti
- Da se akcija izvrši u drugom thread-u

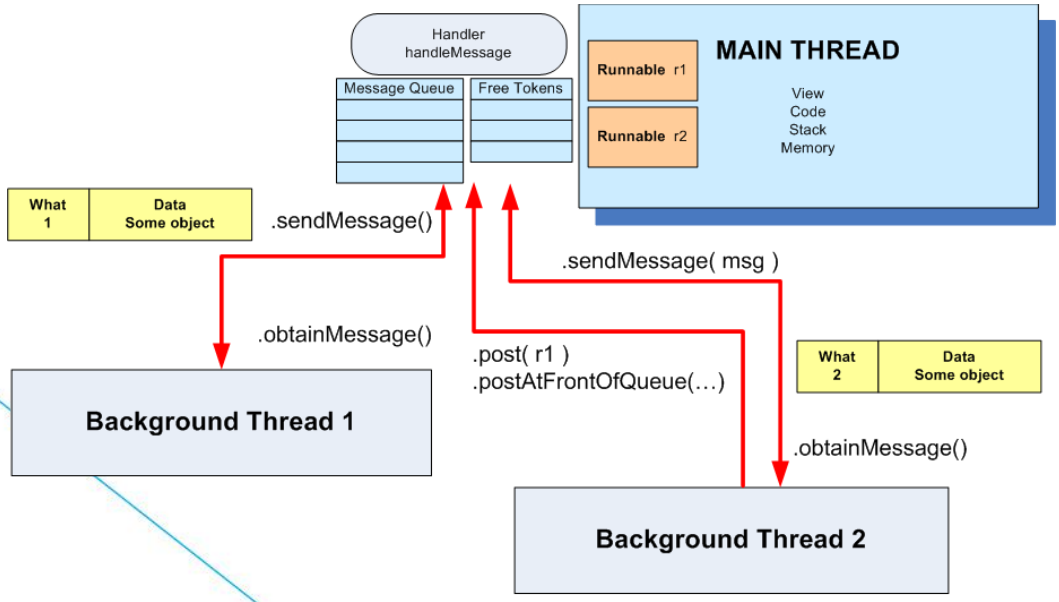
Sekundarni thread koji želi da komunicira sa glavnim prvo zahteva message token sa metodom:

obtainMessage()

Zatim može da popuni message token nekim podacima i da ga doda u message queue metodom:

sendMessage()

Handler u metodi handleMessage() u glavnom thread-u obrađuje poruke. Na ovaj način sekundarni thread može glavnom thread-u da prenese neke podatke ili da zatraži izvršenje runnable objekta metodom post() (slika 4.1)




Slika 4.1 Princip funkcionisanja Android Handler-a

### 4.5 AsyncTask

AsyncTask je čistija alternativa bez Handlera i poruka. AsyncTask predstavlja neku operaciju koja će se obaviti u pozadinskom thread-u i čiji će se rezultat predstaviti u UI thread-u. AsyncTask ima 4 koraka:

- onPreExecute
- doInBackground
- onProgressUpdate
- onPostExecute

AsyncTask se uvek nasleđuje i parametrizuje se sa Params, Progress i Result:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
private class VerySlowTask extends AsyncTask<String, Long, Void> {

    // Begin - can use UI thread here
    protected void onPreExecute() {

    }

    // this is the SLOW background thread taking care of heavy tasks
    // cannot directly change UI
    protected Void doInBackground(final String... args) {
    ... publishProgress((Long) someLongValue);
    }

    // periodic updates - it is OK to change UI
    @Override
    protected void onProgressUpdate(Long... value) {

    }

    // End - can use UI thread here
    protected void onPostExecute(final Void unused) {

    }
}
```

AsyncTask obezbeđuje "čistiju" komunikaciju sa UI thread-om i widget-ima koja ne zahteva korišćenje Handler-a i thread-ova.

| 3 Generic Types | 4 Main States | 1 Auxiliary Method |
|---|---|---|
| Params, Progress, Result | onPreExecute, doInBackground, onProgressUpdate onPostExecute. | publishProgress |

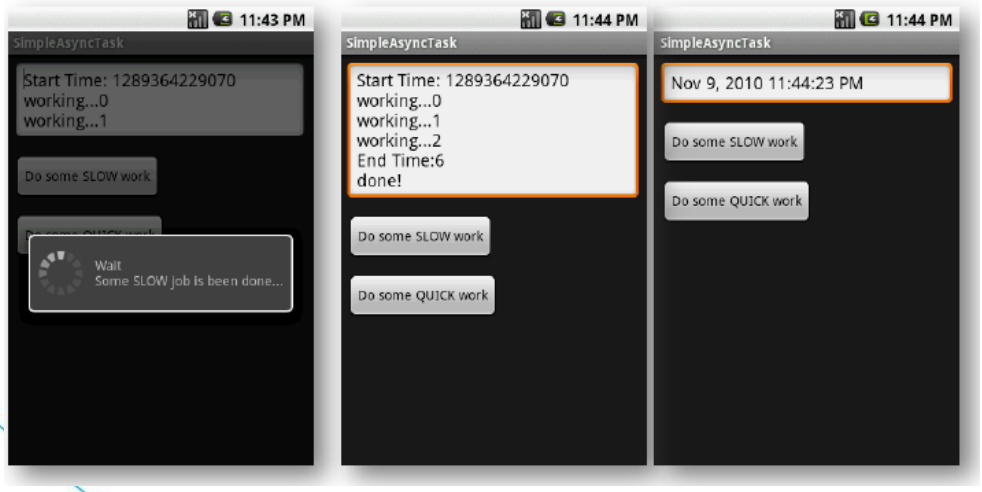
| AsyncTask's generic types | |
|---|---|
| **Params**: | the type of the parameters sent to the task upon execution. |
| **Progress**: | the type of the progress units published during the background computation. |
| **Result**: | the type of the result of the background computation. |

**AsyncTask - Primer**

Glavni task poziva AsyncTask koji izračunava nešto i periodično update-uje UI (upisuje linije teksta) i menja cirkularni progress bar.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




Slika 4.2 Primer AsyncTask-a

```
public class Main extends Activity {
Button btnSlowWork;
Button btnQuickWork;
EditText etMsg;
Long  startingMillis;

@Override
public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      etMsg = (EditText) findViewById(R.id.EditText01);

      btnSlowWork = (Button) findViewById(R.id.Button01);
      // slow work...for example: delete all data from a database or get data from Internet
      this.btnSlowWork.setOnClickListener(new OnClickListener() {
            public void onClick(final View v) {
                  new VerySlowTask().execute();
            }
      });

      btnQuickWork = (Button) findViewById(R.id.Button02);
      // delete all data from database (when delete button is clicked)
      this.btnQuickWork.setOnClickListener(new OnClickListener() {
            public void onClick(final View v) {
                  etMsg.setText((new Date()).toLocaleString());
            }
      });

}// onCreate
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
private class VerySlowTask extends AsyncTask <String, Long, Void> {

      private final ProgressDialog dialog = new ProgressDialog(Main.this);

      // can use UI thread here
      protected void onPreExecute() {
            startingMillis = System.currentTimeMillis();
            etMsg.setText("Start Time: " + startingMillis);
            this.dialog.setMessage("Wait\nSome SLOW job is being done...");
            this.dialog.show();
      }

      // automatically done on worker thread (separate from UI thread)
      protected Void doInBackground(final String... args) {
        try {
            // simulate here the slow activity
            for (Long i = 0L; i < 3L; i++) {
                  Thread.sleep(2000);
                  publishProgress((Long)i);
                  }
        } catch (InterruptedException e) {
                  Log.v("slow-job interrupted", e.getMessage())
        }
        return null;
      }

     // periodic updates - it is OK to change UI
     @Override
     protected void onProgressUpdate(Long... value) {
           super.onProgressUpdate(value);

           etMsg.append("\nworking..." + value[0]);
     }


     // can use UI thread here
     protected void onPostExecute(final Void unused) {

           if (this.dialog.isShowing()) {
                 this.dialog.dismiss();
           }

           // cleaning-up, all done
           etMsg.append("\nEnd Time:"
                     + (System.currentTimeMillis()-startingMillis)/1000);
           etMsg.append("\ndone!");
     }

 }//AsyncTask


}// Main
```

### 4.6 Praktična vežba 4 - Perzistencija podataka i SQLite baza

Na prošloj vežbi je omogućen unos omiljenih lokacija u aplikaciju MyPlaces. Za sada je moguće samo unositi ime lokacije i njen opis. Aplikacija ne pamti unete lokacije između dva poziva (osim ukoliko sama list kontrola nije upamtila stanje, ipak to nije dovoljno). U te svrhe je potrebno izvršiti pamćenje unetih lokacija nekako. Android podržava ugrađeni nezavisan program sqlite3 za rad sa lokalnim bazama na samom telefonu. U ovoj vežbi ćemo posvetiti pažnju korišćenju SQLite baze podataka za pamćenje unetih omiljenih lokacija. Pored toga, potrebno je proširiti osnovne klase i activity-e tako da podržavaju unos geografskih koordinata i dodatnog fajla koji opisuje našu lokaciju. Kao i prethodne vežbe, i ova se oslanja na kod sa prethodnih vežbi.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Ciljevi ove vežbe su:

1. Proširenje klase MyPlace.java tako da pamti longitudu, latitudu i putanju do fajla povezanog sa lokacijom.
2. Kreiranje klasa za rad sa bazom podataka. Potrebno je implementirati MyPlacesDBAdapter.java klasu koja olakšava rad sa bazom podataka.
3. Kreirati klasu MyPlacesApplication.java.
4. Omogućiti pristup bazi podataka iz klase MyPlacesData.java.
5. Dodati mogućnost brisanja omiljene lokacije u kontekstni meni MyPlacesList.java klase.

1. Proširenje klase MyPlace.java
   a. Pokrenuti Eclipse razvojno okruženje I otvoriti klasu MyPlace.java
   b. Dodati u klasu string polja longitude, latitude i filename.
   c. Kreirati odgovarajuće get i set metode za dodata polja.

```
String longitude;
String latitude;
String filename;
public void setLongitude(String lon)
{
    this.longitude = lon;
}
public void setLatitude(String lat)
{
    this.latitude = lat;
}
public void setFilename(String file)
{
    this.filename = file;
}
public String getLongitude()
{
    return this.longitude;
}
public String getLatitude()
{
    return this.latitude;
}
public String getFilename()
{
    return this.filename;
}
```

   d. Dodati u klasu MyPlace.java long ID polje koje će držati odgovarajući, automatski dodeljeni ID lokacije. Dodati i odgovarajuće get i set metode.

```
long ID;
public void setID(long id)
{
    this.ID = id;
}
public long getID()
{
    return ID;
}
```

2. Kreiranje klasa za rad sa bazom podataka. Potrebno je implementirati MyPlacesDBAdapter.java klasu koja olakšava rad sa bazom podataka.
   a. Kreirati klasu MyPlacesDBAdapter.java koja će se koristiti za pristup, kreiranje i izvršavanje upita nad bazom.
   b. U klasu dodati reference na SQLiteDatabase objekat i stringove koje ćemo koristiti za ime baze, ime tabele i imena kolona. Na ovaj način ćemo izbeći greške koje su moguće prilikom unošenja pogrešnih stringova u komande za rad sa bazom.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
package org.elfak.modis.myplaces;

import java.util.ArrayList;

public class MyPlacesDBAdapter {
    public static final String DATABASE_NAME = "MyPlacesDb";
    public static final String DATABASE_TABLE = "MyPlaces";
    public static final int DATABASE_VERSION = 1;

    public static final String PLACE_ID = "ID";
    public static final String PLACE_NAME = "Name";
    public static final String PLACE_DESCRIPTION = "Desc";
    public static final String PLACE_LONG = "Long";
    public static final String PLACE_LAT = "Lat";
    public static final String PLACE_FILENAME="Filename";

    private SQLiteDatabase db;
```

c. Kreiranje tabele je moguće izvršiti programski (kao što je pokazano na računskim vežbama) a moguće je i kopirati bazu sa računara na mobilni uređaj. Android aplikacije se distribuiraju fajlovima sa ekstenzijom .apk I developerima bi bilo jako teško da ubede korisnike da urade još nešto, pored preuzimanja aplikacije sa marketa. Stoga je prvi pristup mnogo bolji osim možda za aplikacije namenjene manjem broju korisnika ili nekoj grupi. Da bi rešili problem kreiranja baze programski prilikom prvog pokretanja aplikacije najbolje je koristiti pomoćnu klasu koju kreiramo nasleđivanjem klase SQLiteOpenHelper. Ona prilikom pristupa bazi automatski proverava da li tražena baza podataka postoji I ako ne izvršava kod za kreiranje. Pored toga ovak pomočna klasa rešava probleme promene verzije baze ili bekapovanja.

d. Kreirati klasu MyPlacesDatabaseHelper.java koja nasleđuje klasu SQLiteOpenHelper i implementirati nasleđene metode onCreate() i onUpgrade(). Dodati i string za kreiranje tabele u bazi podataka.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```java
package org.elfak.modis.myplaces;

import android.content.Context;

public class MyPlacesDatabaseHelper extends SQLiteOpenHelper {
    // SQL naredba za kreiranje nove tabele
    private static final String DATABASE_CREATE = "create table " + MyPlacesDBAdapter.DATABASE_TABLE + " ("
        + MyPlacesDBAdapter.PLACE_ID + " integer primary key autoincrement, "
        + MyPlacesDBAdapter.PLACE_NAME + " text unique not null, "
        + MyPlacesDBAdapter.PLACE_DESCRIPTION + " text, "
        + MyPlacesDBAdapter.PLACE_LONG + " text, "
        + MyPlacesDBAdapter.PLACE_LAT + " text, "
        + MyPlacesDBAdapter.PLACE_FILENAME + " text);";

    public MyPlacesDatabaseHelper(Context context, String name,
            CursorFactory factory, int version) {
        super(context, name, factory, version);
        // TODO Auto-generated constructor stub
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(DATABASE_CREATE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Log the version upgrade.
        Log.w("TaskDBAdapter", "Upgrading from version " + oldVersion + " to " + newVersion
                + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS " + MyPlacesDBAdapter.DATABASE_TABLE);
        onCreate(db);

    }
}
```

e. U klasu MyPlacesDBAdapter.java dodati polje reference na objekat tipa MyPlacesDatabaseHelper. Pošto konstruktor objekta SQLiteOpenHelper zahteva prosleđivanje konteksta, dodati i reference na Context u istu klasu I kreirati odgovarajući konstruktor.

```java
private final Context context;
private MyPlacesDatabaseHelper dbHelper;

public MyPlacesDBAdapter(Context cont) {
    context = cont;
    dbHelper = new MyPlacesDatabaseHelper(context, DATABASE_NAME, null, DATABASE_VERSION);
}
```

f. Implementirati metod za otvaranje konekcije na bazu open().

```java
public MyPlacesDBAdapter open() throws SQLException {
    db = dbHelper.getWritableDatabase();
    return this;
}
```

g. Implementirati metod za zatvaranje konekcije na bazu close().

```java
public void close() {
    db.close();
}
```

h. Implementirati metod za dodavanje novog podatka u bazu insertEntry().



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
public long insertEntry(MyPlace myPlace) {
    ContentValues contentValues = new ContentValues();

    contentValues.put(PLACE_NAME, myPlace.getName());
    contentValues.put(PLACE_DESCRIPTION, myPlace.getDesc());
    contentValues.put(PLACE_LONG, myPlace.getLongitude());
    contentValues.put(PLACE_LAT, myPlace.getLatitude());
    contentValues.put(PLACE_FILENAME, myPlace.getFilename());
    long id = -1;
    db.beginTransaction();
    try{
        id = db.insert(DATABASE_TABLE, null, contentValues);
        db.setTransactionSuccessful();
    } catch(SQLiteException e) {
        Log.w("MyPlacesDBAdapter", e.getMessage());
    } finally {
        db.endTransaction();
    }
    return id;
}
```

i. Implementirati metod za izbacivanje podatka iz baze removeEntry().

```
public boolean removeEntry(long id) {
    boolean success= false;
    db.beginTransaction();
    try{
        success = db.delete(DATABASE_TABLE, PLACE_ID+"=" + id, null) > 0;
        db.setTransactionSuccessful();
    } catch(SQLiteException e) {
        Log.w("MyPlacesDBAdapter", e.getMessage());
    } finally {
        db.endTransaction();
    }
    return success;
}
```

j. Implementirati metod za preuzimanje svih podataka iz baze getAllEntries().

```
public ArrayList<MyPlace> getAllEntries () {
    ArrayList<MyPlace> myPlaces = null;
    Cursor cursor = null;
    db.beginTransaction();
    try{
        cursor = db.query(DATABASE_TABLE, null ,null, null, null, null, null);
        db.setTransactionSuccessful();
    } catch(SQLiteException e) {
        Log.w("MyPlacesDBAdapter", e.getMessage());
    } finally {
        db.endTransaction();
    }
    if(cursor!= null)
    {
        myPlaces = new ArrayList<MyPlace>();
        MyPlace myPlace = null;
        while(cursor.moveToNext()){
            myPlace = new MyPlace(cursor.getString(cursor.getColumnIndex(MyPlacesDBAdapter.PLACE_NAME)));
            myPlace.setDesc(cursor.getString(cursor.getColumnIndex(MyPlacesDBAdapter.PLACE_DESCRIPTION)));
            myPlace.setLongitude(cursor.getString(cursor.getColumnIndex(MyPlacesDBAdapter.PLACE_LONG)));
            myPlace.setLatitude(cursor.getString(cursor.getColumnIndex(MyPlacesDBAdapter.PLACE_LAT)));
            myPlace.setFilename(cursor.getString(cursor.getColumnIndex(MyPlacesDBAdapter.PLACE_FILENAME)));
            myPlaces.add(myPlace);
        }
    }
    return myPlaces;
}
```

k. Implementirati metod za preuzimanje jednog podatka iz baze getEntry().



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
public MyPlace getEntry(long id) {
    MyPlace myPlace = null;
    Cursor cursor = null;
    db.beginTransaction();
    try{
        cursor = db.query(DATABASE_TABLE, null, PLACE_ID +"=" + id, null, null, null, null);
        db.setTransactionSuccessful();
    } catch(SQLiteException e) {
        Log.w("MyPlacesDBAdapter", e.getMessage());
    } finally {
        db.endTransaction();
    }
    if(cursor!= null)
    {
        if(cursor.moveToFirst()){
            myPlace = new MyPlace(cursor.getString(cursor.getColumnIndex(MyPlacesDBAdapter.PLACE_NAME)));
            myPlace.setDesc(cursor.getString(cursor.getColumnIndex(MyPlacesDBAdapter.PLACE_DESCRIPTION)));
            myPlace.setLongitude(cursor.getString(cursor.getColumnIndex(MyPlacesDBAdapter.PLACE_LONG)));
            myPlace.setLatitude(cursor.getString(cursor.getColumnIndex(MyPlacesDBAdapter.PLACE_LAT)));
            myPlace.setFilename(cursor.getString(cursor.getColumnIndex(MyPlacesDBAdapter.PLACE_FILENAME)));
        }
    }
    return myPlace;
}
```

l. Implementirati metod za editovanje jednog podatka u bazi updateEntry().

```
public int updateEntry(long id, MyPlace myPlace) {
    String where = PLACE_ID + "=" + id;

    ContentValues contentValues = new ContentValues();

    contentValues.put(PLACE_NAME, myPlace.getName());
    contentValues.put(PLACE_DESCRIPTION, myPlace.getDesc());
    contentValues.put(PLACE_LONG, myPlace.getLongitude());
    contentValues.put(PLACE_LAT, myPlace.getLatitude());
    contentValues.put(PLACE_FILENAME, myPlace.getFilename());

    return db.update(DATABASE_TABLE, contentValues, where, null);
}
```

3. Kreirati klasu MyPlacesApplication.java. Konstruktor klase SQLiteOpenHelper zahteva prosleđivanje context objekta. Ovaj objekat je nasleđen u Application i Activity klasama I ima za cilj da odredi životni vek objekta kome je context objekat prosleđen. Tako npr. svaki Widget "živi" u memoriji onoliko koliko je aktivan Activity čiji mu je context prosleđen. Pošto ćemo mi pristupati bazi posredno preko više različitih Activity-a, potrebno je konstruktoru MyPlacesDatabaseHelper.java klase proslediti Application context. Application context je moguće preuzeti takođe iz Activity-a. Ipak postoji "workaround" koji omogućuje da se izbegne potreba za instancama Activity objekata u cilju pribavljanja reference na Application context. Ovo se vrši kreiranjem Application klase. Application klasa najčešće nije potrebna u jednostavnijim programima, pa će nama služiti kao pomoćna klasa. Ova klasa treba da omogući pribavljanje reference na Application context (detaljnije o ovome na http://developer.android.com/reference/android/content/Context.html#getApplicationContext() i http://developer.android.com/resources/articles/avoiding-memory-leaks.html .
    a. Kreirati klasu MyApplication.java kao što je prikazano na slici. Klasa ima statičku metodu koja omogućava pribavljanje context-a.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```java
package org.elfak.modis.myplaces;

import android.app.Application;

public class MyPlacesApplication extends Application {
    private static MyPlacesApplication instance;

    public MyPlacesApplication() {
        instance = this;
    }

    public static Context getContext() {
        return instance;
    }
}
```

b. Obavezno modifikovati AndroidManifest.xml tako da u tagu application bude dodat i atribut koji definiše ime odgovarajuće Application klase.

```xml
<application android:name="MyPlacesApplication"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name" >
    <activity
```

4. Omogućiti pristup bazi podataka iz klase MyPlacesData.java.
   a. Dodati referencu na objekat MyPlacesDBAdapter i prilagoditi konstruktor tako da instancira ovaj objekat i iščitava sve objekte iz baze.

```java
MyPlacesDBAdapter dbAdapter;
private MyPlacesData()
{
    dbAdapter = new MyPlacesDBAdapter(MyPlacesApplication.getContext());
    dbAdapter.open();
    this.myPlaces = dbAdapter.getAllEntries();
    dbAdapter.close();
}
```

   b. Metodu addNewPlace() prilagoditi tako da novi element dodaje u bazu podataka i iščitava njegov ID koji smešta u objekat MyPlace radi dalje identifikacije vrste u bazi.

```java
public void addNewPlace(MyPlace place)
{
    myPlaces.add(place);
    dbAdapter.open();
    long ID = dbAdapter.insertEntry(place);
    dbAdapter.close();
    place.setID(ID);

}
```

   c. Prilagoditi i metodu deletePlace() i kreirati metodu updatePlace().

```java
public void deletePlace(int index)
{
    MyPlace place = myPlaces.remove(index);
    dbAdapter.open();
    dbAdapter.removeEntry(place.getID());
    dbAdapter.close();
}
public void updatePlace(MyPlace place)
{
    dbAdapter.open();
    dbAdapter.updateEntry(place.getID(), place);
    dbAdapter.close();
}
```

   d. Povezati EditMyPlaceActivity tako da se vrši ažuriranje podataka o lokaciji.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

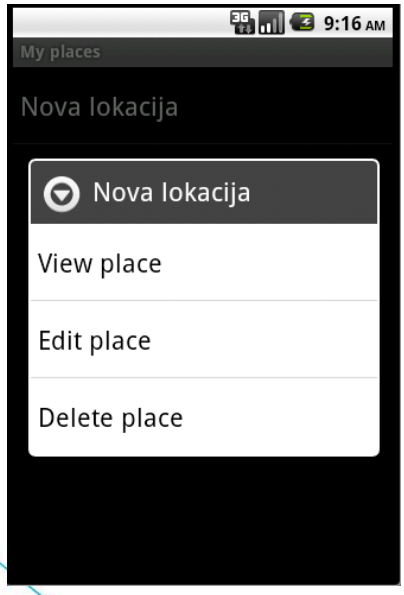
```
public void onClick(View v) {
    switch(v.getId())
    {
        case R.id.editmyplace_finished_button:
        {
            EditText etName = (EditText)findViewById(R.id.editmyplace_name_edit);
            String nme = etName.getText().toString();
            EditText etDesc = (EditText)findViewById(R.id.editmyplace_desc_edit);
            String desc = etDesc.getText().toString();
            if(!editMode)
            {
                MyPlace place = new MyPlace(nme, desc);
                MyPlacesData.getInstance().addNewPlace(place);
            }
            else
            {
                MyPlace place = MyPlacesData.getInstance().getPlace(position);
                place.setName(nme);
                place.setDesc(desc);
                MyPlacesData.getInstance().updatePlace(place);
            }
            setResult(Activity.RESULT_OK);
            finish();
            break;
        }

        case R.id.editmyplace_cancel_button:
        {
            setResult(Activity.RESULT_CANCELED);
            finish();
            break;
        }
    }
}
```

5. Dodati mogućnost brisanja omiljene lokacije u kontekstni meni MyPlacesList.java klase.
    a. Potrebno je dodati novu stavku menija u metodi onCreateContextMenu()
    b. Obraditi klik na dodatu stavku tako da se poziva metod deletePlace() iz klase MyPlacesData.java.

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo)
{
    super.onCreateContextMenu(menu, v, menuInfo);
    AdapterView.AdapterContextMenuInfo info = (AdapterView.AdapterContextMenuInfo)menuInfo;
    menu.setHeaderTitle( MyPlacesData.getInstance().getPlace(info.position).getName());
    menu.add(0, 1, 1, "View place");
    menu.add(0, 2, 2, "Edit place");
    menu.add(0, 3, 3, "Delete place");
}
@Override
public boolean onContextItemSelected(MenuItem item)
{
    AdapterView.AdapterContextMenuInfo info = (AdapterView.AdapterContextMenuInfo)item.getMenuInfo();
    Bundle positionBundle = new Bundle();
    positionBundle.putInt("position", info.position);
    Intent i = null;
    if(item.getItemId() == 1)
    {
        i = new Intent(this, ViewMyPlaceActivity.class);
        i.putExtras(positionBundle);
        startActivity(i);
    }
    else if(item.getItemId() == 2)
    {
        i = new Intent(this, EditMyPlaceActivity.class);
        i.putExtras(positionBundle);
        startActivityForResult(i, 1);
    }
    else
    {
        MyPlacesData.getInstance().deletePlace(info.position);
        setListAdapter(new ArrayAdapter<MyPlace>(this, android.R.layout.simple_list_item_1, MyPlacesData.getInstance().getMyPlaces()));
    }
    return super.onContextItemSelected(item);
}
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES






Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 5. Mobilno pozicioniranje, Google mape i lokaciono-zasnovani servisi (LBS)

## 5.1 Lokaciono zasnovani servisi

Lokaciono zasnovani servisi (LBS – Location Based Service) su mobilni informacioni servisi kojima se obezbeđuju infromacije i servisi korisniku u zavisnosti od njegove lokacije, ili lokacija mobilnih i/ili stacionarnih objekata od njegovog interesa. Osnovna karakteristika ovih servisa je raspoloživost i dostupnost na svakom mestu, u svakom trenutku, kad i gde je potrebno, putem mobilnih uređaja (mobilni telefon, tableti, laptop i nosivi računari).

Primeri LBS:

- Navigacioni i saobraćajni servisi u realnom vremenu za obradu zahteva poput: "Kako stići do mesta/adrese najbržim/najkraćim putem preko ...", ili "Obavestiti o postojanju saobraćajnog zastoja i načinu na koji ga izbeći".
- Urgentni servisi za obradu zahteva poput: "Pokvaren automobil, neophodna pomoć na putu", "Neophodna medicinska pomoć zbog srčanih smetnji!", "Da li postoje zapaljive ili eksplozivne materije u rejonu akcije gašenja požara?".
- Turistički servisi poput: "Najbliže pozorište ili bioskop koji prikazuje određenu predstavu/film", "Gde se u blizini može kupiti/ručati/videti/posetiti ...".
- Marketinška obaveštenja zasnovana na lokaciji potencijalnih klijenata, poput: "Popust sportske garderobe u prodavnici 50 m udaljenoj od vas".
- Nalaženje najbližih prijatelja (dece), poput: "Da li je dete napustilo školu i gde se trenutno nalazi?"
- Lokaciono zasnovane igre, gde se igrač u proširenoj stvarnosti takmiči sa drugim igračima.

## 5.2 Mobilno pozicioniranje

Pozicioniranje/lociranje korisnika/mobilnih objekata predstavlja određivanje lokacije objekta, tačnije njegove pozicije u prostoru u odnosu na određeni prostorni (geografski, kartografski) referentni system. Postoji veliki broj tehnoloških rešenja za mobilno pozicioniranje koja se mogu podeliti u četiri grupe:

- Pozicioniranje zasnovano na mobilnom uređaju (GPS) – (terminal based positioning),
- Pozicioniranje zasnovano na mobilnoj komunikacionoj mreži (network based positioning),
- Hibridno pozicioniranje zasnovano i na mobilnom uređaju i na mobilnoj mreži,
- Pozicioniranje korišćenjem posebno razvijene infrastrukture (obično u zatvorenim prostorima)

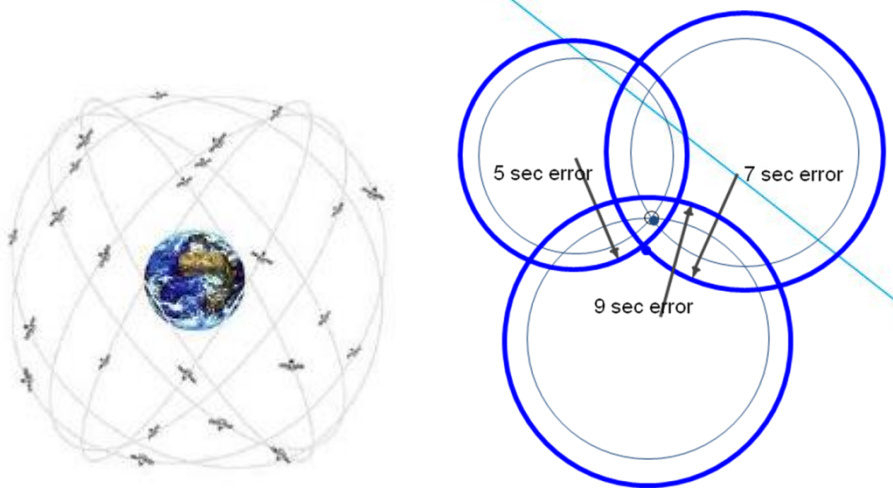
### 5.3 GPS

Sistem za globalno pozicioniranje kontrolisan od SAD. Razvoj sistema započet 1970-ih, potpuno operativan od 1995. godine. Sastoji se od tri segmenta:

- Svemirski segment: sateliti
- Upravljački (kontrolni) segment
- Korisnički segment: svi GPS korisnici, vojni i civilni, sa odgovarajućim GPS prijemnicima

Svemirski segment čini kompletna konstelacija Navstar GPS satelita u Zemljinim orbitama (24 operativna, 7 rezervna), raspoređenih u 6 orbita (po 4 satelita u svakoj) nagnutih 55° u odnosu na Ekvatorsku ravan na 20200 km visine. Jedan obilazak satelita oko Zemlje po orbiti traje 11 časova 58 minuta (brzina oko 11000 km/h). Napajaju se solarnom energijom, a njihov planiran životni vek iznosi 7.5 godina (proizvođač: Rockwell International).

Princip funkcionisanja GPS je sledeći. Sateliti emituju jedinstveni signal koji uključuje prostorne i vremenske informacije. Rastojanje od satelita se određuje merenjem vremena prenosa ovog signala (koji se kreće brzinom svetlosti). Lokacija prijemnika se odeđuje u preseku tri sfere, čiji su poluprečnici rastojanja do satelita metodom triangulacije (trilateracije) (slika 5.1). Četvrti satelit se koristi zbog problema sa sinhronizacijom clock-a i određivanja tačne pozicije. Izvori GPS grešaka su: greška u časovniku satelita/prijemnika, greška u poziciji satelita (ephemeris), uticaj atmosfere na prostiranje satelitskih signala, refrakcija u jonosferi, refrakcija u troposferi, ugao položaja satelita, šumovi, višestruki putevi do prijemnika (Multipath), opstrukcije signala usled selektivne raspoloživosti (S/A).




Slika 5.1 Princip funkcionisanja GPS-a

Tačnost GPS-a može biti poboljšana korišćenjem diferencijalnog GPS (D-GPS), WAAS (EGNOS) i A-GPS. Ove metode se generalno zasnivaju na dodavanju prijemnika sa tačno određenom pozicijom i emitovanju korekcionih podataka putem mreže mobilne telefonije ili geostacionarnih satelita. Skoro



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

svi savremeni mobilni (pametni) telefoni i tableti poseduju ugrađeni GPS prijemnik, što je već standardna karakteristika.

Prednosti pozicioniranja putem GPS-a su: pozicioniranje jezasnovano isključivo na mobilnom uređaju, globalna dostupnost, dobra preciznost. Nedostaci su veliko kašnjenje do dobijanja prvih informacija o poziciji, nemogućnost pozicioniranja u zatvorenim prostorima i prirodnim i urbanim kanjonima i neophodan je dodatan hardver (cena + potrošnja snage).

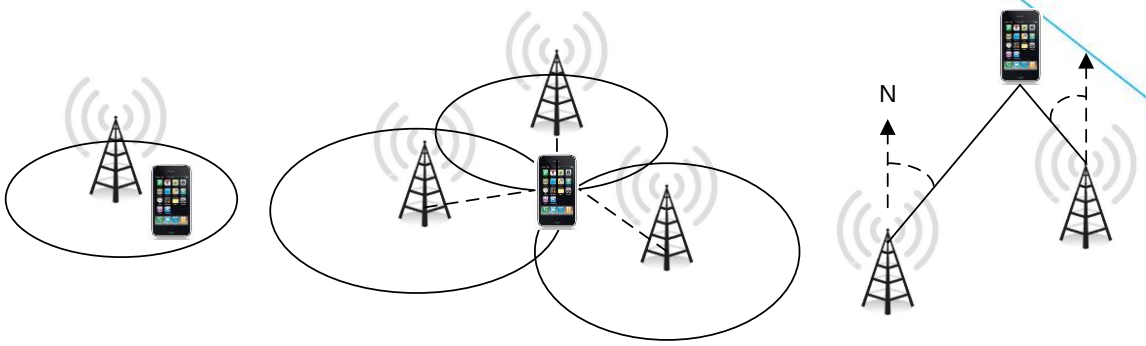
GLONASS sistem je započet kao SSSR ekvivalent GPS-u. Sateliti su lansirani u periodu 1992 – 1996 i funkcionalnu konstelaciju čine 24 satelita. Upravljački segment je u granicama bivšeg SSSR. Počev od 2001 Rusija je započela njegovo ponovno uspostavljanje i u saradnji sa vladom Indije, kao partnerom, postigla punu funkcionalnost i globalnu pokrivenost 2011 godine.

Galileo projekat u okviru EU je započet 1999 godine, sa namerom postizanja pozicioniranja nezavisno od SAD, sistemom koji će biti interoperabilan sa GPS i GLONASS. Galileo čine 30 satelita (27 operativnih i 3 rezervna satelita, na tri cirkularne orbite (Medium Earth Orbits – MEOs) na visini 23 222 km i inklinaciji od 56º u odnosu na Ekvator. Prva dva operativna satelita lansirana u oktobru 2011, inicijalni navigacioni servisi očekuju se 2016, dok se puna funkcionalnost i globalna pokrivenost sistema očekuje do 2019 godine.

### 5.4 Pozicioniranje zasnovano na mobilnoj mreži

Razvijene su različite metode za pozicioniranje mobilnih telefona zasnovane na bežičnoj mreži (slika 5.2):

- Identitet ćelije - Cell-ID (slika 5.2a)
- Timing Advance (TA)
- Time of Arrival (TOA) (slika 5.2b)
- Time Difference of Arrival (TDOA)
- Angle of Arrival (AOA) (slika 5.2c)
- Enhanced Observed Time Difference (E-OTD)




a) b) c)

Slika 5.2 Pozicioniranje mobilnih telefona zasnovano na bežičnoj mreži



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 5.5 Pozicioniranje u zatvorenom prostoru

- Wireless LAN (WiFi) – metoda zasnovana na triangulaciji poput pozicioniranja u mobilnim mrežama
- WiFi fingerprinting
- Ultra-wideband (UWB)
- Bluetooth, Bluetooth LE
- Infracrveni signali (IrDA)
- RFID
- Ultrazvučna identifikacija
- Specijalizovana infrastruktura, predajnici/odašiljači i prijemnici koji se koriste isključivo za pozicioniranje:
    - Active Badge (XeroxParc)
    - Active Bat (AT&T)
    - RADAR - Microsoft Research RF rešenje za lociranje i praćenje

WiFi fingerprinting tehnika se zasniva na tome da distribucija radio signala pokazuje jedinstvene šablone u diskretnim pozicijama nekog prostora (referentne tačke). Radio mapa se dobija snimanjem i pamćenjem podataka koji opisuju šablone i njihovim asociranjem sa diskretnim lokacijama u kojima su podaci snimljeni. Omogućava poređenje tih podataka sa podacima koji će biti snimljeni u budućnosti. Kada mobilni uređaj na nekoj lokaciji snimi odgovarajuće podatke o dostupnim radio signalima, ti podaci se mogu uporediti sa prethodno snimljenim. Kada se pronađe najbolje poklapanje novih podataka sa snimljenim podacima, lokacija na kojoj su snimljeni ti podaci se uzima kao lokacija mobilnog uređaja/korisnika.

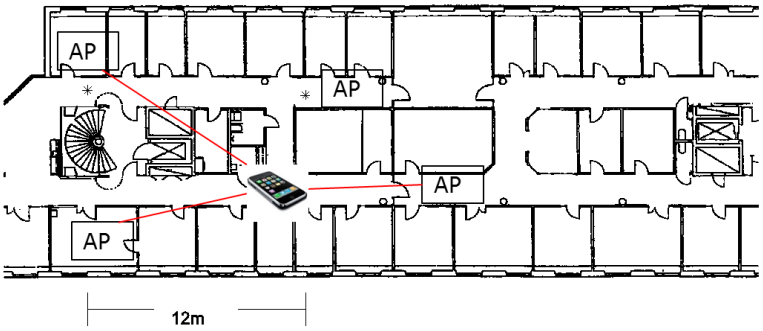
Pomenuti šabloni, odnosno podaci o signalima snimljeni na nekoj lokaciji se nazivaju “otisci” (eng. Fingerprint). Jedan otisak na nekoj lokaciji predstavlja skup informacija (MAC adresa, jačina signala, ime mreže, kanal i dr.) o signalima svih access point-a čiji se signal može registrovati na datoj lokaciji. Na osnovu otisaka za veliki broj diskretnih lokacija na nekom prostoru, kreira se radio mapa tog prostora.

- Faza obuke (Training phase)
- Faza pozicioniranja (Realtime location phase)

Sličnost između otisaka iz radio mape i trenutno snimljenog otiska, najčešće se određuje primenom Euklidovog algoritma rastojanja u prostoru signala. Određivanje lokacije se obavlja nalaženjem najsličnijeg otiska čija se lokacija uzima kao lokacija uređaja ili aproksimacijom u odnosu na lokacije k najsličnijih otisaka (slika 5.3).



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




Slika 5.3 Lociranje korišćenjem WiFi fingerprinting tehnike

### 5.6 Android lociranje

Metode lociranja na Android platformi

- GPS (satelitski, Epson Infineon prijemnik 3x3mm)
- Cellular network (u mreži baznih stanica)
- WiFi (na osnovu RSSI access point-a) - Svi korisnici učestvuju u generisanju podataka za lociranje

Android klase za lociranje:

- *Address*
- *Criteria*
- *Geocoder*
- *GpsSatellite*
- *GpsStatus*
- *Location*
- *LocationManager*
- *LocationProvider*
- *GpsStatus.Listener*
- *GpsStatus.NmeaListener*
- *LocationListener*

***Location*** klasa predstavlja poziciju uređaja određenu u nekom trenutku. Sadrži podatke o geografskoj dužini i širini, UTC timestamp i opciono nadmorsku visinu, brzinu i pravac kretanja. Klasa ima getter metode za sve ove podatke. Zanimljiva je statička metoda:

static void distanceBetween(...)

koja računa rastojanje između dve tačke u sfernom koordinatnom sistemu što inače ne predstavlja jednostavan problem.

***LocationManager*** klasa omogućava pristup Android sistemskim servisima za geolociranje. Ima dva osnovna zadatka:

- Periodično obaveštava o lokaciji mobilnog uređaja
- Emituje Intent kada se korisnik nađe u blizini neke definisane tačke

Ne instancira se direktno već se objekat ove klase dobija kao sistemski servis:

Context.getSystemService(Context.LOCATION_SERVICE)

Konkretnu lokaciju generiše ***LocationProvider***. LocationProvider se pribavlja direktno po imenu ili na osnovu definisanih kriterijuma lociranja (Criteria – potrošnja energije, preciznost itd.)

***LocationListener*** klasa se registruje sa ***LocationManager***-om i po promeni lokacije dobija novu lokaciju. Registracija listener-a se obavlja sledećom metodom:

requestLocationUpdates(Provider, minTime, minDistance, LocationListener)

Značajne callback metode su:

- onLocationChanged(Location) - Nova lokacija korisnika dobijena
- onProviderDisabled(String), onProviderEnabled(String) - Korisnik isključio/uključio location provider
- onStatusChanged(String, int, Bundle) - Promenjen status provider-a (GPS ne vidi dovoljno satelita npr.)

***LocationProvider*** klasa opisuje karakteristike konkretnog location provider-a (metode lociranja). Osim preciznosti, lociranje korišćenjem provider-a može korisniku prouzrokovati dodatne troškove. A-GPS (assisted GPS – koristi mobilni paketni prenos podataka da preuzme ažuran almanah).

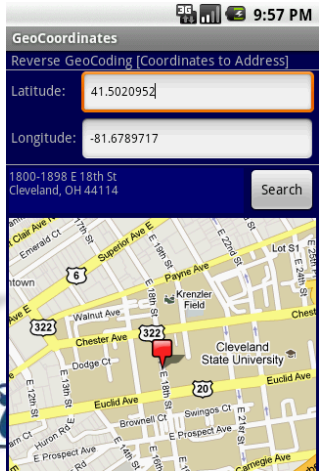
Metode za ispitivanje tipova podataka o lokaciji i kretanju koje provider vraća:

- getAccuracy()
- supportsAltitude()
- supportsBearing()
- supportsSpeed()

Google nudi dodatne lokacione servise za:

- Geokodiranje - Vraća koordinate na osnovu adrese

```
Geocoder gc= new Geocoder(this);
// 5 najpribližnijih lokacija
List<Address> lstFoundAddresses =
        gc.getFromLocationName(txtStreetAddress, 5);
```



- Reverzno geokoridanje - Vraća adresu na osnovu koordinata

```
Geocoder gc = new Geocoder(context, Locale.US);
List<Address> streets =
        gc.getFromLocation(latitude, longitude, 1);
```

U Android emulatoru koordinate se mogu ručno uneti ili se može uneti putanja kao:

- GPX datoteka - često ručni GPS prijemnici mogu snimati putanju kretanja u ovaj format
- KML datoteka - XML format Google Earth

### 5.7 Android i mapiranje

Android koristi Google Maps eksternu biblioteku za rad sa mapama. Klase iz ove biblioteke (paketa) potpuno transparentno preuzimaju (download sa Interneta) segmente mape, iscrtavaju mapu i keširaju segmente mape na lokalnom uređaju.

Google Maps V2 predstavlja potpuno novi concept, dok je V1 zastarela i map key nije moguće dobiti. Glavna klasa u ovom paketu je MapFragment, podklasa Fragment klase koja obezbeđuje prikaz mape. GoogleMap klasa predstavlja ulaznu tačku za sve metode koje se odnose na mapu, a objekat ove klase se dobija metodom getMap() nad MapFragment-om.

Android Google Maps API V2 zavisi od Google Play Service SDK ekstenzije koja se instalira kroz SDK manager. Projekat može da se importuje u Eclipse workspace kao biblioteka:

```
<android-sdk-folder>/extras/google/google_play_services/libproject/google-
play-services_lib
```

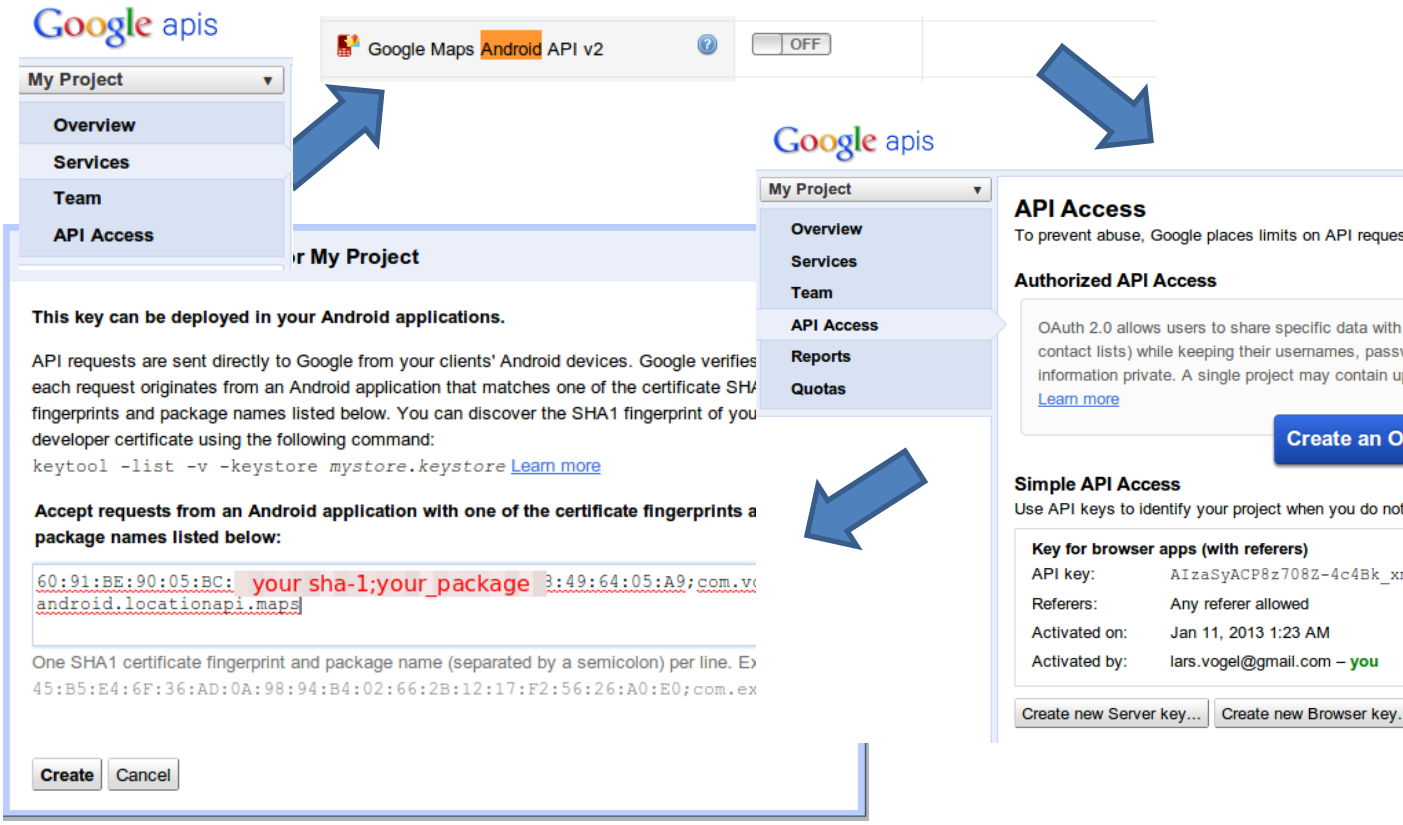
Voditi računa o TargetSDK, a za više detalja pogledati: http://android-er.blogspot.com/2012/12/a-simple-example-using-google-maps.html

Korišćenje Google resursa se prati kroz registraciju aplikacija korišćenjem ključeva. SVAKI development računar ima keystore, Debug i Release. Za dobijanje ključa bitan je SH1 fingerprint. Može se dobiti na korišćenjem komande keytool iz komandne linije:

```
keytool -list -v -keystore <USER_HOME>/.android/debug.keystore -alias
androiddebugkey -storepass android -keypass android
```

ili iz Eclipse IDE-a, opcijom iz menija Window->Preferences->Android ->Build.

Centralizovano upravljanje svim Google servisima kroz Google Developers Console (ranije Google APIs Console) - https://code.google.com/apis/console (slika 5.4)




Slika 5.4 Kreiranje Android ključa za korišćenje Google Maps API V2

Neophodne su izmene i u manifestu aplikacije:

```
    <permission
        android:name="com.example.MAPS_RECEIVE"
        android:protectionLevel="signature" />
```

Custom permission štićen potpisom aplikacije. Verovatno se koristi kao endpoint za komunikaciju od strane Google Maps API

```
    <uses-feature
        android:glEsVersion="0x00020000"
        android:required="true" />
```

Google Maps API V2 koristi OpenGL ES v2 za renderovanje mape

```
    <uses-permission android:name="com.example.MAPS_RECEIVE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <application
        …
        <meta-data
            android:name="com.google.android.maps.v2.API_KEY"
            android:value="your_apikey" />
    </application>

</manifest>
```

Aplikacija koristi registrovan Google Maps V2 API key (**ne može se koristiti V1 key**)

Google Maps API V2 radi sa novijim tipovima UI widget-a:

- ***MapFragment*** (od API level 11 – Android 3.0)



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

- Za starije uređaje se koristi support library - ***SupportMapFragment***

Glavna aktivnost nasleđuje *FragmentActivity*, a ne *Activity*. Sve aplikacije koje koriste ovaj API moraju da prikažu Google Play Services Legal Notices. Klasa GooglePlayServicesUtil nudi helper metodu za preuzimanje aktuelnog teksta licence:

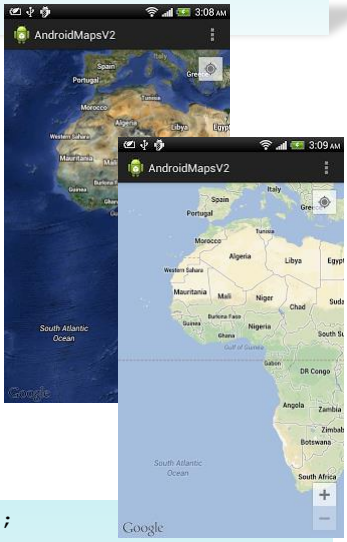
```
String LicenseInfo = GooglePlayServicesUtil.getOpenSourceSoftwareLicenseInfo(
        getApplicationContext());
AlertDialog.Builder LicenseDialog = new AlertDialog.Builder(MainActivity.this);
LicenseDialog.setTitle("Legal Notices");
LicenseDialog.setMessage(LicenseInfo);
LicenseDialog.show();
```

Google Maps API V2 zavisi od Google Play Services komponente pa treba proveriti da li je ova bibloteka instalirana i uključena u projekat.

```
@Override
 protected void onResume() {
  super.onResume();
  int resultCode = GooglePlayServicesUtil.isGooglePlayServicesAvailable(getApplicationContext());
  if (resultCode == ConnectionResult.SUCCESS){
   Toast.makeText(getApplicationContext(), "isGooglePlayServicesAvailable SUCCESS",
     Toast.LENGTH_LONG).show();
  }else{
   GooglePlayServicesUtil.getErrorDialog(resultCode, this, RQS_GooglePlayServices);
  }
```

Za upravljanje mapom se koristi GoogleMap objekat koga treba preuzeti iz fragmenta

```
FragmentManager myFragmentManager = getSupportFragmentManager();
SupportMapFragment mySupportMapFragment
    = (SupportMapFragment)myFragmentManager.findFragmentById(R.id.map);
myMap = mySupportMapFragment.getMap();
```




Moguće je uključiti i opciju prikaza korisnikove trenutne lokacije:

```
myMap.setMyLocationEnabled(true);
```

Postavljanje tipa mape obavlja se korišćenjem sledećih konstanti:

- MAP_TYPE_NONE - Bez pozadinske mape
- MAP_TYPE_NORMAL - Obične mape
- MAP_TYPE_SATELLITE - Satelitski snimak bez labela
- MAP_TYPE_HYBRID - Satelitski snimak sa transparentnim ulicama
- MAP_TYPE_TERRAIN - Reljef

```
FragmentManager myFragmentManager = getSupportFragmentManager();
SupportMapFragment mySupportMapFragment
          = (SupportMapFragment)myFragmentManager.findFragmentById(R.id.map);
myMap = mySupportMapFragment.getMap();
myMap.setMyLocationEnabled(true);
//myMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
//myMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
//myMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
myMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
```

Radi detekcija klika na mapu FragmentActivity treba da implementira *OnMapClickListener*. Override-ujemo metodu onMapClick(LatLng point) i uvezujemo listener sa:

myMap.setOnMapClickListener(this)

```
public class MainActivity extends FragmentActivity implements OnMapClickListener{
  private GoogleMap myMap;
  ...
  myMap.setOnMapClickListener(this);
  ...
  @Override
  public void onMapClick(LatLng point) {
    tvLocInfo.setText(point.toString());
    myMap.animateCamera(CameraUpdateFactory.newLatLng(point));
  }
  ...
```

Dodavanje markera na mapu se ne obavlja kroz overlay slojeve kao u V1 API, već je neophodno definisati poseban listener: *OnMapLongClickListener.*

FragmentActivity implementira ovaj listener i override-uje metodu

onMapLongClick(LatLng point)

Listener se vezuje za mapu metodom myMap.setOnMapLongClickListener(this)

Marker se dodaje direktno na mapu:

```
myMap.addMarker(new MarkerOptions().position(point).title("title")
```

Da bi se obradio klik na marker i crtanje polilinije na mapi, aktivnost treba da implementira:

OnMarkerClickListener

```
@Override
 public boolean onMarkerClick(Marker marker) {
  if(markerClicked){
   if(polyline != null){
    polyline.remove();
    polyline = null;
   }
   rectOptions.add(marker.getPosition());
   rectOptions.color(Color.RED);
   polyline = myMap.addPolyline(rectOptions);
  }else{
   if(polyline != null){
    polyline.remove();
    polyline = null;
   }
   rectOptions = new PolylineOptions().add(marker.getPosition());
   markerClicked = true;
  }
  return true;
 }
```

Poligon se iscrtava na praktično identičan način:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
@Override
 public boolean onMarkerClick(Marker marker) {

  if(markerClicked){

   if(polygon != null){
    polygon.remove();
    polygon = null;
   }

   polygonOptions.add(marker.getPosition());
   polygonOptions.strokeColor(Color.RED);
   polygonOptions.fillColor(Color.BLUE);
   polygon = myMap.addPolygon(polygonOptions);
  }else{
   if(polygon != null){
    polygon.remove();
    polygon = null;
   }

   polygonOptions = new PolygonOptions().add(marker.getPosition());
   markerClicked = true;
  }

  return true;
 }
```

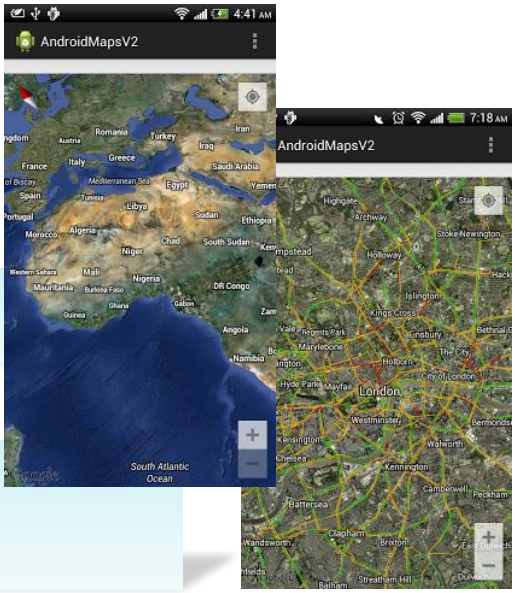
Premeštanje markera (draggable marker) se obavlja na sledeći način:

- Marker mora da bude označen kao draggable  marker.draggable(true)
- Implementirati OnMarkerDragListener  u aktivnosti
- Listener se postavlja za mapu:  myMap.setOnMarkerDragListener(this)
- Override-ovati tri metode listenera: onMarkerDrag, onMarkerDragStart i onMarkerDragEnd

GoogleMap widget podržava standarne kontrole

- Zoom, kompas, myLocation

```
myMap.getUiSettings().setZoomControlsEnabled(true);
myMap.getUiSettings().setCompassEnabled(true);
myMap.getUiSettings().setMyLocationButtonEnabled(true);
```




- Omogućavanje map gestures kontrole (*multi-touch*)

```
myMap.getUiSettings().setRotateGesturesEnabled(true);
myMap.getUiSettings().setScrollGesturesEnabled(true);
myMap.getUiSettings().setTiltGesturesEnabled(true);
myMap.getUiSettings().setZoomGesturesEnabled(true);
//or myMap.getUiSettings().setAllGesturesEnabled(true);
```

- Traffic layer



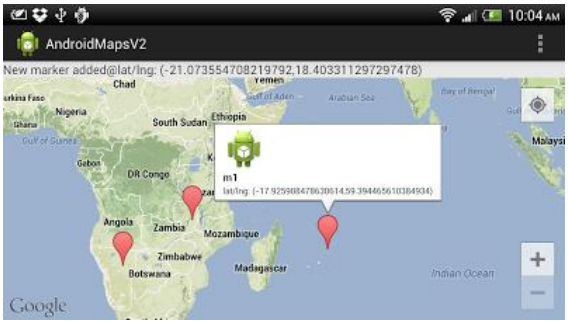
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA  Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
myMap.setTrafficEnabled(true);
```

Custom info za markere se dobija kroz implementaciju InfoWindowAdapter-a.
Dodati layout za info prikaz.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/andr
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginRight="5dp"
            android:adjustViewBounds="true"
            android:src="@drawable/ic_launcher"/>
        <TextView
            android:id="@+id/title"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="12dp"
            android:textStyle="bold"/>
        <TextView
            android:id="@+id/snippet"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="10dp"/>
    </LinearLayout>
</LinearLayout>
```

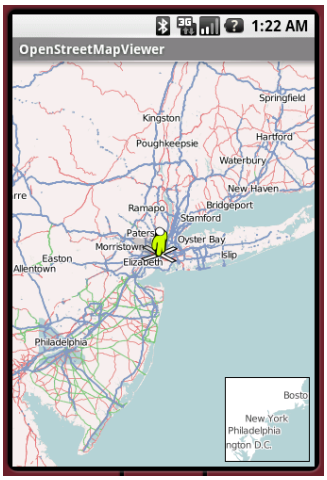
```
public class MainActivity extends Activity
 implements OnMapLongClickListener{

class MyInfoWindowAdapter implements InfoWindowAdapter{
  private final View myContentsView;
  MyInfoWindowAdapter(){
   myContentsView = getLayoutInflater().inflate(
            R.layout.custom_info_contents, null);
  }

  @Override
  public View getInfoContents(Marker marker) {
    TextView tvTitle =
((TextView)myContentsView.findViewById(R.id.title));
    tvTitle.setText(marker.getTitle());
    TextView tvSnippet =
((TextView)myContentsView.findViewById(R.id.snippet));
    tvSnippet.setText(marker.getSnippet());

    return myContentsView;
  }

  @Override
  public View getInfoWindow(Marker marker) {
   // TODO Auto-generated method stub
   return null;
  }

 }
```

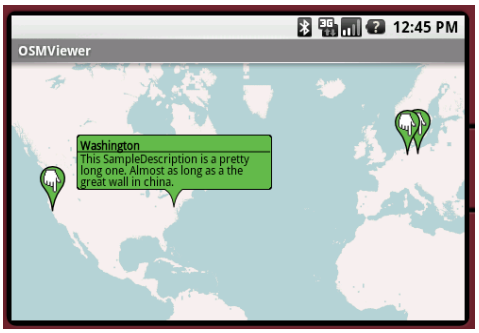



Alternativa Google Maps API V je OpenStreetMap, kao potpuno besplatan map servis (http://www.openstreetmap.org/). OpenStreetMap na Androidu je moguće obezbediti korišćenjem osmdroid biblioteke (https://github.com/osmdroid/osmdroid) koja potpuno imitira GoogleMaps V1 API.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES






### 5.8 Praktična vežba 5 - Rad sa mapama - Google Maps API v2

Na poslednjoj vežbi je omogućeno pamćenje omiljenih lokacija u ugrađenu bazu podataka SQLite. Pored toga, model podataka je proširen tako da podržava pamćenje longitude i latitude (koordinata) omiljene lokacije. Ovo je preduslov za prikaz omiljenih lokacija na mapi. Google nudi programerima svoje rešenje za rad sa mapama u vidu MapView widgeta. Ovim je značajno olakšan rad sa mapama i nije više potrebno biti GIS ekspert da bi se prikaz mape ugradio u aplikacije. Ugrađena podrška za servise za lociranje takođe omogućava lakšu navigaciju mape, očitavanje trenutne lokacije i prikaz željene lokacije na mapi korišćenjem markera. U ovoj vežbi će akcenat biti na radu sa mapama čime aplikacija dobija svoju originalnu svrhu. Kao i prethodne vežbe, i ova se oslanja na kod sa prethodnih vežbi.

Ciljevi ove vežbe su:

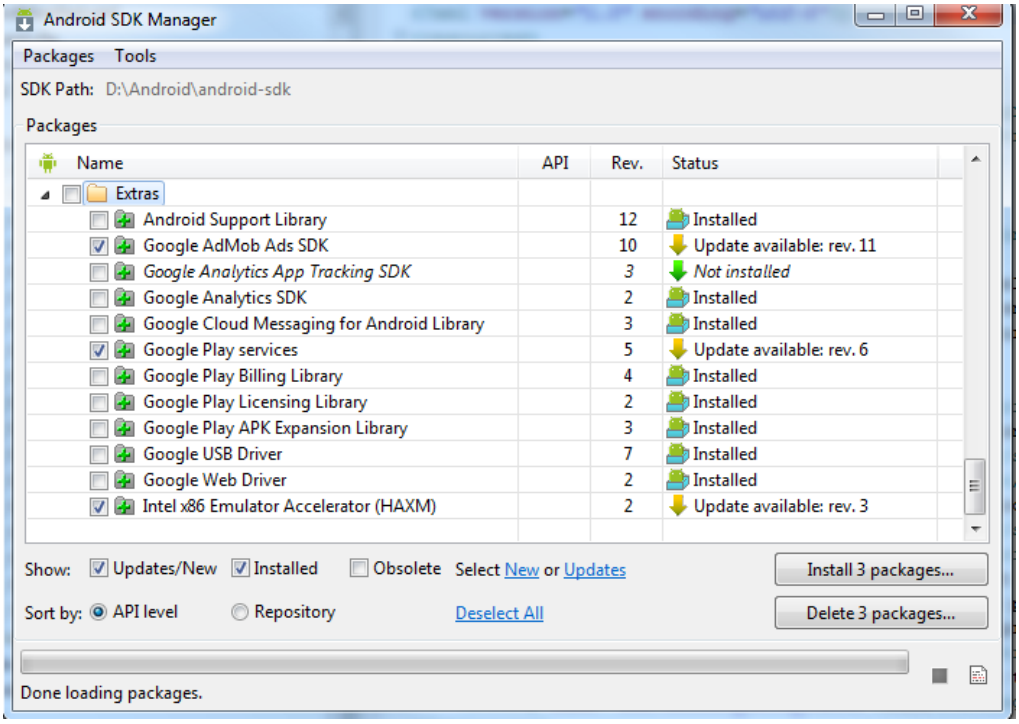
1. Preuzimanje Google Play Services biblioteke i njeno uključivanje u projekat
2. Registrovanje debug ključa za rad sa mapama.
3. Priprema AndroidManifest.xml-a.
4. Omogućavanje rada sa Map fragmentom.
5. Kreiranje MyPlacesMapActivity.java klase i prikaz mape sa trenutnom lokacijom korisnika.
6. Dodavanje kontrola za unos i prikaz koordinata omiljene lokacije u EditMyPlaceActivity.java i podrška za izbor koordinata omiljene lokacije sa mape.
7. Prilagođavanje MyPlacesMapActiviry-a za različita stanja rada (prikaz lokacija na mapi, centriranje prikaza omiljene lokacije na mapi, izbor koordinata omiljene lokacije na mapi).
8. Dodavanje markera omiljenih lokacija na mapi.
9. Prikaz određene omiljene lokacije centrirano na mapi.

1. Preuzimanje Google Play Services biblioteke i njeno uključivanje u projekat.
    a. Ukoliko *google_play_services* nisu instalirane na računaru instalirati ih korišćenjem Android SDK Manager-a.
    b. Pokrenuti Android SDK Manager iz Eclipse-a klikom na Window->Android SDK Manager.

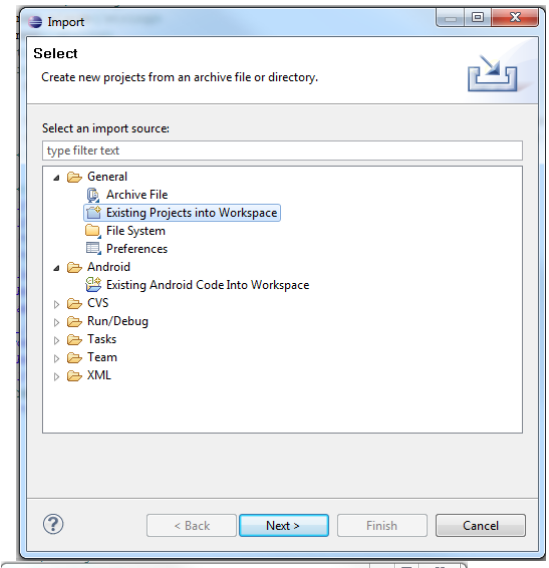


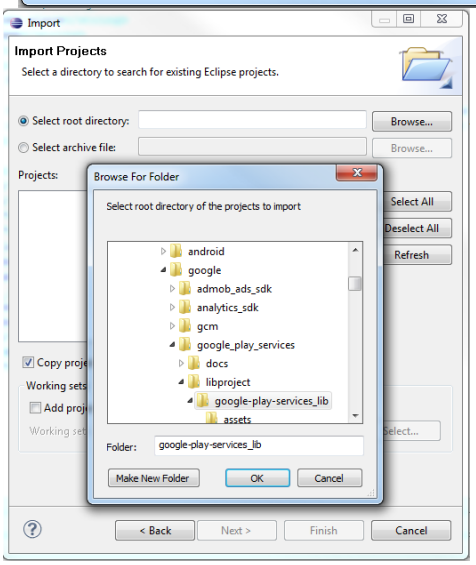
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

c. Skrolovati do dna i čekirati opciju Extras > Google Play services a zatim kliknuti na Install dugme.




d. Nakon što je Google Play services biblioteka preuzeta, potrebno ju je uključiti u Workspace. Kliknuti File -> Import..., izabrati Android -> Existing Android Code into Workspace.



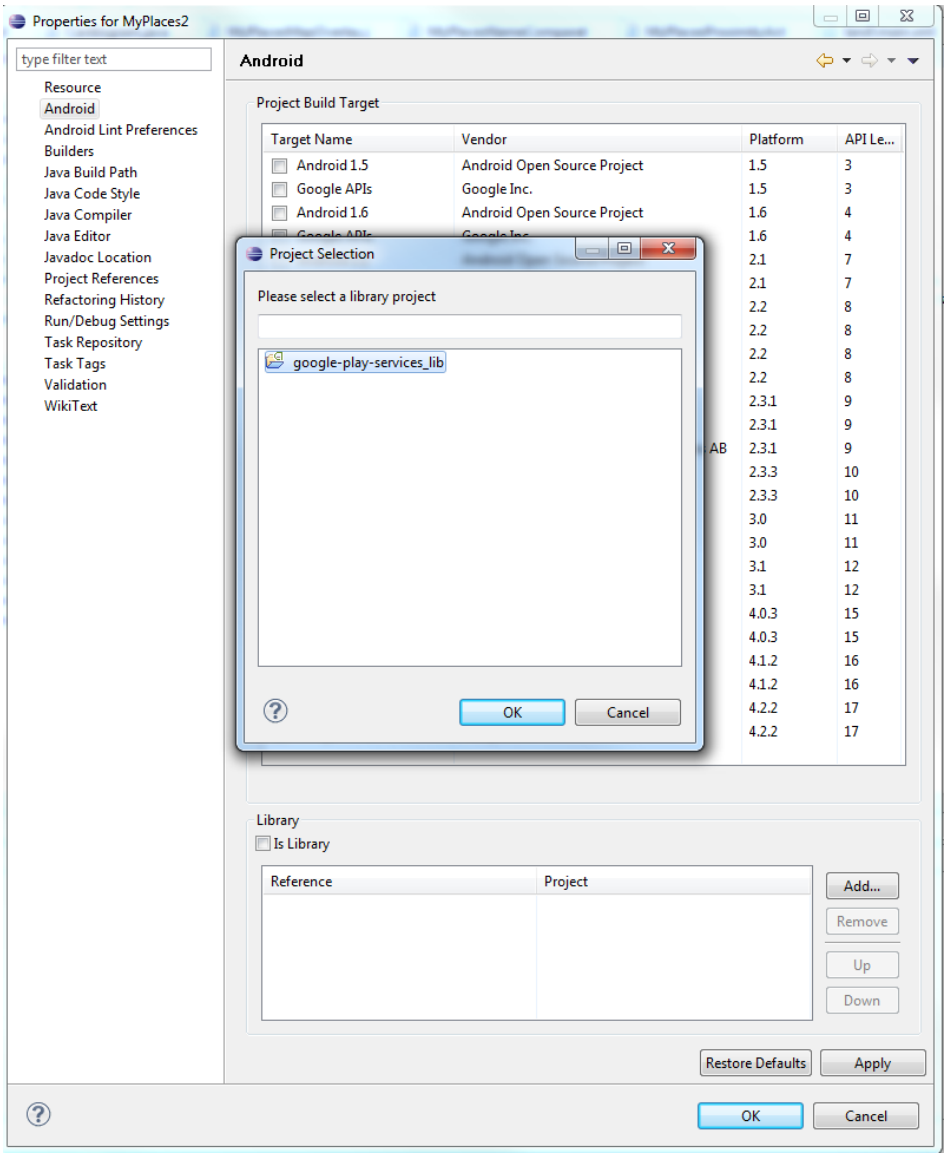




e. U dijalogu izabrati <android-sdk-folder>/extras/google/google_play_services/libproject/google-play-services_lib i kliknuti Import

f. Potrebno je još uključiti biblioteku u projekat. Potrebno je kliknuti na Project -> Properties -> Android -> Library, a zatim na Add I izabrati google-play-services_lib.



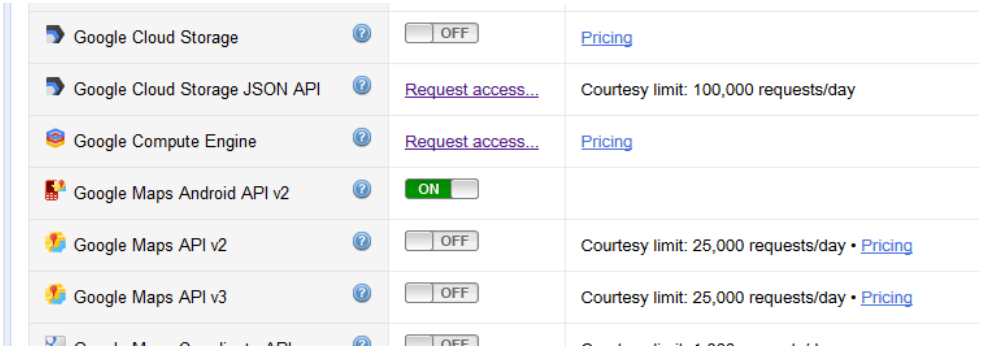
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




2. Registrovanje debug ključa za rad sa mapama.
    a. Da bi koristili Google Maps API V2 potrebno je da registrujete API Key vezan za svoj lični google account.
    b. Idite na Google APIs Console (https://code.google.com/apis/console )
    c. Logujte se sa svojim Google nalogom.
    d. U Services stavci omogućite upotrebu Google Maps Android API v2 i prihvatite uslove korišćenja.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




e. Zatim kliknite na API Access (Pre toga opciono kreirajte API projekat).
f. Zatim kliknite Create New Android Key.
g. Potrebno je uneti vaš SHA-1 fingerprint, njega možete dobiti korišćenjem keytool alata koji se najćešće nalazi tamo gde je smešten Java jre ili jdk, npr. C:\Program Files (x86)\Java\jre7\bin
h. U Command Promptu izvršite navigaciju do navedenog foldera u kome se nalazi keytool.exe i unesite sledeću komandu za kreiranje debug key-a:

   keytool -list -alias androiddebugkey -keystore ~/.android/debug.keystore -storepass android -keypass android –v
   gde je .android folder na Windows-u smešten u User direktorijumu pa ~/.android/debug.keystore postaje najčešće C:\Users\Ime\.android\debug.keystore.
i. Kao rezultat ćete dobiti nešto poput BB:0D:AC:74:D3:21:E1:43:67:71:9B:62:91:AF:A1:66:6E:44:5D:75
j. U developer konzoli na sajtu uneti taj ključ praćen sa “;package.name” pa bi ključ trebalo da izgleda ovako:

   BB:0D:AC:74:D3:21:E1:43:67:71:9B:62:91:AF:A1:66:6E:44:5D:75;org.mad.myplaces
k. I trebalo bi da dobijete api key od 40 karaktera koji je sličan ovom ključu: AIzbSyBaEqeUpYvlsbUe0OqTPn1-ABVMHVbdNxo

3. Priprema AndroidManifest.xml-a.
   a. Dodati u manifest u okviru application tag-a podatak o kreiranom ključu

```
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="your_api_key"/>
```

   b. Dodati i sledeće permission-e. Voditi računa da zapravo prvo kreiramo MAPS_RECEIVE permission a zatim ga koristimo. Kreiramo tako što imenu permission-a prethodi ime paceta aplikacije u kome će se koristiti Google map-a.

```
<permission
     android:name="org.elfak.modis.myplaces.permission.MAPS_RECEIVE"
     android:protectionLevel="signature"/>

<uses-permission android:name="org.elfak.modis.myplaces.permission.MAPS_RECEIVE"/>
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

c. Omogućiti i mod openGlEs-a.

```
<uses-feature
    android:glEsVersion="0x00020000"
    android:required="true"/>
```

4. Omogućavanje rada sa Map fragmentom.
   a. Kreirati MyPlacesMapActivity.java klasu i dodati i ovaj activity u manifest.

```
        <activity android:name=".MyPlacesMapActivity"
            android:theme="@android:style/Theme.Black.NoTitleBar.Fullscreen">
        </activity>
```

   b. Umesto da nasleđuje Activity, prepraviti kod tako da klasa nasleđuje FragmentActivity.

```
public class MyPlacesMapActivity extends FragmentActivity {
```

   c. Potrebno je kreirati i odgovarajući xml fajl za layout pod nazivom myplacesmap.xml. Mapa se prikazuje kao Fragment u okviru Activity-a. Iako je Fragment uveden u Android od verzije 3.0, da bi ovaj Fragment bio korišćen na starijim verzijama Android-a, potrebno je koristiti verziju SupportMapFragment class prilikom navođenja u XML opisu korisničkog interfejsa.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/frameLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <fragment
        android:id="@+id/map"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        class="com.google.android.gms.maps.SupportMapFragment" />

</FrameLayout>
```

   d. U klasu MyPlacesMapActivity.java dodati referencu na GoogleMap-u

```
private GoogleMap map;
```

   e. U onCreate metodi MyPlacesMapActivity-a dodati postavljanje reference na GoogleMap objekat i omogućiti prikaz sopstvene lokacije na mapi. Google Maps će automatski dodati dugme kojim je moguće prikazati korisnikovu trenutnu lokaciju centriranu na mapi.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    setContentView(R.layout.myplacesmap);
    map = ((SupportMapFragment) getSupportFragmentManager().findFragmentById(R.id.map)).getMap();
    map.setMyLocationEnabled(true);
}
```

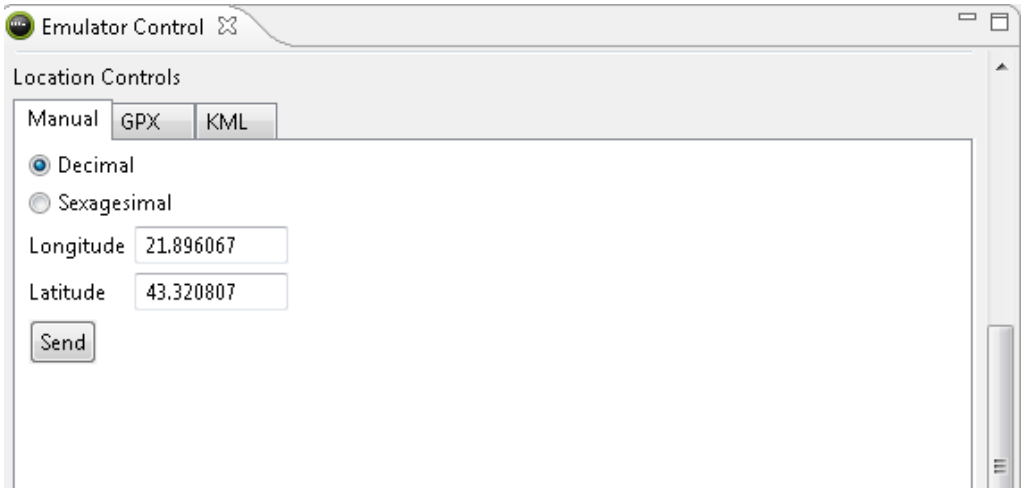
   f. Da bi dobili lokaciju dok testiramo aplikaciju, potrebno je da imamo nekakav izvor trenutne lokacije koji ne može biti GPS jer se nalazimo unutar objekta. U tom slučaju je, ukoliko koristimo emulator (ovo može biti problem jer Google mape API verzije 2 nisu predviđene za testiranje na emulatoru ukoliko se ne koristi “zaobilazni” put), potrebno lokaciju postaviti korišćenjem DDMS pogleda iz Eclipse razvojnog okruženja



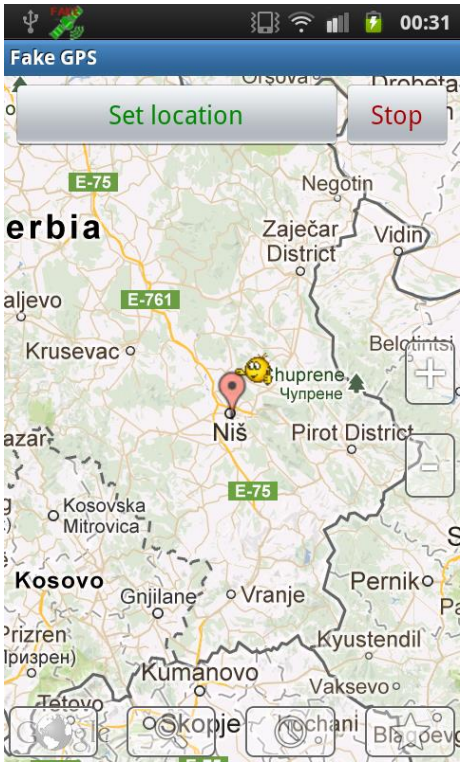
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

dok ukoliko koristimo uređaj, možemo lokaciju dobiti ili od nekog drugog provajdera (WiFi ili GSM) ili koristiti aplikaciju koja emulira lokacijski provajder (preporuka FakeGPS aplikacija).

g. Iz DDMS-a postavljamo lokaciju upisujući longitudu i latitudu u odgovarajuća polja I kliknuti na Send. Ove koordinate će automatski biti dostupne u emulatoru.




h. Korišćenjem FakeGPS aplikacije lokaciju možemo postaviti izborom lokacije na mapi.



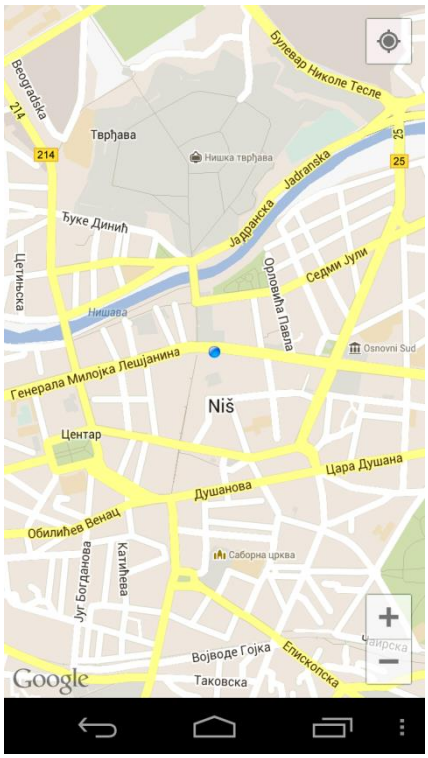

i. Preostalo je još napraviti poziv za napravljeni MyPlacesMapActivity iz osnovnog Activity-a aplikacije. Za ovo je potrebno dodati kreiranje i poziv odgovarajućeg intent-a u klasi MyPlacesActivity.java.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
@Override
public void onClick(View v) {
    Intent i = null;
    switch(v.getId())
    {
        case R.id.my_places_button:
            i = new Intent(this, MyPlacesList.class);
            startActivity(i);
            break;
        case R.id.new_place_button:
            i = new Intent(this, EditMyPlaceActivity.class);
            startActivityForResult(i, 1);
            break;
        case R.id.show_map_button:
            i = new Intent(this, MyPlacesMapActivity.class);
            startActivity(i);
            break;
        case R.id.about_button:
            i = new Intent(this, About.class);
            startActivity(i);
            break;
        case R.id.exit_button:
            finish();
            break;
    }
}
```

j. Probajte aplikaciju. Na ekranu emulatora bi trebalo da vidite sledeću sliku nakon postavljanja validne lokacije:






Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

5. Dodavanje kontrola za unos i prikaz koordinata omiljene lokacije u EditMyPlaceActivity.java i podrška za izbor koordinata omiljene lokacije sa mape.
   a. Potrebno je dodati odgovarajuće kontrole u edimyplace.xml.

```
        <TableLayout
                android:layout_height="wrap_content"
                android:layout_width="wrap_content"
                android:layout_gravity="center"
                android:stretchColumns="*" >
            <TableRow
                 android:layout_width="fill_parent"
                 android:layout_height="wrap_content" >
                 <LinearLayout
                     android:layout_width="fill_parent"
                     android:layout_height="wrap_content"
                     android:orientation="vertical" >
                     <TextView
                        android:layout_width="fill_parent"
                        android:layout_height="wrap_content"
                        android:text="@string/editmyplace_lat_label" />
                     <EditText
                        android:id="@+id/editmyplace_lat_edit"
                        android:layout_width="fill_parent"
                        android:layout_height="wrap_content" />
                     <TextView
                        android:layout_width="fill_parent"
                        android:layout_height="wrap_content"
                        android:text="@string/editmyplace_lon_label" />
                     <EditText
                        android:id="@+id/editmyplace_lon_edit"
                        android:layout_width="fill_parent"
                        android:layout_height="wrap_content" />
                 </LinearLayout>
                 <Button
                     android:id="@+id/editmyplace_location_button"
                     android:layout_width="wrap_content"
                     android:layout_height="wrap_content"
                     android:layout_gravity="center"
                     android:text="@string/editmyplace_location_label" />
            </TableRow>

            <TableRow
                 android:layout_width="fill_parent"
                 android:layout_height="wrap_content" >
                 <Button
                     android:id="@+id/editmyplace_finished_button"
                     android:layout_weight="1" />
                 <Button
                     android:id="@+id/editmyplace_cancel_button"
                     android:layout_weight="1"
                     android:text="@string/editmyplace_cancel_label" />
            </TableRow>
        </TableLayout>
    </LinearLayout>
```

   b. Dodati i promene u kodu tako da se kontrole postavljaju na odgovarajući način u metodama onCreate(), onClick() i kreirati metodu onActivityResult().

```
Button finishedButton = (Button)findViewById(R.id.editmyplace_finished_button);
Button cancelButton = (Button)findViewById(R.id.editmyplace_cancel_button);
Button locationButton = (Button)findViewById(R.id.editmyplace_location_button);
EditText etName = (EditText)findViewById(R.id.editmyplace_name_edit);

cancelButton.setOnClickListener(this);
locationButton.setOnClickListener(this);
etName.addTextChangedListener(this);
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
        case R.id.editmyplace_cancel_button:
        {
            setResult(Activity.RESULT_CANCELED);
            finish();
            break;
        }

        case R.id.editmyplace_location_button:
        {
            Intent i = new Intent(this, MyPlacesMapActivity.class);
            startActivityForResult(i,1);
        }
    }
```

c. U MyPlacesMapActivity.java klasi dodati mogućnost klika na mapu i izbora lokacije. Potrebno je da MyPlacesMapActivity implementira OnMapClickListener interfejs.

```
public class MyPlacesMapActivity extends FragmentActivity implements OnMapClickListener {
```

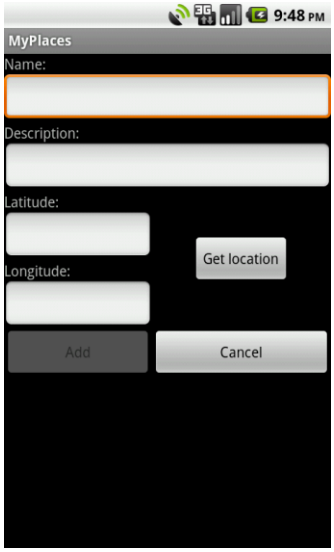
i njegovu metodu onMapClick

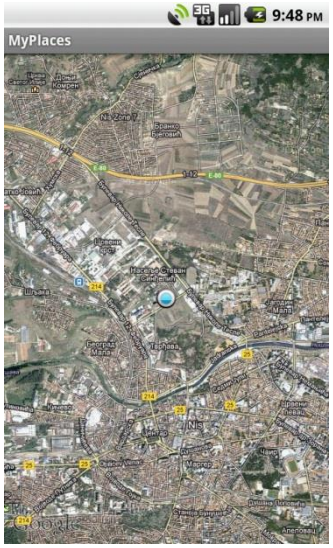
```
@Override
public void onMapClick(LatLng loc) {

}
```

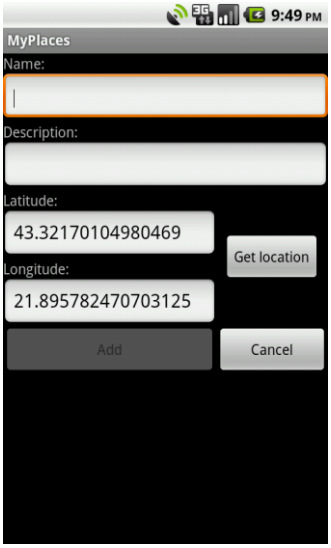
d. Dodati kod koji preuzima lokaciju iz argumenta metode onMapClick i smešta vrednosti longitude I latitude je u Intent kako bi te vrednosti vratio pozivajućem Activity-u.

```
@Override
public void onMapClick(LatLng loc) {
    String lon=Double.toString(loc.longitude);
    String lat=Double.toString(loc.latitude);
    Intent locationIntent = new Intent();
    locationIntent.putExtra("lon", lon);
    locationIntent.putExtra("lat", lat);
    setResult(Activity.RESULT_OK, locationIntent);
    finish();
}
```

e. Sada smo u mogućnosti da preuzmemo lokaciju na koju smo ukazali prstom na mapi. Probajte aplikaciju.




f. Omogućiti smeštanje podataka o longitudi i latitudi (onClick()) u bazu.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
}
else
{
    if(position >=0)
    {
        finishedButton.setText("Save");
        MyPlace place = MyPlacesData.getInstance().getPlace(position);
        etName.setText(place.getName());
        EditText etDesc = (EditText)findViewById(R.id.editmyplace_desc_edit);
        etDesc.setText(place.getDesc());
        EditText latEdit = (EditText)findViewById(R.id.editmyplace_lat_edit);
        latEdit.setText(place.getLatitude());
        EditText lonEdit = (EditText)findViewById(R.id.editmyplace_lon_edit);
        lonEdit.setText(place.getLongitude());
    }
}
finishedButton.setOnClickListener(this);
```

g. Omogućiti i inicijalno postavljanje koordinata prilikom editovanja omiljene lokacije.

```
@Override
public void onClick(View v) {
    switch(v.getId())
    {
        case R.id.editmyplace_finished_button:
        {
            EditText etName = (EditText)findViewById(R.id.editmyplace_name_edit);
            String nme = etName.getText().toString();
            EditText etDesc = (EditText)findViewById(R.id.editmyplace_desc_edit);
            String desc = etDesc.getText().toString();
            EditText latEdit = (EditText)findViewById(R.id.editmyplace_lat_edit);
            String lat = latEdit.getText().toString();
            EditText lonEdit = (EditText)findViewById(R.id.editmyplace_lon_edit);
            String lon = lonEdit.getText().toString();
            if(!editMode)
            {
                MyPlace place = new MyPlace(nme, desc);
                place.setLatitude(lat);
                place.setLongitude(lon);
                MyPlacesData.getInstance().addNewPlace(place);
            }
            else
            {
                MyPlace place = MyPlacesData.getInstance().getPlace(position);
                place.setName(nme);
                place.setDesc(desc);
                place.setLatitude(lat);
                place.setLongitude(lon);
                MyPlacesData.getInstance().updatePlace(place);
            }
            setResult(Activity.RESULT_OK);
            finish();
            break;
        }
```

h. Probati aplikaciju

6. Prilagođavanje MyPlacesMapActivity-a za različita stanja rada (prikaz lokacija na mapi, centriranje prikaza omiljene lokacije na mapi, izbor koordinata omiljene lokacije na mapi).
   a. Trenutno je moguće izabrati koordinate sa mape ali nije moguće panovati i zoomirati mapu. Potrebno je implementirati u MyPlacesMapActivity.java klasi različita stanja, prosleđena putem intent-a tako da se rad sa mapom prilagodi odgovarajućoj nameni u aplikaciji. Prvo je potrebno napraviti stanja i promenljivu koja će ih čuvati. Zgodno je napraviti konstante da ne bi došlo do greške prilikom poziva.

```
public class MyPlacesMapActivity extends FragmentActivity implements OnMapClickListener {

    public static final int SHOW_MAP = 0;
    public static final int CENTER_PLACE_ON_MAP = 1;
    public static final int SELECT_COORDINATES = 2;

    private int state = 0;
    private boolean selCoorsEnabled = false;
    private LatLng placeLoc;

    private GoogleMap map;
```

   b. Prepraviti onCreate metodu tako da očitava stanje Activity-a prosleđeno preko intent i postavlja odgovarajuće promenljive. U zavisnosti od postavljenog stanja, potrebno



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

je i pozvati odgovarajuće metode. Ukoliko je stanje SHOW_MAP omogućiti prikaz trenutne korisnikove lokacije. Za stanje SELECT_COORDINATES omogućiti Click na mapu a za stanje CENTER_PLACE_ON_MAP postaviti kameru zumirati mapu na koordinate omiljenog mesta prosleđene preko Intent-a.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    try
    {
        Intent mapIntent = getIntent();
        Bundle mapBundle = mapIntent.getExtras();
        if(mapBundle != null)
        {
            state = mapBundle.getInt("state");
            if(state == CENTER_PLACE_ON_MAP)
            {
                String placeLat = mapBundle.getString("lat");
                String placeLon = mapBundle.getString("lon");
                placeLoc = new LatLng(Double.parseDouble(placeLat), Double.parseDouble(placeLon));
            }
        }

    }
    catch(Exception e) {
    }
    setContentView(R.layout.myplacesmap);
    map = ((SupportMapFragment) getSupportFragmentManager().findFragmentById(R.id.map)).getMap();
    if(state == SHOW_MAP)
        map.setMyLocationEnabled(true);
    else if(state == SELECT_COORDINATES)
        map.setOnMapClickListener(this);
    else
        map.moveCamera(CameraUpdateFactory.newLatLngZoom(placeLoc, 15));
}
```

c. U slučaju korišćenja mape kada se vrši izbor lokacije, korisniku će inicijalno biti omogućeno da radi sa njom. Nakon što u options meniju izabere odgovarajuće dugme (Select Coordinates), sledeći klik (tap) na mapu će izvršiti izbor koordinata.

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
    if(state == SELECT_COORDINATES && !selCoorsEnabled)
    {
        menu.add(0, 1, 1, "Select Coordinates");
        menu.add(0, 2, 2, "Cancel");
    }
    return super.onCreateOptionsMenu(menu);
}

@Override public boolean onOptionsItemSelected(MenuItem item)
{
    switch (item.getItemId()){
        case 1:
            selCoorsEnabled = true;
            Toast.makeText(this, "Select coordinates", Toast.LENGTH_SHORT).show();
            break;
        case 2:
            setResult(Activity.RESULT_CANCELED);
            finish();
            break;
    }
    return super.onOptionsItemSelected(item);
}
```

d. Potrebno je izvršiti izmene i u metodi onMapClick() tako da se preuzima lokacija samo onda kada smo spremni da izaberemo odgovarajuće koordinate.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
@Override
public void onMapClick(LatLng loc) {
    if(state == SELECT_COORDINATES && selCoorsEnabled)
    {
        String lon=Double.toString(loc.longitude);
        String lat=Double.toString(loc.latitude);
        Intent locationIntent = new Intent();
        locationIntent.putExtra("lon", lon);
        locationIntent.putExtra("lat", lat);
        setResult(Activity.RESULT_OK, locationIntent);
        finish();
    }
}
```

e. Dodajte u odgovarajuće stanja u intent-e koje smo do sada kreirali za pozive MyPlacesMapActivity-a

```
case R.id.editmyplace_location_button:
{
    Intent i = new Intent(this, MyPlacesMapActivity.class);
    i.putExtra("state", MyPlacesMapActivity.SELECT_COORDINATES);
    startActivityForResult(i,1);
}

    ------,
case R.id.show_map_button:
    i = new Intent(this, MyPlacesMapActivity.class);
    i.putExtra("state", MyPlacesMapActivity.SHOW_MAP);
    startActivity(i);
    break;
```

f. Probajte aplikaciju.

7. Prikaz markera omiljenih lokacija na mapi.
   a. Pošto smo pribavili koordinate omiljenih lokacija, možemo lako da ih prikažemo na mapi korišćenjem markera.
   b. Prvo je potrebno dodati ikonicu koju ćemo prikazivati kao marker. Najlakše je kreirati novi folder drawable (bez nastavaka) u resursima i u njega smestiti ikonicu (dostupna na sajtu predmeta).
   c. U onCreate() metodi MyPlacesMapActivity klase dodati kod koji omogućuje dodavanje markera za svako omiljeno mesto koje se nalazi u bazi podataka i mapira objekte dodatih markera sa njihovim pozicijama u listi u HashMap objekat tako da konačno metod izgleda kao što je prikazano na slici.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
private GoogleMap map;

private HashMap<Marker, Integer> markerPlaceIdMap;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    try
    {
        Intent mapIntent = getIntent();
        Bundle mapBundle = mapIntent.getExtras();
        if(mapBundle != null)
        {
            state = mapBundle.getInt("state");
            if(state == CENTER_PLACE_ON_MAP)
            {
                String placeLat = mapBundle.getString("lat");
                String placeLon = mapBundle.getString("lon");
                placeLoc = new LatLng(Double.parseDouble(placeLat), Double.parseDouble(placeLon));
            }
        }

    }
    catch(Exception e) {
    }
    setContentView(R.layout.myplacesmap);
    map = ((SupportMapFragment) getSupportFragmentManager().findFragmentById(R.id.map)).getMap();
    if(state == SHOW_MAP)
        map.setMyLocationEnabled(true);
    else if(state == SELECT_COORDINATES)
        map.setOnMapClickListener(this);
    else
        map.moveCamera(CameraUpdateFactory.newLatLngZoom(placeLoc, 15));
    ArrayList<MyPlace> places = MyPlacesData.getInstance().getMyPlaces();
    markerPlaceIdMap = new HashMap<Marker, Integer>((int)((double)places.size()*1.2));
    for(int i = 0;i<places.size(); i++)
    {
        MyPlace place = places.get(i);
        String lat = place.getLatitude();
        String lon = place.getLongitude();
        LatLng loc = new LatLng(Double.parseDouble(lat), Double.parseDouble(lon));
        MarkerOptions markerOptions = new MarkerOptions();
        markerOptions.position(loc);
        markerOptions.icon(BitmapDescriptorFactory.fromResource(R.drawable.myplace));
        markerOptions.title(place.getName());
        Marker marker = map.addMarker(markerOptions);
        markerPlaceIdMap.put(marker, i);
    }
}
```

d. U listu interfejsa koje implementira MyPlaceMapActivity dodati i OnMarkerClickListener i implementirati njegovu metodu onMarkerClick() tako da za kliknuti marker dobavlja odgovarajuću poziciju i startuje ViewMyPlaceActivity.

```
@Override
public boolean onMarkerClick(Marker marker) {
    Intent intent = new Intent(this, ViewMyPlaceActivity.class);
    int i = markerPlaceIdMap.get(marker);
    intent.putExtra("position", i);
    startActivity(intent);
    return true;
}
```

e. Postaviti u onCrete() metodi da onMarkerClickListener Google mape bude instanca trenutne klase.

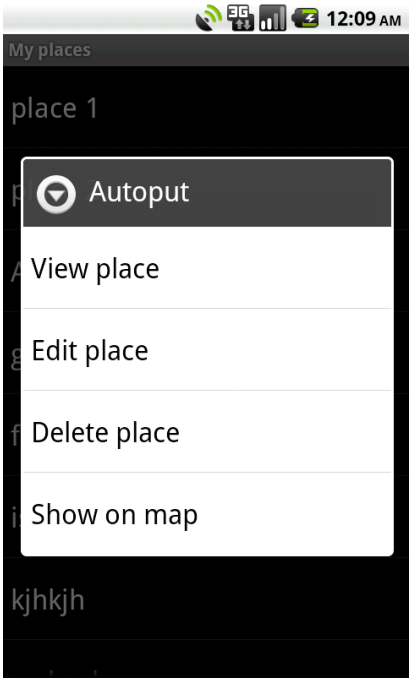
```
map.setOnMarkerClickListener(this);
```

f. Probati aplikaciju



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

8. Dodati mogućnost prikaza omiljene lokacije na mapi dodavanjem opcije u kontekstni meni MyPlacesList.java klase.
    a. Potrebno je dodati novu stavku menija u metodi onCreateContextMenu()
    b. Obraditi klik na dodatu stavku tako da se kreira intent za MyPlaceMapActivity i prosleđuje odgovarajuće stanje (CENTER_PLACE_ON_MAP).




9. Domaći: Dodati prikaz koordinata u ViewMyPlaceActivity

Rad sa Google mapama APi verzije 2 na emulatoru: http://stackoverflow.com/questions/15172597/google-maps-api-v2-not-work-in-emulator



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 6. Mobilna komunikacija i servisi

Vrlo često se u mrežnoj komunikaciji podaci kodiraju u XML format. Rad sa XML dokumentima se zasniva na Document Object Model (DOM) koji se primenjuje na HTML i XML dokumente. Primer podrške za obradu XML sadržaja je Android DOM manager.

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document dom= db.parse(someHttpInputStream);
```

Android za rad sa HTTP protokolom koristi:

- Standardni java.net paket i HttpURLConnection
- Apache HttpClient biblioteku

Za pristup Internet-u je potreban permission android.permission.INTERNET.Sva mrežna komunikacija mora da se obavlja u posebnoj niti.

Apache HttpClient biblioteka sadrži:

- DefaultHttpClient – standardni HttpClient
- AndroidHttpClient - implementacija prekonfigurisana za Android

AndroidHttpClient podržava SSL i GZIP kompresiju podataka i *thread safe* je, što znači da više niti može da koristi jednu instance.

```
HttpClient client = new DefaultHttpClient();
HttpGet request = new HttpGet("http://www.vogella.com");
HttpResponse response = client.execute(request);
// Get the response
BufferedReader rd = new BufferedReader(new InputStreamReader(
         response.getEntity().getContent()));
String line = "";
while ((line = rd.readLine()) != null) {
         textView.append(line);
}
```

Orimer parsiranja XML-a dobijenog preko HTTP-a dat je sledećim programskim kodom:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
  myListView.setOnItemClickListener(new OnItemClickListener() {
    public void onItemClick(AdapterView<?> _av, View _v,
                                int _index, long _id) {
      selectedNewsItem= newsList.get(_index);
      showNiceDialogBox(selectedNewsItem, context);
    }
  });
}//onCreate

@Override
  protected void onResume() {
    super.onResume();
    try{
      URL url = new URL(urlAddress);
      URLConnection connection = url.openConnection();
      HttpURLConnection httpConnection= (HttpURLConnection) connection;
      int responseCode= httpConnection.getResponseCode();
      if(responseCode == HttpURLConnection.HTTP_OK) {
        InputStream in = httpConnection.getInputStream();
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();
        Document dom= db.parse(in);
        Element docEle= dom.getDocumentElement();
        NodeList nl= docEle.getElementsByTagName("item");
```

```
          if((nl != null) && (nl.getLength() > 0)) {
            for(int i= 0; i< nl.getLength(); i++) {
              dissectNode(nl, i);
            }// for
          }// if
        }// if
      int layoutID = R.layout.my_simple_list_item_1;
      ArrayAdapter<SingleNewsItem> aaNews=
                  new ArrayAdapter<SingleNewsItem>(this, layoutID, newsList);
      myListView.setAdapter(aaNews);
    } catch(MalformedURLException e) {
      e.printStackTrace();
    } catch(IOException e) {
      e.printStackTrace();
      Toast.makeText(context, "Trouble!!!", 1).show();
    } catch(ParserConfigurationException e) {
      e.printStackTrace();
    } catch(SAXException e) {
      e.printStackTrace();
    }
  }// onResume
```

Parsiranje XML-a se obavlja na sledeći način:



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
  public void dissectNode(NodeList nl, int i){
    try{
      Element entry = (Element) nl.item(i);
      Element title = (Element) entry.getElementsByTagName("title").item(0);
      Element description = (Element) entry.getElementsByTagName
                                                ("description").item(0);
      Element pubDate = (Element) entry.getElementsByTagName
                                                ("pubDate").item(0);
      Element link = (Element) entry.getElementsByTagName
                                                ("link").item(0);
      String titleValue = title.getFirstChild().getNodeValue();
      String descriptionValue = description.getFirstChild().getNodeValue();
      String dateValue = pubDate.getFirstChild().getNodeValue();
      String linkValue = link.getFirstChild().getNodeValue();
      SingleNewsItem singleItem= new SingleNewsItem(
                 dateValue, titleValue, descriptionValue, linkValue);
      newsList.add(singleItem);
    } catch(DOMExceptione) {
      e.printStackTrace();
    }
  }//dissectNode
```

### 6.1 Servisi

Servis je komponenta aplikacije koje se izvršava u pozadini bez interakcije sa korisnikom i traje neodređeno vreme. I servisi, kao i ostali objekti aplikacije se izvršavaju u GLAVNOJ niti (UI thread) aplikacije. Ako servis treba da obavi neku dugu operaciju MORA da kreira novu nit za tu operaciju. Servis se u manifestu označava <service> tag-om. Servis se startuje sa Context.startService().

Životni ciklus Android servisa zahteva da se definišu tri public (callback) metode:

- void onCreate()
- void onStartCommand(Intent intent, int flags, int startId) – (void onStart(Intent intent, int startId) je zastareo od API5)
- void onDestroy()

Ceo životni ciklus servisa je između poziva onCreate() i onDestroy(). Sva inicijalizacija se obavlja u metodi onCreate() kao kod aktivnosti. Na primer: music player bi u onCreate() kreirao nit za puštanje zvučne datoteke, a u onDestroy() bi uništio tu nit.

Kostur koda Android servisa izgleda ovako:

```
Public class MyService3 extends Service {
  boolean isRunning= true;

  @Override
  public Ibinder onBind(Intent arg0) {
    return null;
  }

  @Override
  public void onCreate() {
    super.onCreate();
  }

  ....
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

### 6.2 Praktična vežba 6 - Mrežna komunikacija korišćenjem HTTP protokola

Nakon što smo obezbedili lako unošenje koordinata i omiljenih mesta i obezbedili njihov prikaz na mapi, vreme je da se posvetimo radu sa udaljenim serverom. Cilj ove vežbe je upoznavanje sa mogućnostima Android operativnog sistema za rad sa HTTP komunikacijom. Proučićemo metode slanja i primanja podataka korišćenjem HTTP POST metode. Za svrhe ove vežbe je razvijen jednostavan server koji prima i obrađuje zahteve dospele iz MyPlaces aplikacije. Pored navedenog, akcenat vežbe je i rad sa nitima u Androidu. Potrebno je omogućiti da se komunikacija sa serverom odvija u posebnim nitima kako ne bi uticala na korisnički interfejs. Kao i prethodne vežbe, i ova se oslanja na kod sa prethodnih laboratorijskih vežbi.

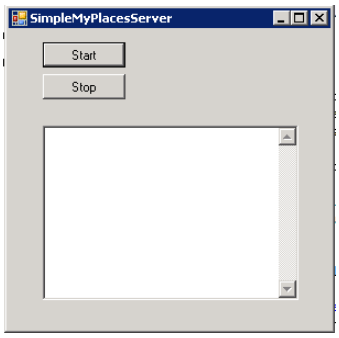
Ciljevi ove vežbe su:

1. Upoznavanje sa jednostavnom serverskom komponentom razvijenom za Windows operativno sistem u C# programskom jeziku.
2. Kreiranje zahteva za slanje podataka o omiljenom mestu na server korišćenjem HTTP POST metode.
3. Prikaz dostupnih podataka o mestima preuzetih sa servera. Prikaz imena lokacija izvršiti u posebnom dijalogu sa listom.
4. Preuzimanje izabranog omiljenog mesta sa servera i njegovo dodavanje u aplikaciju.

1. Upoznavanje sa jednostavnom serverskom komponentom razvijenom za Windows operativno sistem u C# programskom jeziku.
    a. Preuzeti aplikaciju servera sa sajta predmeta.
    b. Prilikom prvog pokretanja, aplikacija kreira folder myplaces u koji će smeštati tekstualne fajlove sa opisom svakog poslatog omiljenog mesta.
    c. Jednostavna serverska aplikacija osluškuje zahteve na portu 8080 i omogućava obradu tri vrste zahteva.
        i. Zahtev za slanjem podataka u omiljenom mestu pri čemu se primljeni podaci snimaju u tekstualni fajl. Fajl se smešta u folder myplaces na raćunaru.
        ii. Zahtev za listanje svih dostupnih omiljenih mesta na serveru. Aplikacija kao odgovor na ovaj zahtev šalje listu imena omiljenih mesta a na osnovu fajlova koji se nalaze u folderu myplaces.
        iii. Zahtev za preuzimanje omiljenog mesta na osnovu prosleđenog imena. Aplikacija vraća podatke o omiljenom mestu (ime, opis i koordinate).
    d. Svi zahtevi su u obliku HTTP POST zahteva pri čemu se u sadržaju POST zahteva navodi vrsta zahteva i odgovarajući podaci.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




2. Kreiranje zahteva za slanje podataka o omiljenom mestu na server korišćenjem HTTP POST metode.
   a. Za izabrano omiljeno mesto je moguće kreirati zahtev za slanje podataka. U kontekstni meni dodati MyPlacesList Activity-a stavku „Upload place“.

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo)
{
    super.onCreateContextMenu(menu, v, menuInfo);
    AdapterView.AdapterContextMenuInfo info = (AdapterView.AdapterContextMenuInfo)menuInfo;
    menu.setHeaderTitle( MyPlacesData.getInstance().getPlace(info.position).getName());
    menu.add(0, 1, 1, "View place");
    menu.add(0, 2, 2, "Edit place");
    menu.add(0, 3, 3, "Delete place");
    menu.add(0, 4, 4, "Show on map");
    menu.add(0, 5, 5, "Upload place");

}
```

   b. Komunikaciju sa serverom traje neko vreme i da ne bi uticala na korisnički interfejs, potrebno ju je izvršavati u posebnoj niti. Pored toga, sva osvežavanja stanja korisničkog interfejsa iz posebne niti je potrebno vršiti tako da bude bezbedno I da ne dođe u koliziju sa glavnom niti korisničkog interfejsa. Za ovo je potrebno koristiti concept Handler-a preko kojih se bezbedni uticaj na nit korisničkog interfejsa.
   c. Deklarisati promenljivu Handler guiThread koju ćemo koristiti za osvežavanje korisničkog interfejsa iz niti koja obavlja komunikaciju. Pored toga, da u ovom slučaju kontekst Activity-a bio dostupan, potrebno je deklarisati I promenljivu u kojoj će kontekst biti zapamćen. Zbog dužine komunikacije koristićemo jednostavan progres dijalog da bi obavestili korisnika da se komunikacija izvršava u pozadini.

```
public class MyPlacesList extends ListActivity{
    Handler guiThread;
    Context context;
    ProgressDialog progressDialog;
     public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setListAdapter(new ArrayAdapter<MyPlace>(this, android.R.layout.simple_list_item_1, MyPlacesData.getInstance().getM
        ListView lv = this.getListView();
        registerForContextMenu(lv);
        guiThread = new Handler();
        context = this;
        progressDialog = new ProgressDialog(this);
    }
```

   d. U metodi onContextItemSelected() dodati stavku koja obrađuje slanje omiljenog mesta.

```
else if(item.getItemId() == 5)
{

}
```

   e. Pribaviti ExecutorService koji ćemo koristiti za izvršavanje niti u pozadini. ExecutorService je moćni concept uveden u Java programski jezik koji je dostupan I u



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Androidu a omogućava jednostavniji rad sa nitima (slično ThreadPool-u kod programskih jezika namenjenih Windows-u). Dovoljno je ovakvom ExecutorService-u proslediti klasu koja implementira intefrejs Runnable i metodu run() da bi nit bila izvršena.

```
else if(item.getItemId() == 5)
{
    ExecutorService transThread = Executors.newSingleThreadExecutor();
    transThread.submit(new Runnable()
    {
        public void run() {

        }
    });
}
```

f. Ono što će biti izvršavano u posebnoj niti je potrebno smestiti unutar run() metode. Imajte na umu da sve promenljive koje su u dosegu ovog metoda a koje želimo da budu korišćene unutar njega moraju biti final. Da bi unutar run() metoda pribavili odgovarajuću omiljenu lokaciju potrebno je pribaviti poziciju mesta u listi a zatim je I pribaviti iz podataka.

```
else if(item.getItemId() == 5)
{
    ExecutorService transThread = Executors.newSingleThreadExecutor();
    final int position = info.position;
    transThread.submit(new Runnable()
    {
        public void run() {

            try {
                MyPlace place = MyPlacesData.getInstance().getPlace(position);

            } catch (Exception e) {
                e.printStackTrace();
            }

        }
    });
}
```

g. Kreirati posebnu klasu MyPlacesHTTPHelper.java. U ovoj klasi će biti implementirani statički metodi za komunikaciju. Dodati u klasu promenljive kojima ćemo identifikovati zahteve koji će biti slati serveru.

```
public class MyPlacesHTTPHelper {
    private static final String GET_MY_PLACE = "1";
    private static final String GET_ALL_PLACES_NAMES = "2";
    private static final String SEND_MY_PLACE = "3";
```

h. Kreirati statičku metodu String SendMyPlace(MyPlace place) u kojoj će biti vršena komunikacija. Obratite pažnju na IP adresu koja je prosleđena konstruktoru HttpPost objekta. Ova IP adresa se koristi ukoliko želimo da iz emulatora pristupamo računaru na kome se emulator izvršava. U ovom slučaju ne koristimo klasičnu localhost ili 127.0.0.1 loopback adresu vec 10.0.2.2. Pored toga navodimo i port na kome komuniciramo (da ne bi ometali podrazumevani HTTP protokol na portu 80).



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
public static String SendMyPlace(MyPlace place)
{
    HttpClient httpclient = new DefaultHttpClient();
    HttpPost httppost = new HttpPost("http://10.0.2.2:8080");
    String retStr;
    try {
        HttpResponse response = httpclient.execute(httppost);
        retStr = inputStreamToString(response.getEntity().getContent()).toString();
    } catch (Exception e) {
        e.printStackTrace();
        retStr = "Error during upload!";
    }
    return retStr;
}
```

i. Kreirati i pomoćnu statičku funkciju inputStreamToString koja čita odgovor sa servera i smešta ga u odgovarajući string.

```
private static StringBuilder inputStreamToString(InputStream is) {
    String line = "";
    StringBuilder total = new StringBuilder();
    BufferedReader rd = new BufferedReader(new InputStreamReader(is));
    try {
        while ((line = rd.readLine()) != null) {
            total.append(line);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return total;
}
```

j. Sada je preostalo da pripremimo sam zahtev. Zahtev treba da sadrži podataka o tipu zahteva ka serveru, ime omiljenog mesta koje šaljemo i podatke o mestu. Za slanje podataka o mestu možemo koristiti npr. XML generisan u aplikaciji na mobilnom. Ipak, često je generalna preporuka kod Android aplikacija korišćenje JSON-a za komunikaciju. Razlog za ovo je što je praktično svaki XML moguće kodirati i kao JSON string pri čemu JSON unosi značajno manje redundatnih podataka i značajno je manji od odgovarajućeg XML fajla. Ovo je vrlo značajno u uslovima ograničene i skupe komunikacije kod mobilnih telefona (srećom, sve je manje skupa i sve je brža).



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
public static String SendMyPlace(MyPlace place)
{
    HttpClient httpclient = new DefaultHttpClient();
    HttpPost httppost = new HttpPost("http://10.0.2.2:8080");
    String retStr;
    try {
        JSONObject holder = new JSONObject();
        JSONObject data = new JSONObject();
        data.put("name", place.getName());
        data.put("desc", place.getDesc());
        data.put("lon", place.getLongitude());
        data.put("lat", place.getLatitude());
        holder.put("myplace", data);

        List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(2);
        nameValuePairs.add(new BasicNameValuePair("req",SEND_MY_PLACE));
        nameValuePairs.add(new BasicNameValuePair("name", place.getName()));
        nameValuePairs.add(new BasicNameValuePair("payload", holder.toString()));
        httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));

        HttpResponse response = httpclient.execute(httppost);
        retStr = inputStreamToString(response.getEntity().getContent()).toString();
    } catch (Exception e) {
        e.printStackTrace();
        retStr = "Error during upload!";
    }
    return retStr;
}
```

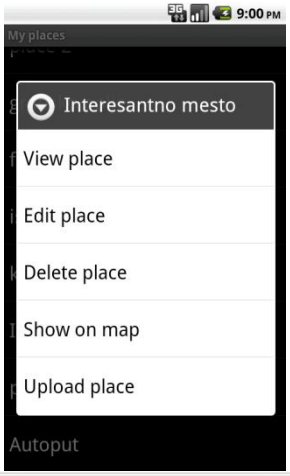
k. U MyPlacesList Activity-u dodati odgovarajući poziv implementiranoj metodi. Dobijeni string je potrebno prikazati kao Toast poruku. Za ovo nam je potrebna posebna funkcija koja može da interreaguje sa niti korisničkog interfejsa. Potrebno je i na početku izvršavanja niti prikazati progres dijalog i ukloniti ga prilikom završetka njenog rada.

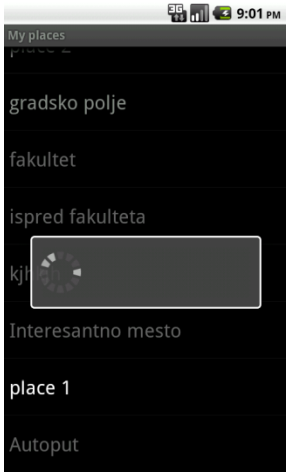
```
else if(item.getItemId() == 5)
{
    ExecutorService transThread = Executors.newSingleThreadExecutor();
    final int position = info.position;
    transThread.submit(new Runnable()
    {
        public void run() {
            guiProgressDialog(true);
            try {
                MyPlace place = MyPlacesData.getInstance().getPlace(position);
                final String message = MyPlacesHTTPHelper.SendMyPlace(place);
                guiNotifyUser(message);
            } catch (Exception e) {
                e.printStackTrace();
            }
            guiProgressDialog(false);
        }
    });
}
private void guiNotifyUser(final String message) {
    guiThread.post(new Runnable() {
        public void run() {
            Toast.makeText(context, message, Toast.LENGTH_SHORT).show();
        }
    });
}
private void guiProgressDialog(final boolean start) {
    guiThread.post(new Runnable() {
        public void run() {
            if(start)
                progressDialog.show();
            else
                progressDialog.dismiss();
        }
    });
}
```

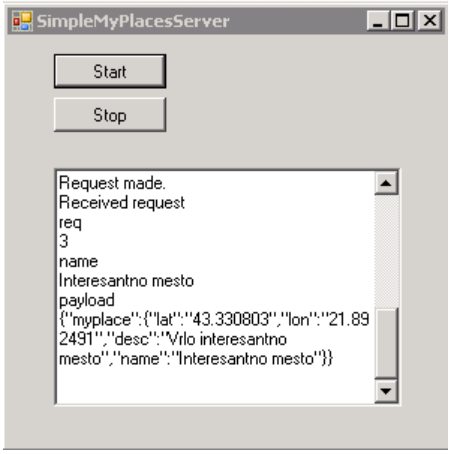


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

l. Probati aplikaciju.




3. Prikaz dostupnih podataka o mestima preuzetih sa servera. Prikaz imena lokacija izvršiti u posebnom dijalogu sa listom.
   a. Napraviti klasu MyPlacesServerList koja nasleđuje Dijalog i implementira interfejs OnClickListener interfejs. U klasi implementirati konstruktor i deklarisati promenljive za rad sa nitima slično kao u prethodnom primeru.

```
public class MyPlacesServerList extends Dialog implements android.view.View.OnClickListener{

    ListView lv;
    private Handler guiThread;
    private ExecutorService transThread;
    Context context;
    private ProgressDialog progressDialog;
    public MyPlacesServerList(Context context) {
        super(context);
        this.context = context;
        progressDialog = new ProgressDialog(context);
    }
```

   b. Kreirati odgovarajući XML layout fajl pod imenom serverlist.xml I dodati odgovarajuće stringove u strings.xml



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <ListView xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/serverlist"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" >
    </ListView>
    <Button
        android:id="@+id/serverlist_done_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="@string/serverlist_done_label" />
</LinearLayout>
```

c. U onCreate() metodi inicijalizovati kontrolu i dodati poziv za preuzimanje spiska imena lokacija dostupnih na server na sličan način kako je to urađeno u odeljku 2.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setTitle(R.string.myplaces_server_list);
    setContentView(R.layout.serverlist);
    lv = (ListView)findViewById(R.id.serverlist);
    registerForContextMenu(lv);
    lv.setOnCreateContextMenuListener(this);
    Button btnDone = (Button)findViewById(R.id.serverlist_done_button);
    btnDone.setOnClickListener(this);
    guiThread = new Handler();
    transThread = Executors.newSingleThreadExecutor();
    transThread.submit(new Runnable()
    {
        public void run() {
            guiProgressDialog(true);
            try {
                final List<String> names = MyPlacesHTTPHelper.getAllPlacesNames();
                guiSetAdapter(names);
            } catch (Exception e) {
                e.printStackTrace();
            }
            guiProgressDialog(false);
        }
    });
}

private void guiSetAdapter(final List<String> names) {
    guiThread.post(new Runnable() {
        public void run() {
            lv.setAdapter(new ArrayAdapter<String>(context, android.R.layout.simple_list_item_1, na
        }
    });
}
private void guiProgressDialog(final boolean start) {
    guiThread.post(new Runnable() {
        public void run() {
            if(start)
                progressDialog.show();
            else
                progressDialog.dismiss();
        }
    });
}
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

d. U klasu MyPlacesHTTPHelper.java dodati metodu getAllPlacesNames() koja preuzima sa server spisak lokacija. Obratiti pažnju da ova metoda od servera dobija spisak lokacija u vidu JSON stringa, parsira ga I priprema ArrayList-u imena lokacija koja će biti priosleđena adapteru za listu.

```
public static List<String> getAllPlacesNames()
{
    HttpClient httpclient = new DefaultHttpClient();
    HttpPost httppost = new HttpPost("http://10.0.2.2:8080");
    List<String> names = new ArrayList<String>();
    try {
        List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(2);
        nameValuePairs.add(new BasicNameValuePair("req",GET_ALL_PLACES_NAMES));
        httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));

        HttpResponse response = httpclient.execute(httppost);
        String str = inputStreamToString(response.getEntity().getContent()).toString();
        JSONObject jsonObject = new JSONObject(str);
        JSONArray jsonArray = jsonObject.getJSONArray("myplacesnames");
        for (int i = 0; i < jsonArray.length(); i++) {
            String name = jsonArray.getString(i);
            names.add(name);
        }
    } catch (Exception e) {
        e.printStackTrace();
        names.add("Problem downloading places' names list");
    }
    return names;
}
```

e. U MyPlacesList Activity-u dodati novu stavku options menija iz koga ćemo pozivati prikaz dijaloga.

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
    menu.add(0, 1, 1, "Add new place");
    menu.add(0, 2, 2, "Check places on server");
    return super.onCreateOptionsMenu(menu);
}

@Override public boolean onOptionsItemSelected(MenuItem item)
{
    Intent i = null;
    if(item.getItemId() == 1)
    {
        i = new Intent(this, EditMyPlaceActivity.class);
        startActivityForResult(i, 1);
    }
    else
    {
        Dialog v = new MyPlacesServerList(this);
        v.show();
    }
    return super.onOptionsItemSelected(item);
}
```

f. Probati aplikaciju.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

4. Preuzimanje izabranog omiljenog mesta sa servera i njegovo dodavanje u aplikaciju.
   a. Dodati konteksni meni kreirati konteksni meni za listu u dijalogu.
   b. Na klik na konteksni meni implementirati preuzimanje izabranog mesta sa servera.

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo)
{
    super.onCreateContextMenu(menu, v, menuInfo);
    AdapterView.AdapterContextMenuInfo info = (AdapterView.AdapterContextMenuInfo)menuInfo;
    String str = (String)lv.getItemAtPosition(info.position);
    menu.setHeaderTitle( str);
    menu.add(0, 1, 1, "Download " + str);
}
@Override
public boolean onMenuItemSelected(int featureId, MenuItem menuItem)
{
    AdapterView.AdapterContextMenuInfo info = (AdapterView.AdapterContextMenuInfo)menuItem.getMenuInfo();
    final String itemName = (String)lv.getItemAtPosition(info.position);
    transThread = Executors.newSingleThreadExecutor();
    transThread.submit(new Runnable()
    {
        public void run() {
            guiProgressDialog(true);
            try {
                final MyPlace place = MyPlacesHTTPHelper.getMyPlace(itemName);
                guiAddPlace(place);
            } catch (Exception e) {
                e.printStackTrace();
            }
            guiProgressDialog(false);
        }
    });

    return super.onMenuItemSelected(featureId, menuItem);
}
private void guiAddPlace(final MyPlace place) {
    guiThread.post(new Runnable() {
        public void run() {
            if(place == null)
                Toast.makeText(context, "Error during downloading of place" , Toast.LENGTH_SHORT).show();
            else
            {
                MyPlacesData.getInstance().addNewPlace(place);
                Toast.makeText(context, "Downloaded place with name:" + place.getName() , Toast.LENGTH_SHORT).show();
            }
        }
    });
}
```

c. U klasu MyPlacesHTTPHelper dodati statički metod getMyPlace() koji služi za preuzimanje izabrane lokacije.

```
public static MyPlace getMyPlace(String itemName)
{
    HttpClient httpclient = new DefaultHttpClient();
    HttpPost httppost = new HttpPost("http://10.0.2.2:8080");
    MyPlace place = null;
    try {
        List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(2);
        nameValuePairs.add(new BasicNameValuePair("req",GET_MY_PLACE));
        nameValuePairs.add(new BasicNameValuePair("name", itemName));
        httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));

        HttpResponse response = httpclient.execute(httppost);
        String str = inputStreamToString(response.getEntity().getContent()).toString();
        JSONObject jsonObject = new JSONObject(str);
        JSONObject jsonPlace = jsonObject.getJSONObject("myplace");
        String name = jsonPlace.getString("name");
        String desc = jsonPlace.getString("desc");
        String lon = jsonPlace.getString("lon");
        String lat = jsonPlace.getString("lat");
        place = new MyPlace(name, desc);
        place.setLongitude(lon);
        place.setLatitude(lat);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return place;
}
```

d. Implementirati i klik na dugme Done kako bi zatvorili dijalog nakon što završimo rad sa njim.

```
@Override
public void onClick(View v) {
    this.dismiss();
}
```

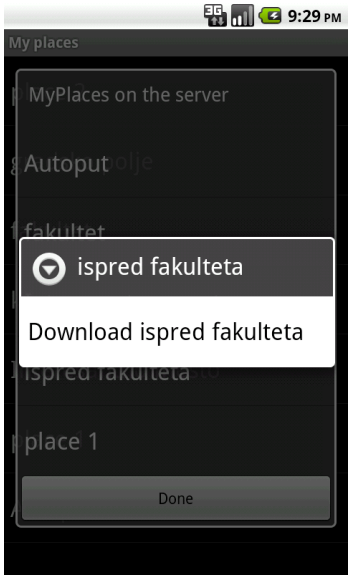
e. Da bi bili u mogućnosti da osvežimo listu nakon što je dijalog ugašen, dodati i postavljanje onDismissListener-a prilikom njegovog poziva u klasi MyPlacesList.java.

```
else
{
    Dialog v = new MyPlacesServerList(this);
    v.setOnDismissListener(new OnDismissListener()
    {
        @Override
        public void onDismiss(DialogInterface dialog) {
            setListAdapter(new ArrayAdapter<MyPlace>(context, android.R.layout.simple_list_item_1, MyPlacesData.getInstance().getMyPlaces()));
        }
    });
    v.show();
}
```

f. Probajte aplikaciju.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

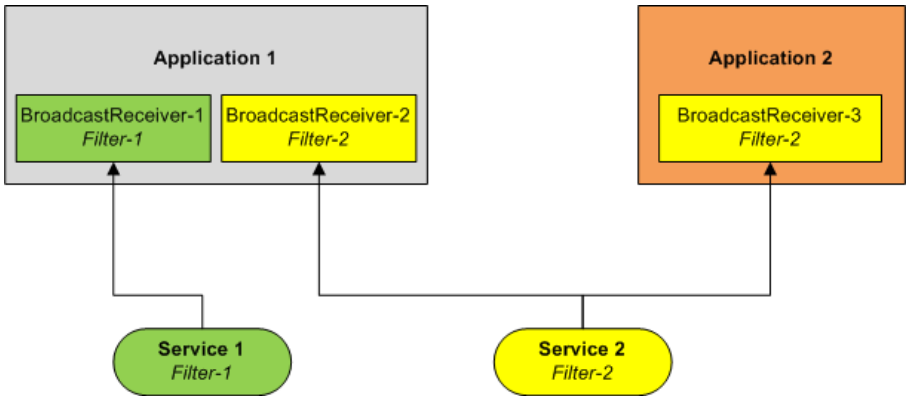


Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

# 7. Broadcast receiver-i i notifikacije

## 7.1 BroadcastReceiver

**BroadcastReceiver** je klasa čiji objekti osluškuju Intent-e koji su emitovani. Ne bave se Intent-ima koji su poslati konkretnoj aplikaciji/aktivnosti. Android runtime sistem isporučuje Broadcast Intent-e svim registrovanim broadcast receiver-ima koji ih obrađuju sekvencijalno (slika 7.1).




Slika 7.1 Funkcionisanje BroadcastReceiver-a

Broadcast receiver se može registrovati kao zainteresovan za neki Intent na dva načina:

- Programski - Context.registerReceiver()
- Statički - <receiver> tagom u manifestu AndroidManifest.xml

Kada nastane događaj za koji je Broadcast receiver registrovan Android sistem poziva callback metod

```
void onReceive(Context curContext, Intent broadcastMsg)
```

Broadcast receiver je aktivan samo dok izvršava onReceive metod. Preko parametra broadcastMsg ovaj metod dobija Intent koji je uhvaćen i koji predstavlja događaj koji je nastao

Definisana su dva tipa broadcast-a:

- Normalan - Šalje se sa Context.sendBroadcast(), potpuno su asinhroni i svi receiver-i se izvršavaju u slučajnom redosledu sekvencijalno.
- Uređeni - Šalje se sa Context.sendOrderedBroadcast(), obrađuju ih redom receiver-i pri čemu svaki receiver može da propagira broadcast dalje ili da prekine lanac.
    - Redosled receiver-a se definiše atributom android:priority odgovarajućeg intent-filter tag-a.
    - Svi receiver-i sa istim prioritetom se izvršavaju u proizvoljnom redosledu.

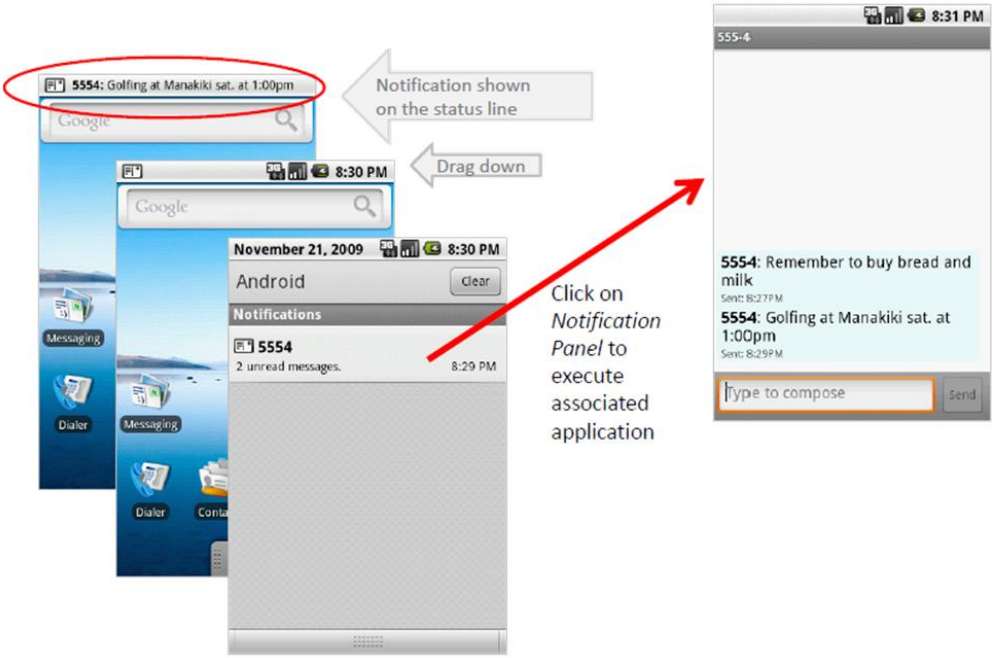
## 7.2 Notifikacija

Notifikacija na Android sistemu je kratka poruka koja se prikazuje na statusnoj liniji (status bar). Tipično na ovaj način obaveštavamo korisnika o nekom događaju u aplikaciji. Kada korisnik otvori notification panel ima mogućnost da klikne na notifikaciju i izvrši aktivnost povezanu za tu



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

notifikaciju. Tipičan primer za notifikaciju je dolazak SMS poruke. Klikom na tu notifikaciju se startuje aplikacija za pregled SMS poruka (slika 7.2).




Slika 7.2 Notifikacija o pristigloj SMS poruci

Klasa **NotificationManager** obaveštava korisnika da se nešto desilo u pozadini. U Androidu su definisana tri oblika notifikacija:

- Trajna ikona prikazana u status bar-u za koju se vezuje Intent i koju korisnik može da izabere.
- Uključeno trajno ili blinkanje LED indikatora na uređaju.
- Skrenuti pažnju korisnika blinkanjem pozadinskog osvetljenja, reprodukcijom zvuka ili vibracijom.

Referenca na **NotificationManager** se uzima kao sistemski servis:

```
notificationManager=(NotificationManager)
                    getSystemService(Context.NOTIFICATION_SERVICE);
```

Klasa **Notification** definiše kako će se prikazati notifikacija. Konstruktor ove klase je:

```
public Notification(int icon, CharSequence tickerText,long when);
```

Argumenti konstruktora su ikona, tekst i vreme koji će biti prikazani u status bar-u. Prikaz notifikacije obavlja NotificationManager metodom:

```
public void notify(int id, Notification notification)
```

Gde je id jedinstveni identifikator notifikacije u okviru aplikacije. Postavljanje detalja notifikacije obavlja se metodom koja je zastarela od API 11:

```
public void setLatestEventInfo(Context context, CharSequence
    contentTitle,CharSequence contentText, PendingIntent contentIntent)
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Argumenti:

- context – kontekst aplikacije
- contentTitle – naslov prikazan u proširenom prikazu
- contentText - tekst prikazan u proširenom prikazu
- contentIntent – Intent koji se šalje kada korisnik klikne na proširenu notifikaciju

Počev od API level 11 za kreiranje i postavljanje Notification objekta koristi se **Notification.Builder.**

```
Notification noti = new Notification.Builder(mContext)
   .setContentTitle("New mail from " + sender.toString())
   .setContentText(subject)
   .setContentIntent(pendIntent)
   .setSmallIcon(R.drawable.new_mail)
   .setLargeIcon(aBitmap)
   .setSound(uriSound)
   ...
   .build();
```

Možemo otkazati notifikaciju metodama:

- public void cancel(int id) gde je id prethodno definisan jedinstveni identifikator notifikacije
- public void cancelAll()

### 7.3 Senzori

Android mobilni telefoni imaju integrisane senzore koji omogućavaju opažanje okruženja telefona. Tipični primeri integrisanih senzora su:

- Akceleracioni senzor sa 3 ose,
- Magnetni senzor sa 3 ose,
- Orijentacioni senzor,
- Senzor blizine (proximity),
- Senzor intenziteta svetlosti,
- Temperaturni sensor.

Sistemskom servisu koji kontroliše senzore se pristupa preko klase **SensorManager.** Instanca klase SensorManager se pribavlja kao i svi sistemski servisi:

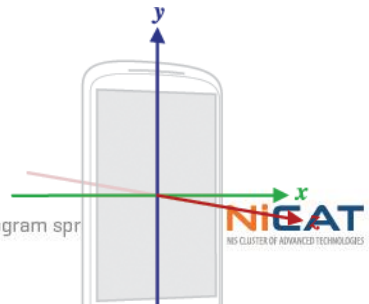
```
getSystemService(Context.SENSOR_SERVICE)
```

Konkretnom senzoru se pristupa preko objekta klase **Sensor**

```
Sensor s = SensorManager.getDefaultSensor(int type)
```

Tipovi senzora (argument type) mogu biti:

- TYPE_ACCELEROMETER
- TYPE_GRAVITY



- TYPE_GYROSCOPE
- TYPE_LIGHT
- TYPE_MAGNETIC_FIELD
- TYPE_PRESSURE
- TYPE_PROXIMITY
- TYPE_TEMPERATURE

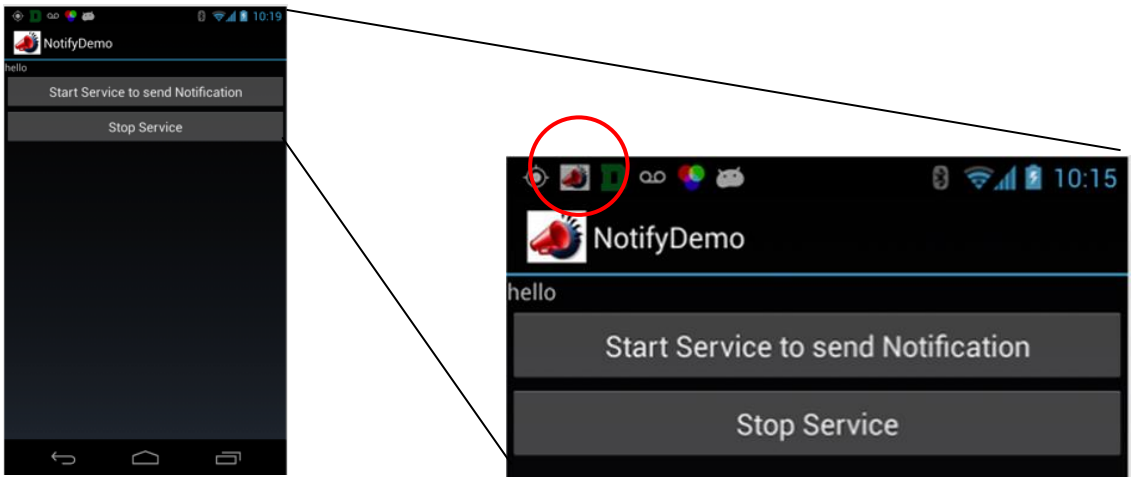
Senzor šalje asinhrono podatke. Klasa SensorEvent sadrži podatke specifične za sensor: tip senzora, vreme uzorkovanja, preciznost, izmerene vrednosti. Koordinatni sistem integrisanih senzora se ne menja sa orijentacijom uređaja (landscape/portrait). Interfejs **SensorEventListener** deklariše callback metode:

- void onAccuracyChanged(Sensor sensor, int accuracy)- Poziva se kada se promeni preciznost senzora.
- void onSensorChanged(SensorEvent event)- Poziva se kada se dobiju novi podaci sa senzora.

Ovaj listener se registruje korišćenjem **SensorManager** instance (objekta)

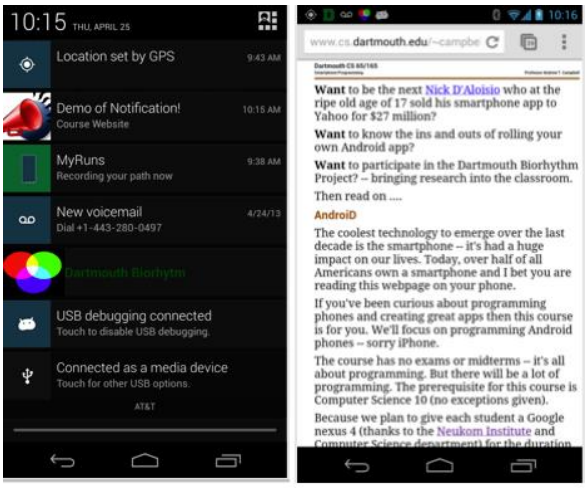
- public boolean registerListener(SensorEventListener listener, Sensor sensor, int rate)
- public void unregisterListener(SensorEventListener listener, Sensor sensor)

Primer aplikacije sa notifikacijom, broadcast receiver-ima i servisima – **NotifyDemo**. Izgled aktivnosti ovog pojekta prikazan je na slici 7.3, a ceo Android projekat se može preuzeti na sledećem linku: (http://www.cs.dartmouth.edu/~campbell/cs65/code/notifydemo.zip)






Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES




Slika 7.3 Izgled i tok aktivnosti u NotifyDemo projektu

**NotifyActivity** klasa je definisana na sledeći način:

```
public class NotifyActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button buttonStartService = (Button)findViewById(R.id.startservice);
        Button buttonStopService = (Button)findViewById(R.id.stopservice);

        buttonStartService.setOnClickListener(new Button.OnClickListener(){

            @Override
            public void onClick(View arg0) {

                Intent intent = new Intent(NotifyActivity.this, NotifyService.class);
                NotifyActivity.this.startService(intent);
            }});

        buttonStopService.setOnClickListener(new Button.OnClickListener(){

            @Override
            public void onClick(View arg0) {

                Intent intent = new Intent();
                intent.setAction(NotifyService.ACTION);
                intent.putExtra(NotifyService.STOP_SERVICE_BROADCAST_KEY,
                            NotifyService.RQS_STOP_SERVICE);

                    // Broadcast the given intent to all interested BroadcastReceivers
                sendBroadcast(intent);

            }});

    }
}
```

Klasa **NotifyService** je definisana sledećim programskim kodom.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```java
public class NotifyService extends Service {

    final static String ACTION = "NotifyServiceAction";
    final static String STOP_SERVICE_BROADCAST_KEY="StopServiceBroadcastKey";
    final static int RQS_STOP_SERVICE = 1;

    NotifyServiceReceiver notifyServiceReceiver;



    private final String myBlog = "http://www.cs.dartmouth.edu/~campbell/cs65/cs65.html";

    @Override
    public void onCreate() {

        notifyServiceReceiver = new NotifyServiceReceiver();
        super.onCreate();
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {


        IntentFilter intentFilter = new IntentFilter();
        intentFilter.addAction(ACTION);
        registerReceiver(notifyServiceReceiver, intentFilter);

        // Send Notification
        String notificationTitle = "Demo of Notification!";
        String notificationText = "Course Website";
        Intent myIntent = new Intent(Intent.ACTION_VIEW, Uri.parse(myBlog));
        PendingIntent pendingIntent
                = PendingIntent.getActivity(getBaseContext(),
                        0, myIntent,
                        Intent.FLAG_ACTIVITY_NEW_TASK);
```

```java
        Notification notification = new Notification.Builder(this)
        .setContentTitle(notificationTitle)
        .setContentText(notificationText).setSmallIcon(R.drawable.icon)
        .setContentIntent(pendingIntent).build();
        NotificationManager notificationManager =
                (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
        notification.flags = notification.flags
                | Notification.FLAG_ONGOING_EVENT;
        notification.flags |= Notification.FLAG_AUTO_CANCEL;

        notificationManager.notify(0, notification);

        return super.onStartCommand(intent, flags, startId);
    }

    @Override
    public void onDestroy() {

        this.unregisterReceiver(notifyServiceReceiver);
        super.onDestroy();
    }

    @Override
    public IBinder onBind(Intent arg0) {

        return null;
    }

    public class NotifyServiceReceiver extends BroadcastReceiver{

        @Override
        public void onReceive(Context arg0, Intent arg1) {

            int rqs = arg1.getIntExtra(STOP_SERVICE_BROADCAST_KEY, 0);

            if (rqs == RQS_STOP_SERVICE){
                stopSelf();
                ((NotificationManager) getSystemService(NOTIFICATION_SERVICE))
                .cancelAll();
            }
        }
    }

}
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

Servis je definisan u manifest-u:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="edu.dartmouth.cs.notifydemo"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="17" />

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name="edu.dartmouth.cs.notifydemo.NotifyActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name="edu.dartmouth.cs.notifydemo.NotifyService"/>
    </application>
</manifest>
```

### 7.4 Praktična vežba 7 - Broadcast receiver-i i proximity Alert-ovi

Poslednja vežba ovog kursa je posvećena radu sa Proximity Alert-ima. Ovo je korisna funkcionalnost koju Android operativni sistem obezbeđuje svojim korisnicima. Kosrišćenje Proximity Alert-a nam daje mogučnost da dobijamo obaveštenja od sistema kada smo napustili ili ušli u postavljeni radijus oko neke lokacije od interesa. U ovoj vežbi ćemo omogućiti dodavanje proximity alert-ova koji obaveštavaju korisnika kada je u blizini ili je napustio svoju omiljenu lokaciju. Vežba podrazumeva postojanje prethodnog koda.

Ciljevi ove vežbe su:

1. Kreiranje Activity-a za unos Proximity Alert-a vezanog za određenu lokaciju.
2. Prepravljanje baze tako da se u njoj pamte podaci o dodatom Proximity Alertu.
3. Dodavanje logike unosa Proximity Alert-a.
4. Kreiranje Broatcast Receiver-a koji obaveštava korisnika o događaju.

1. Da bi mogli da vršimo unos Proximity Alert-a za određenu omiljenu lokaciju, potrebno je kreirati odgovarajući Activity.
    a. Za svaki Proximity Alert je moguće definisati radijus. Najzgodnije je koristiti predefinisane vrednosti. Za ove namene možemo kreirati poseban xml fajl u koji ćemo smestiti predefinisane vrednosti za radijus. Kreirati fajl distances.xml u folderu values.
    b. U fajl distances.xml dodati odgovarajući string-array i integer-array.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="radiusStrings">
        <item name="1">1 meters</item>
        <item name="2">2 meters</item>
        <item name="3">3 meters</item>
        <item name="5">5 meters</item>
        <item name="10">10 meters</item>
        <item name="25">25 meters</item>
        <item name="50">50 meters</item>
        <item name="100">100 meters</item>
        <item name="250">250 meters</item>
        <item name="500">500 meters</item>
        <item name="1000">1000 meters</item>
    </string-array>
    <integer-array name="radiusValues">
        <item name="1">1</item>
        <item name="2">2</item>
        <item name="3">3</item>
        <item name="5">5</item>
        <item name="10">10</item>
        <item name="25">25</item>
        <item name="50">50</item>
        <item name="100">100</item>
        <item name="250">250</item>
        <item name="500">500</item>
        <item name="1000">1000</item>
    </integer-array>
</resources>
```

c. Kreirati odgovarajući layout pod imenom proximity.xml.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
     <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/proximity_messagetext_label"
        />
     <EditText
        android:id="@+id/proximity_messagetext_edit"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
         />
     <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/proximity_distance_label"
        />
     <Spinner
        android:id="@+id/proximity_distance_spinner"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:entries="@array/radiusStrings"
        android:prompt="@string/proximity_distance_prompt" />
     <TableRow
         android:layout_width="fill_parent"
         android:layout_height="wrap_content" >
         <Button
             android:id="@+id/proximity_finished_button"
             android:layout_weight="1" />
         <Button
             android:id="@+id/proximity_cancel_button"
             android:layout_weight="1"
             android:text="@string/proximity_cancel_label" />
     </TableRow>
</LinearLayout>
```

d. Kreirati klasu MyPlacesProximityActivity.java i u njoj metod onCreate koji postavlja odgovarajući view.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
public class MyPlacesProximityActivity extends Activity


    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.proximity);
```

2. Prepraviti bazu podataka i kreirati klasu za pamćenje jednog ProximityAlert-a.
   a. Kreirati klasu MyPlaceProximity.java.

```
package org.elfak.modis.myplaces;

public class MyPlaceProximity {
    long placeId;
    String alertText;
    float radius;

    public long getPlaceID()
    {
        return this.placeId;
    }
    public String getAlertText()
    {
        return this.alertText;
    }
    public float getRadius()
    {
        return this.radius;
    }
    public void setPlaceID(long id)
    {
        this.placeId = id;
    }
    public void setAlertText(String alert)
    {
        this.alertText = alert;
    }
    public void setRadius(float rad)
    {
        this.radius = rad;
    }
}
```

   b. U klasi MyPlacesDBAdapter.java dodati stringove za ime nove tabele i imena njenih kolona.

```
public static final String DATABASE_TABLE2 = "Proximity";
public static final String PROXIMITY_PLACE_ID = "Place_ID";
public static final String PROXIMITY_ALERTTEXT = "Alert";
public static final String PROXIMITY_RADIUS = "Dist";
```

   c. Izmeniti klasu MyPlacesDatabaseHelper.java tako da podržava novu tabelu.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```java
package org.elfak.modis.myplaces;

import android.content.Context;

public class MyPlacesDatabaseHelper extends SQLiteOpenHelper {
    // SQL naredba za kreiranje nove tabele
    private static final String DATABASE_CREATE = "create table " + MyPlacesDBAdapter.DATABASE_TABLE + " ("
        + MyPlacesDBAdapter.PLACE_ID + " integer primary key autoincrement, "
        + MyPlacesDBAdapter.PLACE_NAME + " text unique not null, "
        + MyPlacesDBAdapter.PLACE_DESCRIPTION + " text, "
        + MyPlacesDBAdapter.PLACE_LONG + " text, "
        + MyPlacesDBAdapter.PLACE_LAT + " text, "
        + MyPlacesDBAdapter.PLACE_FILENAME + " text);";
    private static final String DATABASE_CREATE2 = "create table " + MyPlacesDBAdapter.DATABASE_TABLE2 + " ("
        //+ MyPlacesDBAdapter.PROXIMITY_ID + " integer, "
        + MyPlacesDBAdapter.PROXIMITY_PLACE_ID + " integer primary key, "
        + MyPlacesDBAdapter.PROXIMITY_ALERTTEXT + " text, "
        + MyPlacesDBAdapter.PROXIMITY_RADIUS + " float);"


    public MyPlacesDatabaseHelper(Context context, String name,
            CursorFactory factory, int version) {
        super(context, name, factory, version);
        // TODO Auto-generated constructor stub
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        try{
            db.execSQL(DATABASE_CREATE);
            db.execSQL(DATABASE_CREATE2);
        } catch(SQLiteException e) {
            Log.w("MyPlacesDatabaseHelper", e.getMessage());
        }
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Log the version upgrade.
        Log.w("TaskDBAdapter", "Upgrading from version " + oldVersion + " to " + newVersion
                + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS " + MyPlacesDBAdapter.DATABASE_TABLE);
        db.execSQL("DROP TABLE IF EXISTS " + MyPlacesDBAdapter.DATABASE_TABLE2);
        onCreate(db);

    }
}
```

d. Dodati metode za unos i čitanje iz baze podataka o Proximity Alert-ovima.

```java
public long insertProximityEntry(MyPlaceProximity prox) {
    ContentValues contentValues = new ContentValues();

    contentValues.put(PROXIMITY_PLACE_ID, prox.getPlaceID());
    contentValues.put(PROXIMITY_ALERTTEXT, prox.getAlertText());
    contentValues.put(PROXIMITY_RADIUS, prox.getRadius());
    long id = -1;
    db.beginTransaction();
    try{
        id = db.insert(DATABASE_TABLE2, null, contentValues);
        db.setTransactionSuccessful();
    } catch(SQLiteException e) {
        Log.w("MyPlacesDBAdapter", e.getMessage());
    } finally {
        db.endTransaction();
    }
    return id;
}

public boolean removeProximityEntry(long myPlaceId) {
    boolean success= false;
    db.beginTransaction();
    try{
        success = db.delete(DATABASE_TABLE2, PROXIMITY_PLACE_ID+"=" + myPlaceId, null) > 0;
        db.setTransactionSuccessful();
    } catch(SQLiteException e) {
        Log.w("MyPlacesDBAdapter", e.getMessage());
    } finally {
        db.endTransaction();
    }
    return success;
}
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
public MyPlaceProximity getProximityEntry(long myPlaceId) {
    MyPlaceProximity proximity = null;
    Cursor cursor = null;
    db.beginTransaction();
    try{
        cursor = db.query(DATABASE_TABLE2, null, PROXIMITY_PLACE_ID +"=" + myPlaceId, null, null, null, null);
        db.setTransactionSuccessful();
    } catch(SQLiteException e) {
        Log.w("MyPlacesDBAdapter", e.getMessage());
    } finally {
        db.endTransaction();
    }
    if(cursor!= null)
    {
        if(cursor.moveToFirst()){
            proximity = new MyPlaceProximity();
            proximity.setPlaceID(cursor.getLong(cursor.getColumnIndex(MyPlacesDBAdapter.PROXIMITY_PLACE_ID)));
            proximity.setAlertText(cursor.getString(cursor.getColumnIndex(MyPlacesDBAdapter.PROXIMITY_ALERTTEXT)));
            proximity.setRadius(cursor.getFloat(cursor.getColumnIndex(MyPlacesDBAdapter.PROXIMITY_RADIUS)));
        }
    }
    return proximity;
}

public int updateProximityEntry(long myPlaceId, MyPlaceProximity proximity) {
    String where = PROXIMITY_PLACE_ID + "=" + myPlaceId;

    ContentValues contentValues = new ContentValues();

    contentValues.put(PROXIMITY_ALERTTEXT, proximity.getAlertText());
    contentValues.put(PROXIMITY_RADIUS, proximity.getRadius());

    return db.update(DATABASE_TABLE2, contentValues, where, null);
}

  public ArrayList<MyPlaceProximity> getAllProximityEntries () {
      ArrayList<MyPlaceProximity> proximities = null;
      Cursor cursor = null;
      db.beginTransaction();
      try{
          cursor = db.query(DATABASE_TABLE2, null ,null, null, null, null, null);
          db.setTransactionSuccessful();
      } catch(SQLiteException e) {
          Log.w("MyPlacesDBAdapter", e.getMessage());
      } finally {
          db.endTransaction();
      }
      if(cursor!= null)
      {
          proximities = new ArrayList<MyPlaceProximity>();
          MyPlaceProximity proximity = null;
          while(cursor.moveToNext()){
              proximity = new MyPlaceProximity();
              proximity.setPlaceID(cursor.getLong(cursor.getColumnIndex(MyPlacesDBAdapter.PROXIMITY_PLACE_ID)));
              proximity.setAlertText(cursor.getString(cursor.getColumnIndex(MyPlacesDBAdapter.PROXIMITY_ALERTTEXT)));
              proximity.setRadius(cursor.getFloat(cursor.getColumnIndex(MyPlacesDBAdapter.PROXIMITY_RADIUS)));
              proximities.add(proximity);
          }
      }
      return proximities;
  }
```

e. Dodati odgovarajuće metode i u klasu MyPlacesData.java za rad sa Proximity Alert-ima i prepraviti konstruktor.

```
private ArrayList<MyPlace> myPlaces;
private ArrayList<MyPlaceProximity> proximities;
private static MyPlacesData instance;
MyPlacesDBAdapter dbAdapter;
private MyPlacesData()
{
    dbAdapter = new MyPlacesDBAdapter(MyPlacesApplication.getContext());
    dbAdapter.open();
    this.myPlaces = dbAdapter.getAllEntries();
    this.proximities = dbAdapter.getAllProximityEntries();
    dbAdapter.close();
}
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
public MyPlace getMyPlaceById(long myPlaceId)
{
    MyPlace found = null;
    int i = 0;
    while(i<myPlaces.size() && found == null)
    {
        if(myPlaces.get(i).getID() == myPlaceId)
            found = myPlaces.get(i);
        i++;
    }
    return found;
}
public ArrayList<MyPlaceProximity> getProximities()
{
    return proximities;
}
public MyPlaceProximity getProximity(int index)
{
    return proximities.get(index);
}
public void addNewProximity(MyPlaceProximity proximity)
{
    proximities.add(proximity);
    dbAdapter.open();
    dbAdapter.insertProximityEntry(proximity);
    dbAdapter.close();
}
public MyPlaceProximity getProximityForPlace(long myPlaceId)
{
    MyPlaceProximity found = null;
    int i = 0;
    while(i<proximities.size() && found == null)
    {
        if(proximities.get(i).getPlaceID() == myPlaceId)
            found = proximities.get(i);
        i++;
    }
    return found;
}
public void deleteProximityByPlaceID(long placeID)
{
    MyPlaceProximity proximity = getProximityForPlace(placeID);
    proximities.remove(proximity);
    dbAdapter.open();
    boolean success = dbAdapter.removeProximityEntry(placeID);
    dbAdapter.close();
}
public void updateProximity(MyPlaceProximity proximity)
{
    dbAdapter.open();
    dbAdapter.updateProximityEntry(proximity.getPlaceID(), proximity);
    dbAdapter.close();
}
```

3. Dodavanje logike unosa Proximity Alert-a.
    a. Dodati odgovarajuće promenljive u klasu MyPlacesProximityActivity.java za LocationManager i podršku za editovanje/unos (slično kao u EditMyPlaceActivity.java).



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```java
public class MyPlacesProximityActivity extends Activity implements OnClickListener, TextWatcher{
    boolean editMode = true;
    int myPlacePosition;
    LocationManager locationManager;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.proximity);
        locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
        try
        {
            Intent listIntent = getIntent();
            Bundle positionBundle = listIntent.getExtras();
            if(positionBundle != null)
            {
                myPlacePosition = positionBundle.getInt("myplaceposition");
                if(myPlacePosition>=0)
                {
                    MyPlace place = MyPlacesData.getInstance().getPlace(myPlacePosition);
                    MyPlaceProximity proximity = MyPlacesData.getInstance().getProximityForPlace(place.getID());
                    if(proximity!=null)
                        editMode = true;
                    else
                        editMode = false;
                }
            }
            else
                editMode = false;
        }
        catch(Exception e) {
            editMode = false;
        }
        Button finishedButton = (Button) findViewById(R.id.proximity_finished_button);
        Button cancelButton = (Button) findViewById(R.id.proximity_cancel_button);
        EditText alertText = (EditText) findViewById(R.id.proximity_messagetext_edit);
        finishedButton.setOnClickListener(this);
        if(!editMode)
        {
            finishedButton.setEnabled(false);
            finishedButton.setText("Add");
        }
        else
        {
            finishedButton.setText("Save");
            MyPlace place = MyPlacesData.getInstance().getPlace(myPlacePosition);
            MyPlaceProximity proximity = MyPlacesData.getInstance().getProximityForPlace(place.getID());
            alertText.setText(proximity.getAlertText());
            Spinner distanceSpinner = (Spinner)findViewById(R.id.proximity_distance_spinner);
            int[] distancesArray = getResources().getIntArray(R.array.radiusValues);
            int pos = -1;
            int i = 0;
            while(pos==-1 && i<distancesArray.length)
            {
                if(distancesArray[i] == (int)proximity.getRadius())
                    pos = i;
                i++;
            }
            if(pos!=-1)
                distanceSpinner.setSelection(pos);
        }
        finishedButton.setOnClickListener(this);
        cancelButton.setOnClickListener(this);
        alertText.addTextChangedListener(this);
    }
```

b. Dodati onClick, afterTextChanged, beforeTextChanged i onTextChanged metode (dodati i implementaciju odgovarajućih Listenera Interface-a). OnClick metod je suštinski bitan i u njemu, u zavisnosti od toga da li editujemo ili kreiramo Proximity Alert pozivamo odgovarajuće metode za kreiranje ili brisanje Proximity Alert-ova.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
@Override
public void onClick(View v) {
        switch(v.getId())
        {
            case R.id.proximity_finished_button:
            {
                EditText alertText = (EditText)findViewById(R.id.proximity_messagetext_edit);
                String alertString = alertText.getText().toString();

                if(!editMode)
                {
                    MyPlaceProximity proximity = new MyPlaceProximity();
                    proximity.setAlertText(alertString);
                    Spinner distanceSpinner = (Spinner)findViewById(R.id.proximity_distance_spinner);
                    int[] distancesArray = getResources().getIntArray(R.array.radiusValues);
                    int distance = distancesArray[distanceSpinner.getSelectedItemPosition()];
                    proximity.setRadius(distance);
                    MyPlace place = MyPlacesData.getInstance().getPlace(myPlacePosition);
                    proximity.setPlaceID(place.getID());
                    MyPlacesData.getInstance().addNewProximity(proximity);
                    addProximity(MyPlacesData.getInstance().getPlace(myPlacePosition), proximity);
                }
                else
                {
                    MyPlace place = MyPlacesData.getInstance().getPlace(myPlacePosition);
                    MyPlaceProximity proximity = MyPlacesData.getInstance().getProximityForPlace(place.getID());
                    if(proximity!=null)
                    {
                        proximity.setAlertText(alertString);
                        Spinner distanceSpinner = (Spinner)findViewById(R.id.proximity_distance_spinner);
                        int[] distancesArray = getResources().getIntArray(R.array.radiusValues);
                        int distance = distancesArray[distanceSpinner.getSelectedItemPosition()];
                        MyPlacesData.getInstance().updateProximity(proximity);
                        removeProximityIntent(place);
                        addProximity(place, proximity);
                    }
                }
                setResult(Activity.RESULT_OK);
                finish();
                break;
            }

            case R.id.proximity_cancel_button:
            {
                setResult(Activity.RESULT_CANCELED);
                finish();
                break;
            }
        }
}
@Override
public void afterTextChanged(Editable arg0) {
    // TODO Auto-generated method stub

}
@Override
public void beforeTextChanged(CharSequence arg0, int arg1, int arg2,
        int arg3) {
    // TODO Auto-generated method stub

}
@Override
public void onTextChanged(CharSequence s, int start, int before,int count) {
    EditText alertText = (EditText)findViewById(R.id.proximity_messagetext_edit);
    Button finishedButton = (Button)findViewById(R.id.proximity_finished_button);
    if(alertText.getText().toString().length()>0)
        finishedButton.setEnabled(true);
    else
        finishedButton.setEnabled(false);
}
```

c. Kreirati odgovarajuće metode za kreiranje i brisanje Proximity Alert-a.



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
private static final String PROX_ALERT_INTENT = "org.elfak.modis.myplaces.ProximityAlert";
public void addProximity(MyPlace place, MyPlaceProximity proximity)
{
    Intent intent = new Intent(PROX_ALERT_INTENT);
    intent.putExtra("placeID", place.getID());
    PendingIntent proximityIntent = PendingIntent.getBroadcast(this, (int)place.getID(), intent, 0);
    double lat = Double.parseDouble(place.getLatitude());
    double lon = Double.parseDouble(place.getLongitude());
    float radius = proximity.getRadius();
    locationManager.addProximityAlert(lat, lon, radius, -1, proximityIntent);
}
public void removeProximityIntent(MyPlace place)
{
    Intent intent = new Intent(PROX_ALERT_INTENT);
    intent.putExtra("placeID", place.getID());
    PendingIntent proximityIntent = PendingIntent.getBroadcast(this, (int)place.getID(), intent, 0);
    locationManager.removeProximityAlert(proximityIntent);
}
```

d. Potrebno je u MyPlacesList.java klasi povezati poziv kreiranog Activity-a…

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo)
{
    super.onCreateContextMenu(menu, v, menuInfo);
    AdapterView.AdapterContextMenuInfo info = (AdapterView.AdapterContextMenuInfo)menuInfo;
    MyPlace place = MyPlacesData.getInstance().getPlace(info.position);
    menu.setHeaderTitle( place.getName());
    menu.add(0, 1, 1, "View place");
    menu.add(0, 2, 2, "Edit place");
    menu.add(0, 3, 3, "Delete place");
    menu.add(0, 4, 4, "Show on map");
    menu.add(0, 5, 5, "Upload place");
    if(MyPlacesData.getInstance().getProximityForPlace(place.getID()) == null)
        menu.add(0, 6, 6, "Add proximity");
    else
    {
        menu.add(0, 6, 6, "Edit proximity");
        menu.add(0, 7, 7, "Delete proximity");
    }
}
```

e. …i dodati nove stavke u metodi onContextItemSelected

```
else if(item.getItemId() == 6)
{
    i = new Intent(this, MyPlacesProximityActivity.class);
    i.putExtra("myplaceposition", info.position);
    startActivityForResult(i, 3);
}
else
{
    MyPlace place = MyPlacesData.getInstance().getPlace(info.position);
    removeProximityIntent(place);
    MyPlacesData.getInstance().deleteProximityByPlaceID(place.getID());
}
```

f. Obratiti pažnju i da u istoj metodi, brisanje omiljene lokacije bi trebalo da bude usklađeno i da omogućava brisanje i Proximity Alert-a. Stoga je potrebno metodu za brisanje Proximity Alert-a iskopirati i u ovu klasu.

```
else if(item.getItemId() == 3)
{
    MyPlace place = MyPlacesData.getInstance().getPlace(info.position);
    if(MyPlacesData.getInstance().getProximityForPlace(place.getID())!=null)
    {
        removeProximityIntent(place);
        MyPlacesData.getInstance().deleteProximityByPlaceID(place.getID());
    }

    MyPlacesData.getInstance().deletePlace(info.position);
    setListAdapter(new ArrayAdapter<MyPlace>(this, android.R.layout.simple_list_item_1, MyPlacesData.getInstance().getMyPlaces()));
}
```



Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
        return super.onContextItemSelected(item);
    }
    private static final String PROX_ALERT_INTENT = "org.elfak.modis.myplaces.ProximityAlert";
    public void removeProximityIntent(MyPlace place)
    {
        Intent intent = new Intent(PROX_ALERT_INTENT);
        intent.putExtra("placeID", place.getID());
        PendingIntent proximityIntent = PendingIntent.getBroadcast(this, (int)place.getID(), intent, 0);
        LocationManager locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
        locationManager.removeProximityAlert(proximityIntent);
    }

    private void guiNotifyUser(final String message) {
```

4. Kreiranje Broatcast Receiver-a koji obaveštava korisnika o događaju.
   a. Sada je sve spremno da dodamo našu implementaciju BroadcastReceiver-a koja će da hvata događaje i obaveštava korisnika.

```
public class MyPlaceProximityIntentReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        boolean entering = intent.getBooleanExtra(LocationManager.KEY_PROXIMITY_ENTERING, false);
        NotificationManager notificationManager = (NotificationManager) context.getSystemService(Context.NOTIFICATION_SERVICE);

        long placeID = intent.getExtras().getLong("placeID");
        Notification notification = createNotification();
        MyPlace place = MyPlacesData.getInstance().getMyPlaceById(placeID);
        MyPlaceProximity proximity = MyPlacesData.getInstance().getProximityForPlace(placeID);
        if(proximity!=null)
        {
            PendingIntent pendingIntent = PendingIntent.getActivity(context, 0, null, 0);
            String placeString = null;
            if(entering)
            {
                placeString = "Approaching to place:" + place.getName();
            }
            else
            {
                placeString = "Leaving the place:" + place.getName();
            }
            notification.setLatestEventInfo(context, placeString, proximity.getAlertText(), pendingIntent);
            notificationManager.notify(500, notification);
        }
    }


    private Notification createNotification() {
        Notification notification = new Notification();

        notification.icon = R.drawable.ic_launcher;
        notification.when = System.currentTimeMillis();

        notification.flags |= Notification.FLAG_AUTO_CANCEL;
        notification.flags |= Notification.FLAG_SHOW_LIGHTS;

        notification.defaults |= Notification.DEFAULT_VIBRATE;
        notification.defaults |= Notification.DEFAULT_LIGHTS;

        notification.ledARGB = Color.WHITE;
        notification.ledOnMS = 1500;
        notification.ledOffMS = 1500;

        return notification;
    }
}
```

   b. Potrebno je kreirani BroatcastReceiver i registrovati u manifestu aplikacije kako bi stalno imali obaveštenja a ne samo dok je aktivan Activity iz koga bi bio programski registrovan.

```
        <activity android:name=".MyPlacesProximityActivity">
        </activity>
        <receiver android:name=".MyPlaceProximityIntentReceiver" android:enabled="true">
            <intent-filter>
                    <action android:name="org.elfak.modis.myplaces.ProximityAlert"/>
            </intent-filter>
        </receiver>
```

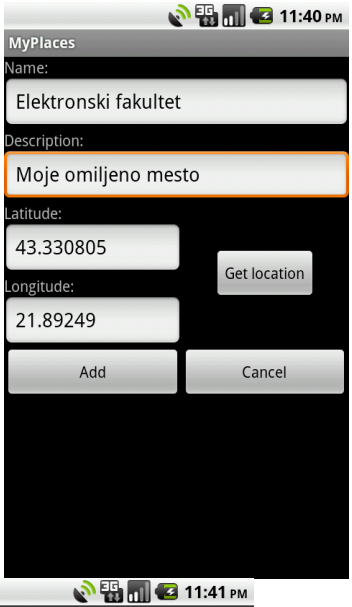
   c. Dodajmo i još dva permission-a.

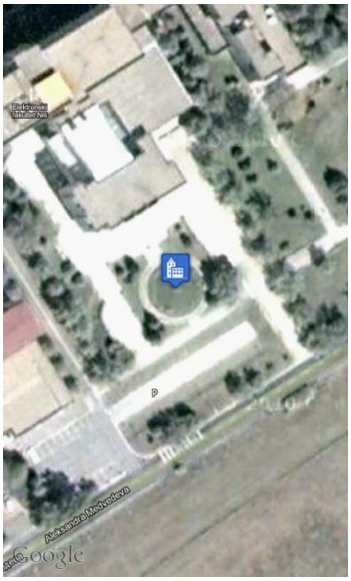


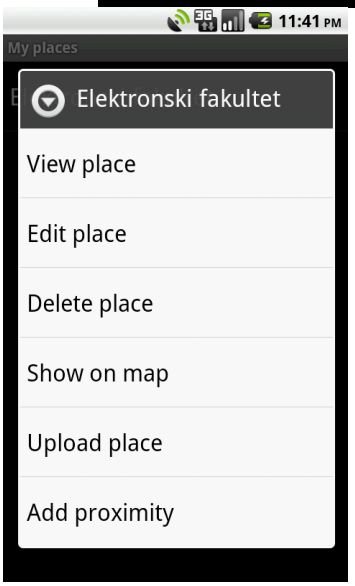
Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES

```
<uses-permission
    android:name="android.permission.ACCESS_MOCK_LOCATION" />
<uses-permission
    android:name="android.permission.VIBRATE" />
```

d. Pokrenimo i probajmo aplikaciju.

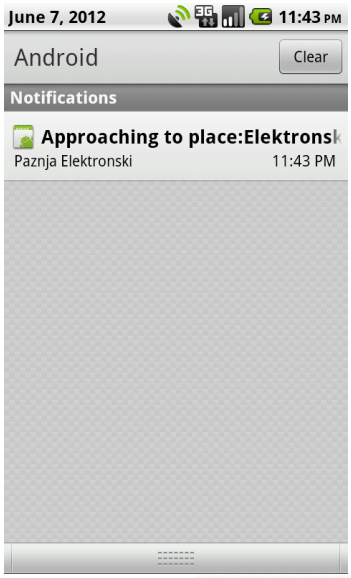


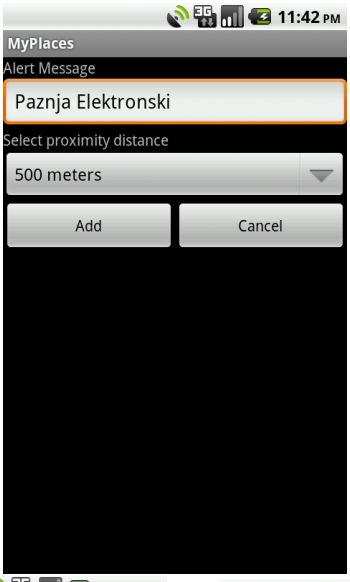



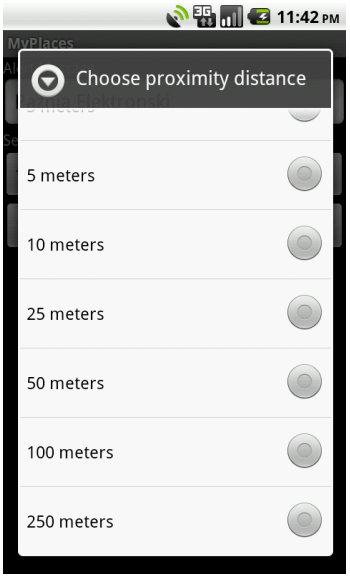


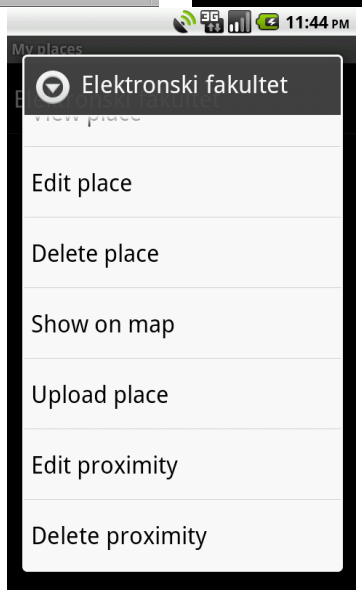


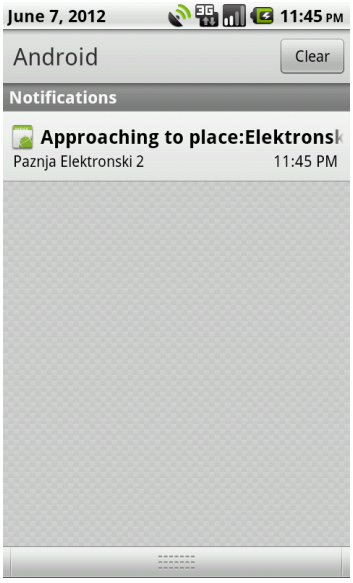

Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES



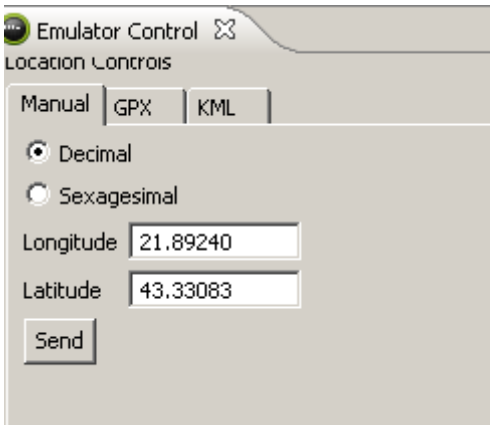




e. Isprobajmo aplikaciju promenom koordinata u odnosu na koodrinate u okolini Elektronskog fakulteta-a.






Program finansira: USAID | SRBIJA OD AMERIČKOG NARODA
Program sprovodi: NiCAT NIS CLUSTER OF ADVANCED TECHNOLOGIES