SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Ivan Tržić

SEMANTIČKA INTEGRACIJA WEB USLUGA U MOBILNOM OKRUŽJU

DIPLOMSKI RAD

Varaždin, 2012.

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Ivan Tržić
Redoviti student
Broj indeksa: 39563/10-R.
Smjer: Informacijsko i programsko inženjerstvo
Diplomski studij

SEMANTIČKA INTEGRACIJA WEB USLUGA U MOBILNOM OKRUŽJU

DIPLOMSKI RAD

Mentor:
Doc.dr.sc. Markus Schatten, docent

Varaždin, lipanj 2012.

Sadržaj

1. UVOD **1**

2. ANDROID OPERATIVNI SUSTAV **2**

2.1. UVOD 2
2.2. STRUKTURA ANDROID OPERACIJSKOG SUSTAVA 3
2.2.1. Aplikacije (eng. Applications) *3*
2.2.2. Aplikacijski okvir (eng. Application framework) *4*
2.2.3. Biblioteke (eng. Library) *4*
2.2.4. Android radno okruženje (eng. Android runtime) *4*
2.2.5. Linux jezgra (eng. Linux Kernel) *4*
2.3. VERZIJE ANDROID OPERACIJSKOG SUSTAVA 4
2.3.1. Gingerbread *4*
2.3.2. Honeycomb *5*
2.3.3. Ice Cream Sandwich *5*
2.4. DVM (ENG. DALVIK VIRTUAL MACHINE) 5
2.5. APLIKACIJE KOJE DOLAZE SA ANDROID OPERACIJSKIM SUSTAVIMA 6

3. SEMANTIČKA INTEGRACIJA **7**

3.1. UVOD 7
3.2. MOBILNO OKRUŽJE 7

4. XQUERY **9**

4.1. OSNOVNI POJMOVI 10
4.2. FLOWR 10
4.3. XPATH 11
4.3.1. Primjer korištenja XPath – a *11*

5. WEB SERVISI **13**

5.1. XML 14
5.2. SOAP 14
5.3. WSDL 15
5.4. UDDI 15

6. EXIST BAZA PODATAKA **16**

6.1. PRIMJERI RADA S BAZOM 16

7. RAZVOJNI ALATI **20**

7.1. NETBEANS 20
7.2. APPCELERATOR TITANIUM 20

8. OPIS SUSTAVA **22**

9. PRVA APLIKACIJA, COLLECTOR-HJP **24**

9.1. OPIS 24
9.2. OBJAŠNJENJA RADA LIBRARYA I VANJSKIH PROJEKATA 25
9.2.1. Html Cleaner *25*
9.2.2. HtmlUnit *27*
9.2.3. Vjezba_03_02 ..... *28*
9.3. OBJAŠNJENJA RADA POJEDINIH KLASA 28
9.3.1. Collector-HJP *28*
9.3.2. Dretva *29*
9.3.3. HTMLCistac *29*
9.3.4. Spajanje *29*
9.3.5. StringoviObrada *30*
9.3.6. XpathDOM *32*

9.3.7. Vrijeme *32*
9.3.8. Vlakovi *35*
9.3.9. Avioni *38*
9.3.10. Autobusi *39*

10. DRUGA APLIKACIJA, WEBSERVIS-HJP 44

11. TREĆA APLIKACIJA, HRVATSKI JAVNI PRIJEVOZ 47

11.1. OPIS MOBILNE APLIKACIJE 47
11.2. POJAŠNJENJE KODA MOBILNE APLIKACIJE 50

12. ZAKLJUČAK 55

LITERATURA 56

KNJIGE 56
PREZENTACIJE 56
STRUČNI ČLANCI 56
INTERNET 57

1. Uvod

Kao što se može primjetiti kroz vijesti i statističke podatke u zadnjih nekoliko godina Android je ostvario ogroman rast u broju novih korisnika pametnih mobilnih uređaja. Tako na primjer u zadnjem kvartalu 2011. godine više od 50% prodanih uređaja otpada upravo na Android uređaje.[1] Sukladno s time povećan je i broj aplikacija koje se nude na Android marketu te broj preuzimanja aplikacija s Android marketa.

Nabavkom Android pametnog mobilnog uređaja, autor ovog rada je krenuo u istraživanje tog novog područja. Istraživanjem Google Play – a [2] je ustanovljen manjak kvalitetnih aplikacija koje pružaju informacije o putovanjima u Republici Hrvatskoj. Autor ovog rada smatra da je time uskraćena vrlo korisna usluga turistima kojima bi se na ovaj način olakšalo kretanje turista u turističkoj sezoni. Ovdje je napomena na inozemne turiste iz tog razloga što velika većina lokalnog stanovništva je određenom mjeri upoznato s načinima putovanja dok to s turistima nije slučaj. Nadalje, postoji mogućnost da bi se i u određenom mjeri smanjile gužve na informativnim šalterima, a možda čak i uklonila potreba za postojanjem takvih šaltera budući da bi sve informacije bile udaljene jedan potez prsta.S tom idejom autor ovog rada je krenuo u potragu za web servisima koji bi mogli pružiti takve informacije tako da se napravi aplikacija koja bi samo to prikazivala. Kako autor ovog rada nije uspio pronaći prikladne web servise ova ideja se razvila u diplomski rad koji za praktični dio ima cijelu infrastrukturu koja podržava rad mobilne aplikacije.

[1] Introduction, Using Android NDK, Ivan Švogor, Željko Šmaguc, prezentacija sa konferencije CASE24 koja je održana 05.06.2012. godine u Zagrebu

[2] Google Play prijašnji Android Market je servis koji omogučuje preuzimanje raznog multimedijskog sadržaja na Android uređaje između ostalog i aplikacije. Moguće mu je pristupiti preko linka https://play.google.com/store ili pokretanjem istoimene aplikacije na Android uređaju.

2. Android operativni sustav

2.1. Uvod

Andy Rubin iz tvrtke Google opisuje Android kao:
„[3]*The first truly open and comprehensive platform for mobile devices, all of the software to run a mobile phone but without the proprietary obstacles that have hindered mobile innovation*[4].“

Android operacijski sustav je mobilni sustav otvorenog koda koji je izgrađen na podlozi Linux operativnog sustava. Glavni razlozi zbog korištenja operacijskog sustava Linux su: sigurnosni model, mrežni sustav, mogućnost upravljanja memorijom i procesima, pogonska podrška, robusnost, stalni razvoj i unapređivanje sustava. Nastao je 2003. godine. u gradu Palo Alto. Kreirali su ga četiri programera, Andy Rubin, Rich Miner, Nick Sears i Chris White. Najkraće rečeno Android je kombinacija tri vrlo poželjne komponente. On je besplatan operacijski sustav otvorenog koda za mobilne uređaje, platforma otvorenog koda za stvaranje mobilnih aplikacija te uređaji koji rade na Android operativnom sustavu i aplikacije koje su stvorene za njega.[5]

Detaljnije[6] Android sustav je stvoren od nekoliko potrebnih i ovisnih dijelova kao:

- Linux operacijski sustav koji pruža sučelje s sklopovljem, upravljanje memorijom, kontrolu procesa, a sve to je optimizirano za mobilne uređaje.
- Opis sklopovlja kojim se opisuju mogućnosti mobilnih uređaja kako bi mogli podržavati aplikacije.
- Biblioteke otvorenog koda za razvoj aplikacija uključujući SQLite, WebKit, OpenGL itd.
- Korisničko sučelje koje je stvoreno za spremanje i pokretanje aplikacija.
- Prethodno instalirane aplikacije zapakirane kao dio operacijskog sustava.
- Sadrži Dalvik virtualni stroj i jezgrene biblioteke čime se pruža funkcionalnost specifična za Android.
- Alat za razvoj programskih rješenja koji omogućuje stvaranje aplikacija, a sadrži dodatne alate, dodatke i dokumentaciju.

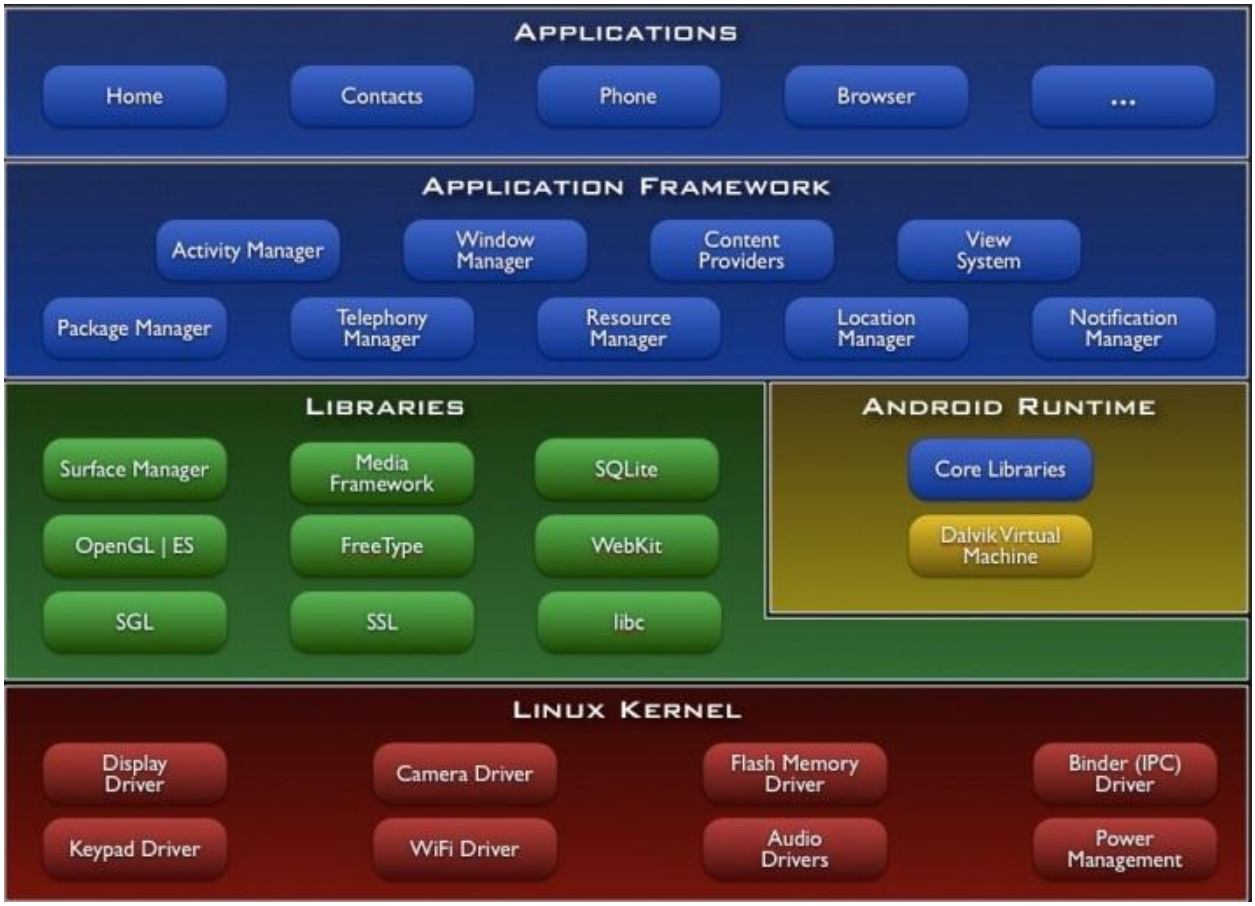
[3] „Prva otvorena i opsežna platforma za mobilne uređaje, sa svim programskim rješenjima koji pokreću mobilne uređaje ali bez prepreka zatvorenih licenci koje ometaju mobilnu inovaciju. “

[4] Preuzeto sa: http://googleblog.blogspot.com/2007/11/wheres-my-gphone.html
[5] Professional Android 2 Application Development, Reto Meier, str. 4
[6] Professional Android 2 Application Development, Reto Meier, str. 4

Što Android čini uistinu očaravajućim je otvorena filozofija, koja omogućava popravljanje bilo kakvim pogrešaka u korisničkom sučelju ili dizajnu matičnih (eng. native) aplikacija na taj način da se napiše ekstenzija ili zamjena. Android pruža programerima priliku implementirati sučelje mobilnog uređaja i izgleda aplikacija koje će izgledati i funkcionirati upravo kako su zamišljene.

2.2. Struktura Android operacijskog sustava

Na slici 2.1. možemo vidjeti cjeline Android operacijskog sustava koje se dijele na 5 dijelova[7].



Slika 2.1. Arhitektura Android operacijskog sustava[8]

2.2.1. Aplikacije (eng. Applications)

Ova cjelina su aplikacije koje dolaze s Android operativnim sustavom, a bit će detaljnije objašnjenje u poglavlju 2.4.

[7] Preuzeto sa: http://developer.android.com/guide/basics/what-is-android.html
[8] Slika preuzeta sa: http://developer.android.com/guide/basics/what-is-android.html

2.2.2. Aplikacijski okvir (eng. Application framework)

Aplikacijski okvir implementira skup zajedničkih programskih rutina koje služe kao pomoć programerima pri izradi aplikacija.To znači da programeri mogu koristiti već gotove funkcije što skraćuje vrijeme programiranja budući da ne moraju programirati osnovne funkcionalnosti.

2.2.3. Biblioteke (eng. Library)

Android uključuje skup C/C++ biblioteka koje potom koriste razni dijelovi Android operacijskog sustava.

2.2.4. Android radno okruženje (eng. Android runtime)

Android radno okruženje su osnovne biblioteke koje nam omogućuju veću iskoristivost osnovnih biblioteka programskog jezika Java. Svaka aplikacija se izvršava u svom vlastitom procesu, s vlastitom instancom Dalvik virtualnog stroja. Dalvik je napisan na taj način da može efikasno izvoditi nekoliko virtualnih strojeva. Dalvik virtualni stroj izvršava datoteka u dex[9] formatu koji je optimiziran za korištenje minimalnog dijela memorije.

2.2.5. Linux jezgra (eng. Linux Kernel)

Android se oslanja na Linux verziju 2.6. za jezgrene sustavne servise poput sigurnosti, upravljanje memorije, procesa, mreže i pogonski programima (eng. drivers).

2.3. Verzije Android operacijskog sustava

Kako rast Android korisnika raste velikom brzinom može se reći da raste broj programera koji podržavaju Android te doprinose njegovom rastu i kvaliteti. Taj rast prati i brzo smjenjivanje verzija Android sustava i svakim izdavanjem se pokušava poboljšati kvaliteta sustava. Zanimljivost je da se od verzije 1.5 pod nazivom Cupcake i svaka sljedeća verzija se naziva po nekom desertu. Tako zadnje tri verzije su 2.3 Gingerbread, 3.0 Honeycomb i 4.0 Ice Cream Sandwich.

2.3.1. Gingerbread

Kod verzije 2.3 poboljšano je korisničko sučelje, meka tipkovnica, kopiranje, poboljšana podrška matičnom (eng. native) kodu, dodana SIP[10] podrška koja služi za VoIP[11] pozive.

[9] Dex format – Prilikom izvršavanja na DVM – u .class datoteke se pretvaraju u .dex datoteke točnije Dalvik bajtni kod
[10] SIP (eng. Session Initiation Protocol)
[11] VoIP (eng. Voice over Internet Protocol) je protokol koji omogućava prijenost zvučne komnikacije preko interneta

2.3.2. Honeycomb

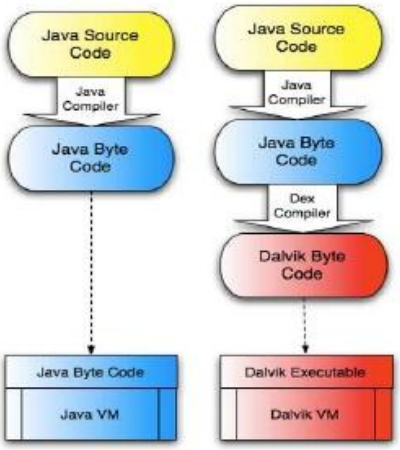
3.0 Verzija je izdana specifično za tablete. Podržava uređaje većeg zaslona te dodaje pregršt novih mogućnosti za korisnike, podršku za procesore s više jezgri, sklopovsko ubrzanje za grafiku i potpunu enkripciju sustava.

2.3.3. Ice Cream Sandwich

Verzija 4.0 je donijela mogućnosti verzije 3.0 napokon na mobilne uređaje. U trenutku pisanja ovog rada trenutno je najnovija verzija Android operacijskog sustava na tržištu. Nove mogučnosti su brava sa prepoznavanjem lica, praćenje prometa podataka preko računalne mreže, poboljšanja fotografija, dodane mape itd.

2.4. DVM (eng. Dalvik Virtual Machine)

Dalvik virtualni stroj[12] je razvio Dan Bornstein. Prilikom objašnjavanja rada DVM – a potrebno je znati da svaka aplikacija ima svoj proces, a svaki proces se izvodi u svom virtualnom stroju. Budući da govorimo o Android aplikacijama koje se programiraju u Javi logično bi bilo da se izvode na JVM – u (eng. Java Virtual Machine). Problem je taj što JVM zahtjeva puno procesorske snage i resursa memorije te se kao takav ne može izvoditi na mobilnim uređajima. Za tu svrhu je razvijena poseban virtualni stroj, a nazvan je DVM. Taj virtualni stroj ne izvršava Java bajt kod već postoji još jedan korak gdje se Java bajte kod pakira u .dex datoteku koju DVM pokreće što možemo vidjeti na slici ispod. Na ovaj način veličina koda se smanjuje gotovo 50%.



Slika 2.2. Koraci izgradnje aplikacija za JVM i DVM[13]

[12] Dalvik, Using Android NDK, Ivan Švogor, Željko Šmaguc, prezentacija sa konferencije CASE24 koja je održana 05.06.2012. godine u Zagrebu

[13] Slika preuzeta sa: Dalvik, Using Android NDK, Ivan Švogor, Željko Šmaguc, prezentacija sa konferencije CASE24 koja je održana 05.06.2012. godine u Zagrebu

2.5. Aplikacije koje dolaze sa Android operacijskim sustavima

Mobilni uređaju opremljeni Android operacijskim sustavom imaju paket prethodno instaliranih aplikacija na operacijskom sustavu :

- E – mail klijent
- Aplikacija za upravljanjem SMS – ova
- Kalendar
- Aplikaciju za listu kontakata
- Internet preglednik
- Aplikaciju za reprodukciju glazbe
- Preglednik galerije slika
- Kameru i aplikaciju za snimanje
- Kalkulator
- Početni zaslon
- Budilicu

U mnogim slučajevima Android uređaji dolaze i sa sljedećim Google mobilnim aplikacijama:

- Google Play klijent koji služi za preuzimanje Android aplikacija
- Gmail e – mail klijent[14]
- Youtube klijent[15]
- Google Maps aplikaciju[16]

Podaci koje spremaju te aplikacije obično su dostupni i ostalim aplikacijama koje instaliravamo na mobilni uređaj.

[14] E – mail preglednik koji omogućuje izravno spajanje na Gmail račun te omogučava izvođenje operacija poput slanja poruka, pregledavanje poruka itd.
[15] Youtube klijent nam omogučava izravno spajanje na http://www.youtube.com/ stranicu što je stranica za dijeljenje video sadržaja
[16] Google maps je web servis koji na kartama prikazuje karte ulica, omogućuje planiranje putovanja raznim sredstvima itd.

3. Semantička integracija

3.1. Uvod

Internet je stvoren kao prostor za razmjenu informacija ali s ciljem da se razmjena informacija ne događa samo između ljudi već da računala mogu korisnicima pomoći u razmjeni informacija. Glavna prepreka je to što je velika većina informacija koja se pružaja na Internetu je namjenjena upravo ljudima. Računala dobro koriste pažljivo strukturirano podatke, a većina podataka su prikazani na taj način da je to razumljivo samo ljudima. Koncept dokumenata koji računala mogu „razumjeti“ se ne može riješiti umjetnom inteligencijom već se treba osloniti na sposobnost računala kojim rješavaju točno definirane probleme točno definiranim operacijama na dobro strukturiranim podacima. Stoga umjesto da tražimo od računala da razumiju naš jezik prilikom definiranja novih tehnologija bilo je potrebno uložiti određeni trud da se ti podaci bolje strukturiraju. Nove tehnologije kojima se izgrađuje Internet dopuštaju računalima pristup podacima na Internetu.

3.2. Mobilno okružje

U novije vrijeme javila se potreba za razvojem mobilnih aplikacija zato što one imaju specifičnosti koje stolne aplikacije ne mogu pružiti. Najbitniji motiv je mobilnost. Postoji velik dio poslova koji se obavljaju na terenu i prije razvoja mobilnih uređaja informatizacija tih područja nije bilo moguća ali to se sada mijenja. Svakim danom se smišljaju nove aplikacije koje bi poboljšale poslovanje i to je pogotovo bitno u područjima koja imaju podatke koji se brzo mijenjaju. U tom slučaju brzi pregled i ažuriranje mogu biti od ključne važnosti.

Bitno je napomenuti da razvojem mobilnih tehnologija cijena uređaja pada tako da su sve jači uređaji pristupačniji po nižim cijenama nego prije.

Kako su uređaji puno manji od standardnih desktop računala ovo predstavlja određene prednosti i nedostatke kako za korisnike tako i za programera mobilnih aplikacija. Budući da su mobilni uređaju ograničeni prostorom na kojima se prikazuju informacije to predstavlja problem za razvojni tim budući da se potrebno puno više planiranja oko informacija i funkcionalnosti koji će se prikazati te način na koji će se prikazati kako bi zauzeo što manje prostora. Ako su programeri taj dio dobro napravili to predstavlja prednost za korisnika budući da neće biti zatrpan mnoštvom mogučnosti i informacija te će se moći bolje posvetiti poslu koji treba obaviti. Kako su mobilni uređaji limitirani procesorskom snagom i radnom memorijom potreba za semantičkom integracijom sa Web uslugama je jasna. Mobilni uređaji nisu stvoreni za procesiranje velikog broja podataka stoga ih je vrlo dobro spojiti s ostalim uslugama kao na primjer web servisima.

Tako je vrlo dobro napraviti obradu podataka na poslužitelju, a mobilnom uređaju kao rezultat vratiti selektirane podatke koji se potom prikazuju.

Daljne probleme pri razvoju mobilnih aplikacija predstavljaju veći broj operacijskih sustava koji postoje na tržištu te još veći broj različitih uređaja od kojih svaki ima svoje posebnosti. Ciklus razvoja aplikacije zahtjeva dugotrajno vrijeme testiranja zbog tako velikog obujma raznih modela mobilnih uređaja na tržištu.

4. Xquery

1998. izmišljen je novi jezik za postavljanje upita nad XML (eng. Extensible Markup Language) datotekama, izmislili su ga Jonathan Robie i Joe Lapp te su ga nazvali zvao XQL[17](eng. XML Query Language). Daljnjim razvijanjem XQL te spajanjem sa XML-QL[18]-om nastao je XQuery te postao W3C[19] standard.

XQuery je primjer funkcijskog programskog jezika. Da bi se moglo programirati u ovom jeziku potrebna su predznanja iz HTML – a (eng. HyperText Markup Language), XPath-a 2.0 (eng. XML Path Language) [20], XML-a.

Slika 4.1. Prikaz Xquery strukture[21]

XQuery može obavljati upite nad različitim strukturama podataka, rekurzivne je prirode te je zbog toga izuzeto pogodan za vršenje upita nad strukturama poput stabla i grafova. Naredbe u XQueryu dosta su kraće od onih u SQL-u (eng. Structured Query Language) ili XSLT-u (eng. Extensible Stylesheet Language Transformations). Upiti se mogu vršiti nad hijerarhijskim i tabličnim podacima. Xquery ima dosljednu/konzistentnu sintaksu te je lako primjenjiv nad raznim XML standardima.

Najčešće se koristi za:

- Izvlačenje informacija za korištenje iz web servisa
- Generiranje izvještaja
- Transformiranje podataka iz XML – a u XHTML (eng. eXtensible HyperText Markup Language)

[17] Xquery tutorial, dostupno na: http://www.w3schools.com/xquery/default.asp
[18] XML-QL (eng. A Query Language for XML) – XML-QL je vrlo različit od XPath – a jer ne koristi putanje zapisane kao izraze već koristi uzorke kako bi pronašao rezultate
[19] W3C (eng. World Wide Web Consortium) – glavna internacionalna organizacija za standard za World Wide Web
[20] XPath 2.0 verzija trenutna verzija XPath – a u vrijeme pisanja ovog rada. Definiran od strane W3C, a postao je preporuka 23.01.2007. godine
[21] Slika preuzeta sa: http://www.w3schools.com/xquery/xquery_intro.asp

- Pretragu Internet stranica za izvlačenje bitnih podataka

4.1. Osnovni pojmovi[22]

Pojmovi koje je potrebno znati prije nego što se upustimo u programiranje u Xquery - u su:

- Čvorovi (eng. nodes) – elementi XML-a
- Atomna vrijednost (eng. atomic values) – čvorovi bez roditelja i potomaka
- Stavke (eng. Items) – atomne vrijednosti ili čvorovi

Također potrebno je poznavati odnose između čvorova:

- Roditelj (eng. parent) – svaki element ima jednog roditelja (osim ako se radi o atomnoj vrijednosti)
- Potomak (eng. child) – čvorovi mogu imati jednog ili više potomaka
- Braća (eng. siblings) – čvorovi koji imaju zajedničkog roditelja
- Preci (eng. Ancestors) – roditelj nekog čvora, roditelji roditelja itd.
- Nasljednici(eng. Descendants) – čvorovi koji su djeca, djeca djece itd.

Pravila sintakse:

- Moramo paziti na velika i mala slova
- Elementi, atributi i varijable moraju biti važeći XML nazivi
- Znakovni nizovi se mogu staviti pod jednostruke ili dvostruke navodnike
- Varijabla se definira sa znakom $
- Komentiramo na sljedeći način (: ovo je komentar :)

Znakovi za uspoređivanje:

- =, eq – jednako (eng. equal)
- !=, ne – nejednako (eng. not equal)
- <, lt – manje od (eng. less then)
- <, gt – veće od (eng. greater then)
- <=, le – manje ili jednako (eng. less or equal)
- >=, ge – više ili jednako (eng. greater or equal)

4.2. FLOWR

FLOWR je akronim za riječi (eng. For, Let, Where, Order by, Return). To su naredbe koje dodajemo ako želimo da nam se određena akcija obavi, a svaka akcija ima drugo značenje. „For“

[22] Xquery terms, dostupno na: http://www.w3schools.com/xquery/xquery_terms.asp

naredbom označavamo određene elemente i spremamo ih u specifičnu varijablu. Naredbom „where“ omogučujemo označavanje određenih elemenata ukoliko oni ispunjavaju neki uvjet. „Let“ naredbom izbjegavamo ponavljanje nekih izraza više puta. Naredba „order by“ nam omogučava sortiranje rezultata po određenim kriterijima. „Return“ naredba određuje što se treba vratiti kao rezultat.

4.3. XPath

XPath je upitni jezik za adresiranje različitih dijelova XML dokumenata i za izračunavanje vrijednosti baziranih na sadržaju XML dokumenta. XQuery 1.0 i XPath 2.0 koriste zajednički model podataka i podržavaju iste funkcije i operatore. Može se reći da se XQuery zasniva na XPath 2.0 jeziku.

4.3.1. Primjer korištenja XPath – a

XPath - om možemo zadati putanju do željene informacija bilo ona prikazana u XML obliku ili u HTML obliku. Znači XPath ne radi razlike između to dvoje te se podaci mogu dohvatiti.

Jedan od načina za pravilno dohvaćanje putanja specifičnih podataka je instalacija Firebug addon – a za Mozzilu Firefox. Firebug je jedan od najpoznatijih alata za Web razvoj. Moguće je praćenje zahtjeva, podataka, pregled HTML source koda, pregled CSS (eng. Cascading Style Sheet)[23] koda te čak i promjena istih. Što se tiče XPath-a Firebug ima dodatak nazvan Firepath. Firepath nam se tada prikazuje kao još jedna kartica u Firebugu.

Način na koji se dohvaća XPath putanja je pronalazak traženog dijela stranice, označavanje, desni klik i „Inspect element with Firebug“, nakon toga taj dio označimo dolje u Firebug konzoli opet desni klik i „Inspect in Firebug tab“.

Slika 4.2. Prikaz dohvata XPath putanje

[23] CSS – jezik koji pruža veću stilističku kontrolu od one koju pruža HTML za prikaz na Internetu.

Kao rezultat toga je da dobijemo naprimjer
`html/body/div[2]/div[2]/div/div/div/div[3]/div/div[1]/div[2]/div/em[3]`.
Znači sada smo dobili XPath putanju određenog traženog podatka na HTML stranici.
Ovakvi izrazi se mogu reducirati i drukčije napisati poput
`//table/tbody/tr/td[3]` gdje „//“ označava da pretraga izvršava od bilo kojeg čvora u stablu.
Ovim primjerom `//table/tbody/tr/td[3]` moramo paziti da jedinstveno označavamo određeni dio HTML stranice jer kao rezultat možemo dobiti još neke stvari označene što može dovesti onda do pogrešnih podataka. Ovaj XPath izraz možemo dopuniti na način `//table/tbody/tr/td[3]/text()` i on će nam vratiti znakovni niz označenog dijela preko text() funkcije.

Postoji veliki broj funkcija koji se može koristiti u različite svrhe, a popis svih se može pronaći na stranici[24].

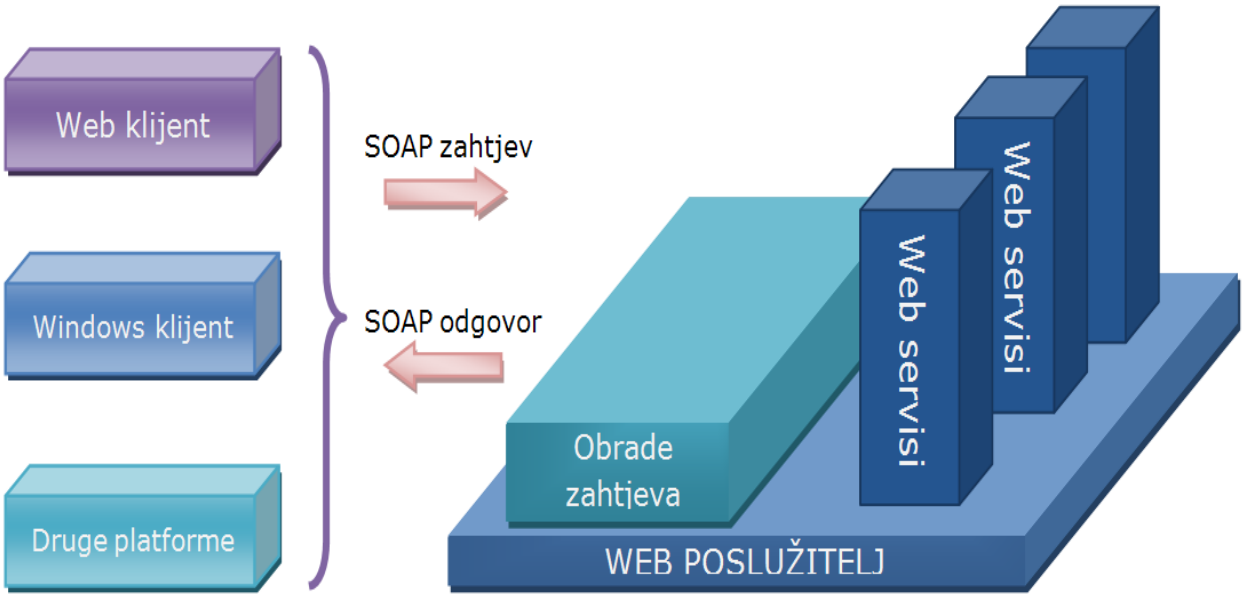
[24] Xpath functions, dostupno na: http://www.w3schools.com/xquery/default.asp

5. Web servisi

Web servisi[25] su klijentske i poslužiteljske aplikacije koje komuniciraju putem www protokola HTTP (eng. HyperText Transfer Protocol). Oni su karakterizirani po svojoj velikoj interoperabilnosti i proširivosti kao i po računalno obradivim opisima, zahvaljujući korištenju XML – a. Na konceptualnoj razini web servis je softverska komponenta koja stoji na raspolaganju putem mrežno dostupne točke.

Mogu postojati web servisi koji preuzimaju jednu od 3 glavne uloge, a to su :

1. Pružatelj web servisa
2. Registar servisa
3. Korisnik servisa.



Slika 5.1. Arhitektura web servisa

Prilikom rada sa web servisima 3 najbitnije operacije koje su uključene u njih jest objavljivanje web servisa, traženje web servisa te spajanje na web servis tj. njegovo korištenje.

Osnovni standardi za web servise:

- XML
- SOAP (Simple Object Access Protocol)
- WSDL (Web Services Description Language)

[25] Predavanja s kolegija Napredne web tehnologije i servisi, Dr.sc. D.Kermek, Fakutet organizacije i informatike

- UDDI (Universal Description, Discovery and Integration)

5.1. XML

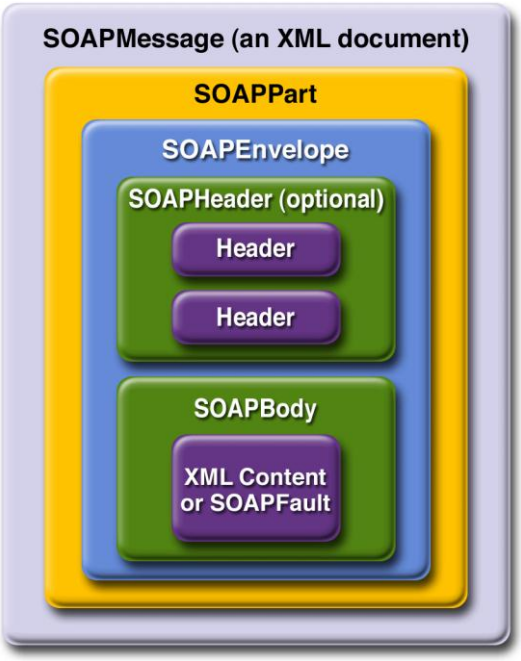
Korisnik poziva web servis slanjem upita, a poslužitelj mu kao rezultat vraća rezultat obrade u XML obliku. XML se koristi za definiranje elemenata koji se koriste u komunikacijama web servisa i svi dokumenti web servisa su pisani u XML – u.

Slika 5.2. Primjer web servisa[26]

5.2. SOAP

SOAP se koristi za komuniciranje sa web servisima. Zahtjev i odgovor su SOAP poruke, a povezuje korisnika sa web servisom.

[26] Web servis tečajna lista PBZ – a, dostupno na: http://www.pbz.hr/Downloads/PBZteclist.xml



Slika 5.3. SOAP poruka

Najvažnije karakteristike SOAP – a su neutralnost, nezavisnost i fleksibilnost.

5.3. WSDL

WSDL su kreirali IBM i Microsoft. On opisuje web servis te definira funkcije koje su izložene u web servisu. Moguće ga je proširiti, a definira XML gramatiku koja se koristi u porukama.

5.4. UDDI

UDDI su kreirali IBM, Microsoft i Ariba. To je registar za upravljanje informacijama o web servisima. On se koristi za registriranje i pretraživanje servisa u središnjem registru. Pružatelj servisa tamo može objaviti informaciju o svojem poslovanju i servise koje nudi. Korisnik servisa može pronaći servise koji su na raspolaganju preko poslovanja, kategorije servisa i specifičnosti servisa.

6. Exist baza podataka

Exist baza[27] je open source, služi za upravljanje bazom podataka temeljenoj na XML tehnologiji. Sprema XML podatke prema XML modelu podataka. Izradio ga je Wolfgang Meier krajem 2000. godine. Ovoj bazi se dodaju XML kolekcije podataka. XML kolekcije podataka koje dodajemo i koristimo možemo usporediti sa tablicama u SQL bazi podataka.

6.1. Primjeri rada s bazom

Prilikom izrade primjera smo radili sa XML datotekom „books.xml“ koja je preuzeta sa stranice[28], a oblika zapisa u njoj je:

```
<bookstore>
<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>
</bookstore>
```

Primjeri:

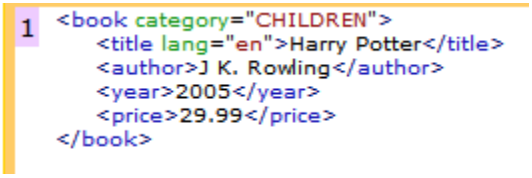
Funkcija doc nam služi za otvaranje određenog dokumenta. Kao što vidimo ispod moramo joj kao parametar proslijediti naziv dokumenta s pripadajućom putanjom. Putanju koju smo napisali nakon zagrade određuje koje ćemo podatke uzimati, a te putanje možemo mijenjati po potrebama.

1. `doc('//db/Primjer/books.xml')/bookstore/book/author` – dohvaća sve autore koji se nalaze u XML datoteci „books.xml“.

[27] Exist database, dostupno na: http://exist-db.org/exist/index.xml
[28] Xquery example, dostupno na: http://www.w3schools.com/xquery/xquery_example.asp

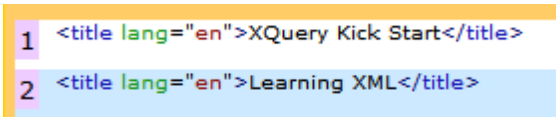
Slika 6.1. Imena autora kao rezultat upita

2. `doc('//db/Primjer/books.xml')/bookstore/book[price<30]` – kao rezultat dobivamo knjige čija je cijena manja od 30



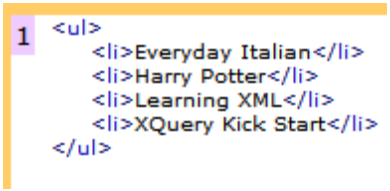
Slika 6.2. Rezultati upita gdje su knjige sa cijenom ispod 30

3. `doc('//db/Primjer/books.xml')/bookstore/book[price>30]/title` – kao rezultat dobivamo naslov svih knjiga čija je cijena veća od 30



Slika 6.3. Rezultati upita gdje su knjige s cijenom iznad 30

4.
```
<ul> {
for $x in
doc('//db/Primjer/books.xml')/bookstore/book/title
order by $x return <li>{data($x)}</li>
} </ul>
```
- kao što vidimo u ovom primjeru upiti se mogu koristiti u kombinaciji sa HTML –om, a tu su rezultati sortirani naslovi knjiga ispisani u obliku HMTL liste.



Slika 6.4. Rezultati sortirani u listu

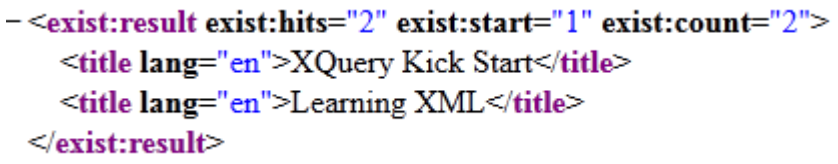
Ovo su bili primjeri dohvata podataka preko Exist – ovog „sandbox - a“. Za dohvat, promjenu podataka i ostale operacije postoji i drugi način, a to je direktni dohvat preko poveznice ili slanje upita preko HTTP GET protokola. Ispod su navedeni nekoliko primjera tog način.

Primjeri upita preko poveznice:

1. http://localhost:8080/exist/servlet/db/Primjer/?_query=for%20$knjiga%20in%20doc%28%22books.xml%22%29/bookstore/book%20return%20$knjiga - pristupanje „books.xml“ dokumentu te ispis cjelokupnoga u pregledniku

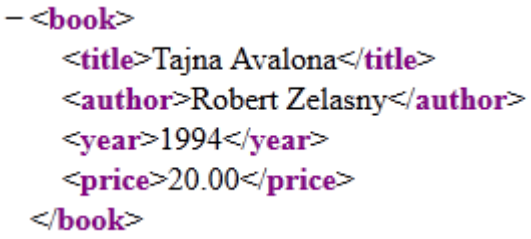
Slika 6.5. Pregled book.xml dokumenta preko upita

2. http://localhost:8080/exist/servlet/db/Primjer/?_query=for%20$x%20in%20doc%28%22books.xml%22%29/bookstore/book%20where%20$x/price%3E30%20return%20$x/title – prikaz knjiga kojima je cijena manja od 30



Slika 6.6. Prikaz knjiga gdje je cijena manja od 30 preko upita

3. http://localhost:8080/exist/servlet/db/Primjer/?_query=let%20$baza%20:=%20doc%28%22books.xml%22%29%20return%20%28%20update%20insert%20%3Cbook%3E%3Ctitle%3ETajna%20Avalona%3C/title%3E%3Cauthor%3ERobert%20Zelasny%3C/author%3E%3Cyear%3E1994%3C/year%3E%3Cprice%3E20.00%3C/price%3E%3C/book%3Einto%20$baza/bookstore,%20$baza%29 – upisuje novu knjigu pod nazivom Tajna Avalona, autora Roberta Zelasny – a, izdane godine 1994. i cijene 20.



Slika 6.7. Upisivanje nove knjige preko upita

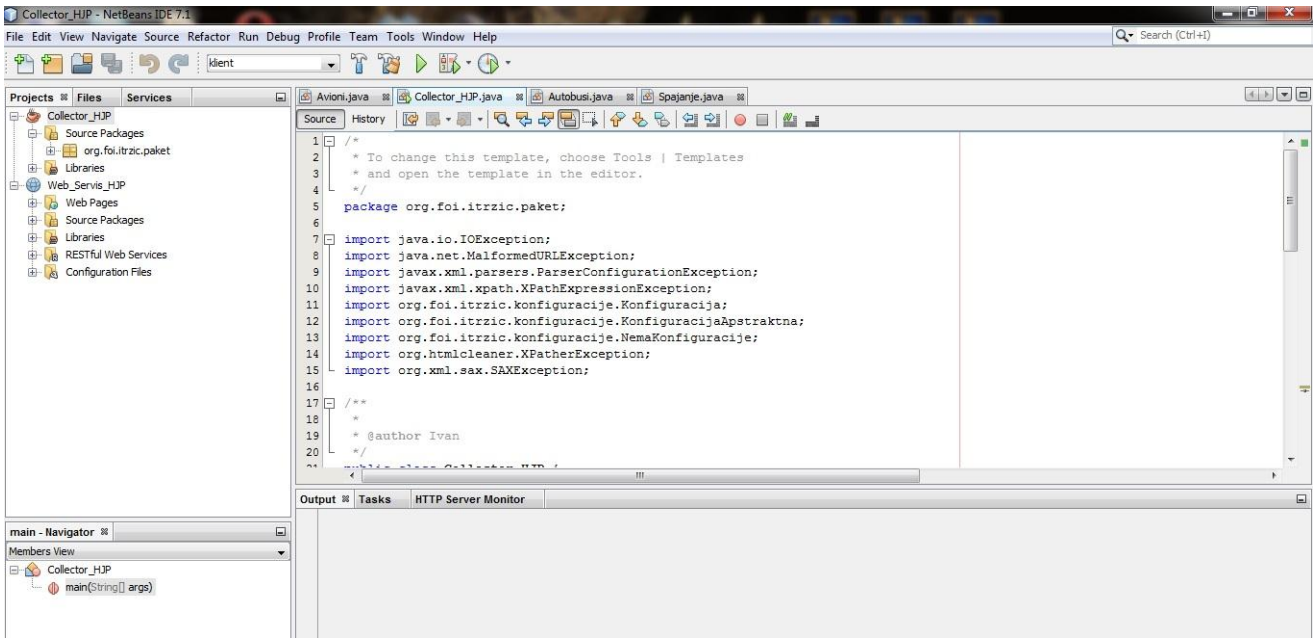
7. Razvojni alati

7.1. NetBeans

NetBeans[29] je IDE (eng. Integrated development environment) za razvoj Java, Javascript, PHP i Python aplikacija. Neatbeans IDE je napisan u Javi i može se pokretati na Windowsima, Mac OS – u, Linuxu Solarisu i ostalim platformama koje su kompatibilne sa JVM – om (eng. Java virtual machine).

Rad na NetBeansima je počeo 1996. godine kao studenski projekt pod vodstvom Fakulteta matematike i fizike u Pragu.

Podržava kompletne alate za razvoj najnovijih Java EE 6 standarda, uključujići i Enterprise Java Beans, servlete, web servise, anotacije, Java Persistance API. Također podržva JSF 2.0 (Facelets), JavaServer Pages (JSP), Hibernate, Spring i Strut. Dolazi sa GlassFish i Apache Tomcat poslužiteljima.



Slika 7.1. Prikaz NetBeans IDE

Diplomski rad je rađen na NetBeans IDE verziji 7.1.

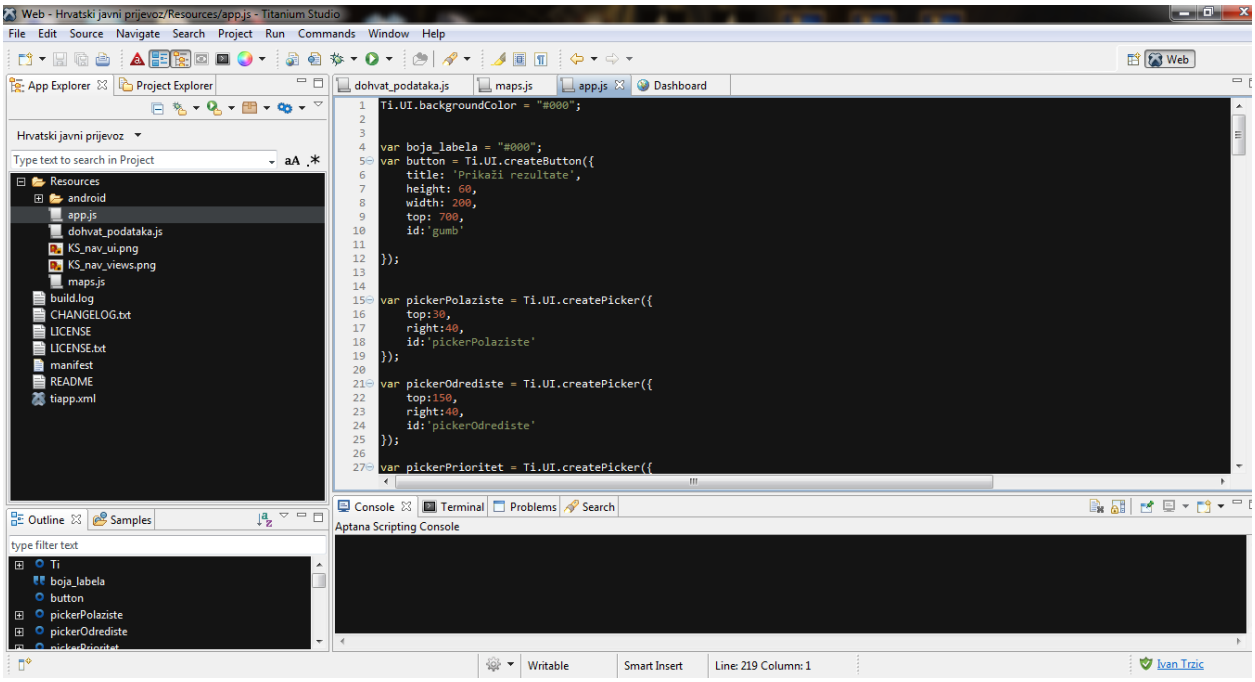
7.2. Appcelerator Titanium

Appcelerator Titanium[30] je platforma za razvoj stolnih aplikacija te aplikacija namjenjene za mobilne uređaje i tablete. Appcelerator Titanium je razvila tvrtka Appcelerator Inc. Podržava

[29] NetBeans početna stranica, dostupno na: http://netbeans.org/
[30] Appcelerator Titanium početna stranica, dostupno na: http://www.appcelerator.com/

razvoj za Android, iPhone i Blackberry mobilne aplikacije. Također podržava razvoj za iPad tablete. Appcelerator Titanium Mobile je razvojno sučelje koji koristi Javascript, CSS i HTML kod za razvoj matičnih (eng. native) mobilnih aplikacija. Takav pristup omogučava programerima brz razvoj matičnih mobilnih aplikacija bez potrebe učenja dva različita jezika jer Appcelerator Titanium omogučava pokretanje istog koda na više različitih operacijskih sustava. Premda implementira postojeće funkcije Javascripta, HTML – a i CSS – a ima također svoj Titanium API[31] koji je drukčiji u određenim aspektima od sintakse ostalih jezika. U potpunosti podržava Jquery, YUI, MooTools, Scriptaculous. Reklamira se pod sloganom: „Programiraj jednom, pokreni na više operacijskih sustava“. Ovo je većim dijelom točno ali ne u potpunosti. Kada se krene malo s programiranjem te proučavanjem njihove dokumentacije može se primjetiti da su određeni elementi specifični za Android ili iOS operacijski sustav. Stoga je potrebno paziti tokom programiranja da se provjerava na kojem se sustav pokreće aplikacija pa da se sukladno s tim izvode određeni segmenti koda. Ukoliko je moguće poželjno je izbjegavati te specifične elemente ili ako se programira samo za jedan operacijski sustav jednostavno pripaziti na to.

U lipnju 2011. Apcelerator je izdao Titanium Studio i Titanium Mobile 1.7. Titanium Studio je IDE baziran na Eclipsu. Titanium Desktop ima podršku za PHP, Python i Ruby.



Slika 7.2. Prikaz Titanium Studio IDE

[31] Titanium Appcelerator API docs, dostupno na: http://developer.appcelerator.com/apidoc/mobile/latest

8. Opis sustava

Prilikom izrade ovog diplomskog rada glavni fokus bio je na praktičnom dijelu rada. Ideja je bila da se napravi turistička aplikacija koja će na jednostavan, brz te instinktivan način korisniku omogućiti odabir polazišne i odredišne lokacije s ciljem pomaganja u odabiru najbolje mogućnosti putovanja s obzirom na određene preference korisnika. Preference mogu biti najjeftinije putovanje, najbrže putovanje ili odabir određene vrste putovanja. Oblici mogućih putovanja su putovanje autobusom, vlakom ili avionom. Aplikacija bi kao rezultat trebala izbaciti određeni broj najboljih opcija za putovanje s obzirom na korisnikove želje. Rezultati će sadržavati polazište, odredište, vrijeme polaska, vrijeme dolaska, vrijeme trajanja putovanja, cijenu itd.

Odabirom pojedinog putovanja na mobilnom uređaju će se prikazati lokacija polazišnog kolodvora bržeg i boljeg snalaženja u putovanju.

Aplikacija je pogodna za strane turiste koji bi se na taj način puno lakše snašli sa putovanjima unutar Republike Hrvatske stoga za komercijalnu upotrebu bi bilo najbolje realizirati dvojezičnost aplikacije ali za potrebe diplomskog rada aplikacija je napravljena na hrvatskom jeziku.

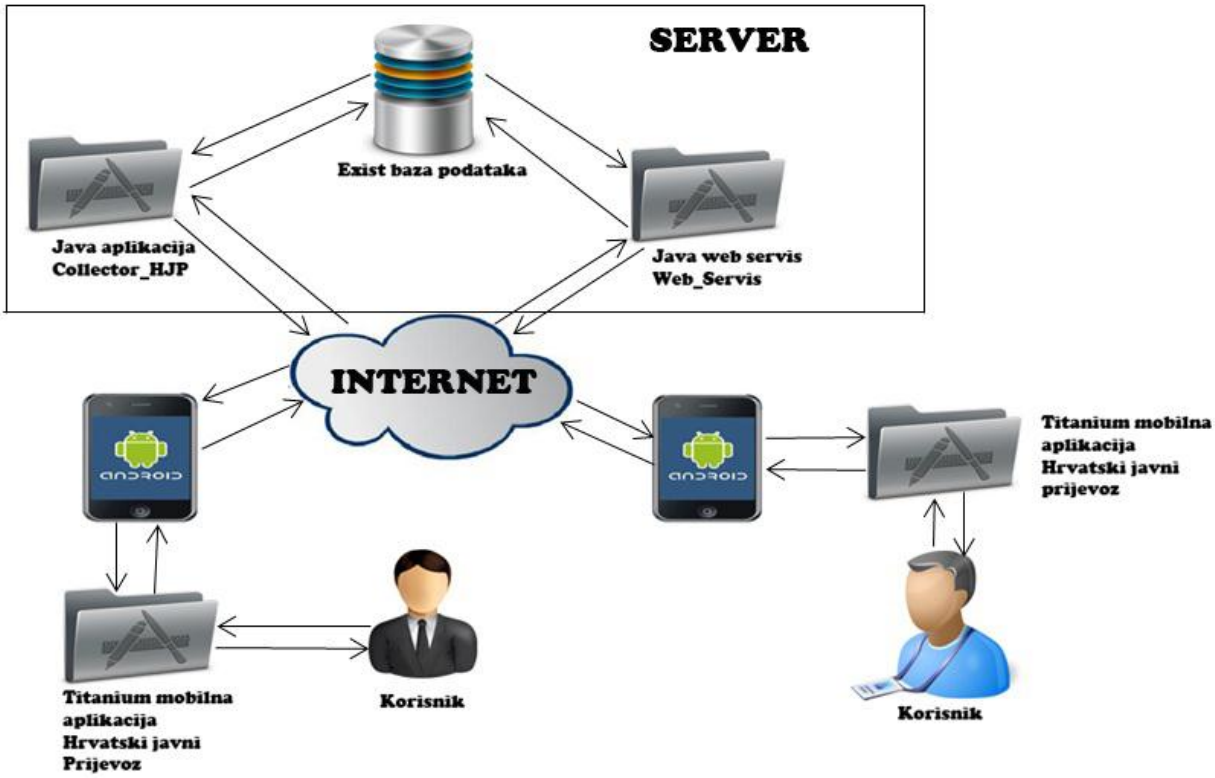
Stoga je primarna ideja bila da se napravi samo mobilna aplikacija tu je došlo na nekoliko problema. Prvi problem je bio što ne postoje nikakvi javni web servisi koji bi pružali takve informacija. Obzirom na trenutačne mogućnosti procesiranja mobilnih uređaja drugi problem je što bi korisniku bilo neprihvatljivo čekati dok se podaci prikupe s 3 različite web stranice, semantički obrade na uređaju te se prikažu na ekranu.

Stoga je osmišljen sustav koji se sastoji od 3 potpuno različite aplikacije, a sve u svrhu stvaranja dobrog finalnog proizvoda u ovom slučaju mobilne aplikacije. Svaka aplikacija će biti u detalje objašnjena kroz pojašnjenje svrhe aplikacije i načina izvođenja.

Aplikacije :

1. Collector-HJP – aplikacija kojoj je primarna funkcija dohvaćanje dnevnih voznih redova za određene gradove, parsiranje tih podataka te spremanje u Exist bazu podataka.
2. Web-Servis-HJP – ovo je aplikacija koja dohvaća podatke s obzirom na određene parametre iz Exist baze podataka te pruža web servis koji će poslužiti mobilnoj aplikaciji za prikazivanje tih rezultata na mobilnom uređaju.

3. Hrvatski javni prijevoz – mobilna aplikacija koja ima početnu formu gdje se odabiru potrebni zahtjevi koji se proslijeđuju web servisu te prikazuje podatke koje dobiva od web servisa u tablici.



Slika 8.1. Prikaz skice kompletnog sustava

9. Prva aplikacija, Collector-HJP

9.1. Opis

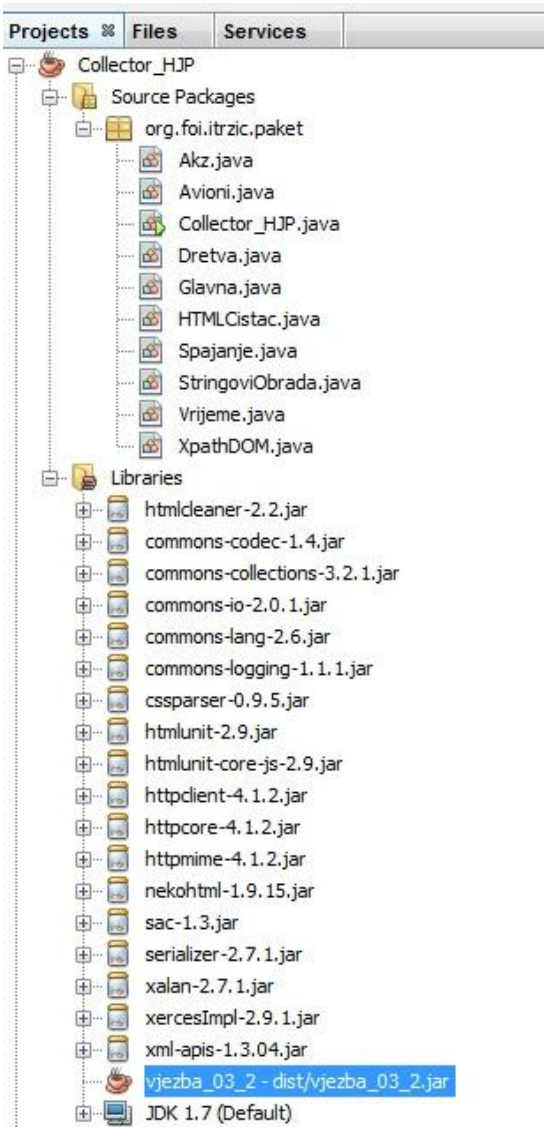
Collector-HJP je stolna Java aplikacija koja služi kao prikupljač podataka za mobilnu aplikaciju, stoga takav naziv. Njezin glavni zadatak je da za određeni konačan broj stanica dohvaća dnevne podatke od busevima, vlakovima te avionima koji se kreću između tih gradova. Te podatke potom sintaktički analizira na adekvatan način te sprema u Exist bazu podataka. Aplikacija radi na taj način da se prikupljač podataka pokreće u 00:01 i prikuplja podatke za tekući dan. S obzirom da su podaci u HTML obliku koristim vanjske biblioteke koji čiste HTML pa je kasnije lakše izvući podatke. Glavne klase su Autobusi, Avioni i Vlakovi. Autobusi je klasa koja prikuplja podatke sa stranica autobusnog kolodvora Zagreb, klasa Avioni prikuplja podatke sa stranica Croatia Airlines – a, a klasa Vlakovi prikuplja podatke sa stranica hrvatskih željeznica. Ispod imamo popis klasa sortirano abecednim redoslijedom, a prilikom objašnjavanja prvo ću objasniti manje „važne“ klase pošto se one koriste u glavnim klasama.

Popis klasa u aplikaciji:

1. Autobusi
2. Avioni
3. Collector_HJP
4. Dretva
5. Vlakovi HTMLCistac
6. Spajanje
7. StringoviObrada
8. Vrijeme
9. XpathDOM

Popis biblioteka i vanjskih projekata:

1. HtmlCLeaner
2. HtmlUnit
3. Vjezba_03_02



Slika 9.1. Prikaz svih klasa i librarya

9.2. Objašnjenja rada librarya i vanjskih projekata

9.2.1. Html Cleaner

HtmlCleaner[32] je analizator sintakse otvorenog koda napisan u Javi. Može poslužiti za više stvari kao na primjer za popravljanje loše strukturiranog HTML koda. Pri bilo kakvoj ozbiljnoj obradi html dokumenata prvo je potrebno pročistiti i urediti čvorove, atribute i običan tekst. HtmlCleaner prati slična pravila koja većina web preglednika koriste kako bi kreirali DOM (eng. Document Object Model). Html Cleaner se može koristiti u Java kodu, može se pokrenuti kao alat s komandne linije ili kao ant zadatak[33]. Oblikovan je na takav način da je malen, nezavisan i fleksibilan. Prilikom kreiranja je zamišljen s namjenom da priprema obični HTML kod za XML

[32] Html Cleaner početna stranica, dostupno na: http://htmlcleaner.sourceforge.net/
[33] Ant zadatak ima najveću upotrebu za izgradnju Java aplikacija. Preko njega se na efektivan način mogu izgrađivati i ostale aplikacije poput onih napisanih u C ili C++ kodu ali se najviše koristi u Javi pošto je to Java biblioteka.

procesiranje s Xpathom, Xqueryem i XSLT - om i u ovom diplomskom radu se koristi upravo na taj način ali se može koristiti na još mnogo načina. Kroz objašnjavanje koda aplikacije će biti napomenuto gdje se točno koristi i na koji način.

Primjer loše struktiriranog HTML koda koji sadrži nezatvorene čvorove vidimo na slici 9.2.

```
<table id=table1 cellspacing=2px
    <h1>CONTENT</h1>
    <td><a href=index.html>1 -> Home Page</a>
    <td><a href=intro.html>2 -> Introduction</a>
```

Slika 9.2. Primjer loše strukturiranog HTML koda[34]

Kada HtmlCleaner obradi ovaj isječak HTML koda dobije se strukturiran kod kao na slici 9.3.

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
   <head />
   <body>
      <h1>CONTENT</h1>
      <table id="table1" cellspacing="2px">
         <tbody>
            <tr>
               <td>
                  <a href="index.html">1 -&gt; Home Page</a>
               </td>
               <td>
                  <a href="intro.html">2 -&gt; Introduction</a>
               </td>
            </tr>
         </tbody>
      </table>
   </body>
</html>
```

Slika 9.3. Obrađeni HTML kod[35]

Mogućnosti HtmlCleaner – a:

- Sintaktički obrađuje uneseni HTML i generira stablastu strukturu prikladnu za daljnju obradu.
- Mogući izlazi iz DOM strukture su XML, HTML, DOM ili JDOM.
- Faze analize sintakse se oslanjaju na opise oznaka koje korisnik može podesiti.
- Ponašanje mu se može podesiti kroz veliki broj parametara
- Jedna instanca dretve može čistiti nekoliko HTML izvora u isto vrijeme.
- Zahtjeva JRE 1.5+

[34] Slika preuzeta sa: http://htmlcleaner.sourceforge.net/
[35] Slika preuzeta sa: http://htmlcleaner.sourceforge.net/

9.2.2. HtmlUnit

HtmlUnit[36] je jedan od skupa preglednika bez korisničkog sučelja. HtmlUnit je orginalno napisano Mike Bowler – a iz Gargoyle Software – a i izdan je pod Apache 2 licencom. Od tada doživio je mnoga dorađivanja od strane drugih programera. Preglednik bez korisničkog sučelja je u biti preglednik bez grafičkog sučelja. Šta to točno znači? To je programski proizvod koji simulira ponašanje korisnika, no rezultate ne prikazuje samom korisniku, već ih koristi za svoje interne potrebe. Njegove potrebe ovise o načinu na koji se preglednika bez korisničkog sučelja koristi. Tipično se koristi za testiranja ili dohvaćanja informacija s internet stranica.
Na primjer takav preglednik se može koristiti za prikupljanja podataka koji nisu odmah prikazani u HTML izvornom kodu već dinamički dohvaćeni JavaScriptom ili AJAXom. Može simulirati Firefox ili Internet Explorer preglednik, a to zavisi o tome koja se konfiguracija želi koristiti. Čak je sposoban sam ispunjavati određene obrasce te prikupljati podatke koje dolaze nakon što se dohvate rezultati tog upita. On dozvoljava API koji omogućuje pozive web stranica, popunjavanje obrazaca, klikanje na poveznice itd., kao da se to izvodi u „normalnom“ pregledniku.

Mogućnosti HtmlUnit – a[37]:

- Podrška za HTTP i HTTPS protokole
- Podrška za kolačiće (eng. Cookies)
- Podrška za POST i GET metode
- Mogućnost podešavanja zaglavlja koja se šalju na poslužitelj
- Podrška za HTML odgovore
    - Laki dohvat informacija koje se nalaze unutar HTML stranica
    - Podrška za slanje obrazaca
    - Podrška za klikanje poveznica
    - Podrška za prolaz kroz DOM model tekućeg HTML dokumenta
- Podrška za proxy poslužitelje
- Podrška za osnovnu i NTLM autentikaciju
- Izvrstan za Javascript podršku

Primjeri funkcionalnosti HtmlUnit – a:

1. Način na koji se određuje kako imitirati određeni preglednik

[36] HtmlUnit početna stranica, dostupno na: http://htmlunit.sourceforge.net/
[37] HtmlUnit početna stranica, dostupno na: http://htmlunit.sourceforge.net/

2. Pronalaženje specifičnog elementa na stranici po identifikatoru div elementa

```
@Test
public void homePage_Firefox() throws Exception {
    final WebClient webClient = new WebClient(BrowserVersion.FIREFOX_2);
    final HtmlPage page = webClient.getPage("http://htmlunit.sourceforge.net");
    assertEquals("HtmlUnit - Welcome to HtmlUnit", page.getTitleText());

    webClient.closeAllWindows();
}
```

Slika 9.4. Prikaz koda kojim se imitira određeni preglednik

```
@Test
public void getElements() throws Exception {
    final WebClient webClient = new WebClient();
    final HtmlPage page = webClient.getPage("http://some_url");
    final HtmlDivision div = page.getHtmlElementById("some_div_id");
    final HtmlAnchor anchor = page.getAnchorByName("anchor_name");

    webClient.closeAllWindows();
}
```

Slika 9.5. Prikaz koda koji pronalazi specifični div element

9.2.3. Vjezba_03_02

Vjezba_03_02 je Java stolna aplikacija napravljena na vježbama iz predmeta napredne web tehnologije i servisi koja je prilagođena potrebama diplomskog projekta. To je Java projekt ima funkcionalost učitavanja konfiguracijskih postavki iz .xml ili .txt datoteka.

Na primjer imamo u datoteci konfiguracija.txt postavke datum=17.05.12. i interval=230 koje se učitavaju u aplikaciju Collector-HJP te se u njoj koriste.

To je korisno jer ako se neke stvari koriste na više mjesta u projektu nema smisla jednoznačno odrediti jer ako se ta vrijednost promijeni bit će ju potrebno mijenjati na više mjesta (sve dok je to 3-4 mjesta nije problem, ali ako je preko 10 npr 200-300 onda već jest).

Tako na primjer možemo varijablu interval koristiti kada želimo odrediti koliko dugo nam dretva želi pauzirati tj. „spavati“ (eng. Sleep).

9.3. Objašnjenja rada pojedinih klasa

9.3.1. Collector-HJP

Collector-HJP je glavna klasa Java projekta Collector-HJP. Ona ne prima nikakve parametre, a instancira klasu Dretva. Prije nego što se instancira klasa Dretva dohvaća se konfiguracijsku datoteku konfiguracija.txt ukoliko ona postoji. U njoj se nalaze određeni konfiguracijski parametri koji su bitni za rad aplikacije. Ukoliko konfiguracijske datoteke ispisuje se poruka na konzoli da nema konfiguracijske datoteke i ostatak aplikacije se ne izvodi. Nakon što se dohvate konfiguracijski parametri spremaju se u varijablu konfig koji je instanca klase Konfiguracija.

Konfig se potom proslijeđuje kao parametar u klasu Dretva, a naposljetku i se proslijeđuje na isti način u 3 glavne klase Vlakovi, Autobusi i Avioni. Kao što vidimo klasa Dretva se izvodi u 00:01 budući da je tada prvi puta pokrenuti te zatim pauzira 24h i tako svaki dan.

```
String konfDatoteka = "konfiguracija.txt";
Konfiguracija konfig = null;
 try {
     konfig = KonfiguracijaApstraktna.preuzmiKonfiguraciju(konfDatoteka);
 } catch (NemaKonfiguracije ex) {
   System.out.println("Nema konfiguracijske datoteke");
 }

 Dretva dretva = new Dretva(konfig);
 dretva.start();
 Dretva.sleep(86400*1000);
```

Slika 9.6. Odsječak koda iz klase Dretva

9.3.2. Dretva

Klasu Dretva je specijalizacija klase Thread. Ona pokreće tri glavne klase koje funkcioniraju neovisno jedna o drugoj.

9.3.3. HTMLCistac

U klasi HTMLCistac koristimo već spomenutiu biblioteku HtmlCleaner. Kao i ostale „manje“ klase to je izdvojeno u posebnoj klasi i posebnoj metodi budući da se koristi na više mjesta u aplikaciji.

Kodom na slici ispod postavljamo određene postavke koje su nam bitne kao priprema za čišćenje HTML stranice.

```
HtmlCleaner cleaner = new HtmlCleaner();
CleanerProperties props = cleaner.getProperties();
props.setAllowHtmlInsideAttributes(true);
props.setAllowMultiWordAttributes(true);
props.setRecognizeUnicodeChars(true);
props.setOmitComments(true);

return cleaner;
```

Slika 9.7. Prikaz koda kojim se postavljaju postavke Html Cleaner – a

9.3.4. Spajanje

Klasa Spajanje je primarno zamišljena kao klasa u kojoj će preko poveznice dohvatiti sadržaj stranice te poveznice koju smo proslijedili te će se pomoću xpath izraza vratiti pročišćeni podaci pogodni za daljnju obradu upravo kao što možemo vidjeti slici 9.8. u metodi dohvatURL(String

link, String xpath). Kasnije je klasa Spajanje proširena još jednom metodom izvlačenjeIzStringa(String obrada, String xpath) koju vidimo na slici 9.9. kada se javila potreba da selektiranjem određenih podataka iz veće količina znakovnih podataka koji su dohvaćeni preko drugim metoda. Dakle metoda izvlačenjeIzStringa(String obrada, String xpath) je varijacija metode dohvatURL(String link, String xpath) gdje se izbacuje potreba dohvaćanja podataka preko poveznice.

```
HTMLCistac cistac = new HTMLCistac();
cleaner = cistac.cisti();

 URL url = new URL (link);
  URLConnection conn = url.openConnection();
  node_privremeni = cleaner.clean(new InputStreamReader(conn.getInputStream()));
  Object [] node = node_privremeni.evaluateXPath(xpath);

  return node;
```

Slika 9.8. Prikaz metode dohvatURL

```
HTMLCistac cistac = new HTMLCistac();
cleaner = cistac.cisti();

node_privremeni = cleaner.clean(obrada);
Object [] node = node_privremeni.evaluateXPath(xpath);

  return node;
```

Slika 9.9. Prikz metode izvlacenjeIzStringa

9.3.5. StringoviObrada

Prilikom rada se javila potreba za manipulacijom određenih znakovnih nizova. Kao primjer ćemo navesti podatke o vlakovima koji se mogu dohvaćati preko poveznica ali prilikom slanja naziva grada kao odredišta ili polazišta hrvatska slova Ž, Š, Đ, Č, Ć se mijenjaju drukčijim znakovima. Način na koji se to riješava je metoda koja prepoznaje te znakove i automatski ih mijenja u slučaju da grad ima ta slova. U suprnotnom slučaju ne bi se dobile nikakve povratne informacije o vlakovima koji putuju između tih gradova. Primjer takve metode možemo vidjeti ispod.

```
public String zamjenaHrvZnakovaVlak(String parametar){
    this.rijec = parametar;

    rijec = rijec.replace(" ", "+");
    rijec = rijec.replace("Ž", "%8E");
    rijec = rijec.replace("Š", "%8A");
    rijec = rijec.replace("Đ", "%D0");
    rijec = rijec.replace("Č", "%C8");
    rijec = rijec.replace("Ć", "%C6");


    return rijec;
}
```

Slika 9.10. Prikaz metode zamjenaHrvZnakovVlak

Također prilikom rada sa Exist bazom ista ta slova se nemogu zapisati već zbog razlike u načinu kodiranja znakova dolazi do nečitljivosti podataka. Iz tog razloga se tim slovima pridružuje slovo X kako bi se moglo raspoznati na kojim mjestima se trebaju vratiti određena slova. Naravno za svaku ovu metodu postoji suprotnu metodu koja vraća znakovni niz u prijašnje stanje.

```
public String zamjenaUXZnak(String parametar){
    this.rijec = parametar;

    rijec = rijec.replace(" ", "+");
    rijec = rijec.replace("Ž", "ZX");
    rijec = rijec.replace("Š", "SX");
    rijec = rijec.replace("Đ", "DX");
    rijec = rijec.replace("Č", "CX");
    rijec = rijec.replace("Ć", "CXX");


    return rijec;
}
```

Slika 9.11. Prikaz metode zamjenaUXZnak

Unutar klase StringoviObrada imamo 6 sličnih metoda koje obavljaju isti zadatak kao metoda na slici 9.11:

- zamjenaHrvZnakovaVlak(String parametar) – zamjenjuje hrvatske znakove u klasi Vlakovi
- zamjenaNazadUHrvZnakoveVlak(String parametar) – vraća znakove u hrvatske u klasi Vlakovi
- zamjenaHrvZnakovaBus(String parametar) – zamjenjuje hrvatske znakove u klasi Autobus

- zamjenaNazadUHrvZnakoveBus(String parametar) - vraća znakove u hrvatske u klasi Vlakovi
- zamjenaUXZnak(String parametar) – dodaje znak X uz određena hrvatska slova kako bi se moglo ispravno zapisati u bazu.
- zamjenaZamjenaIzXZnak(String parametar) – prilikom dohvaćanja znakova iz baze vraća ih nazad u hrvatska slova.

9.3.6. XpathDOM

XpathDOM klasa sadrži metodu kreiranjeXMLFilea (String obrada) koja kao parametar dobiva znakovni niz sa stranice autobusnog kolodvora Zagreb dohvaćenu iz tablice „rezultati“. Kodom na slici ISPOD taj string obrada ubacujemo u datoteku koju privremeno kreiramo pod nazivom Obrada.xml kako bi se moglo izvršiti dodatna selekcija podataka preko Xpath – a u glavnoj klasi Autobusi.

```
public void kreiranjeXMLFilea (String obrada){
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

    DocumentBuilder builder;
    try {
        builder = factory.newDocumentBuilder();


        Document document = builder.parse(new InputSource(new StringReader(obrada)));

        TransformerFactory tranFactory = TransformerFactory.newInstance();
        Transformer aTransformer = tranFactory.newTransformer();
        Source src = new DOMSource(document);
        Result dest = new StreamResult(new File("Obrada.xml"));
        aTransformer.transform(src, dest);
    } catch (Exception e) {

        e.printStackTrace();
    }

}
```

Slika 9.12. Prikaz metode kreiranjeXMLFilea

9.3.7. Vrijeme

Klasa Vrijeme je klasa koja se sadrži tri metode koje se bave obradom vremena. Budući da u mobilnoj aplikaciji želimo napraviti sortiranje rezultata po određenim kriterijima poput brzine trajanja putovanja i cijene kako bi povećala fleksibilnost i uporabljivost aplikacije razvijeni su algoritami izračuna vremena trajanja putovanja. Prilikom dohvata podataka za vlakove i avione informacija o trajanju putovanja je već prikazana stoga se ova metoda koristi samo u klasi Autobusi. Metoda obradaVremena(String vOd, String vOd) prima kao parametre vrijeme

polaska i vrijeme dolaska. Prvo se izvlače minute i sati posebno od svakoga vremena i pretvaraju u cjelobrojne vrijednosti jer nam je oblik podatka u početku znakovni niz. Nakon toga se vrijeme pretvara u minute. Postoje određeni slučajevi kada autobus kreće kasno navečer pa dolazi drugi dan. Ako dođe do toga slučaja vremenu dolaska se prirodaje 1440 minuta tj. jedan dan. Nakon toga preostaje samo oduzeti minute te pretvoriti nazad u sate i minute. Na kraju se taj rezultat pretvara u znakovni niz i vraća nazad. Naglasak je na tome ako je broj jednoznamenkasti formatira se na način da mu se dodaju 0 ispred tog broja jer su na taj način prikazani sva ostala vremena putovanja.

```
public String obradaVremena(String vOd, String vDo) {
  this.vrijemeOd = vOd;
  this.vrijemeDo = vDo;

  h1_priv = vrijemeOd.substring(0, 2);
  m1_priv = vrijemeOd.substring(3, 5);

  h2_priv = vrijemeDo.substring(0, 2);
  m2_priv = vrijemeDo.substring(3, 5);


  h1 = Integer.parseInt(h1_priv);
  m1 = Integer.parseInt(m1_priv);
  m1 = m1 + (h1 * 60);

  h2 = Integer.parseInt(h2_priv);
  m2 = Integer.parseInt(m2_priv);
  m2 = m2 + (h2 * 60);


  if (m2 < m1) {
      m2 = m2 + 1440;
  }


  vrijeme_putovanja = m2.floatValue() - m1.floatValue();
  vrijeme_putovanja = vrijeme_putovanja / 60;

  Float razlika = vrijeme_putovanja - vrijeme_putovanja.intValue();
  h = vrijeme_putovanja.intValue();
  String hours = String.format("%02d", h);

  m = razlika * 60;
  Integer minute = m.intValue();
  String minutes = String.format("%02d", minute);

  vrijeme = hours + ":" + minutes;

  return vrijeme;

}
```

Slika 9.13. Prikazuje metodu obradaVremena

Ova metoda uzima današnji datum formatira ga na način dd.MM.yy, ispisuje to na konzoli radi provjere i vraća nazad. Metoda datumDanasZaAvione je slična samo što sadrži drukčiji oblik formatiranja.

```
    public String datumDanasZaVlak(){
        Calendar currentDate = Calendar.getInstance();
SimpleDateFormat formatter= new SimpleDateFormat("dd.MM.yy");
String datum = formatter.format(currentDate.getTime());
System.out.println("Današnji datum je:  " + datum);
return datum;
    }
```

Slika 9.14. Prikaz metode datumDanasZaVlak

9.3.8. Vlakovi

Klasa Vlakovi je klasa koja se bavi dohvatom rasporeda vlakova za odabrani broj gradova. Primjer dohvaćenih podataka na njihovoj stranici možemo vidjeti na slici ispod.

Relacija: ZAGREB GL. KOL. – VARAŽDIN
Datum polaska: 19.05.12.

SAMO IZRAVNI VLAKOVI

Okvirni izraèun cijene prijevozne karte

Polazak iz kolodvora ZAGREB GL. KOL.	Vlak, kategorija i naziv	Dolazak	Trajanje putovanja	Smjer
04:34	3000	07:42	03:08	VARAŽDIN 07:42
07:16	992	09:41	02:25	VARAŽDIN 09:41
09:34	3004	12:06	02:32	VARAŽDIN 12:06
11:16	3006	13:53	02:37	VARAŽDIN 13:53
14:09	994	16:37	02:28	VARAŽDIN 16:37
15:13	790	17:23	02:10	VARAŽDIN 17:23
15:26	3010	18:13	02:47	VARAŽDIN 18:13
18:24	3014	21:06	02:42	VARAŽDIN 21:06

Slika 9.15. Primjer rezultata na relaciji Zagreb - Varaždin[38]

Na početku metode dohvatPodataka() koja ne prima nikakve parametre instanciraju se klase Vrijeme, StringoviObrada, HtmlCistac i Spajanje. Obavljaju se potrebne pripreme kao dohvaćanje današnjega datuma, postavke za bazu, postavljanje svojstva čistača.. Potom se iz baze dohvaćaju gradovi za koje se prikupljaju podaci.

[38] Slika preuzeta sa:
http://vred.hznet.hr/hzinfo/?category=hzinfo&service=vred3&nkod1=&nkdo1=&lang=hr&screen=4

```
Vrijeme vrijeme = new Vrijeme();
datum = vrijeme.datumDanasZaVlak();
//dohvaca postavke iz datoteka
baza = konfig.dajPostavku("baza");

//inicijalizacija klase StringoviObrada
StringoviObrada stringovi = new StringoviObrada();

//inicijalizacija i podesavanje HTMLCleanera
  HTMLCistac cistac = new HTMLCistac();
  cleaner = cistac.cisti();


  //dohvacanje stanica iz baze
Spajanje url = new Spajanje();
node_stanice = url.dohvatURL(baza+"/?_query=for%20$stanica%"
        + "20in%20doc%28%22stanice.xml%22%29/stanice/stanica"
          + "%20where%20@category=%27vlak%27%20return"
        + "%20$stanica/naziv", xpath_stanice);
```

Slika 9.16. Instaciranje objekata i spajanje na bazu

Nakon toga se radi dvostruku petlja kako bi prošlo kroz sve kombinacije gradova i dohvatili moguće podatke za njih. Kao što je već napomenuo tu je nužna pretvorba znakovnih nizova gradova zbog dohvata iz baze i slanje zahtjeva preko poveznice.

```
for(int i=0; i<node_stanice.length; i++){
    for (int j=0; j<node_stanice.length; j++){
        stanicaOd = node_stanice[i].toString();
        stanicaDo = node_stanice[j].toString();


stanicaOd = stringovi.zamjenaIzXZnak(stanicaOd);
stanicaOd = stringovi.zamjenaHrvZnakovaVlak(stanicaOd);
System.out.println(stanicaOd);
stanicaDo = stringovi.zamjenaIzXZnak(stanicaDo);
stanicaDo = stringovi.zamjenaHrvZnakovaVlak(stanicaDo);
System.out.println(stanicaDo);


URL urlovi = new URL("http://vred.hznet.hr/hzinfo/Default.asp?NKOD1=" + stanicaOd + "&ddList1=" + stanicaOd +
        "&ODH=&NKDO1=" + stanicaDo + "&ddList2=" +stanicaDo + "&DOH=&K1=&K2=&DT=" + datum +
        "&DV=D&Category=hzinfo&Service=vred3&LANG=hr&SCREEN=2&SESSIONID=1332037843ICE1087");
URLConnection konekcija = urlovi.openConnection();
URLConnection konekcija1 = urlovi.openConnection();
URLConnection konekcija_izravnosti = urlovi.openConnection();
```

Slika 9.17. Zamjena znakovnih nizova i spajanje na stranice Hrvatskih željeznica

Tu se naravno radi i provjera izravnosti vlakova. U obzir dolaze samo izravni vlakovi, ako dolazi do presjedanja ti podaci se ne spremaju jer nas zanimaju samo direktne linije. Ispod možemo vidjeti dohvat koji služi za provjeru ispravnosti.

```
node_izravnost = cleaner.clean(new InputStreamReader(konekcija_izravnosti.getInputStream()));
Object [] izravnost_obj = node_izravnost.evaluateXPath(xpath_izravnost);
izravnost = izravnost_obj[0].toString();
izravnost = izravnost.replace(" ", "");
```

Slika 9.18. Provjera izravnosti dohvaćenih podataka

Na način ispod iz baze se dohvaća interni identifikator polazište i odredišta koji služi za dohvaćanje cijene putovanja. Dohvaćena cijena se izdvaja i radi se obradu nad njom.

```
Object[] node_interni2 = url.dohvatURL(baza+"/?_query=for%20$interni%20in%20doc"
        + "%28%22stanice.xml%22%29/stanice/stanica[@category=%27vlak%27]%20where%20$interni/"
        + "naziv=%27"+ stanicaDo+"%27%20return%20$interni/id_interni", xpath_interni);
interni_do = Integer.parseInt(node_interni2[0].toString());


Object [] node_cijena1 = url.dohvatURL("http://vred.hznet.hr/hzinfo/Default.asp?OD="+interni_od+"&DO="+interni_do+"&smjer=1&"
        + "DOD=250312&bp1=001&POV1=01&bp2=000&POV2=&DOP=&Via1=&Via2=&Vir1=&Vir2="
        + "&raz1=2&vrvA=4&vrvR=&Category=hzinfo&Service=cijK&SCREEN=2&LANG=HR", xpath_cijena);
cijena = node_cijena1[0].toString();
cijena = cijena.replace(" ", "");
cijena = cijena.replace("Zaplatiti:", "");
cijena = cijena.replace("kn", "");
cijena = cijena.replace("\r\n", "");
System.out.println("Cijena putovanja: "+cijena);
```

Slika 9.19. Dohvaćanje internih identifikatora i cijene putovanja

Nakon toga se dohvaćaju brojevi redaka tablice koja dolazi kao rezultata upita na web stranici. To služi kao ograničenje kada se prolazi kroz pojedine retke tablice, izdvajaju podaci te se zapisuju u bazu podataka. Implementaciju toga možemo vidjeti na slici 9.20.

```
node_broj_redaka = cleaner.clean(new InputStreamReader(konekcija.getInputStream()));
Object[] node_redci = node_broj_redaka.evaluateXPath(xpath_broj_redaka);
broj_redaka = Integer.parseInt(node_redci[0].toString());
System.out.println(broj_redaka);

  if(broj_redaka>1){

node = cleaner.clean(new InputStreamReader(konekcija1.getInputStream()));
for(int k=2; k<=broj_redaka; k++){
Object[] node_print = node.evaluateXPath("//table[2]/tbody/tr["+k+"]/td/text()");



        vrijeme_polazak = node_print[0].toString();
        vrijeme_dolazak = node_print[2].toString();
        vrijeme_dolazak = vrijeme_dolazak.replace("\r\n", "");
        vrijeme_trajanje = vrijeme_dolazak.substring(5, 10);
        vrijeme_dolazak = vrijeme_dolazak.substring(0,5);
```

Slika 9.20. Izvlačenje podataka pojedinog putovanja

Naposljetku sa svim dohvaćenim podacima se rezultat zapisuje u bazu podataka.

```
URL urls = new URL(baza+"/?_query=let $upis:= doc('baza.xml') "
        + "return ( update insert "
        + "<putovanja category='vlak'><polaziste>" + stanicaOd + "</polaziste>"
        + "<odrediste>" + stanicaDo + "</odrediste>"
        + "<datum>" + datum + "</datum>"
        + "<vrijeme_polazak>" + vrijeme_polazak + "</vrijeme_polazak>"
        + "<vrijeme_dolazak>" + vrijeme_dolazak + "</vrijeme_dolazak>"
        + "<vrijeme_trajanje>" + vrijeme_trajanje + "</vrijeme_trajanje>"
        + "<cijena>" + cijena + "</cijena></putovanja>"
        + "into $upis/baza, $upis)");

URLConnection konekcija_upis_u_exist = urls.openConnection();
privremeno = cleaner.clean(new InputStreamReader(konekcija_upis_u_exist.getInputStream()));
```

Slika 9.21. Zapis dohvaćenih rezultata u bazu podataka

9.3.9. Avioni

Metoda dohvatPodataka() u klasi Avioni je dosta nalik metodi dohvatPodataka() u klasi Vlakovi ali se opet razlikuju u nekim bitnim segmentima koji se vežu uz specifičnosti pojedinih web stranica Croatia Airelines – a i Hrvatskih željeznica. Na početku se također dohvaća današnje vrijeme ali je potrebna drukčija obrada točnije posebno izvlačenje dana i mjeseca zbog specifičnosti poveznice za dohvat podataka. Potom se dohvaćaju aerodromi iz baze za koje želimo dohvaćati podatke te se pomoću XPath – a izvlače nazivi. Upit možemo vidjeti na slici 8.22.

```
node_stanice = url.dohvatURL(baza+"/?_query=for%20$stanica%20in%20"
        + "doc%28%22stanice.xml%22%29/stanice/stanica"
        + "%20where%20@category=%27avion%27%20return%"
        + "20$stanica/id_interni", xpath_stanice);
```

Slika 9.22. Dohvaćanje naziva stanica

Kod dohvata ovih podataka također se broji koliko postoji redaka na stranici te se od ukupnog broja redaka oduzima 6 budući da na stranici bez podataka postoji već 6 redaka. Nakon toga se svaki redak dohvaća posebno, radi se obrada i zapisuje u bazu podataka. Prilikom obrade imamo određene provjere zbog toga što se u rezultatima mogu pronaći neizravni letovi te letovi partnerskih kompanija koje ne uzimamo u obzir prilikom spremanja podataka. Potom se izvlače polazište, odredište, vrijeme polaska, vrijeme dolaska, radi obrada koja uključuje vrijeme polaska i dolaska kako bi se dobilo vrijeme putovanja kao što možemo vidjeti na slici 9.23.

```
vrijeme_polazak = podaci.substring(8,10) + ":" + podaci.substring(10,12);
vrijeme_dolazak = podaci.substring(15,17) + ":" + podaci.substring(17,19);
vrijeme_trajanje = vrijeme.obradaVremena(vrijeme_polazak, vrijeme_dolazak);
```

Slika 9.23. Izvlačenje vremena polaska, dolaska i računanje vremena trajanja putovanja

Nakon što su svi potrebni podaci koji opisuju jedan let dohvaćeni let se zapisuje u bazu.

```
    URL urlovi = new URL(baza+"/?_query=let $upis:= doc('baza.xml') "
+ "return ( update insert "
+ "<putovanja category='avion'><polaziste>" + stanicaOdNova + "</polaziste>"
+ "<odrediste>" + stanicaDoNova + "</odrediste>"
+ "<datum>" + datum + "</datum>"
+ "<vrijeme_polazak>" + vrijeme_polazak + "</vrijeme_polazak>"
+ "<vrijeme_dolazak>" + vrijeme_dolazak + "</vrijeme_dolazak>"
+ "<vrijeme_trajanje>" + vrijeme_trajanje + "</vrijeme_trajanje>"
+ "<cijena>" + cijena + "</cijena></putovanja>"
+ "into $upis/baza, $upis)");

URLConnection konekcija_upis_u_exist = urlovi.openConnection();
InputStreamReader upis = new InputStreamReader(konekcija_upis_u_exist.getInputStream());
System.out.println("Upisano u bazu!");
upis.close();
```

Slika 9.24. Upisivanje dohvaćenih rezultata u bazu podataka

Kao što možemo vidjeti po upitu zapisi u pojedinim glavnim klasama se razlikuju po kategoriji. To možemo primjetiti u trećem retku upita <putovanja category='avion'>. S obzirom na različite klase category je: vlak, avion, autobus.

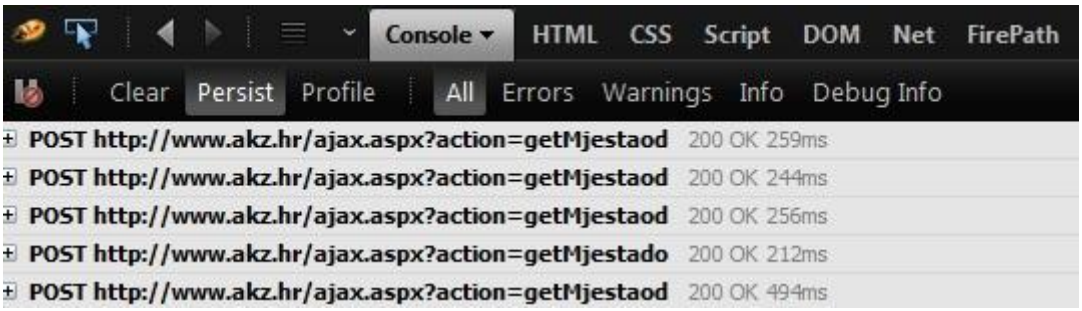
9.3.10. Autobusi

Klasa Autobusi ima također metodu dohvatPodataka() ali je u samoj srži vrlo različita implementacija od ostalih dviju glavnih klasa. To je iz razloga što je sama stranica drukčije napravljena. Budući da je napravljena HTML – om, CSS – om, JavaScript - om, AJAX - om način koji smo do sada dohvaćali podatke, a to je dohvaćanje podataka slanjem zahtjeva u poveznici te izvlačenje podataka više nije moguće. Rasporede autobusa možemo pronaći na stranici autobusnog kolodvora Zagreb[39]. Da bi se podaci dohvatili potrebno je prethodno ispuniti formu.

[39] Vozni red autobusnog kolodovora Zagreb, dostupno na: http://www.akz.hr/default.aspx?id=7

Slika 9.25. Prikaz stranice autobusnog kolodvora Zagreb[40]

Prilikom upisivanja gradova možemo u Firebugu pratiti da se preko AJAX zahtjeva dohvaćaju gradovi s obzirom na slova koja se unose.



Slika 9.26. Post zahtjevi za nazivima polazišta i odredišta prikazani u Firebugu

Rezultati se potom dohvaćaju AJAX - om u div element rezultati kao što možemo vidjeti na slici 9.26.

[40] Slika preuzeta sa: http://www.akz.hr/default.aspx?id=7

U cijenu su uključeni rezervacija i kolodvorska usluga.

19.05.2012 - subota >>>

Polazak	Dolazak	Veze	Cijena kn	Prijevoznik		
19.05. 06:15	19.05. 07:35		87,00	Autobusni promet d.d. u stečaju	Stanice	Popusti
19.05. 07:15	19.05. 08:35		87,00	Autobusni promet d.d. u stečaju	Stanice	Popusti
19.05. 08:00	19.05. 09:46		65,00	Croatia bus d.d. u stečaju	Stanice	Popusti
19.05. 08:45	19.05. 10:30		87,00	Autobusni promet d.d. u stečaju	Stanice	Popusti
19.05. 09:30	19.05. 11:15		87,00	Autobusni promet d.d. u stečaju	Stanice	Popusti
19.05. 11:00	19.05. 12:45		87,00	Autobusni promet d.d. u stečaju	Stanice	Popusti
19.05. 12:25	19.05. 14:10		87,00	Autobusni promet d.d. u stečaju	Stanice	Popusti
19.05. 13:00	19.05. 14:46		65,00	Croatia bus d.d. u stečaju	Stanice	Popusti
19.05. 13:45	19.05. 15:30		87,00	Autobusni promet d.d. u stečaju	Stanice	Popusti

Slika 9.27. Primjer rezultata za relaciju Zagreb - Varaždin [41]

Taj problem je riješen upotrebom već spomenute biblioteke, HtmlUnit – a koji je preglednik bez korisničkog sučelja. Spomenuto je već kako se pomoću njega mogu ispunjavati obrasci te dohvaćati podatke i upravo na taj način se dolazi do podataka sa ove stranice. Naravno tu su još implementirani već od prije poznati mehanizmi izvlačenja specifičnih podataka iz skupa podataka.

Na početku metode se instanciraju objekti klasa StringoviObrada, Vrijeme i Spajanje. Kasnije se još instancira i objekt klase XPathDOM u trenutku kada je potreban. Prvo se dohvaćaju sve stanice za koje se prikupljanju podaci te se rade potrebne promjene znakova zbog dohvaćanja iz baze kako je već objašnjeno u prijašnoj klasi samo šta se ovdje koristi specifična metoda za autobuse. Naposljetku se dolazi do ispunjavanja obrasca preko preglednika bez korisničkog sučelja. Prvo se kreira novi WebClient i spajamo se na stranicu autobusnog kolodovora Zagreb. Potom pronalazimo potrebni obrazac te se ispunjavaju potrebni elemente. Vrijednosti elemenata koji se upisuju se dohvaćaju iz baze. Nakon što je sve popunjeno „pritisnemo“ gumb prikaži koji ima identifikator pod nazivom „_submit“.

[41] Slika preuzeta sa: http://www.akz.hr/default.aspx?id=7

```
final WebClient webClient = new WebClient();
final HtmlPage page = webClient.getPage("http://www.akz.hr/default.aspx?id=7");

//pronalazenje forme vozni red i elemenata mjestood, skrivenog elementa mjestood_id, m
final HtmlForm frmVozniRed = page.getFormByName("voznired");
final HtmlTextInput input_od = frmVozniRed.getInputByName("mjestood");
final HtmlHiddenInput input_od_hidden = frmVozniRed.getInputByName("mjestood_id");

final HtmlTextInput input_do = frmVozniRed.getInputByName("mjestodo");
final HtmlHiddenInput input_do_hidden = frmVozniRed.getInputByName("mjestodo_id");

//postavljanje polazista i još bitnije postavljanje internog ID-a u skrivene elemente
System.out.println("Polaziste je: " +stanicaOd+ " s internim: " + interni_od);
input_od_hidden.setValueAttribute(interni_od);
 input_od.setValueAttribute(stanicaOd);

System.out.println("Odredište je: " +stanicaOd+ " s internim: " + interni_od);
input_do.setValueAttribute(stanicaDo);
input_do_hidden.setValueAttribute(interni_do);

//submiting forme pritiskom na gumb Prikaži
HtmlPage page2 = frmVozniRed.getInputByName("_submit").click();
```

Slika 9.28. Prikaz koda kojim popunjavamo obrazac

Nakon toga u string obrada se dohvaćamju rezultati iz div elementa rezultati. Ovdje se kreira novi privremena XML datoteka nazvana Obrada.xml u koji se upisuju ti podaci te se nakon toga izvlače na već standardni način pomoću XPath – a kao što možemo vidjeti na slici 9.29.

```
final HtmlDivision divRezultati = page.getHtmlElementById("rezultati");
String obrada = divRezultati.asXml();


XpathDOM dom = new XpathDOM();
dom.kreiranjeXMLFilea(obrada);


XPath xpath = XPathFactory.newInstance().newXPath();

InputSource inputSource = new InputSource("Obrada.xml");
```

Slika 9.29. Dohvaćanje rezultata i obrada

Pomoću XPath – a se dohvaća broj redaka i opet prolazimo kroz svaki redak pojedinačno i novim XPath izrazom izvlačimo podatke te ih prilagođavamo standardnom zapisu koji je određen za bazu podataka. Tu opet postoje provjere izravnosti autobusnih linija, da li uopće postoje busevi za ta odredišta itd.

Ispod vidimo način na koje je napravljeno izvlačenje pojedinih podataka te njihovo formatiranje.

```
vrijeme_polazak = (nodes.item(0).getTextContent());
vrijeme_polazak = vrijeme_polazak.trim();
if(vrijeme_polazak.length()>11){
datum = vrijeme_polazak.substring(0, 6);
datum = datum + "12.";
vrijeme_polazak = vrijeme_polazak.substring(7, 12);

vrijeme_dolazak = (nodes.item(1).getTextContent());
vrijeme_dolazak = vrijeme_dolazak.trim();
  if(vrijeme_dolazak.length()>11){
vrijeme_dolazak = vrijeme_dolazak.substring(7, 12);
cijena = (nodes.item(3).getTextContent());
cijena = cijena.trim();
cijena = cijena.replace(",00", "");

vrijeme_trajanje = vrijeme.obradaVremena(vrijeme_polazak, vrijeme_dolazak);
```

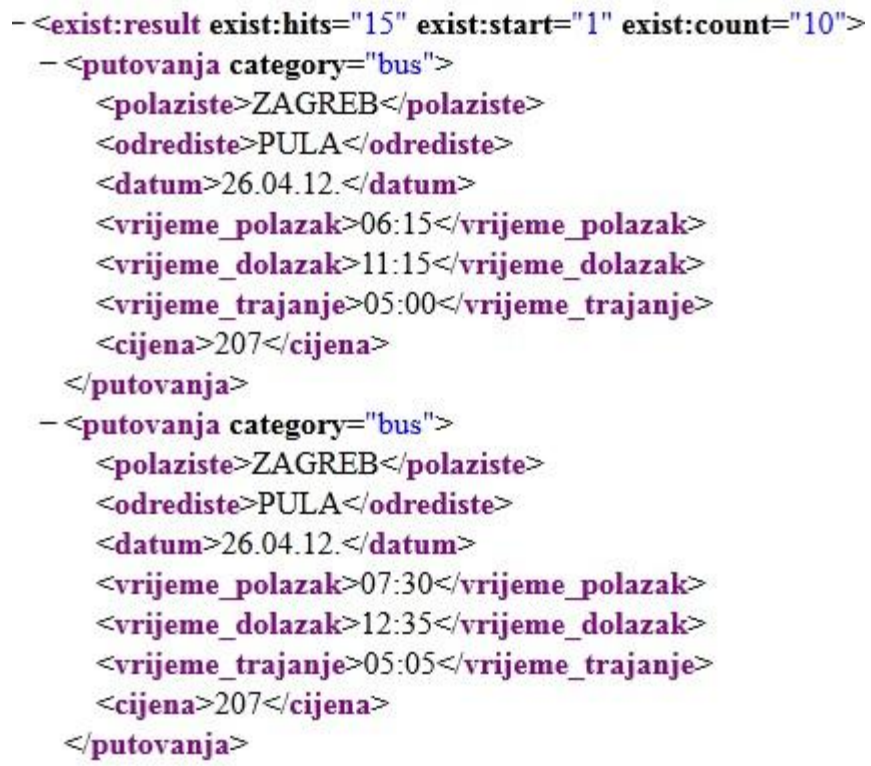
Slika 9.30. Izvlačenje potrebnih podataka

Nakon što su svi podaci dohvaćeni i formatirani na već poznati način se rezultati upisuju u bazu podataka.

10. Druga aplikacija, WebServis-HJP

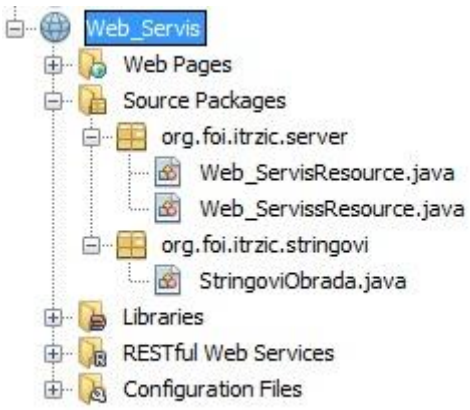
Druga aplikacija je Java Web aplikacija koja pruža web servis mobilnoj aplikaciji za podacima iz baze. Aplikacija se pokreće na poslužitelju Apache Tomcat, a web servis je RESTful.

Primjer rezultata koje nam vrati web servis za specifične parametre: http://localhost:8084/Web_Servis/resources/ws/ZAGREB/PULA/26.04.12./bus za rezultate dobivamo ovakav izgled kao na slici 10.1.



Slika 10.1. Primjer rezultata koji nam pruža web servis

Aplikacija WebServis-HJP sadrži dva paketa, org.foi.itrzic.server i org.foi.itrzic.stringovi. U org.foi.itrzic.stringovi paketu imamo klasu StringoviObrada koja je kopirana iz aplikacije Collector-HJP zbog već poznatog problema kada se gube određena slova hrvatske abecede prilikom slanja znakova preko url – a i prilikom spremanja u bazu podataka. U paketu org.foi.itrzic.server imamo klase Web_ServisResource i Web_ServissResource koji se automatski generiraju prilikom kreiranja web servisa. Modifikacije su napravljene u klasi Web_ServisResource kako bi oblikovao servis po potrebama ovog diplomskog rada.



Slika 10.2. Popis klasa koje se koriste u drugoj aplkaciji

U konstruktoru klase Web_servisResource primaju se parametri te se odrađuju potrebne promjene koje je potrebno napraviti da se podaci iz baze podataka dohvate pravilno.

```
this.polaziste = polaziste;
this.odrediste = odrediste;
this.datum = datum;
this.prioritet = prioritet;
this.prioritet = prioritet.toLowerCase();

StringoviObrada stringovi = new StringoviObrada();
this.polaziste = stringovi.zamjenaNazadUHrvZnakovaBus(this.polaziste);
this.odrediste = stringovi.zamjenaNazadUHrvZnakovaBus(this.odrediste);

this.polaziste = stringovi.zamjenaUXZnak(this.polaziste);
this.odrediste = stringovi.zamjenaUXZnak(this.odrediste);
```

Slika 10.3. Zamjena znakovnih nizova

Prilikom dohvata podataka imamo upit koji dohvaća rezultate osim po ostalim parametrima i po parametru prioritet koji može biti vlak, avion, autobus. Kao prioritet na mobilnoj aplikaciji može se još odabrati brzina putovanja i cijena putovanja u kojem slučaju imamo još dva uvjeta selekcije koji provjeravaju prioritet te vraćaju sortirane rezultate od najmanjeg prema najvećem.

```
URL link = new URL("http://localhost:8080/exist/servlet/db/Database/? "
        + "query=for%20$putovanja%20in%20doc%28%22baza.xml%22%29/baza/putovanja"
        + "%20where%20polaziste"
        + "=%22"+ polaziste +"%22%20and%20odrediste=%22"+odrediste+
        "%22%20and%20datum=%22"+ datum +""
        + "%22%20and%20@category=%27"+ prioritet+ "%27%20return%20$putovanja");

if (prioritet.equals("brzina putovanja")){
   link = new URL("http://localhost:8080/exist/servlet/db/Database/? "
           + "query=for%20$putovanja%20in%20doc%28%22baza.xml%22%29/baza/putovanja%20"
           + "where%20polaziste=%22"+ polaziste +"%22%20and%20odrediste=%22"+ odrediste
           +"%22%20and%20datum=%22"+ datum +"%22%20"
           + "order%20by%20$putovanja/vrijeme_trajanje%20return%20$putovanja");
}

 if (prioritet.equals("cijena putovanja")){
   link = new URL("http://localhost:8080/exist/servlet/db/Database/? "
           + "query=for%20$putovanja%20in%20doc%28%22baza.xml%22%29/baza/putovanja%20"
           + "where%20polaziste=%22"+ polaziste +"%22%20and%20odrediste=%22"+ odrediste
           +"%22%20and%20datum=%22"+ datum +"%22%20"
           + "order%20by%20$putovanja/cijena%20return%20$putovanja");
}
```

Slika 10.4. Provjera i dohvat podataka

Ispod imamo način na koji se cijeli dohvaćeni XML ubacuje u znakovni niz te prikazuje preko web servisa.

```
BufferedReader in = new BufferedReader(
new InputStreamReader(link.openStream()));

String stranica = "";
String inputLine;
while ((inputLine = in.readLine()) != null){
    System.out.println(inputLine);
stranica += inputLine;
}
in.close();
return stranica;
```

Slika 10.5. Kreiranje web prikaza za dohvaćene podatke

11. Treća aplikacija, Hrvatski javni prijevoz

11.1. Opis mobilne aplikacije

Hrvatski javni prijevoz je mobilna aplikacija napravljena u alatu Titanium Appcelerator. Kao što je već objašnjeno u Titanium Appceleratoru se programira pomoću JavaScript, HTML i CSS koda, a moguće je pokretati nativne aplikacije i na Android sustavu i na iOS – u.

Možemo reći da su ove sve aplikacije prije bila prethodnica ovoj aplikaciji budući da se tek tu to može opipljivo vidjeti tj. korisnik može kontrolirati aplikaciju. Aplikacija Collector-HJP je tu da skuplja podatke za mobilnu aplikaciju, a WebServis-HJP je napravljena tako da pruža podatke mobilnoj aplikaciji preko interneta.

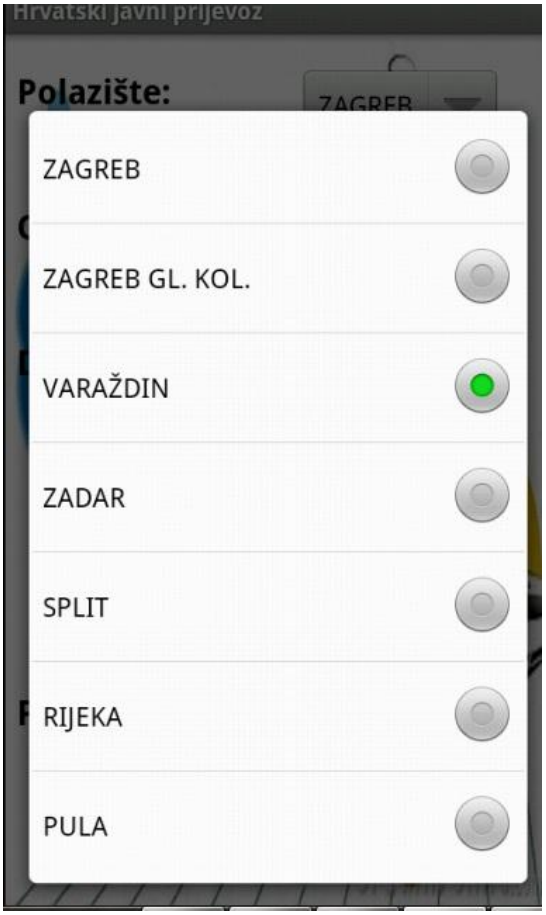
Prvo ćemo objasniti što korisnik može vidjeti i šta se može uraditi s time, a nakon toga kako aplikacija uistinu radi tj. objasniti kod koji se krije u pozadini i omogućava nam tu funkcionalnost.

Sučelje je napravljeno prilično jednostavno za korištenje. Nema nikakvih upisivanja što će većina korisnika cijeniti. Izgled početnog zaslona aplikacije možemo vidjeti na slici 10.1.

Slika 11.1. Početni obrazac mobilne aplikacije

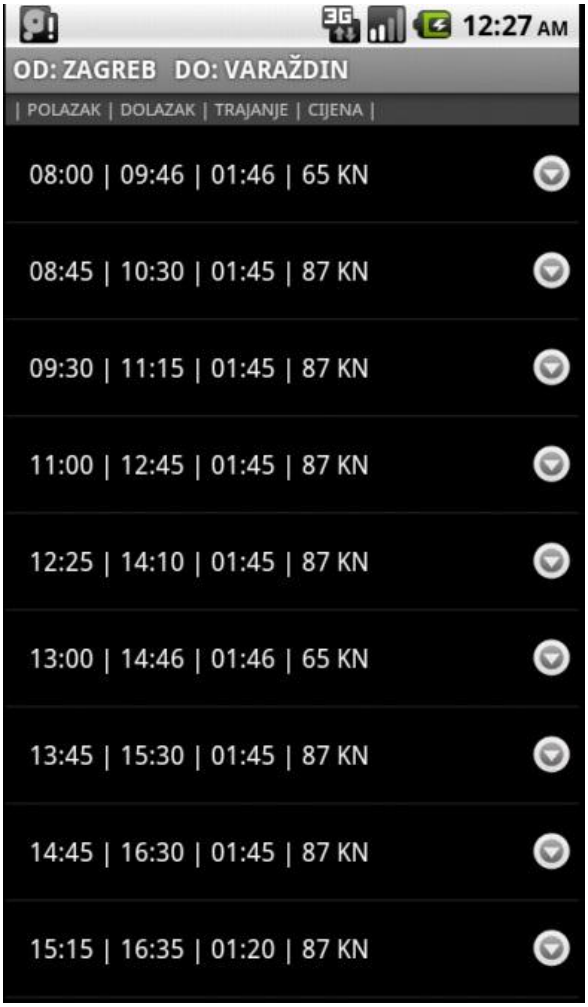
Kao što možemo vidjeti određujemo polazište, odredište iz ograničenog skupa gradova. U aplikaciji možemo odabrati iz skupa glavnih gradova hrvatske, a je to napravljeno na način da se odabire preko komponente za odabir (eng. Picker widget).

Na slici 11.2. možemo vidjeti kako izgleda odabiranje gradova.



Slika 11.2. Način odabira polazišta i odredišta

Nakon što se odaberu polazište i odredište potrebno je još podesiti datum za pretraživanje i prioritet. Prioritet je isto komponenta za odabir koji ima mogućnosti odabira putovanja vlakom, autobusom, avionom ili sortiranja prema brzini putovanja ili cijene putovanja. Ispod možemo vidjeti rezultate zahtjeva za putovanjem između Zagreba i Varaždina na dan 20.05.2012. gdje smo kao prioritet odredili autobus. Na slici 11.3. možemo vidjeti rezultate.



Slika 11.3. Prikaz dobivenih rezultata

Nakon što smo dobili rezultate može se još kliknuti na pojedino putovanje pa će nam se otvoriti novi prozor sa prikazom lokacije stanice od kuda kreće putovanje.

Slika 11.4. Prikaz početnog polazišta na karti

11.2. Pojašnjenje koda mobilne aplikacije

Aplikacija se sastoji od 3 JavaScript datoteke: app.js, dohvat_podataka.js i maps.js. Obzirom da se određeni elementi ponavljaju objasnit ćemo samo samo reprezentativne dijelove, a ne aplikaciju u potpunosti.

Početna je app.js koja se izvršava pokretanjem aplikacije. Kao što smo vidjeli u prvom prozoru aplikacije imamo obrazac koju ispunjavamo kako bi mogli poslati zahtjev za rezultatima stoga se taj dio više manje sveo na uređivanje pomoću CSS koda. Ispod vidimo na koji način kreiramo pojedine elemente na obrascu. Možemo primjetiti sad se oni svi nalaze u istom modulu Ti.UI.

Titanium.UI modul ili skraćeno Ti.UI modul se može podijeliti na 3 glavna područja :

1. Pogledi
2. Grafičke komponente
3. Prozori

Svaku kreirana kontrola se mora dodijeliti određenoj varijabli kako bi se kasnije mogla vršiti manipulacija nad njima. Na slici 11.5. vidljivo je da se prilikom kreiranja mogu dodijeliti CSS svojstva poput širine, dužine, identifikatora, udaljenosti od pojedinih margina, naslov itd.

```
var boja_labela = "#000";
var button = Ti.UI.createButton({
    title: 'Prikaži rezultate',
    height: 60,
    width: 200,
    top: 700,
    id:'gumb'

});


var pickerPolaziste = Ti.UI.createPicker({
    top:30,
    right:40,
    id:'pickerPolaziste'
});
```

Slika 11.5. Odsječak koda koji izvršava kreiranje kontrola

Ispod se vidi na koji način se kreira polje prioritet i puni podacima. Na isti način su napravljeni ostale komponente za odabir. Kasnije u kodu varijabla prioritet koja je oblika polje se pridružuje njegovoj komponenti za odabir. Na slici 11.6. se vide načini kreiranja labela te svojstava koja mu se zadaju.

```
var prioritet = [];
prioritet[0] = Ti.UI.createPickerRow({title:'VLAK'});
prioritet[1] = Ti.UI.createPickerRow({title:'AUTOBUS'});
prioritet[2] = Ti.UI.createPickerRow({title:'AVION'});
prioritet[3] = Ti.UI.createPickerRow({title:'BRZINA PUTOVANJA'});
prioritet[4] = Ti.UI.createPickerRow({title:'CIJENA PUTOVANJA'});




var lblPolaziste = Titanium.UI.createLabel({
    top:30,
    left:10,
    height:'auto',
    width:'auto',
    text:'Polazište: ',
    textAlign:'left',
    color:boja_labela,
    font:{fontFamily:'Arial Black', fontSize:30, fontWeight:'bold'}

});
```

Slika 11.6. Punjenje varijable prioritet i uređivanje labele

Na slici 11.7. vidimo pridruživanje polja njihovim komponente za odabir te kako se na isti način sve kontrole moraju dodati na scrollView, a scrollView u prozor. Kada bi se sve kontrole

direktno dodavale na prozor ako u ekran mobilnog uređaja ne bi stao prozor u cijelosti ne bi se mogao pomicati pogled. Na ovaj način je taj problem riješen.

```
pickerPrioritet.add(prioritet);
pickerPrioritet.selectionIndicator = true;


scrollView.add(pickerPolaziste);
scrollView.add(pickerOdrediste);
scrollView.add(pickerPrioritet);
scrollView.add(pickerDatum);
scrollView.add(button);
scrollView.add(lblPolaziste);
scrollView.add(lblOdrediste);
scrollView.add(lblPrioritet);
scrollView.add(lblDatum);

win.add(scrollView)
win.open();
```

Slika 11.7. Dodavanje svih kontrola na scrollView

Kodom na slici 11.8. određujemo koje će opcije biti selektriranje prilikom prvog učitavanja aplikacije. Ispod toga vidimo slušaće događaja koji osluškuju i upisuju pravilne vrijednosti ukoliko dođe do promjene odabrane vrijednosti kod bilo koje komponente za odabir na obrascu.

```
var polaziste = pickerPolaziste.getSelectedRow(0,0,false).title;
var odrediste = pickerOdrediste.getSelectedRow(0,2,false).title;
var prioritet = pickerPrioritet.getSelectedRow(0,3,false).title;

pickerDatum.addEventListener('change', function(e){
     datum = e.value;

});


pickerPolaziste.addEventListener('change', function(e){
    polaziste = e.row.title;
});
```

Slika 11.8. Postavljanje početnih vrijednosti pickera

Kod na slici 11.9. se nalazi unutar slušača događaja koji reagira na pritisak gumba s naslovom „Prikaži rezultate“. Ovdje kreiramo novi prozor koji pokreće dohvat_podataka.js te mu kao naslov šaljemo polazište i odredište. Na način win2.polaziste = polaziste dodjeljujemo mu vrijednost polazista iz trenutnog prozor u drugi prozor tj. win2, a u dohvat_podataka.js mu pristupamo na način Ti.UI.currentWindow.polaziste.

```
    var win2 = Ti.UI.createWindow({
    url:'dohvat_podataka.js',
    title:"OD: " + polaziste + "   DO: " + odrediste,
    modal:true

});

win2.polaziste = polaziste;
win2.odrediste = odrediste;
win2.prioritet = prioritet;
win2.datumNovi = datumNovi;
win2.prozor = win;
win2.open();
```

Slika 11.9. Omogučavanje pristupa pojedinih varijabli u prozoru dva

Prilikom otvaranja novog prozora kreiramo novi HTTP klijent pod nazivom xmlDohvat na način prikazan na slici 11.10.

```
var table = Ti.UI.createTableView({data:[], backgroundColor:'#000', headerTitle:"| POLAZAK | DOLAZAK

var data = [];


var xmlDohvat = Titanium.Network.createHTTPClient();
```

Slika 11.10. Kreiranje novog HTTP klijenta

Nakon toga dohvaćamo podatke preko poveznice web servisa aplikacije WebServis-HJP. Na slici 11.11. vidimo mehanizam na koji je napravljena izvlačenje podataka. U varijable se upisuju rezultati koji se izvlače preko davanja imena čvora tj. `doc.getElementByTagName („naziv_čvora“)` . Nakon toga se petljom prolazi kroz rezultate koji se potom pojedinačno izdvajaju u retke.

```
var client = Ti.Network.createHTTPClient({
     onload : function(e) {
       Ti.API.info("Received text: " + this.responseText);
       var doc = this.responseXML.documentElement;
   var element_polazak = doc.getElementsByTagName("vrijeme_polazak");
   var element_dolazak = doc.getElementsByTagName("vrijeme_dolazak");
   var element_trajanje = doc.getElementsByTagName("vrijeme_trajanje");
   var element_cijena = doc.getElementsByTagName("cijena");

   for(var i=0; i<element_polazak.length; i++){
       var podaci_polazak = element_polazak.item(i);
       var podaci_odlazak = element_dolazak.item(i);
       var podaci_trajanje = element_trajanje.item(i);
       var podaci_cijena = element_cijena.item(i);
       var row = Ti.UI.createTableViewRow({
           hasChild:true,
           height:80,
           title:
           "   " + podaci_polazak.text + " | " + podaci_odlazak.text + " | "
           + " | " + podaci_cijena.text + " KN ",
           color:'#fff'
       });
```

Slika 11.11. Izvlačenje dohvaćenih podataka

Ti podaci se tada pridružuju tablici, tablica prozoru te se prozor prikazuje. U tom prozoru tada imamo prikazane rezultate u tablici koje smo dohvatili preko web servisa. Nakon toga se može kliknuti na pojedini redak tablice koji će potom pokrenuti sljedeću JavaScript datoteku, maps.js. Dohvaća se polazište i prioritet te se šalje ponovno zahtjev u aplikaciju WebServis-HJP. Kao odgovor na ovaj zahtjev dobivamo geografsku širinu i dužinu traženog polazišta. Na temelju dobivenih podataka kreiramo novu anotaciju koju pridružujemo novoj karti. Na taj način se prikazuje polazna stanica na karti, a kod možemo vidjeti na slici 11.12.

```
    if(sirina != null){
    var annotation = Titanium.Map.createAnnotation({
    latitude:sirina.text,
    longitude:duzina.text,
    title:"Polazna stanica "+Ti.UI.currentWindow.polaziste,
    pincolor:Titanium.Map.ANNOTATION_GREEN,

    animate:true

})

var mapView = Titanium.Map.createView({
    mapType: Titanium.Map.SATELLITE_TYPE,
    region:{latitude: sirina.text, longitude:duzina.text,
        latitudeDelta:0.01, longitudeDelta:0.01},

    regionFit:true,
    userLocation:true,
    animate:true,
    annotations:[annotation]

});


Window.add(mapView);
```

Slika 11.12. Stvaranje karte gdje se prikazuje početna stanica

12. Zaključak

Izradom ovog diplomskog zadatka je u potpunosti realiziran zadani projektni zadatak koji je bio semantička integracija web usluge u mobilnom okruženju. Kao što smo već mogli kroz diplomski rad vidjeti usluga je pružanje rasporeda putovanja za nekolicinu skupina javnog prijevoza. Osim osnovne funkcionalosti aplikacija je obogaćenja pružanjem informacija o cijeni, određenim sortiranjima te prikazom lokacije s koje putovanje započinje na karti.

Rad na Android operacijskom sustavu je odabran nadasve zbog pristupa takvom mobilnom uređaju što je naposljetku rezultiralo lakšim razvojem i lakoćom testiranja. Kao dodatna pogodnost prilikom odabira operativnog sustava je da možemo sa sigurnošću reći da je Android sustav prisutan i većoj mjeri kod korisnika u Hrvatskoj nego bilo koji drugi sustav.

Nakon završetka izrade praktičnog rada smatram da bi se ovakav pristup razvoja trebao primjenjivati i u ostalim aplikacijama budući da ovakvo razdvajanje zadataka i grupiranje ih u različite aplikacije rasterećuje mobilni uređaj od procesiranja velikog broja podataka i općenito izbjegava bilo kakvo usporavanje mobilnog uređaja. Također ovakav pristup razvoja omogućava izdvajanje i zamjenu određenih funkcijskih područja, a da pritom aplikacija nema problema sa radom.

U ovom radu su se integrirale tri usluge koje su bile različite kako sintaktički tako i semantički. Bila je potrebna kombinacija upravo tih tri usluga kako bi se kreirao sustav koji može konkurirati ostalim komercijalnim mobilnim aplikacijama jer omogučuje brzu uslugu, točne informacije te dodatne pogodnosti poput pozicioniranja na karti radi lakšeg snalaženja.

Literatura

Knjige

1. Professional Android 2 Application Development, Reto Meier
2. Visualizing the Semantic Web, XML – Based Internet and Information Visualization, Vladimir Geroimenko, Chaomei Chen

Prezentacije

1. Predavanja s kolegija Napredne web tehnologije i servisi, Dr.sc. D.Kermek, Fakutet organizacije i informatike
2. Using Android NDK, Ivan Švogor, Željko Šmaguc, prezentacija sa konferencije CASE24 koja je održana 05.06.2012. godine u Zagrebu
3. Komponentni pristup razvoju Android aplikacije, Ivan Švogor, Tomislav Tušek, Zlatko Stapić, prezentacija sa konferencije CASE24 koja je održana 05.06.2012. godine u Zagrebu

Stručni članci

1. Scientific publishing on the semantic web, Tim Berners – Lee, James Hendler, 2001, dostupno na: http://www.nature.com/nature/debates/e-access/Articles/bernerslee.htm
2. Semantic Web Based Services for Intelligent Mobile Construction Collaboration, Aziz Zeeshan, Anumba Chimay, Ruikar Darshan, Carrillo Patricia, Bouchlaghem Dino, 2004, dostupno na: http://itcon.org/data/works/att/2004_26.content.00370.pdf
3. RDF Based Architecture for Semantic Integration of Heterogeneous Information Sources, Richard Vdovjak, Geert – Jan Houben, dostupno na: http://iwayan.info/Research/Interoperability/Papers_Research/SemanticMediation/wiiw01.pdf
4. Semantic Integration Through Invariants, Michael Gruninger, Joseph B. Kopena, 2005, dostupno na: https://www.aaai.org/ojs/index.php/aimagazine/article/view/1795/1693

Internet

1. What is Android, http://developer.android.com/guide/basics/what-is-android.html
2. Android SDK, http://developer.android.com/sdk/index.html
3. Xquery, : http://www.w3schools.com/xquery/default.asp
4. Xpath functions: http://www.w3schools.com/xquery/default.asp
5. Exist database: http://exist-db.org/exist/index.xml
6. NetBeans: http://netbeans.org/
7. Titanium Appcelerator: http://netbeans.org/
8. Titanium Appcelerator API docs: : http://developer.appcelerator.com/apidoc/mobile/latest
9. Html Cleaner: http://htmlcleaner.sourceforge.net/
10. Html Unit: http://htmlunit.sourceforge.net/