UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

David Sedlar

RAZVOJ ENOSTAVNE SPLETNE APLIKACIJE Z UPORABO FLEKSIBILNEGA OGRODJA NA ODPRTOKODNIH KNJIŽNICAH

DIPLOMSKO DELO

NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Borut Robič

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01689/2010
Datum: 01.09.2010

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **DAVID SEDLAR**

Naslov: **RAZVOJ ENOSTAVNE SPLETNE APLIKACIJE Z UPORABO FLEKSIBILNEGA OGRODJA NA ODPRTOKODNIH KNJIŽNICAH**

DEVELOPING A BASIC WEB APPLICATION USING A FLEXIBLE FRAMEWORK OF OPEN SOURCE LIBRARIES

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Na podlagi izkušenj in dobre prakse s področja informacijskih sistemov razvijte v programskem jeziku Java čim bolj fleksibilno ogrodje spletnih aplikacij. Ogrodje naj v največji meri uporablja sodobne odprtokodne knjižnice in skrbno dokumentira proces vključitve in združitve. Izberite primer, ki v vseh pogledih posnema zahteve realnih informacijskih sistemov, primer implementirajte z vašim ogrodjem, in opišite proces izdelave ter funkcionalnost.

Mentor:

prof. dr. Borut Robič

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani/-a **David Sedlar**,

z vpisno številko **63020140**,

sem avtor/-ica diplomskega dela z naslovom:

Razvoj enostavne spletne aplikacije z uporabo fleksibilnega ogrodja na odprtokodnih knjižnicah

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  prof. dr. Borut Robič

  in somentorstvom (naziv, ime in priimek)

  /

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne _____ Podpis avtorja/-ice: _____

Zahvala

Zahvaljujem se prof. dr. Borut Robiču za izredno uspešno in prijazno mentorstvo, za pomoč pri pisanju diplomske naloge in uporabne nasvete ter njegov trud nasploh.

Zahvaljujem se tudi moji družini, domačim in sorodnikom, prijateljem in še vsem tistim, ki so me pri pisanju diplomske naloge in pri študijski poti podpirali in me spodbujali.

Zahvalil bi se tudi podjetju Normacom Plus d.o.o., katero mi je v času študija in pisanja diplomske naloge nudilo znanje in izkušnje.

Kazalo

Seznam uporabljenih kratic in simbolov

JSF - JavaServer Faces technology (ogrodje spletih aplikacij)

UML - Unified Modeling Language (jezik za modeliranje)

TRR - Tekoči Transakcijski Račun

PDF - Portable Document Format

AJPES - Agencija Republike Slovenije za javnopravne evidence in storitve

HTML - HyperText Markup Language (skriptni jezik za oblikovanje spletnih strani)

CSS - Cascading Style Sheets (jezik za dizajn spletnih strani)

MVC - Model–View–Controller (tip arhitekture)

GUI - Graphical user interface (uporabniški vmesnik)

XHTML - eXtensible HyperText Markup Language (razširjen jezik *HTML*)

GET - način pošiljanja podatkov iz obrazca strežniku

POST - način pošiljanja podatkov iz obrazca strežniku

JSP - JavaServer Pages (predloge za oblikovanje strani v Javi)

ORM - Object-relational mapping (povezava med objekti in relacijami)

SQL - Structured Query Language (jezik za poizvedbe v podatkovnih bazah)

GNU – Operacijski sistem, ki temelji na Unix-u

XML - Extensible Markup Language (jezik za strukturiranje podatkov)

HQL - Hibernate Query Language (jezik, ki ga uporablja komponenta *Hibernate*)

SOAP - Simple Object Access Protocol (protokol spletnih storitev)

W3C - The World Wide Web Consortium (skupina, ki se ukvarja s standardi)

JSTL - JavaServer Pages Standard Tag Library (knjižnica za delo s predlogami *JSP*)

JCP - Java Community Process (skupina, ki se ukvarja s standardi)

JSR - Java Specification Requests (zahtevki za spremembe standardov)

SLF4J - Simple Logging Facade for Java (knjižnica za beleženje dogodkov)

API - Application Programming Interface (vmesnik)

PDM – Tip datoteke, ki jo uporablja *PowerDesigner*

BPMN - Business Process Modeling Notation (jezik za modeliranje procesov)

RDBMS - Relational Database Management System (sistem za upravljanje s podatkovnimi bazami)

IDE - Integrated Development Environment (razvojno okolje)

JDT - Java development Tools (vtičnik za orodje *Eclipse*)

IT – Informacijska Tehnologija

UNICODE – Eden izmed tipov črkovnih kodnih tabel

ZIP – Datotečni format s kompresijo

WAR - Web application ARchive (datotečni format podoben *ZIP*)

SSL - Secure Sockets Layer (protokol za enkripcijo)

HTTPS - Hypertext Transfer Protocol Secure (protokol *HTML* z vgrajeno enkripcijo)

WSDL - Web Services Description Language (jezik za opisovanje spletnih storitev)

URL - Uniform Resource Locator (naslov nekega vira na internetu)

JAX-WS – Ena izmed knjižnic, ki omogoča implementacijo spletnih storitev

RI - Reference Implementation (referenčna implementacija)

IP - Internet Protocol (internetni protokol)

Povzetek

Glavni namen te diplomske naloge je, da bralcu, ki že ima neke osnove programskega jezika Java ter tudi pozna nekaj osnovnih pojmov spletnih aplikacij, predstaviti eno izmed možnih alternativ izgradnje spletne aplikacije. Poudarek te naloge ni toliko v opisu samih tehnologij, ampak je bolj praktične narave, in sicer skozi implementacijo nekega testnega primera voditi bralca, da bo lahko potem tudi sam bil zmožen na podlagi spremenjenih zahtev zgraditi ustrezno rešitev. Sama vsebina temelji na dosedanjem delu v gospodarskem sektorju, kjer je avtor v roku nekaj let pridobil izkušnje na področju tako zajemanja uporabniških zahtev, kot tudi kasneje celotnega razvoja informacijskih sistemov, med njimi tudi spletnih aplikacij. Bralcu smo tako v strnjeni obliki podali del tega znanja, ga opozorili na kritične točke pri tovrstnem delu ter mu navsezadnje omogočili, da kasneje tudi izboljša ta proces na podlagi svojih izkušenj in znanj.

V uvodnem delu smo tako opisali področje diplomske naloge, problematiko ter cilje, ki izhajajo iz njih, pa tudi samo motivacijo za izbor te problematike. Dotaknili smo se tudi testnega primera, katerega zahteve smo določili tako, da čim bolj posnemajo zahteve, ki jih srečamo v realnem svetu, obenem pa so še vedno dovolj splošne, da končni sistem ne bo preobsežen in da ga bo lahko bralec tudi nadgradil.

V nadaljevanju smo podrobneje opisali tehnologije, orodja ter komponente, ki smo jih tekom razvoja sistema in pisanja diplomske naloge uporabili. Bralcu so tako podane bistvene značilnosti vseh uporabljenih sestavnih delov, kar je pomembno za razumevanje nadaljnjih poglavij.

V samem jedru pa smo podrobneje opisali, kako je sam razvoj potekal, od načrtovanja uporabniških zahtev ter podatkovnega modela, do končnega produkta ter postavitve na strežnik, na koncu pa se dotaknemo tudi pregleda implementacije celotnega sistema z razlago delovanja.

Končni rezultat je torej konkretna implementacija izbranega primera, ki je zgrajen nad fleksibilnim ogrodjem odprtokodnih knjižnic, katerega izvorna koda je v celoti na voljo bralcu in se mu priporoča, da jo za najboljše razumevanje tematike tudi sam preuči, v pomoč pa naj mu bo seveda vsebina te diplomske naloge.

Ključne besede

spletna aplikacija, orodja, komponente, ogrodje, uporabniške zahteve, znanje, izkušnje, primer implementacije

Abstract

The main purpose of this thesis is to present one of the possible alternatives to building a web application to the reader that already has some basic Java programming language experience and also knows some basic concepts of web applications. The focus of this task is not so much in the description of technologies themselves, but is more practical in nature, namely to lead the reader through the implementation of a test case scenario, to be able to then build an appropriate solution, based on the revised requirements. Content itself is based on author's previous work in the commercial sector, where the author has within a few years gained experience in the field by capturing user requirements, as well as the overall development of information systems later on, including web applications. The reader is now presented with parts of this knowledge, he is pointed to the critical points in this type of work, and he is eventually encouraged to later improve this process, based on their experience and knowledge.

In the introductory section we describe the scope of this work, issues and objectives that come as a result, as well as the motivation for the selection of these issues. We also mention the test case scenario, the requirements of which were determined so as to mimic as closely as possible the requirements that arise in the real world, while still being general enough that the final system will not be overly complex and that the reader can also upgrade it later on.

In the following chapter we describe in detail the techniques, tools and components that we have used throughout the system development and writing of this thesis. The reader is thus given the essential characteristics of all the components, which is important for understanding subsequent chapters.

At the very core, we describe in detail how the development itself took place, from planning the user requirements and data model to the final product and installation on the server, and finally we review the implementation of the entire system, with helpful commentary on how the system is to be used.

The end result is a concrete implementation of the selected test case, which was built over a flexible frame of open source libraries. The source code is fully available to the reader, and he is recommended to study it for the best understanding of the system itself, with the help of the contents of this thesis.

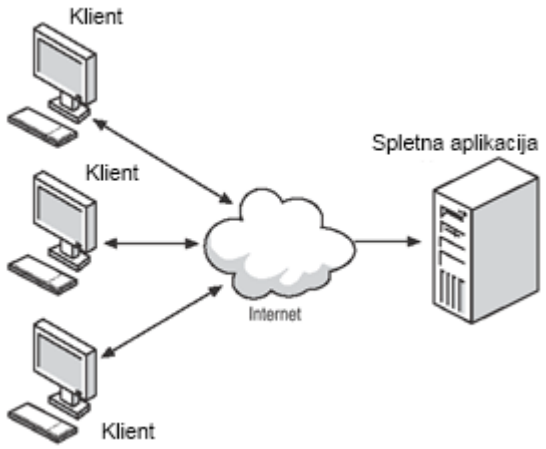
Key words

web application, tools, components, framework, user requirements, knowledge, experience, implementation example

1 Uvod

1.1 Področje

Področje, ki ga ta diplomska naloga pokriva, je precej široko, saj vsebuje velik nabor raznoraznih tehnologij, ki se uporabljajo v svetu informacijskih tehnologij. Vendar, če se osredotočimo predvsem na bistvo, obsega področje te diplomske gradnjo spletnih aplikacij ter vse povezane aktivnosti, ki spadajo zraven. Gradnja spletne aplikacije pa ni nič drugega kot gradnja novega informacijskega sistema in kot takega, se je tudi spletnih aplikacij potrebno lotiti organizirano in premišljeno.



Slika 1: Shematski prikaz spletne aplikacije

Na sliki (Slika 1) lahko vidimo nek zelo poenostavljen pogled na spletno aplikacijo, kot ga vidimo z vidika omrežja, odjemalcev in strežnikov. Navadno aplikacijo si človek po navadi predstavlja kot nek skupek datotek, lahko bi rekli neko komponento, ki se nahaja na njegovem računalniku in služi nekemu namenu. Tako aplikacijo je potrebno namestiti, zagnati, po potrebi nadgraditi in podobno. Tu pa se pojavijo prednosti spletnih aplikacij. Namreč, spletne aplikacije so po svoji naravi centralizirane, kar pomeni, da aktivnosti iz prejšnjega stavka opravimo samo enkrat in na eni lokaciji. Sprva se to mogoče ne zdi tako uporabno za domačega uporabnika, vendar se je potrebno zavedati, da uporabnost takega sistema narašča s številom uporabnikov te aplikacije in je sistem kot tak idealen za večja podjetja in veje državnih organov.

Beseda spletna v besedni zvezi spletna aplikacija pa nam da vedeti, da je aplikacija spletne narave in kot taka omogoča dostop vsem, ki so preko omrežja povezani s strežnikom, kjer se ta aplikacija nahaja. To omrežje je lahko poljubne oblike, lahko je privatno ali pa tudi javno, največkrat pa je to kar omrežje internet. Prej smo omenili strežnik, na katerem se nahaja aplikacija. Temu strežniku zaradi svoje narave pravimo aplikacijski strežnik, čeprav se v svoji grobi funkcionalnosti bistveno ne razlikuje od spletnega strežnika, vsaj za nekega laičnega uporabnika, ki na tak strežnik dostopa preko svojega internetnega brskalnika.

To je torej na kratko opisano področje, ki ga zajema ta diplomska naloga, če pa bi to področje nekako razdelili še na manjše dele oziroma korake, bi bili ti sestavljeni iz naslednjih elementov:

- Zajem zahtev
- Izbira tehnologij, orodij ter metod
- Načrtovanje in planiranje
- Izdelava ali implementacija
- Namestitev

Vsi ti elementi bodo nekako skozi tok celotne diplomske naloge predstavljeni, in sicer na takšnem nivoju podrobnosti, da se bralec ne bo počutil, da mu manjka informacij ali pa da je informacij preveč.

1.2 Cilji in problematika

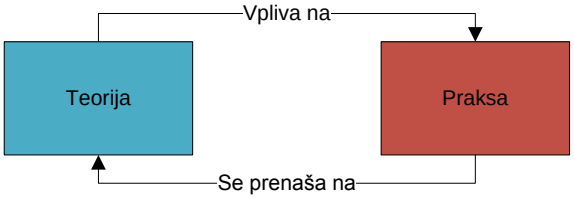
Glavni namen te diplomske naloge je, da bralcu, ki že ima neke osnove programskega jezika Java ter tudi poznavanje osnovnih pojmov spletnih aplikacij, predstaviti eno izmed možnih alternativ izgradnje spletne aplikacije. Ogrodje *Stripes* se zadnje časa uveljavlja kot resna alternativa sicer že bolj zrelim tehnologijam, ko so na primer *JSF*, ogrodje *Spring* in podobni. Poudarek naloge ni toliko v opisu samih tehnologij (čeprav zaradi narave le teh jih na kratko vseeno moramo opisati), ampak je bolj praktične narave, in sicer skozi implementacijo nekega testnega primera voditi bralca, da bo lahko potem tudi sam bil zmožen na podlagi spremenjenih zahtev zgraditi ustrezno rešitev. Testni primer bo določen tako, da bo čim bolj posnemal zahteve, ki jih srečamo v realnem svetu, obenem pa bo še vedno dovolj splošen, da ne bo preobsežen in da ga bo lahko bralec tudi nadgradil. Priložena bo tudi celotna izvorna koda, seveda ustrezno dokumentirana na mestih, kjer bi se lahko pojavile nejasnosti. Tako bo lahko bralec tudi sam preizkusil delovanje brez večjih težav, diplomska pa mu bo dala še nekaj dodatnih informacij o samem načrtovanju in izgradnji.

Sama vsebina temelji na mojem dosedanjem delu v gospodarskem sektorju, kjer sem v roku nekaj let pridobil izkušnje na področju tako zajemanja naročniških zahtev, kot tudi kasneje celotnega razvoja informacijskih sistemov, med njimi tudi spletnih aplikacij. Bralcu bi tako v strnjeni obliki podal del tega znanja, ga opozoril na kritične točke pri tovrstnem delu, razne trike, ki se jih je treba posluževati ter navsezadnje mu omogočil, da kasneje tudi izboljša ta proces na podlagi svojih izkušenj.

1.3 Motivacija

Motivacija te diplomske izhaja predvsem iz lastnih dognanj in izkušenj s podobnimi procesi dela v podjetjih in tudi nasploh. Dejstvo je, da je danes na voljo zelo širok spekter vseh različnih vrst tehnologij, kar velja tudi za področje informacijskih tehnologij. Za nekega začetnika, ki v ta svet šele vstopa, je torej kritično, da se v tej poplavi nekako usmeri, oziroma da zna izmed množice vseh tehnologij izbrati tiste, ki mu bodo omogočile svoje cilje uresničiti karseda optimalno. Tu pa se po navadi stvari že zapletejo, saj so med tehnologijami precejšnje razlike, tako v sami kvaliteti komponente, kot tudi v dokumentaciji. In predvsem kvaliteta slednje je za nekoga, ki tehnologije ne pozna, največjega pomena.

Velikokrat se soočamo s težavo, da nekatere tehnologije, ki jih želimo ali bi želeli uporabiti, niso dokumentirane ali pa so dokumentirane zelo skopo. To bodočemu razvijalcu ne pušča dosti možnosti in ga prisili v to, da se znajde sam, ali z eksperimentiranjem ali pa z iskanjem nekega primera, kjer je morebiti ta tehnologija že vgrajena. Na drugi strani pa imamo tehnologije, kjer je dokumentacije na pretek, kar je za izkušenega mojstra zelo dobro, vendar se izkaže da prevelika količina informacij oziroma prevelika krivulja potrebnega znanja (*angl. learning curve*) ne vodi nujno do večjega razumevanja tehnologije [2]. Eden od vzrokov motivacije je torej dati bralcu ravno zadostno količino informacij, da se v njih ne izgubi, hkrati pa vseeno pridobi potrebna znanja, da lahko svojo aplikacijo, grajeno na tem ali podobnem ogrodju, razvija naprej.



Slika 2: Medsebojni vpliv prakse in teorije

Drugi del motivacije izvira iz dejstva, da se dognanja v gospodarskem sektorju zaradi narave dela vse premalokrat prenesejo tudi v neko pisno oziroma drugo obliko, iz katerega bi potem črpala tudi teorija (Slika 2). Narava dela dandanes pa je pogojena predvsem s finančnimi rezultati ter roki za dokončanje, ki pa so po izkušnjah skoraj vedno preveč optimistični. To pomeni, da dejansko skoraj ne pridemo do te možnosti, da bi si po ali med samim projektom vzeli zadosten čas, da bi nekatere stvari tudi formalno zapisali, opredelili, analizirali in na koncu tudi optimizirali. Tako se moramo velikokrat zadovoljiti z rešitvami, ki so »ad hoc« narave, v upanju, da se na koncu to na kvalitetni ne po poznalo preveč, oziroma da bo rešitev še vedno ustrezala minimalnim pogojem naročnika oziroma končnega uporabnika. V tej diplomski nalogi se bo poizkusilo torej vsaj malo obiti ta začarani krog in del izkušenj iz gospodarskega sektorja prenesti tudi na akademski svet, oziroma krajše rečeno, iz prakse v teorijo.

1.4 Uporabniške zahteve aplikacije

Ta del je eden najpomembnejših členov v razvoju aplikacije, predvsem zato, ker naročnik navadno nima zelo natančno določenih potreb. Zato je izredno pomembno, da se še pred samim pričetkom razvoja s stranko pogovorimo o njenih zahtevah in jih poizkusimo karseda natančno povzeti, pri tem si seveda lahko pomagamo tudi z diagrami, primerni so na primer *UML* diagrami primerov uporabe.

Ker je poglavitni cilj te diplomske naloge spoznavanje in izgradnja fleksibilnega ogrodja spletne aplikacije, ki bo služilo tudi za nadaljnji razvoj, bomo tu skušali podati take zahteve, ki kar najbolje povzamejo realne zahteve, ki bi jih dobili od naročnika in so plod nekajletnih izkušenj na tem področju. Predvsem je pomembno, da se zahteve dotaknejo vseh možnih področij in tipov zahtev, se pravi da so naravnane v spekter širine, bolj kot same globine. To pomeni, da bo uporabniških predlog, vnosnih polj in podobnega le za vzorec in za ponazoritev funkcionalnosti, izogibali pa se bomo podvajanju, nepotrebni redundanci in nasploh vsemu, kar ne bi bistveno doprineslo k bralčevemu razumevanju tematike. Dovolj je na primer, da bralec razume povezovanje z nekim zunanjim informacijskim sistemom, namesto da se zahteva implementacija z več podobnimi sistemi, saj bi to po nepotrebnem dodalo balast naši aplikaciji.

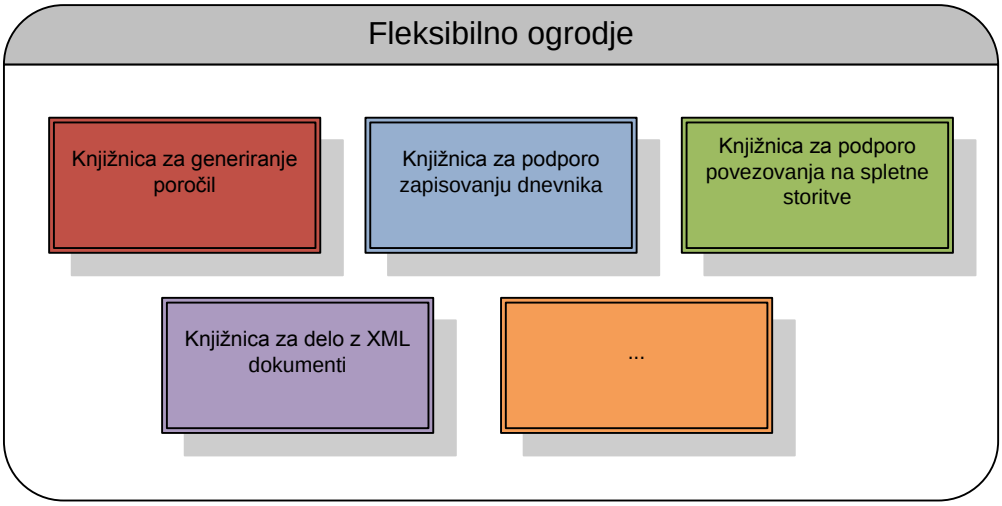
Lotimo se torej kar bistva, se pravi samih zahtev. Naš primarni cilj bo izdelava »centralnega« registra strank, oseb ter poslovnih partnerjev (v nadaljevanju subjektov), ki lahko v večjem podjetju opravlja vlogo osrednjega registra. Register mora biti na voljo vsem zaposlenim, prav tako pa mora biti omogočen avtomatiziran dostop za vso povezano programsko opremo, ki dostopa do podatkov našega registra. Bolj podrobno lahko zahteve razdelimo po naslednjih alinejah.

- Potrebno je izdelati register za vodenje podatkov o subjektih, pri čemer mora biti register organiziran centralno (brez podvajanja podatkov).
- Obstajata 2 osnovna tipa subjektov: fizične in pravne osebe.
- Poleg osnovnih podatkov o subjektu (naziv, naslov itd.) mora biti možnost dodajanja posameznih vlog subjektom: npr. pravna oseba je lahko v vlogi kupca, dobavitelja; fizična oseba je v vlogi zaposlenega, avtorja (dela po avtorski pogodbi), pogodbenega sodelavca, študenta.
- Glede na vlogo pripadajo subjektu dodatni atributi.
- Posamezni subjekt ima lahko hkrati več vlog. Omogočeno mora biti dinamično dodajanje ter odvzemanje vlog subjektom. Vsaka vloga ima začetek in konec.
- Vsak subjekt ima lahko več tekočih računov (*TRR*-jev) in kontaktnih oseb.
- Potrebno je dodati revizijsko sled nad registrom.
- Nad registrom je potrebno narediti uporabniški vmesnik za urejanje, ki omogoča vsaj naslednje operacije: vnos novih zapisov, urejanje, dodajanje ter odvzemanje vlog, iskanje.

- Prijava v uporabniški vmesnik naj temelji na uporabniškem imenu in geslu ali digitalnem potrdilu.
- Pri urejanju subjektov se doda nekaj poslovnih pravil, vsaj naslednje:
    - Vsak subjekt mora imeti vnesen naziv in naslov (obvezni podatki).
    - Ni dovoljeno vnašanje subjektov z isto davčno (potrebno je opozoriti uporabnika), s spletnim servisom na *AJPES*-u pa naj se preveri, če vnesena davčna številka obstaja.
    - Podobno je pri fizičnih osebah, kjer se izvede preverjanje na enakost (ime + priimek + naslov) – ne razlikujemo med malimi in velikimi črkami (*angl. case-insensitive*).
    - V primeru, da prihaja pri fizičnih osebah do časovnega prekrivanja vlog (oseba je hkrati delavec in študent), naj vmesnik onemogoči dodajanje takšne vloge.
- Omogoči se izvoz podatkov v obliki dokumentov *Excel, Word in PDF* (možnost izvoza po različnih kriterijih).
- Omogočiti je potrebno dostop do podatkov preko spletnih storitev (*angl. web service*).
- Programsko kodo je potrebno ustrezno dokumentirati (namestitev, konfiguracija, uporaba).

1.5 Na kratko o fleksibilnem ogrodju

Fleksibilno ogrodje kot pojem, ki se omenja v tej diplomski nalogi je pravzaprav bolj abstraktne narave. To pomeni, da ne predstavlja neke specifične, točno določene tehnologije ali komponente, temveč je v nekem smislu skupek več tehnologij. Pojem ogrodje je predvsem v svetu informacijskih tehnologij kar malo preveč uporabljen, tako da za nekega bralca izgubi pravi pomen. Predvsem se dostikrat ta pojem zameša s pojmom komponenta, kajti meja je tu precej zamegljena. Kdaj je neka komponenta dovolj kompleksna, da bi jo bilo logično imenovati ogrodje ter kdaj je ogrodje tako homogeno ali enostavno, da ga je bolje imenovati komponenta? Na takšna vprašanja je seveda več možnih odgovorov, ki so odvisni od interpretacije.



Slika 3: Poenostavljen prikaz zasnove fleksibilnega ogrodja

Mi bomo torej za namen te diplomske naloge ogrodje poimenovali kot nek heterogen sestav več komponent, ki so med seboj slabo povezane ali celo neodvisne, vseeno pa z njihovim povezovanjem pridobimo neko funkcionalno zaključeno celoto (Slika 3). Poglavitni namen ogrodja pa je, da nad njem lahko prosto gradimo in dopolnjujemo funkcionalnost. Od tod tudi izvira ime fleksibilno, saj nam ogrodje, ki je zacementirano in nam ne omogoča večjih posegov vanj, omogoča precej manj proste roke pri nadgradnji.

Cilj je torej zgraditi tako ogrodje, ki nam bo omogočalo zgornjo funkcionalnost, tako bo morebitni bralec, ki bo imel namen nad tem ogrodjem ustvariti svojo aplikacijo, imel čim manj dela s stvarmi, ki niso del same poslovne logike. Zato je potrebno za uspešno uresničitev tega cilja zgraditi ogrodje, ki bo prosto nadgradljivo in nastavljivo ter nam bo omogočalo, da bo čas od začetka projekta pa do tiste prve funkcionalnosti oziroma do prvega uporabnega obrazca, čim krajši. Za izgradnjo pa bomo uporabljali knjižnice, ki so odprtokodne (*angl. open source*), te pa so v svetu programskega jezika Java na srečo precej razširjene [7]. Več o samih knjižnicah, ki so uporabljene pa bomo napisali v naslednjem poglavju (Metode, orodja in uporabljene tehnologije).

1.6 Rezultati in rešitev problema

Rezultat te diplomske naloge je rešitev problemov in izpolnitev ciljev, ki smo si jih zadali ob začetku. To pomeni, da smo v procesu izdelave uporabili izkušnje in dobre prakse iz dela na dosedanjih informacijskih sistemih in razvili karseda fleksibilno ogrodje spletnih aplikacij v programskem jeziku Java. To ogrodje nam je služilo kot temelj za nadaljnji razvoj. Pri tem smo v največji meri uporabljali sodobne odprtokodne knjižnice in sam proces skrbno dokumentirali, tako v sami izvorni kodi, kot tudi v diplomski nalogi. Obrazložene so bile tiste temeljne podrobnosti, ki jih bralec potrebuje za dosego istega cilja, se pravi kako točno te komponente vključiti v sistem ter jih združiti in uporabljati kot celoto.

Za dosego teh ciljev pa smo si izbrali tudi primer, ki v vseh aspektih posnema zahteve realnih informacijskih sistemov. To je pomemben del te diplomske naloge, saj preko primera bralec dosti lažje vidi in dojema stvari, ki bi se mogoče sicer zdele preveč abstraktne, nejasne in težko predstavljive. Vzeli smo torej nek realen primer, navedli zanj vse pomembnejše uporabniške zahteve, ki naj bi jih tak sistem imel, potem pa smo primer implementirali z uporabo tega ogrodja, podobno kot v prejšnjem odstavku pa tudi tukaj poskrbeli za opis procesov same izdelave oziroma načrtovanja in pa tudi opis funkcionalnosti in hkrati tudi navodila za uporabo.

1.7 Avtorske pravice in obseg diplomske naloge

V tem poglavji bi na kratko samo namenili par stavkov avtorskim pravicam, ki se tičejo izvorne kode našega informacijskega sistema. Vse avtorske pravice knjižnic, ki smo jih pri gradnji uporabili, so last njihovih avtorjev. Oblika strani, se pravi dizajn in koda v obliki *HTML, CSS* in nekateri deli *Javascript* programske kode pa so last podjetja *Normacom Plus d.o.o*. To je predvsem posledica tega, da je osredotočenost te diplomske naloge na samo funkcionalnost in ozadju sistema, pri čemer bi nam izdelava nove oblikovne sheme vzela nekaj dodatnega časa, medtem ko bralec od tega nebi imel posebne koristi. Sama izvorna koda pa je delo avtorja te diplomske naloge, razen manjših izjem, kjer je avtor oziroma so-avtor tudi ustrezno zapisan v glavi same izvorne datoteke. Tu gre predvsem za nekatere predloge, ki so bile knjižnicam že priložene in jih je bilo potrebno za delovanje sistema prilagoditi.

2 Metode, orodja in uporabljene tehnologije

V tem poglavju bi na kratko predstavili metode, orodja in uporabljene tehnologije oziroma komponente. Poglavje ni namenjeno podrobni predstavitvi vseh tehničnih karakteristik in zmogljivosti prej omenjenih stvari, ampak služi uporabniku, ki s tehnologijami še ni seznanjen, da jih na nekem grobem nivoju spozna, vidi zakaj se stvar uporablja in kakšno funkcionalnost ponuja.

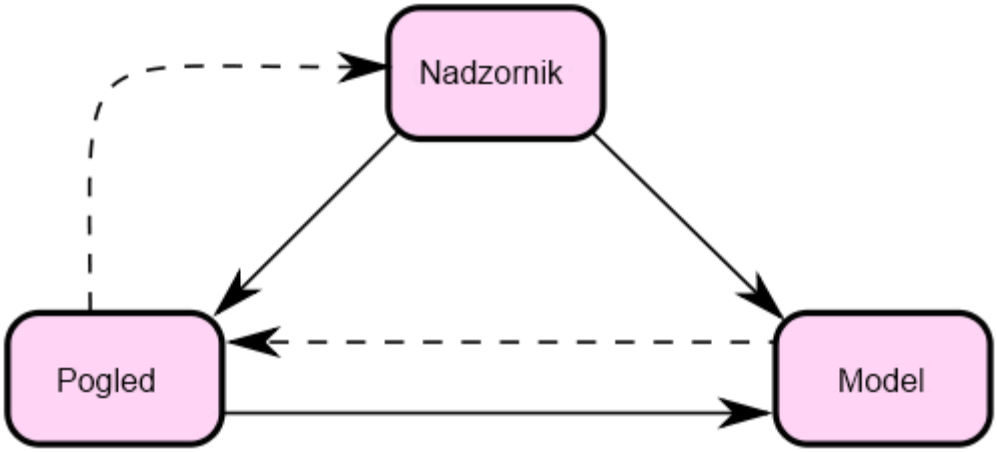
2.1 MVC arhitektura

Model-View-Controller (MVC) je programska arhitektura [9], ki se trenutno šteje za arhitekturni vzorec in se uporablja pri razvoju programske opreme. Imamo torej tri pojme, ki sodelujejo skupaj, to so model (*angl. model*), pogled (*angl. view*) ter pa nadzornik (*angl. controller*). Ta vzorec izolira domensko logiko (aplikacijska logika za uporabnike oziroma poslovna logika) od uporabniškega vmesnika (vhodni podatki in predstavitev), kar omogoča neodvisen razvoj, preizkušanje in vzdrževanje vsakega dela posebej (ločevanje problema in skrbi na več delov).

Model upravlja vedenje aplikacije in pa upravlja s podatki, odgovarja na zahteve za informacije o stanju, in se odziva na navodila za spremembo stanja (ponavadi od nadzornika). V sistemu, ki ga vodijo dogodki (*angl. event-driven systems*), model obvesti opazovalce (običajno pogled) kadar se informacije spremenijo, tako da se lahko ustrezno odzovejo (Slika 4).

Pogled spremeni model v obliko, primerno za interakcijo, običajno je to element uporabniškega vmesnika. Lahko obstaja več pogledov za enoten model, seveda za različne namene.

Nadzornik prejme vhod in sproži odziv, tako da kliče objekte na modelu. Nadzornik sprejme vnos ali podatke od uporabnika in naroči modelu in pogledu, da izvedeta ustrezne ukrepe ali operacije, ki temeljijo na tem vhodu. Aplikacija, ki uporablja arhitekturo *MVC* je pravzaprav lahko zbirka več kombinacij modelov, pogledov ter nadzornikov, vsaka kombinacija pa je odgovorna za različne elemente uporabniškega vmesnika. Na primer pri komponenti sistema *Swing GUI* (ki je del Java ogrodja), skoraj vsak vmesnik predstavlja posamezno kombinacijo *MVC* arhitekture, oziroma deluje kot nek svoj sistem v malem.



Slika 4: Slikovni prikaz arhitekture MVC

MVC arhitekturo je pogosto videti v spletnih aplikacijah, kjer je pogled oblike *HTML* ali *XHTML*, ki ga generira aplikacija. Nadzornik prejme *GET* ali *POST* vhod [6] in se odloči, kaj storiti z njim, navadno pa ga izroči modelu, ki vsebuje poslovna pravila in ve, kako poteka izvajanje posebnih nalog, kot je na primer obdelava novega naročniškega razmerja. Ta model pa nato poda odgovornost naprej komponentam, ki generirajo *HTML* oziroma *XHTML*, to pa so na primer sistemi za upravljanje s predlogami (angl. *template engine*), sistemi za upravljanje z *XML* podatki, itd.

Sam model še ni baza podatkov, model v *MVC* arhitekturi so dejansko kombinacija podatkov samih in pa tudi poslovne logike, ki je potrebna, da manipuliramo s podatki v aplikaciji. Številne aplikacije uporabljajo obstojne shranjevalne mehanizme, kot je na primer podatkovna baza za shranjevanje podatkov. *MVC* posebej ne definira sloja za dostop do podatkov, saj se razume, da leži ta sloj pod modelom oziroma ga zaobjema. Modeli niso objekti za dostop do podatkov, vendar v zelo preprostih aplikacijah, ki imajo malo poslovne logike, pravzaprav ni prave razlike med njimi. Na primer *Active Record* je že uveljavljen načrtovalni vzorec, ki združuje področje poslovne logike in kode za dostop do podatkov, je pravzaprav model, ki shranjuje svoje stanje tudi dolgoročno.

Java platforma definira *MVC* nekako takole [12]:

- Model je zbirka razredov Java, ki skupaj predstavljajo programsko aplikacijo, namenjeno za skladiščenje, in po možnosti tudi ločevanje podatkov. Lahko pa je tudi en sam razred, ki lahko komunicira z vsemi uporabniškimi vmesniki (na primer konzola, grafični uporabniški vmesnik, ali spletna aplikacija).
- Pogled predstavlja *JSP* (*Java Server Page*) stran, pri čemer se podatki prenašajo na stran v obliki *HttpServletRequest* ali *HttpSession*.
- Nadzornik komunicira s prednjim delom (*angl. front end*) modela in naloži *HttpServletRequest* ali *HttpSession* z ustreznimi podatki, nato pa posreduje *HttpServletRequest* in *HttpServletResponse* na *JSP* stran z uporabo *RequestDispatcher*.

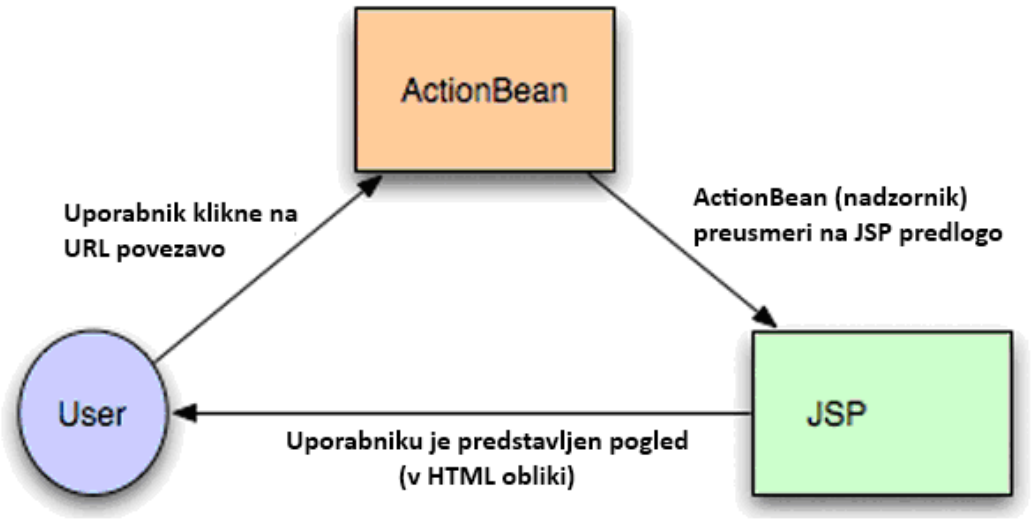
Servlet je razred Java, ki komunicira in sodeluje z modelom, ni pa mu treba ustvarjati *HTML* ali *XHTML* izhodnih podatkov. *JSP* stranem ni potrebno komunicirati z modelom, ker jim že *Servlet* daje informacije in se lahko osredotočijo na ustvarjanje izhodnih podatkov.

2.2 Opis tehnologij oziroma komponent

V tem podpoglavju bomo na kratko predstavili glavne komponente, ki bodo sestavljale naše ogrodje. V prejšnjem podpoglavju je že bila predstavljena *MVC* arhitektura, ki nam predstavlja nek temelj oziroma smer, v kateri bomo gradili naš sistem. Ne bodo pa zaradi omejenega obsega predstavljene popolnoma vse uporabljene knjižnice, predvsem ne tiste, ki so potrebne samo za delovanje drugih knjižnic.

2.2.1 Komponenta za realizacijo arhitekture MVC Stripes

Stripes je odprtokodno ogrodje spletnih aplikacij, ki temelji na arhitekturi *MVC*. Njegov namen je biti ogrodje, ki ni tako masivno kot ostale alternative (na primer ogrodja *Struts, Spring*) To doseže z uporabo Java tehnologij, kot so zaznamki (*angl. annotations*) in generiki (*angl. generics*), ki so bili uvedeni v Java verziji 1.5, s tem pa posredno gradi filozofijo uporabnosti nad konfiguracijo. Tako poudarja idejo, da lahko iz preprostih konvencij, uporabljenih v ogrodju kot celoti, zmanjšamo čas in stroške v režiji, ki bi jih sicer imeli s konfiguracijo. V praksi to pomeni, da *Stripes* aplikacije skorajda ne potrebujejo nastavitvenih datotek, kar zmanjšuje čas razvoja in vzdrževalnih del. Na spodnji sliki (Slika 5) lahko vidimo poenostavljen pogled na delovanje tega ogrodja iz pogleda uporabnika (*angl. user*).



Slika 5: Slikovni prikaz delovanja ogrodja Stripes

Cilji *Stripes* ogrodja:

- Narediti razvoj spletnih aplikacij v Javi enostaven
- Zagotoviti enostavno, vendar zmogljivo rešitev za nekatere skupne probleme
- Zmanjšati začetni čas uporabe (*angl. learning curve*) za nove razvijalce na manj kot 30 minut
- Poenostaviti proces za razširjanje samega *Stripes* ogrodja, ne da bi bilo potrebno vsako majhno stvar konfigurirati

Ključne lastnosti:

- Skoraj nič zunanje konfiguracije na eno stran oziroma na en nadzorni razred (*ActionBean* kontrolni razredi so samodejno zaznani, nastavljeni pa z uporabo zaznamkov v programski kodi)
- Zmogljivo ogrodje za povezovanje, ki gradi zapletene strukture spletnih objektov iz parametrov uporabniških spletnih zahtev (*angl. user web request*)
- Enostaven sistem za uporabo validacije (vključno z lokalizacijo) in pretvorbe podatkovnih tipov
- Sistem za lokalizacijo, ki deluje tudi, ko so uporabljene direktne povezave na JSP strani (za razliko od tistih, ki gredo preko nazornega razreda)
- Sposobnost ponovne uporabe nadzornih razredov *ActionBeans* kot pomočnikov pri izgradnji pogledov
- Zelo enostavna uporaba indeksiranih atributov v objektih (kot so seznami, tabele, itd.)
- Vgrajena podpora za več dogodkov na enem samem obrazcu
- Transparentna uporaba zmogljivosti za prenos datotek k strežniku (*angl. file upload*)
- Podpora za inkrementalno obliko razvoja (npr. najprej lahko zgradimo in preizkušamo svoje *JSP* strani, preden sploh začnemo razmišljati o nadzornem razredu *ActionBean*, ki bo s to stranjo upravljal)
- Veliko vgrajene prožnosti, ki se nam jo sicer ni potrebno zavedati, a je tam ko jo potrebujemo

Stripes poskuša zagotoviti podobno izkušnjo, kot jo ima na primer lastnik Apple strojne opreme ali pa lastnik luksuznega nemškega avtomobile, seveda bistveno ceneje. Stvari preprosto delujejo tako kot morajo in včasih je uporabnik presenečen ko ugotovi, da ima stvar še kakšno dodatno funkcionalnost, za katero si ne bi mislili da bi jo tak sistem imel. V tej diplomski nalogi bomo uporabljali verzijo 1.5.3 te komponente, je pa v času pisanja na voljo že novejša verzija.

2.2.2 Komponenta za upravljanje s podatkovno bazo Hibernate

Hibernate je *ORM* (*angl. object-relational mapping*) knjižnica za jezik Java, ki ponuja ogrodje za povezovanje objektno usmerjenega domenskega modela na tradicionalno relacijsko bazo podatkov (Slika 6). *Hibernate* rešuje težave neusklajenosti med objekti in relacijami, to pa naredi z zamenjavo neposrednega povezovanja na bazo podatkov z dostopi preko funkcij na visoki ravni abstrakcije, oziroma z dostopi preko funkcij, ki upravljajo z objekti. *Hibernate* je brezplačna programska oprema, ki se razširja pod licenco *GNU Lesser General Public License*.

Primarna funkcionalnost *Hibernate* je preslikava iz razredov Java do relacijskih tabel zbirke podatkov (in iz podatkovnih tipov Java v podatkovne tipe *SQL*). *Hibernate* ponuja tudi funkcionalnost za izvajanje poizvedb in iskanja nad podatki. *Hibernate* generira *SQL* klice in poizvedbe in lajša razvijalčevo delo, ki bi sicer obsegalo ročno obdelavo rezultatov podatkovne baze in pretvorbo v objekte. Tako postane aplikacija prenosna na vse podprte *SQL* podatkovne baze, medtem pa ta prenosljivost nima velikega vpliva na zmogljivost, kar je pri delu s podatki zelo pomembno.

Slika 6: Predstavitev strukture komponente Hibernate

Povezovanje Java razredov s tabelami zbirke podatkov je mogoče z nastavitveno datoteko *XML* ali z uporabo Java zaznamkov (*angl. annotations*). Če uporabljamo datoteko *XML*, lahko *Hibernate* ustvari ogrodje izvorne kode za vztrajnostne (oziroma obstojne) razrede, to so razredi, katerih vrednost se usklajujejo s podatki v podatkovni bazi. To ni potrebno, kadar

se uporabljajo zaznamki. *Hibernate* lahko uporablja datoteko XML ali zaznamke, da ohranja shemo podatkovnega modela, če tako določimo v nastavitvah. Tako nam tudi morebitne nenačrtovane spremembe sheme ne povzročajo težav. Prav tako je na voljo funkcionalnost, ki poskrbi za relacije ena proti mnogo in mnogo proti mnogo med razredi. Poleg upravljanja z asociacijami med objekti, lahko *Hibernate* upravlja tudi z refleksivnimi asociacijami, če ima objekt relacijo tipa ena proti mnogo sam s seboj. *Hibernate* podpira povezovanje tudi s podatkovnimi tipi, ki jih določimo po meri.

Povezane objekte lahko nastavimo tako, da imajo operacije kaskadno obliko, se pravi da se prenašajo z enega objekta na drug. Na primer, imamo očetovski objekt *Album*, ki je nastavljen tako, da prenašajo svojo funkcijo za shranjevanje in brisanje na svojega sina *Skladba*. To lahko zmanjša čas razvoja in zagotovi referenčno integriteto. S funkcionalnostjo umazanega preverjanja (*angl. dirty checking*) se izognemo nepotrebnemu pisanju v bazo, tako da se opravljajo *SQL* posodobitve le na spremenjenih atributih vztrajnostnih objektov. *Hibernate* implementira po *SQL-u* povzet jezik imenovan *Hibernate Query Language (HQL)*, ki omogoča pisanje *SQL-u* podobnih poizvedb, ki se izvedejo na *Hibernate* vztrajnostih objektih. *Criteria Queries* poizvedbe so implementirane kot objektno usmerjena alternativa zgoraj omenjenemu *HQL*, ponujajo pa skoraj isto funkcionalnost. *Hibernate* se lahko uporablja tako v samostojnih aplikacijah Java ali pa v Java EE aplikacijah z uporabo *Servlet* komponent ali *EJB* tehnologije. Lahko pa je tudi vključen kot funkcionalna baza v drugih programskih jezikih. Na primer, *Adobe* je integriral *Hibernate* v različico 9 programa *ColdFusion* (ki teče na Java aplikacijskem strežniku), k temu pa je dodal svojo plast abstrakcije z novo funkcionalnostjo.

V sami diplomski nalogo se uporablja verzija 3.5 te komponente, na internetu pa je na voljo tudi zelo obsežna dokumentacija in primeri.

2.2.3 Komponenta za povezovanje na spletne storitve Axis

Apache Axis je odprtokodna, na *XML* tehnologiji temelječa komponenta za spletne storitve. Sestavljena je iz Java in C++ implementacije strežnika *SOAP*, in različnih pripomočkov ter API vmesnika za ustvarjanje in nameščanje spletnih aplikacij, ki uporabljajo spletne storitve. Z uporabo *Apache Axis* lahko razvijalci ustvarijo interoperabilne ter porazdeljene računalniške aplikacije. *Axis* je razvit pod okriljem *Apache Software Foundation*.

SOAP je na *XML* osnovi temelječ komunikacijski protokol ter format kodiranja za komunikacijo med več aplikacijami. Prvotno so si ga zamislili podjetji *Microsoft* in *Userland software* in se je razvil skozi več generacij. Trenutna specifikacija je različica *SOAP* 1.2, čeprav je različica 1.1 bolj razširjena. *W3C* delovna skupina za *XML* protokol (*angl. XML Protocol working group*) je zadolžena tudi za specifikacijo tudi *SOAP* protokola. *SOAP* je splošno gledano temelj za novo generacijo računalniških aplikacij, ki so neodvisne od jezika ter platforme in so porazdeljene, imenujemo pa jih spletne storitve.

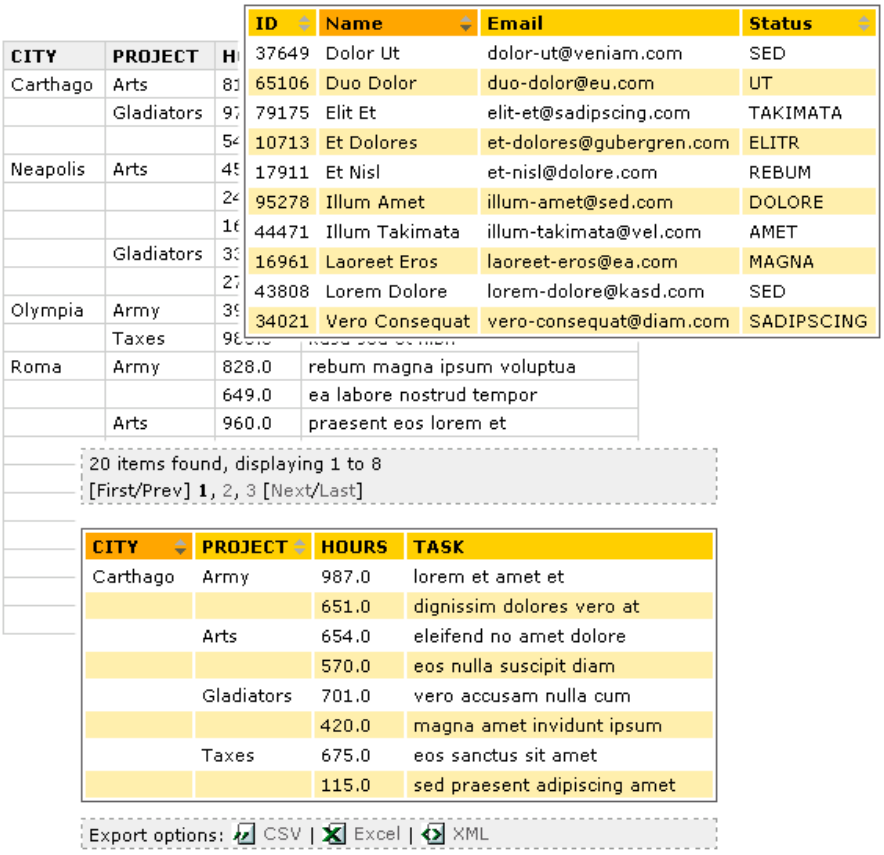
Axis je v bistvu *SOAP* okvir (*angl. SAOP engine*) oziroma ogrodje za izgradnjo *SOAP* procesorjev, kot so odjemalci, strežniki, prehodi (*angl. gateways*), itd. V tej diplomski nalogi bomo uporabljali knjižnico *Axis* verzije 1.4.

Ampak *Axis* ni le *SOAP* okvir, temveč vključuje tudi:

- preprost samostojni strežnik
- strežnik, ki se vklopi v nek drugi obstoječ aplikacijski strežnik, na primer *Apache Tomcat*
- široka podpora za *Web Service Description Language (WSDL)*
- orodje, ki ustvarja Java razrede iz *WSDL*
- nekaj vzorčnih programov
- orodje za spremljanje *TCP/IP* paketov.

2.2.4 Komponenta za prikazovanje tabelaričnih podatkov na strani Displaytag

Displaytag knjižnica je odprtokodni paket različnih oznak (*angl. tags*), ki zagotavljajo visoko raven vzorcev spletne predstavitve, ki deluje v modelu *MVC* arhitekture. Knjižnica vsebuje znatno količino funkcionalnosti, medtem ko je še vedno enostavna za uporabo. Pravzaprav je glavni namen *Displaytag* knjižnice izpis oziroma prikaz tabel. Knjižnici damo na voljo seznam objektov, knjižnica pa bo poskrbela za ravnanje s prikazom stolpcev, sortiranje, razdeljevanje na strani, obrezovanje, združevanje, izvoz podatkov, pametno povezovanje in dekoracijo tabele v prilagodljivem slogu *XHTML*. Tabele z vzorca slike spodaj (Slika 7) so bile ustvarjene s pomočjo te knjižnice.



Slika 7: Prikaz funkcionalnosti knjižnice Displaytag

Najenostavnejši način uporabe *Displaytag* je tako, da ga usmerimo na polje Java tipa *List*. *Displaytag* bo izvedel iteracijo preko seznama in prikazal po en stolpec za posamezne atribute, ki jih vsebuje objekt. Navadno bi tak način uporabili samo v času razvoja zaradi lažjega iskanja napak ali podobnih diagnostičnih zadev. Za proizvodnjo je dobro vedno določiti točno kateri stolpci se bodo prikazovali za posamezne atribute, ki jih vsebuje objekt.

Displaytag pa podpira več izvorov podatkov kot samo *List*, se pravi:

- seznam tipa *Collection*
- seznam tipa *Enumeration*
- seznam tipa *Map* (vrednosti so prikazane v vrstici)
- seznam tipa *Dictionary* (vrednosti so prikazane v vrstici)
- navado polje (*angl. array*)
- seznam tipa *Iterator*
- katerikoli objekt, ki implementira metodo *iterator()*
- ostali objekti (oziroma njihovi atributi) pa so prikazani v eni vrstici

V diplomski nalogi se bo uporabljala knjižnica verzije 1.2, ki je tudi zadnja različica, ki je na voljo.

2.2.5 Knjižnica za dodatno funkcionalnost JSP predlog JSTL

JavaServer Pages Standard Tag Library (JSTL) knjižnica je sestavni del Java EE spletne platforme za razvoj. Komponenta razširja specifikacijo *JSP* strani z dodatnimi knjižnicami *JSP* oznak za pogosta opravila, kot so obdelave podatkov *XML*, pogojno izvajanje stavkov, zanke in internacionalizacija. *JSTL* je bil razvit pod okriljem *Java Community Process (JCP)* pod imenom *Java Specification Request (JSR) 52*. Maja 2006 pa je bila izdana verzija *JSTL 1.2*.

JSTL zagotavlja učinkovit način, da vgradimo logiko na stran *JSP*, brez uporabe vgrajene Java kode (*angl. embedded Java code*) neposredno. Uporaba standardiziranih oznak namesto bolj zapletenih vstopov in izstopov v vgrajeno Java kodo vodi do lažjega vzdrževanja kode in pri razvoju omogoča ločitev skrbi med aplikacijsko oziroma poslovno logiko ter uporabniškim vmesnikom.

Prednost, kot je omenjeno že zgoraj, je torej ta, da lahko celotno *JSP* predlogo oziroma stran pišemo v istem stilu, se pravi *HTML* oziroma *XHTML*, tako se poslovna logika nekako zlije skupaj z logiko za prikaz. Tak stil razvoja se je izkazal za hitrejšega in bolj preglednega. Ena od morebitnih slabosti uporabe te knjižnice pa je ta, da nam pri razvoju z uporabo vgrajene Java kode morebitne napake lahko zazna oziroma odpravi že razvojno orodje samo (oziroma prevajalnik), medtem ko to ponavadi ni možno z uporabo *JSTL* (se navadno prevede šele ob dostopu na stran). V diplomski nalogi se bo uporabljala verzija 1.1.2 te knjižnice.

2.2.6 Knjižnica za beleženje dogodkov log4j

Apache log4j je na Javi temelječa knjižnica za zapisovanje oziroma beleženje dogodkov (*angl. event logging*). Prvotno jo je napisal Ceki Gülcü in je zdaj projekt pod okriljem *Apache Software Foundation*. *Apache log4j* je ena izmed mnogih podobnih knjižnic, ki temeljijo na Java tehnologiji in beležijo dogodke. Gülcü, avtor prvotne verzije, je od takrat začel projekta *SLF4J* in *Logback*, z namenom ponuditi naslednika *log4j*, vseeno pa *log4j* še vedno ostaja na prvem mestu po popularnosti.

V spodnji tabeli so opredeljene ravni sporočil, ki jih generira log4j, v padajočem vrstnem redu strogosti. Na levi strani so navedene označbe ravni v log4j, na desni ob oznaki pa je kratek opis vsake ravni.

- *FATAL* - Hude napake, ki povzročajo prezgodnje odpovedi. Pričakuje se, da bodo takoj vidne na statusni konzoli.
- *ERROR* - Druge napake v izvajanju (*angl. runtime errors*) ali nepričakovane razmere. Pričakuje se, da bodo takoj vidne na statusni konzoli.
- *WARN* - Uporaba odsvetovanih *API-jev*, slaba uporaba *API-jev*, »skoraj« napake tipa ERROR, drugi primeri v izvajanju, ki so neželeni ali nepričakovani, vendar ne nujno »narobe«. Pričakuje se, da bodo takoj vidne na statusni konzoli.

- *INFO* – Zanimivi dogodki v izvajanju (na primer pri zagonu ali ustavitvi). Pričakuje se, da bodo takoj vidne na statusni konzoli, tako da moramo z njihovo uporabo biti konservativni.
- *DEBUG* - Podrobne informacije o pretoku skozi sistem. Pričakuje se, da se bodo ti dogodki zapisali samo v datoteko.
- *TRACE* - Podrobnejše informacije, najnižji nivo in tudi največja količina informacij. Pričakuje se, da se bodo ti dogodki zapisali samo v datoteko. Dodano šele v različici 1.2.12.

Obstajata dva načina za konfiguracijo *log4j*. Eden je z navadno tekstovno datoteko, drugi pa z datoteko oblike *XML*. V obeh primerih lahko določimo 3 glavne sestavine: *Loggers*, *Appenders* in *Layouts*. Konfiguriranje preko datoteke (za razliko od konfiguracijo v sami kodi) nam daje prednost pri vklopu ali izklopu beleženja, kar lahko storimo brez spreminjanja aplikacije same. Aplikaciji se lahko dovoli, da teče z izklopljenim beleženjem, dokler ne pride do problema, nato pa lahko ponovno vklopimo beleženje in to le s spremembo nastavitvene datoteke.

Loggers so logična imena datotek za beleženje. To so imena, ki so znana aplikaciji Java, ki jo razvijamo. Vsak *Logger* je neodvisno nastavljiv glede trenutne ravni beleženja (*FATAL*, *ERROR*, itd). V zgodnjih različicah *log4j* sta bila *Logger* in *raven* imenovana *kategorija* in *prednost*, zdaj pa so njuno poimenovanje spremenili. Dejanski rezultat beleženja se opravi oziroma zapiše z *Appenders*. Obstajajo številni tipi *Appender*, vsak s svojo funkcijo in opisnimi imeni, kot so *FileAppender*, *ConsoleAppender*, *SocketAppender*, itd. Več tipov *Appender* je lahko pritrjenih na katerikoli *Logger*, tako da je možno beleženje iste informacije na več izhodov, na primer v datoteko na lokalni ravni in vtičnico poslušalca na drugem računalniku. *Appenders* uporabljajo *Layouts* za formatiranje zapisov beleženja. Zelo priljubljen način za formatiranje enostavnih datotek za beleženje je *PatternLayout*, ki uporablja vzorčni tekst (*angl. pattern string*), podobno kot *C* funkcija *printf*. Obstajata tudi *HTMLLayout* in *XMLLayout* formata za uporabo, kadar je *HTML* ali *XML* format bolj primeren za uporabo.

V tej diplomski nalogi se bo uporabljala verzija 1.2.15 te komponente.

2.2.7 Komponenta za delo z XML datotekami XMLBeans

XMLBeans je orodje, ki omogoča dostop do vse moči, ki jo ponuja *XML*, na Javi prijazen način. Ideja je, da izkoristimo bogastvo in lastnosti *XML* in *XML* sheme (*angl. XML schema*) in te značilnosti povežemo karseda naravno v enakovredne elemente in objekte v jeziku Java. *XMLBeans* uporablja *XML* sheme za generiranje Java vmesnikov in razredov, ki se lahko nato uporabijo za dostop in spreminjanje *XML* podatkov. Uporaba *XMLBeans* je podobna kot uporaba kateregakoli drugega Java vmesnika ali razreda, se pravi z metodami, kot so na primer *getAttribute* ali *setAttribute*, prav tako kot pri delu z Javo. Medtem ko je glavni namen

uporabe *XMLBeans* dostop do podatkov *XML* z razredi Java, obstajajo tudi *API* vmesniki, ki omogočajo dostop do celotnega *XML Infoset* (*XMLBeans* ohranja celotni *XML Infoset*), kot tudi refleksijo (iskanje preko imena) v samo *XML* shemo, in sicer skozi XML Schema Object model [13].

Značilnosti *XMLBeans* so široka podpora *XML* shemam ter tudi široka podpira za *XML Infoset*. *XMLBeans* v celoti podpira *XML* sheme in ustrezni Java razredi zagotavljajo konstrukte za vso večjo funkcionalnost *XML* shem. To je zelo pomembno, saj pogosto nimamo nobenega nadzora nad značilnostmi *XML* sheme, ki pa jih potrebujemo za delo s podatki v Javi. Prav tako lahko aplikacije, ki so usmerjene v *XML* sheme, v celoti izkoristijo moč sheme *XML* in se ne omejujejo na podmnožico funkcionalnosti. Ko odvijemo (*angl. unmarshalling*) *XML* instanco je ohranjen polni *XML Infoset* in je na razpolago razvijalcu. To je pomembno zato, ker je tista podmnožica *XML* težko zastopana v Javi. Na primer, v neki aplikaciji bomo potrebovali vrstni red elementov ali komentarjev, kar brez poznavanje izvorne strukture podatkov ne bi bilo mogoče.

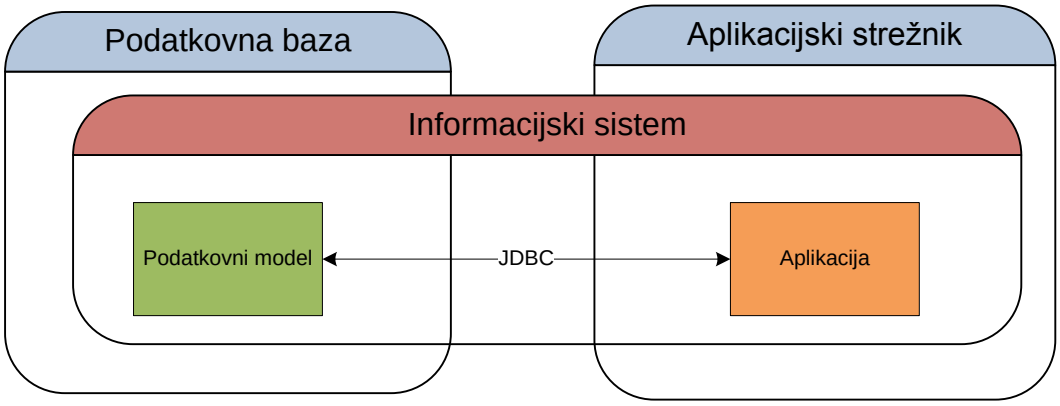
Eden izmed glavnih ciljev *XMLBeans* je bil tudi, da se uporablja v vseh ne-tekočih (*angl. non-streaming*) situacijah programiranja *XML* (se pravi da so podatki na voljo v pomnilniku). Razvijalcu naj bi bilo omogočeno, da iz svoje *XML* sheme lahko generira ustrezen nabor razredov Java, ob tem pa naj bi vedel:

- da lahko uporabi *XMLBeans* za vse sheme s katerimi se srečuje
- da je mogoč dostop do *XML* podatkov na katerikoli ravni brez pomoči drugih orodij

Za primer naše diplomske naloge se bo uporabljala verzija 2.5.0 te knjižnice.

2.3 Podatkovna baza in aplikacijski strežnik

V tem podpoglavju bi se na kratko dotaknili dveh stvari, in sicer podatkovne baze ter aplikacijskega strežnika. Iz čisto teoretičnega vidika bi lahko rekli, da ta dva pojma dejansko ne spadata v spekter našega informacijskega sistema, temveč bolj v zunanje dejavnike. Vendar pa pogosto odločitev, katero podatkovno bazo bomo izbrali ali pa na katerem aplikacijskem strežniku bo naš informacijski sitem tekel, vseeno vpliva na naš način razvoja.



Slika 8: Prikaz razmerja med IS, podatkovno bazo ter informacijskim sistemom

Iz zgornje slike (Slika 8) je videti, da naš informacijski sistem obstaja kot samostojna celota, neodvisna od platform, na kateri se poganja. Vendar je stvar navadno malo bolj kompliciranа, veliko informacijskih sistemov v Javi je togih in omejenih na določene proizvajalce, kar pomeni višje stroške vzdrževanja, posebej če se kdaj odločimo katero od platform zamenjati.

Glede podatkovne baze smo v diplomski nalogi sprejeli odločitev uporabiti eno najbolj razširjenih podatkovnih baz na svetu, to je *Oracle Database*. Pravzaprav bi za tako majhno aplikacijo brez težav lahko izbrali tudi katerokoli od bolj enostavnih ali cenejših opcij. *Oracle Database* podatkovna baza je za komercialno uporabo namreč plačljiva in stroški niso majhni, dolgoročno se nekaterim podjetjem to vseeno splača, saj se zanašajo na to, da bodo ob težavah deležni tudi hitre strokovne pomoči. V tej diplomski nalogi se uporablja zadnja verzija, to je 11g .

Kar pa se tiče aplikacijskega strežnika, imamo tu prav tako na izbiro nekaj dobrih alternativ. Prav tako kot ima *Oracle* nek prevladujoč položaj v svetu podatkovnih baz, ima velik delež trga tudi njihov aplikacijski strežnik *Oracle Application Server*. Vendar zaradi enostavnosti in zaradi zelo dobre integracije z razvojnimi orodji smo izbrali *Apache Tomcat*. Ta aplikacijski strežnik je za razvoj zelo primeren zaradi majhne velikosti, fleksibilnosti in podpore najnovejšim standardom, kot so najnovejša verzija Jave, najnovejši *Servlet API* vmesnik ter *JSP* specifikacija. Mi bomo za potrebe razvoja uporabljali verzijo 6.0.26.

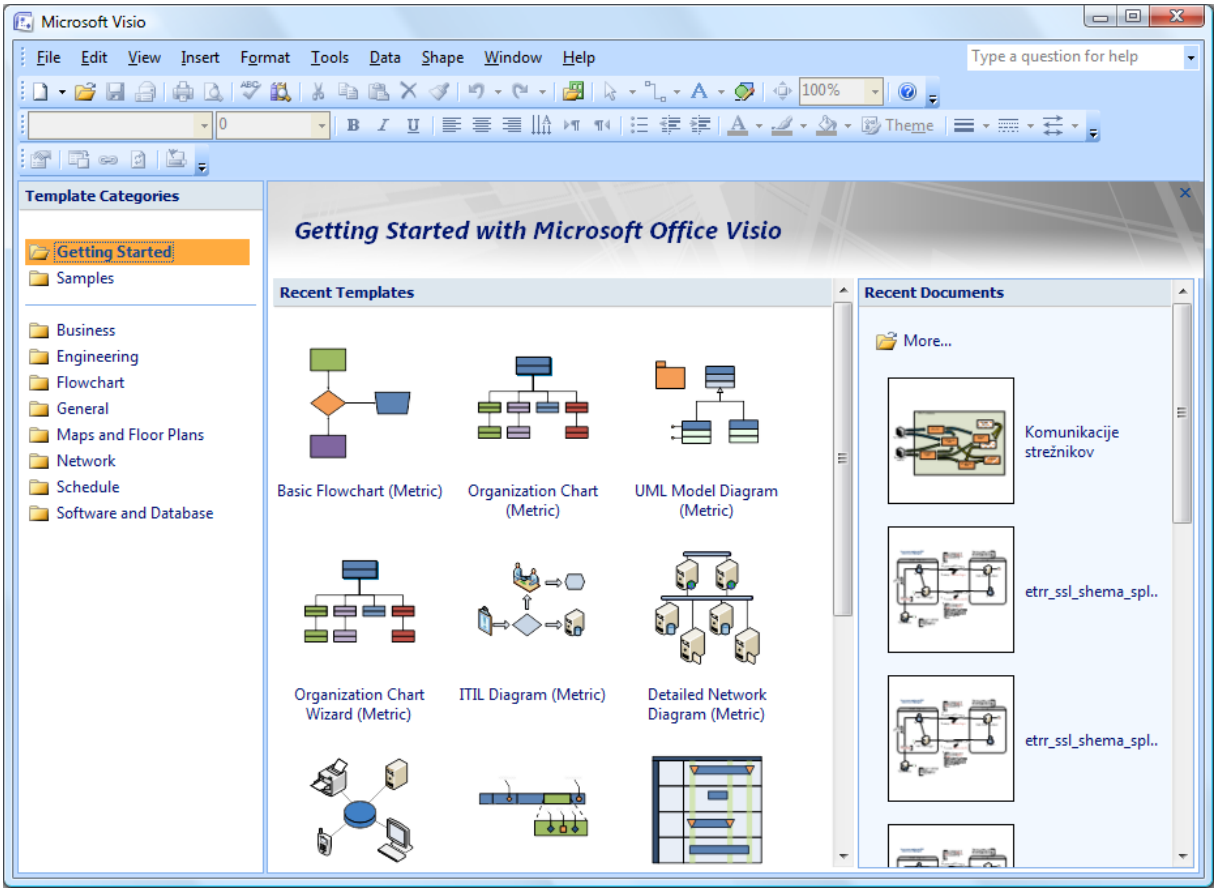
Dejstvo pa je, da se tu ne bomo preveč poglobili v podrobnosti teh dveh platform, saj za njiju na srečo obstaja precej obsežna dokumentacija, ki jo je moč dobiti na internetu. Poleg tega bi lahko zaradi fleksibilnosti ogrodja, na katerem je naš informacijski sistem grajen, naš sistem namestili na več kombinacij obeh platform. Zaradi uporabe knjižnice *Hibernate*, opisane v prejšnjem podpoglavju, smo namreč precej neodvisni od implementacije podatkovne baze, saj *Hibernate* podpira zelo veliko število relacijskih podatkovnih baz. Prav tako pa zaradi uporabe in prilagajanja k vmesniku *Servlet API* in *JSP* specifikaciji lahko izbiramo med skorajda vsemi implementacijami aplikacijskih strežnikov Java. Ti dve stvari torej omogočata našemu sistemu veliko mero robustnosti.

2.4 Uporabljena orodja za razvoj

Tu bi radi naredili kratko predstavitev orodij, ki smo jih tekom te diplomske naloge uporabljali za razvoj in jih bi morebitni bralec prav tako lahko uporabljal, če bi se odločil naš informacijski sistem preučiti ali pa ga nadgraditi.

2.4.1 Orodje za risanje diagramov Microsoft Visio

Napredno orodje za risanje diagramov *Microsoft Visio* (Slika 9) nam pomaga poenostaviti kompleksnost teh opravil z dinamično, podatkovno usmerjeno vizualizacijo in z novimi načini, da to delimo na spletu v realnem času. Začnemo lahko z gradnjo našega diagram s profesionalnim videzom predlog in modernim pristopom ter z vnaprej sestavljenimi oblikami. Nato lahko povežemo diagrame do priljubljenih virov podatkov (na primer *Excel*). Podatki se bodo nato samodejno osvežili kar v našem diagramu, ki pa se odraža v živahni vizualnosti, kot so ikone, simboli, barve, in grafi. Končno lahko z le nekaj kliki objavimo svoje podatkovno povezane diagrame na *SharePoint*, ter zagotovimo dostop tudi drugim na spletu, tudi če nimajo *Visio* orodja. Skupno torej preprostost, podatkovno usmerjene oblike in spletne izmenjave naredijo orodje *Visio* eno izmed najmočnejših in najpreprostejših načinov za prikaz in razumevanje pomembnih informacij.



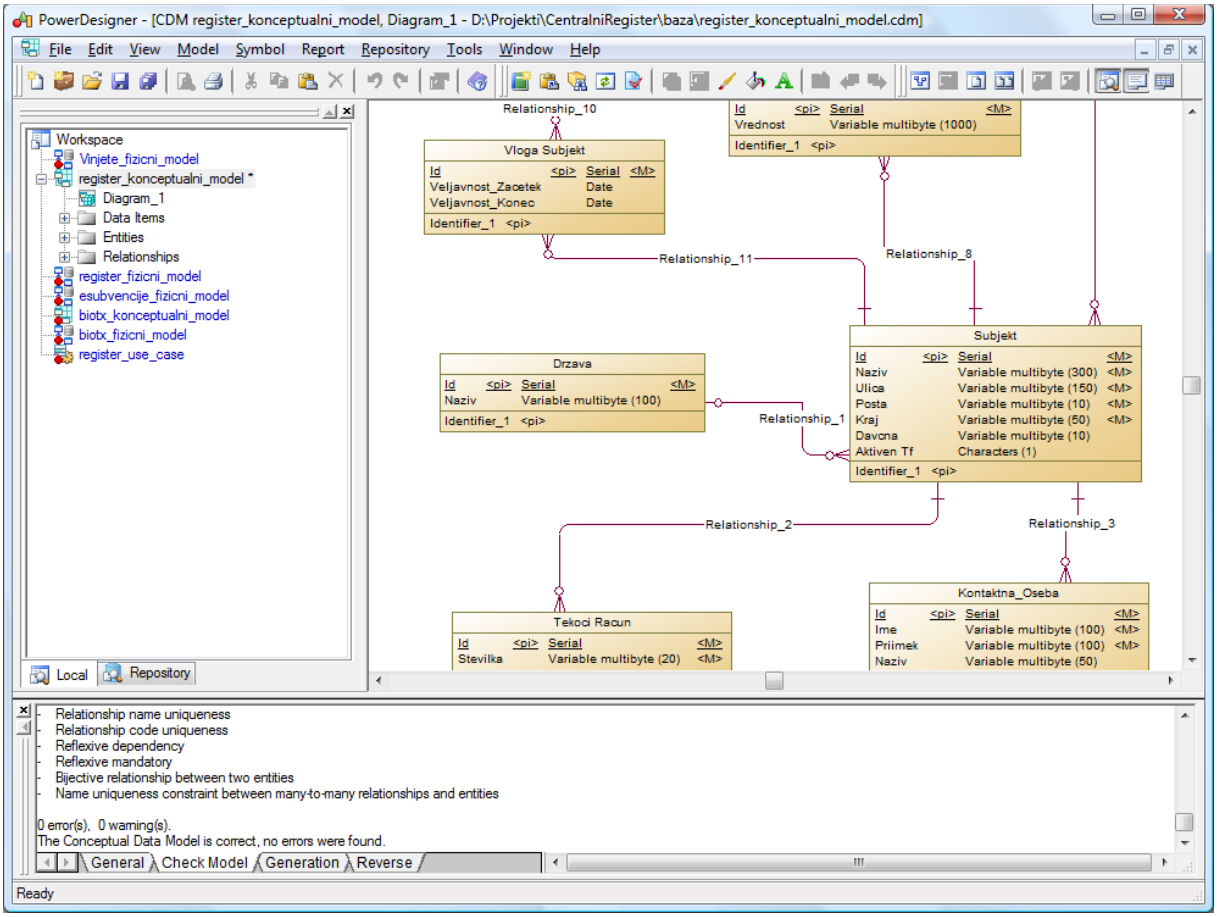
Slika 9: Izgled razvojnega orodja Microsoft Visio

V tej diplomski nalogi bomo smo orodje uporabili za izris nekaterih vrst diagramov, kot so navadni diagrami poteka, diagram primerov uporabe, itd. Izrisali smo tudi nekatere diagrame, ki sicer nimajo neke standardne oblike, so pa vseeno primerni za vizualizacijo kakšne stvari, ki jo samo z besedilom težko opišemo. Tako smo v vsakem poglavju skušali stvari vsaj malce opisati tudi na grafični način. Uporabljali pa smo verzijo *Microsoft Visio 2007*, zunaj pa je

tudi novejša verzija, in sicer *Microsoft Visio 2010.* To orodje je del večjega kompleta orodij *Microsoft Office*, ni pa na voljo v vseh izdajah, predvsem ne v cenejših verzijah.

2.4.2 Orodje za oblikovanje podatkovnih modelov PowerDesigner

PowerDesigner je skupinsko in poslovno orodje za modeliranje, ki ga razvija podjetje *Sybase*. *PowerDesigner* teče pod *Microsoft Windows* kot navadna aplikacija (Slika 10) in prav tako pod orodjem *Eclipse* kot vtičnik (*angl. plugin*). *PowerDesigner* podpira modelno usmerjeno načrtovanje arhitekture programske opreme. *PowerDesigner* uporablja PDM format datoteke. Tržni delež tega orodja za podatkovno modeliranje je v letu 2002 znašal 39%. *PowerDesigner* ima sicer kar visoko ceno licenc, ki znašajo tam od 3000 USD do 7500 USD na delovno mesto razvijalca.



Slika 10: Prikaz izgleda orodja PowerDesigner

PowerDesigner vključuje podporo za:

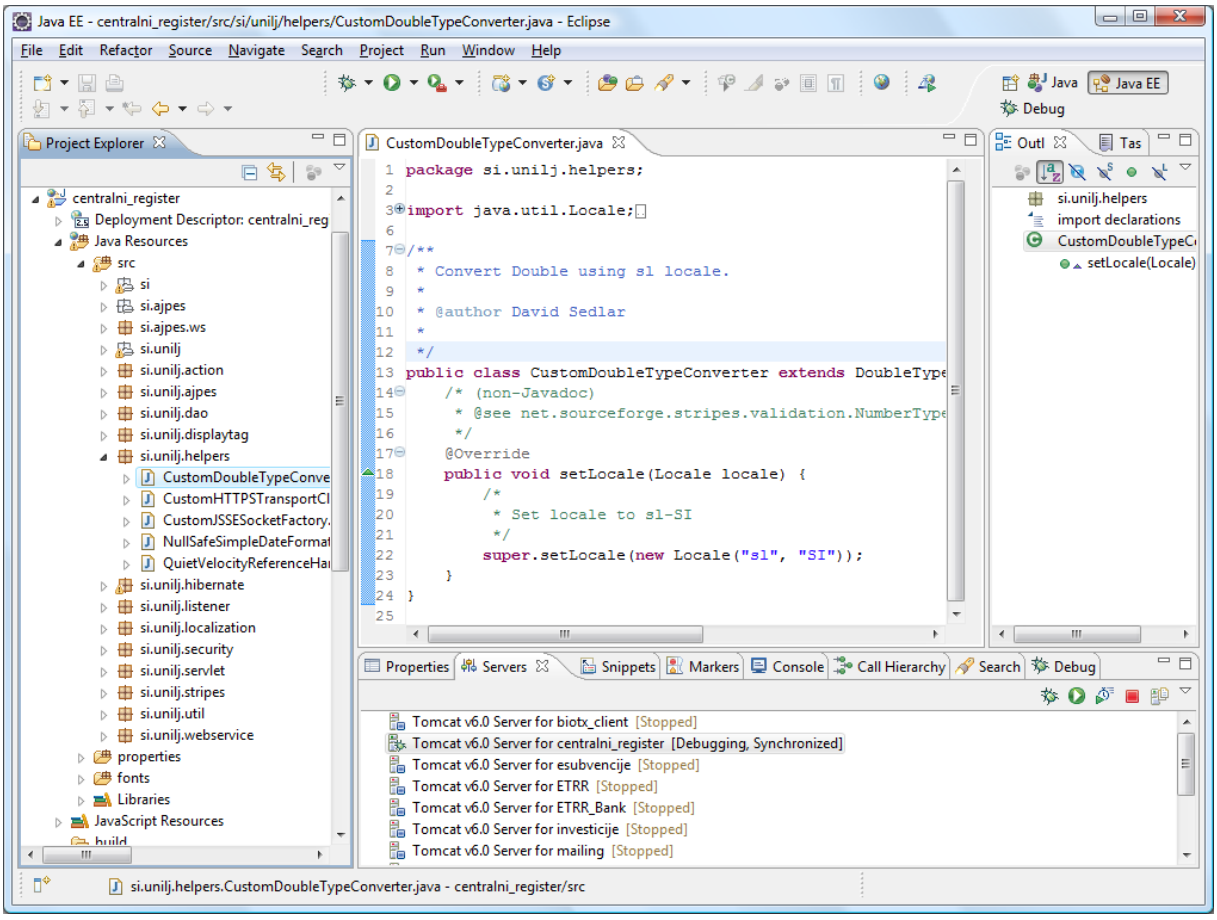
- Modeliranja poslovnih procesov (*ProcessAnalyst*) s podporo za *BPMN*
- Generiranje kode (*Java, C #, VB. NET, Hibernate, EJB3, PowerBuilder*, itd.)
- Podatkovno modeliranje (deluje z večino večjih sistemov *RDBMS*)
- *Eclipse* vtičnik
- Objektno modeliranje (*UML* diagrami 2.0)
- Generiranje poročil
- Repozitorij
- Analiza zahtev

To seveda ni spisek celotnih funkcij, kajti orodje samo res ponuja veliko. V tej diplomski ga bomo uporabili izključno za izdelavo podatkovnega modela in tudi kasnejšo pretvorbo modela v skripto za izgradnjo baze. Uporabljamo pa verzijo 15.2 tega orodja.

2.4.3 Orodje za razvoj aplikacij Eclipse

Eclipse je večjezično orodje za razvoj programske opreme, ki vsebuje tako integrirano razvojno okolje (*IDE*), kot tudi razširljiv sistem vtičnikov (*angl. plugin*). Orodje je večinoma napisano v Javi in se lahko uporablja za razvoj aplikacij v Javi ter z uporabo različnih vtičnikov tudi za razvoj v drugih programskih jezikih, vključno z *Ada, C, C++, COBOL, Perl, PHP, Python, Ruby* (vključno z *Ruby on Rails* ogrodjem), *Scala*, in *Scheme*.

Začetni razvoj izvira iz programskega orodja *VisualAge*. V svoji privzeti obliki je orodje namenjeno razvijalcem Jave, to privzeto obliko sestavlja *Java Development Tools (JDT)* dodatek. Uporabniki lahko razširijo to funkcionalnost z namestitvijo različnih vtičnikov napisanih za ogrodje programske opreme *Eclipse*, kot so že prej omenjeni vtičniki razvojnih orodij za druge programske jezike. Uporabniki pa lahko napišejo in prispevajo svoje vtičnike in module. *Eclipse* je izdan pod pogoji *Eclipse Public License*, *Eclipse* je brezplačna in odprtokodna programska oprema.



Slika 11: Prikaz razvojnega orodja Eclipse

Orodje ima zares veliko paleto funkcionalnosti in je zelo zmogljivo, za razvoj Java aplikacij pa skorajda nima konkurence (eno izmed teh je na primer *Oracle JDeveloper*). V diplomski nalogi se bo uporabljala verzija *Eclipse Java EE IDE for Web Developers* na platformi verzije 3.5.2.

3 Zajem zahtev oziroma diagram primerov uporabe

V tem poglavju se bomo na kratko ustavili pri uporabniških zahtevah in modeliranju le teh. Uporabniške zahteve so namreč zelo pomemben aspekt vsake aplikacije, saj so temelj, iz katerega izhajajo vsi preostali načrti. Na nek način so tudi edina direktna povezava med naročnikom oziroma uporabnikom informacijskega sistema, ter tistim, ki ta sistem razvija. Zato je torej zelo pomembno, da te zahteve skrbno spravimo tudi v formalno obliko. Taka formalna oblika zapisa nam kasneje tudi omogoča lažjo nadaljnjo komunikacijo ter usklajevanje novih ali spremenjenih zahtev. Ne smemo si privoščiti prevelikega odstopanja v katerikoli smeri, se pravi da bi sami dodali funkcionalnost, ki ni zahtevana ali pa ne bi implementirali ali slabo implementirani funkcionalnost, ki bi morala biti tam.

Zajem zahtev je treba opraviti čim bolj zgodaj v razvojnem procesu, narava projektov v *IT* je sicer taka, da se velikokrat zahteve rade spreminjajo tudi v sami fazi razvoja. V nasprotju z recimo načrtovanjem zgradb, kjer velike spremembe v fazi gradnje skoraj niso več mogoče, je to pri razvoju informacijskih sistemov zaradi »mehke« narave programske opreme žal vse prevečkrat realnost. Zato je treba v tej fazi pri zajemu zahtev tudi na grobo oceniti, koliko bodo take spremembe stale v različnih fazah razvoja, tako bo imel tudi naročnik na voljo več podatkov za odločanje.

3.1 Diagram primerov uporabe

Za že prej omejen zajem zahtev bomo tu uporabili standardizirano obliko diagrama, ki mu pravimo diagram primerov uporabe (*angl use-case diagram*) [1]. Nekaj več o orodjih za modeliranje *UML* tehnik [8] je napisanega že v prejšnjih poglavjih, tu pa se osredotočimo zgolj na samo implementacijo in vsebino. Diagram primerov uporabe je zelo primerna tehnika, saj je precej preprost in je tako lahko razumljiv tudi za naročnika oziroma končnega uporabnika in nam služi tudi za nadaljnje dogovarjanje o zahtevah. Diagram lahko dopolnimo tudi z opisi samih primerov uporabe, kar bomo na kratko poizkusili uresničiti v naslednjem poglavju. Tak diagram, opremljen s temi opisi, se nato imenuje model primerov uporabe.

Našega diagrama se bomo lotili tako, da bomo pregledali uporabniške zahteve in iz njih poizkusili izluščiti primere uporabe (Slika 12). To pomeni, da bomo morali kakšno zahtevo razčleniti ali pa več podobnih zahtev združiti. V tem koraku bomo morali tiste zahteve, ki se nanašajo bolj na implementacijo in poslovna pravila, izpustiti oziroma bodo te zahteve implicitno vključene v nekatere bolj splošne primere uporabe.

Slika 12: Diagram primerov uporabe, pridobljen iz uporabniških zahtev

3.2 Kratek opis primerov uporabe

Tu se bomo na kratko ustavili pri že prej omenjenemu diagramu primerov uporabe na zgornji sliki, pri čemer bomo opisali večinoma vsebinski del. Iz uporabniških zahtev lahko tako razberemo, da bomo potrebovali okvirno dva akterja. Akterji so tisti konstrukti, ki bodo te naše primere uporabe dejansko tudi uporabljali, cel sistem, ki ga modeliramo, pa mora biti opredeljen iz vidika teh akterjev. Zato moramo v tem koraku tudi izpustiti vse tiste podrobnosti, ki se na akterje ne nanašajo direktno.

Identificirana akterja v našem sistemu sta navadni uporabnik sistema ter zunanja aplikacija. Večino funkcionalnosti oziroma primerov uporabe bo uporabljal uporabnik, in sicer preko uporabniškega vmesnika. Zunanja aplikacija pa bo dostopala do sistema samo preko spletne storitve, zato je tudi direktno povezana samo s tem primerom uporabe.

Naslednji primer uporabe, ki ga lahko razberemo je prijava v sistem. Zahteve narekujejo, da sta na voljo dva različna načina za prijavo v sistem, in sicer z uporabo uporabniškega imena in gesla ter z uporabo certifikata. To na diagramu ponazorimo tako, da primarnemu primeru uporabe *Prijava v sistem* določimo dva podrejena primera uporabe, ki se nanj navezujeta preko relacije *razširja* (*angl extends*).

Naslednje primere uporabe lahko izluščimo kot funkcionalnost upravljanje s šifranti, na primer *Dodajanje/urejanje vlog* ter *Dodajanje/urejanje atributov*. Ta dva primera uporabe sta precej neodvisna od drugih in tudi najbolj enostavna za realizacijo. Poglavitni del funkcionalnosti pa se nanaša na subjekte. Tako je centralni primer uporabe *Dodajanje/urejanje subjektov*, na tega pa se preko relacije *razširja* nanaša tudi *Določanje vrednosti atributom*, katerega bomo realizirali na istem obrazcu. Ker se pri dodajanju in urejanju subjektov povezujemo tudi na zunanji sistem, ga tukaj ponazorimo kot akterja zunaj sistema, ki pa za razliko od ostalih akterjev ni začetnik komunikacije, saj ta izhaja iz sistema samega. Tu imamo še primere uporabe, ki se nanašajo na obdelovanje podatkov o subjektih, se pravi *Dodeljevanje vlog subjektom, Dodajanje/urejanje tekočih računov* ter *Dodajanje/urejanje kontaktov*. Prav tako je iz zahtev razvidno, da potrebujemo *Iskanje po subjektih*.

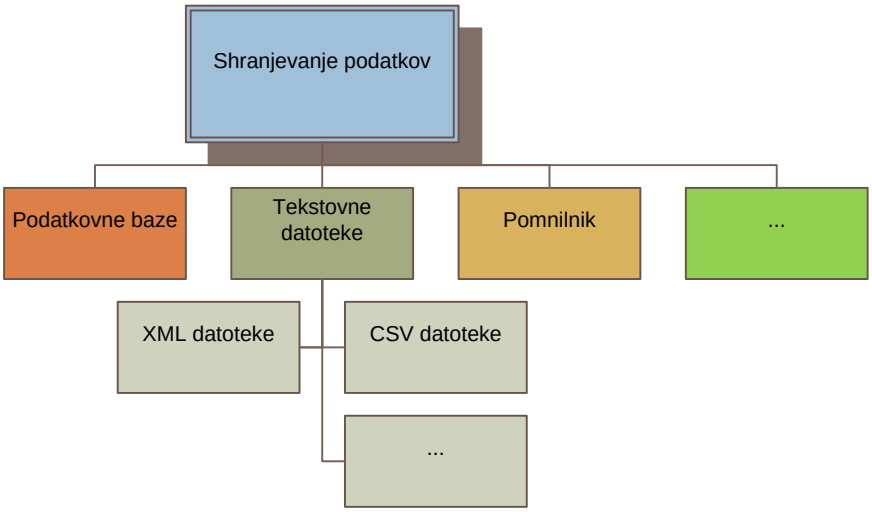
Zadnje dva primera uporabe se navezujeta na bolj administrativne postopke, se pravi *Pregled dnevnika/sledi*, kjer pregledujemo spremembe na samih podatkih, ter *Izvoz podatkov*, ki je funkcionalnost, ki jo bo potrebno implementirati na več obrazcih.

Potrebno je omeniti še meje sistema, v našem diagramu so predstavljene s pravokotnikom, ki obdaja skupino primerov uporabe, vse izven tega pravokotnika smatramo kot okolje sistema, to je tisti del, ki ga ne razvijamo in ki samo vpliva na naš informacijski sistem.

4 Podatkovni model in podatkovna baza

4.1 Splošno o shranjevanju podatkov

V tem poglavju se bomo ustavili pri pomembnejšem delu vsake aplikacije, tako spletne kot neke druge. In to je shranjevanje podatkov. Cilj vsakega informacijskega sistem mora biti, da vse informacije, ki jih shranjuje za nadaljnjo uporabo, to stori čim bolj učinkovito. In primerno orodje, ki nam to omogoča so podatkovne baze. Vseeno pa je treba poudariti, da to še zdaleč ni edini način, kako podatke shranjevati in imeti dostopne, ko jih potrebujemo (Slika 13). Navsezadnje je na primer dosti hitreje, če imamo vse podatke shranjene v glavnem pomnilniku, ampak je ta neprimerno bolj prostorsko omejen, kot kakšne druge rešitve, poleg tega pa tak zapis ni stalen, oziroma se ne ohrani ob izgubi napetosti. Tu se sicer vedno krešejo mnenja, ali je bolje imeti bolj ali manj pomnilniško potratno aplikacijo in katera je na koncu res hitrejša.



Slika 13: Možni načini shranjevanja podatkov

Da pa ne skrenemo iz podanega poglavja, naj povemo, da smo se v tej diplomski nalogi odločili za uporabo podatkovne baze za shranjevanje podatkov, in sicer je podatkovna baza *Oracle Database 11g*, nekaj več smo o tem in ostalih uporabljenih tehnologijah napisalo že v 2. poglavju. V tem poglavju se bomo torej bolj oddaljili od fizičnega pogleda na shranjevanje, saj za to skrbi že sama baza, in se osredotočili na logični pogled, se pravi na podatkovne tabele in relacije med njimi.

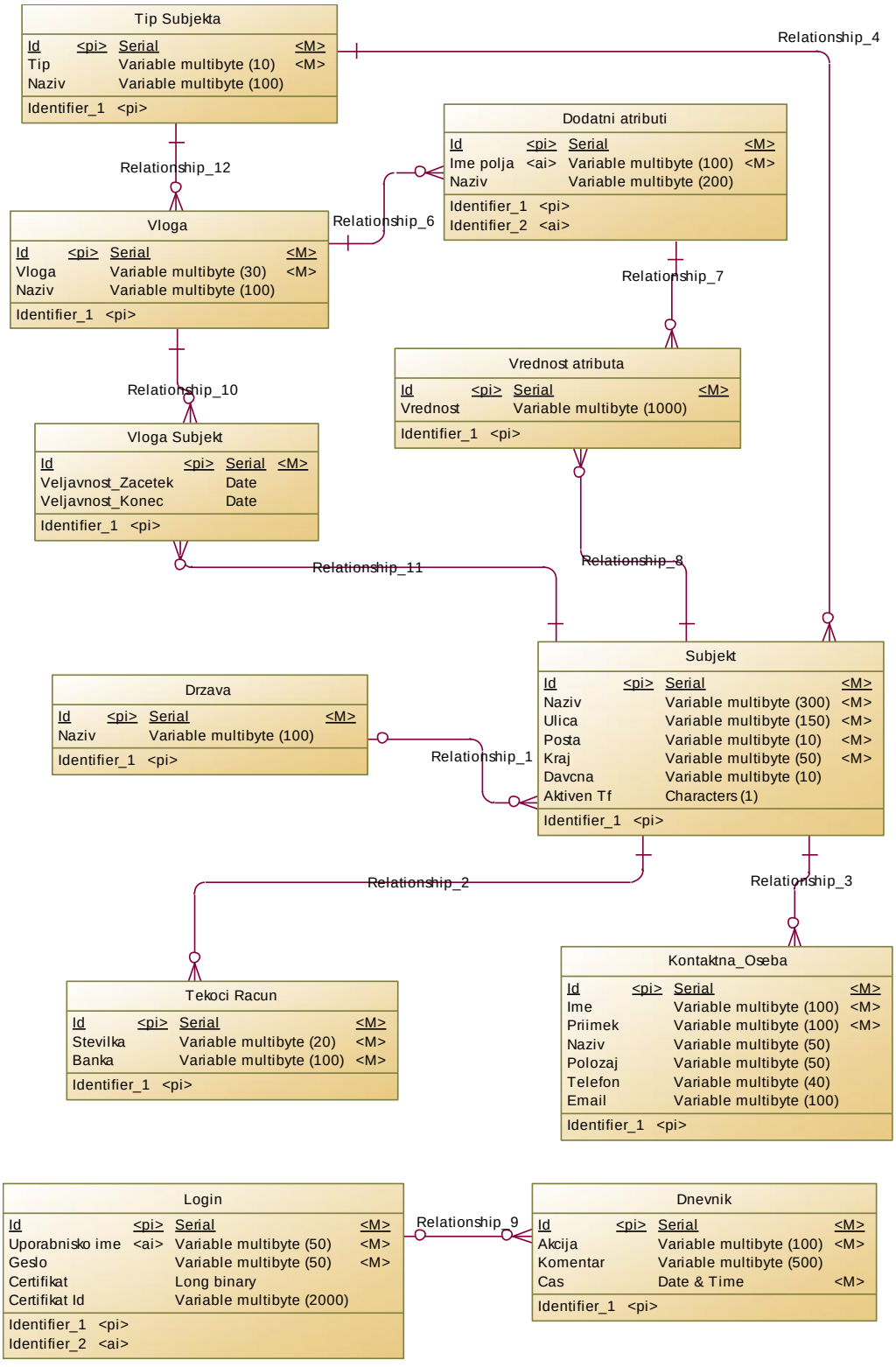
4.2 Podatkovni modeli

Predstavljena bosta dve obliki podatkovnih modelov, to sta konceptualni podatkovni model ter fizični podatkovni model. Oba modela sta na kratko opisana že v drugem poglavju, tu pa bi se lotili dejanske implementacije oziroma izdelave teh dveh modelov. Ta korak je seveda zelo dobro narediti dosti prej, preden se lotimo same kode aplikacije, saj lahko tudi majhne spremembe v podatkovnih modelih propagirajo in nastanejo resen časovni problem kasneje, ko je aplikacija že zrela.

4.2.1 Konceptualni podatkovni model

Konceptualni model podatkovne baze je neodvisen od implementacije baze in je zato najboljši za prikaz vseh tabel. Najboljši v tem smislu, da bralec za njegovo razumevanje načeloma ne potrebuje specifičnega znanja o sami implementaciji podatkovne baze. Tak model je torej lahko implementiran na več načinov, nas pa tu bolj zanima struktura relacijskih tabel in seveda odvisnosti med njimi.

Kar se tiče opisa posameznih tabel, bodo malo podrobneje opisane v nadaljevanju poglavja, tu bi le omenili, da gre za model, ki je v svoji zasnovi precej preprost, kakor je preprosta tudi sama aplikacija (Slika 14) . To pomeni, da nimamo nekih kompleksnih relacij. Na prvi pogled lahko opazimo, da je model dejansko ločen na dva dela, prvi del predstavljajo subjekti in vsi podatki, ki se tičejo njih. Medtem pa je v drugem (spodnjem) delu diagrama vidno, da sta tabeli za varnost oziroma upravljanje z uporabniki ter tabela za shranjevanje dnevnika sprememb ločeni. To je seveda namensko in izhaja iz uporabniških zahtev, kajti ne obstaja neka logična povezava med temi entitetami in zato povezava ne obstaja tudi v modelu.



Slika 14: Konceptualni podatkovni model našega sistema

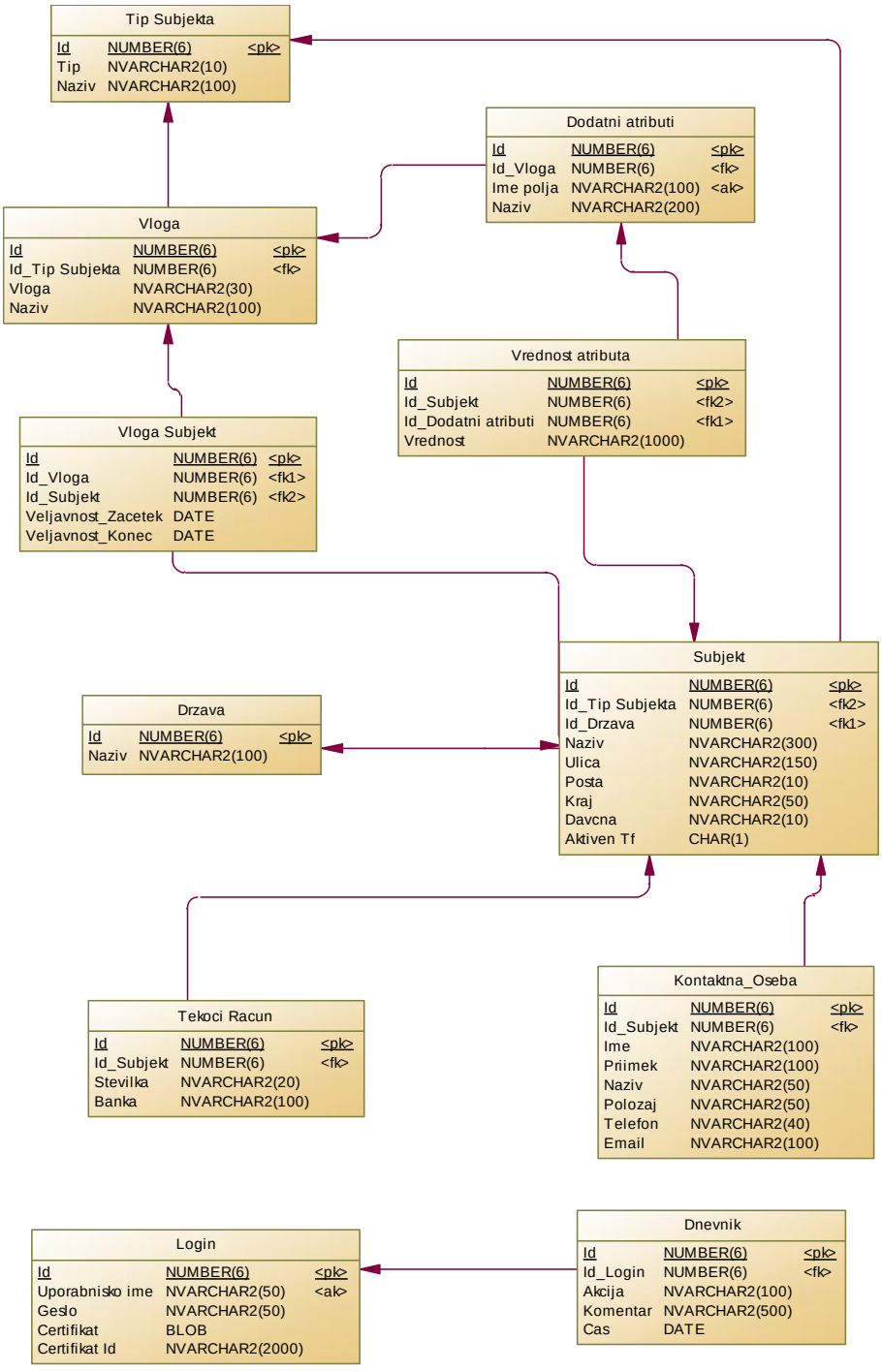
4.2.2 Fizični podatkovni model

Fizični podatkovni model temelji na specifikaciji podatkovne baze *Oracle Database 11g*, kakor smo omenili že poprej. To pomeni, da moramo zgornji konceptualni model transformirati v tako obliko, da bo združljiv z našo podatkovno bazo (Slika 15). Redkokdaj pa je potrebno to operacijo delati na roke, saj nam današnja orodja, eno izmed katerih smo uporabili tudi mi in je opisano v prejšnjih poglavjih, omogočajo, da ta korak avtomatiziramo. Kar sicer še ne pomeni, da s tem nimamo nič dela, saj je fizični model nemalokrat potrebno še ročno prilagoditi, avtomatizacija namreč še vedno ne more predvideti vseh naših želja. Tako moramo na primer ročno določiti, če hočemo da se nek podatkovni tip spremeni v točno tistega, ki nam ustreza, se pravi da ima vse karakteristike, kot so natančnost, dolžina, podporo večjezični abecedi (*UNICODE*) in podobne stvari.

V našem primeru ni bilo potrebno uvesti večjih sprememb, ena od stvari, ki jih na primer naša podatkovna baza ne pozna, so avtomatično naraščajoča polja, ki jih v našem primeru uporabljamo za primarni ključ in s tem tudi enolično identifikacijo. Tako da smo morali za potrebe te funkcionalnosti dodati za vsako tabelo še objekt tipa sekvenca, ki to počne namesto nas. To sicer v spodnjem modelu še ni vidno, je pa vidno v skripti, ki jo uporabimo za izgradnjo same podatkovne baze in je tudi del izvorne kode, priložene k diplomski nalogi.

Tu so še ostale preslikave in spremembe, ki smo jih uporabili:

- Tekstovni podatkovni tip *Variable Multibyte* se preslika v ekvivalenten podatkovni tip *NVARCHAR* v fizičnem modelu. Tak tipa podpira večji črkovni nabor tipa *UNICODE*. Natančnost oziroma dolžina polja ostane ista.
- Podatkovna tipa za datum in čas *Date* ter *Date & Time* se preslikata v tip *DATE*
- Podatkovni tip *Characters* se preslika v ekvivalent *CHAR*
- Podatkovni tip za hrambo podatkov *Long binary* se preslika v podatkovni tip *BLOB*, ki je pravzaprav namenjen hranjenju eksplicitno binarnih podatkov. Za hranjenje daljših tekstovnih podatkov kot so razne *XML* datoteke in podobno je bolje uporabiti tip *CLOB*.
- Ustvarijo se ustrezni tuji ključi, v tabele pa se dodajo ustrezna polja, kamor ti ključi kažejo.



Slika 15: Fizični podatkovni model, transformiran iz konceptualnega modela

4.3 Kratek opis tabel

Sam podatkovni model je, kot smo že omenili, precej preprost. Tu bomo le na kratko preleteli vse tabele in jih opisali. Začnimo kar s prvo tabelo, in sicer se osredotočimo najprej na drugi del modela, ki je neodvisnem od samih poslovnih podatkov (Slika 15). Tu imamo podatkovno tabelo *Login*, kamor se shranjujejo podatki o samih uporabnikih aplikacije, se pravi to niso poslovni subjekti marveč le uporabniki z dostopom do našega informacijskega sistema. V tabeli zabeležimo uporabniško ime in geslo, za primer prijave s certifikatom pa imamo še prostor za shranitev javnega dela certifikata ter enolični identifikator certifikata.

Ta tabela se navezuje na naslednjo, to je *Dnevnik*. V tej tabeli se shranjuje zgodovina dogodkov nad vsemi tabeli, na primer za vsak nov vnos, urejanje ali brisanje iz tabele se tu zabeleži podatek o tipu dogodka, datumu in času. S povezavo na prejšnjo tabelo pa tako zabeležimo tudi samega uporabnika. Tak način povezave je zelo uporaben s stališča zelo dobre prostorske izkoriščenosti, saj se na uporabnika sklicujemo preko tujega ključa. Problem bi lahko nastal, če bi enega izmed teh uporabnikov izbrisali, potem se v dnevniku dogodkov ne bi več videlo, kdo je to akcijo izvedel. Take težave lahko srečamo velikokrat pri delu z relacijskimi bazami in veliko vnaprejšnjega načrtovanja je potrebnega, da se takim stvarem uspešno izognemo.

Potem imamo tu še drugi, poslovni del podatkov. Ta vsebuje najprej bolj preproste tabele, imenovane šifranti, kamor shranjujemo podatke, ki se tekom življenjskega cikla aplikacije ne bodo kaj dosti spreminjali. Najbolj enostaven primer take tabele je tabela *Drzava*, ki vključuje samo polje za naziv države. Podobno imamo tu tabelo *Tip Subjekta*, kjer se hrani tip subjekta (ali gre za fizično ali pravno osebo). Tudi tabelo *Vloga* bi lahko šteli med šifrante, le da je odvisna tudi od tabele *Tip Subjekta*. To pomeni, da se vloge razlikujejo za fizične in pravne osebe. Ker pa ima lahko en subjekt več možnih dodeljenih vlog in ena vloga lahko pripada več subjektom, potrebujemo tudi vmesno tabelo *Vloga Subjekt*. V to tabelo tudi dodamo atributa za začetek in konec veljavnosti vloge, ki je nekomu dodeljena. Sama logika preverjanja prekrivanja veljavnosti vlog se izvaja na aplikativnem nivoju.

Potem je tu podatkovna logika za dodatne atribute k vlogam, ime tabele je *Dodatni atributi*, ki vsebuje samo opis in naziv dodatnih atributov. Ker so atributi vezani na vlogo je tudi tabela povezana s tabelo *Vloga*, medtem ko je zaradi razmerja več proti več s tabelo *Subjekt* povezana preko vmesne tabele, ki smo jo poimenovali *Vrednost* atributa. Ta tabela nam služi tudi kot hramba vrednosti teh dodatnih atributov.

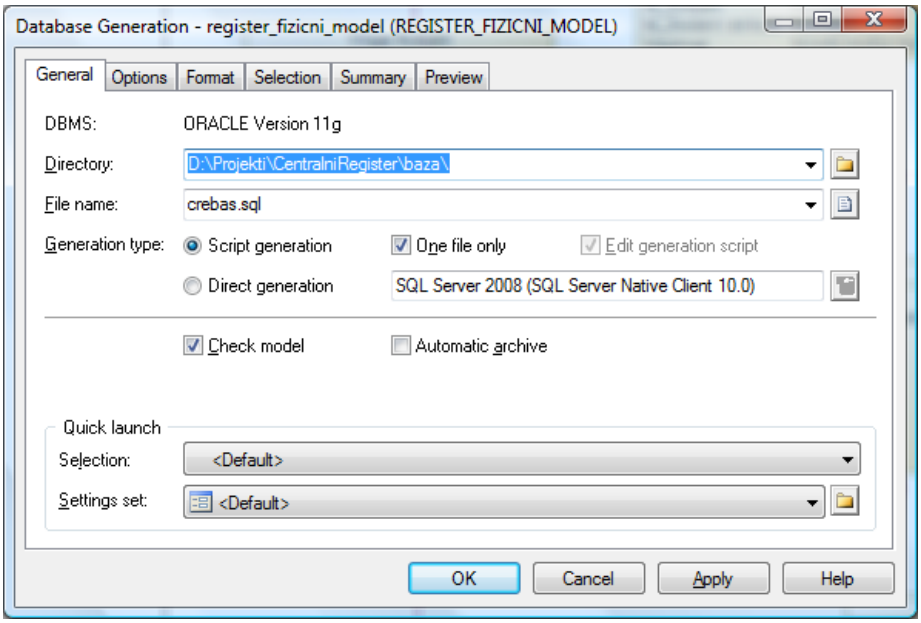
Dotaknimo pa se še predzadnjih dveh tabel, in sicer za hrambo tekočih računov potrebujemo tabelo *Tekoci Racun*, kamor shranimo številko tekočega računa in banko, kjer je račun odprt. Tabela je povezana s tabelo *Subjekt*, kot je povezana tudi druga zmed tabel, *Kontaktna oseba*. Ta pa nam služi hrambi podatkov o kontaktnih osebah za nek subjekt, za neko podjetje je na primer pogosto, da ima določenih več kontaktih oseb.

Čisto nazadnje pa omenimo še tabelo, ki je v osredju samega podatkovnega modela, to je tabela *Subjekt*, na katero se povezuje tudi večina že prej omenjenih tabel. Sem shranjujemo, podobno kot pri kontaktnih podatkih, podatke o samem subjektu, poleg tega pa se nahaja tu še atribut, ki nam pove, ali je nek subjekt aktiven. Subjekte namreč lahko tudi izbrišemo oziroma deaktiviramo, tako sicer še vedno ostanejo v sami podatkovni bazi, niso pa več aktualni za obdelavo. Tak način je zelo primeren predvsem iz razloga, da je brisanje podatkov, ki so med seboj zelo prepleteni, težavno in nemalokrat se zgodi, da zaradi nepazljivosti pri teh zadevah pridemo v stanje, ko je baza nekonsistentna ali pa imamo več vnosov, ki so izgubili svoje nadrejene ali podrejene vnose.

4.4 Skripta za generiranje podatkovne baze

To poglavje zaključi naš podatkovni model, in sicer tako, da zgornje modele, ki so nastali pri načrtovanju prevedemo v dejansko skripto oziroma nek skupek ukaznih stavkov, ki nam naš zamišljeni model tudi ustvarijo na podatkovni bazi, ki smo si jo izbrali. Podatkovni diagrami so sicer izredno uporabni za človeško predstavo podatkovnega modela, v nasprotju s skripto, ki vsebuje stavke *SQL*, ki so sicer kot samostojna enota razumljivi za nekoga, ki to področje pozna, vendar ko kompleksnost modela naraste, tudi taka skripta postane nepregledna. Zato tudi po navadi ta korak sledi za načrtovanjem samega modela, še večkrat pa nam to skripto zgradi že orodje samo.

Za naš primer smo uporabljali orodje *Powerdesigner*, o katerem je nekaj več napisanega v prejšnjih poglavjih, tu pa bi podali postopek, ki ga moramo izvesti, da na koncu dobimo kot rezultat to skripto. V programu moramo ob odprtju fizičnega podatkovnega modela najti ukaz *Generate database (Ctrl+G)*, ki nam odpre ustrezno okno z nastavitvami (Slika 16).



Slika 16: Nastavitveno okno za kreiranje skripte za podatkovno bazo

Tu je veliko možnosti, vendar zaradi enostavnosti ne bomo šli v podrobnosti, kajti za tako enostaven model, kot je naš so tu dovolj že privzete nastavitve. Paziti moramo le, da je izbran pravi proizvajalec in verzija podatkovne baze, v našem primeru *Oracle Database 11g*. Ko orodje poženemo s klikom na *OK* gumb, nam skripto shrani v datoteko, katero smo določili.

```
/*==============================================================*/
/* DBMS name:      ORACLE Version 11g                           */
/* Created on:     9.12.2010 22:18:55                           */
/*==============================================================*/


alter table DNEVNIK
   drop constraint FK_DNEVNIK_RELATIONS_LOGIN;

alter table DODATNI_ATRIBUTI
   drop constraint FK_DODATNI__RELATIONS_VLOGA;+

/* (se nadaljuje) */
```

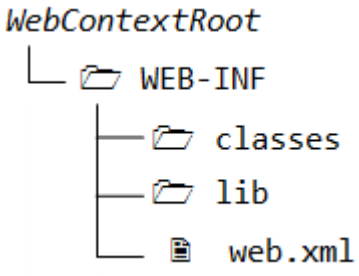
Slika 17: Primer kreirane skripte

Kreirano skripto nato poženemo na naši podatkovni bazi, uporabniška shema mora biti v podatkovni bazi že ustvarjena, in sicer z ustreznimi pravicami za izvajanje. Primer generirane skripte vidimo na zgornji sliki (Slika 17), celotna skripta pa je predolga za vsebino te diplomske naloge, zato jo najdemo ob sami izvorni kodi, in sicer se nahaja v datoteki *\config\database_initialize.sql.*

5 Postavitev aplikacije na strežnik

5.1 Splošno o strukturi spletne aplikacije in postavitvi na strežnik

Začeli bomo s kratkim primerom, kako pripraviti spletno aplikacijo za namestitev na aplikacijski strežnik. Osnovne podrobnosti o aplikacijskem strežniku, ki ga uporabljamo so navedene v prejšnjih poglavjih, vendar zaradi standardnih okvirjev , v katerih so spletne aplikacije grajene, je struktura zelo podobna, če ne kar ista. Je pa seveda potrebno vedeti, da to velja samo za aplikacije, ki se prilagajajo vmesniku spletnih aplikacij Java (*angl. Java Servlet API*) [4]. Obstajajo pa seveda tudi alternative, kot tudi drugi programski jeziki, ki so morda že v začetku bolj spletno naravnani (na primer *PHP*).



Slika 18: Običajna struktura spletne aplikacije

Na zgornji sliki (Slika 18) imamo primer, kakšna naj bi bila struktura tipične Java spletne aplikacije. Čisto na vrhu imamo mapo, ki predstavlja kontekst naše spletne aplikacije (*angl. web context*), ta mapa je na sliki označena z *WebContextRoot*. Pod to mapo se nahaja mapa z imenom *WEB-INF*, v kateri se nahajajo za našo aplikacijo pomembne datoteke in mape. Najprej imamo tu mapo *classes*, ki vsebuje že prevedeno izvorno kodo, tipično so to datoteke s končnico **.class*, v mapi *lib* se nahajajo knjižnice oziroma komponente, ki jih aplikacija za svoje delovanje potrebuje. Te so običajno v datotekah s končnico **.jar*. Na koncu pa imamo še za aplikacijski strežnik najpomembnejšo datoteko, in sicer *web.xml*. V tej datoteki se pravzaprav nahajajo ukazi aplikacijskemu strežniku, kako naj se naša aplikacija izvaja, kdaj naj se izvaja in pod kakšnimi pogoji, dolžina seje in podobno. Ta datotečna struktura se nato zapečati v ZIP datoteko s končnico **.war*, to je končnica, ki jo aplikacijski strežnik prepozna in naloži.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
        http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">
    <display-name>Centralni register</display-name>

    <!-- Session config -->
    <session-config>
        <session-timeout>120</session-timeout>
    </session-config>

    <welcome-file-list>
        <welcome-file>index.action</welcome-file>
    </welcome-file-list>
</web-app>
```

Slika 19: Primer običajne web.xml datoteke

Dotaknimo pa se še datoteke *web.xml*. Na zgornji sliki imamo primer te datoteke (Slika 19), ki vsebuje samo najbolj osnovne stvari, ki jih aplikacija potrebuje. Podano imamo ime aplikacije, to ime povzame strežnik, da jo lažje identificiramo. Potem se tu nahaja še podatek o dolžini seje, in sicer v minutah, v našem primeru seja poteče po dveh urah neaktivnosti.

Seveda je še nemalo ostalih nastavitev, ki pa so vse opisane v specifikacijah vmesnika spletnih storitev v Javi. Tu smo zaradi obsega te diplomske opisali samo tipične, oziroma tiste nastavitve, ki jih aplikacija potrebuje za delovanje.

5.2 Konfiguracija aplikacija pred namestitvijo in uporabo

Aplikacija sama se nahaja v *WAR* datoteki ali pa je to datoteke potrebno še zgraditi z uporabo priložene skripte, ki se nahaja v datoteki: *\build.xml*. Ta skripta je pisana v obliki, ki jo lahko poganja komponenta Ant [5], ki je v svetu Jave najbolj razširjen način za ta namen. V sami strukturi aplikacije se datoteka za nastavitev nahaja na lokaciji: *\WebContent\WEB-INF\config\config.properties*.

Če jo odpremo s tekstovnim urejevalnikom vidimo naslednje nastavitve (Slika 20).

```
# AJPES nastavitve (vse nastavitve so potrebne)
ws.ajpes.url=https://wwwt.ajpes.si/wsPrsInfo/PrsInfo.asmx
ws.ajpes.username=wsPrsInfoTest
ws.ajpes.password=wsprsinfo
ws.ajpes.sifraStoritve=PRS_SN_P
ws.ajpes.trustStorePath=/WEB-INF/certs/truststore.jks
ws.ajpes.trustStorePass=changeit
# JKS, PKCS12, ...
ws.ajpes.trustStoreType=JKS

# AJPES Proxy nastavitve (neobvezne, ce so nastavljene se upostevajo)
#ws.ajpes.proxyHost=62.77.49.68
#ws.ajpes.proxyPort=8080
#ws.ajpes.proxyUser=
#ws.ajpes.proxyyPass=

# Register web service host, port and path
ws.register.hostname=localhost
ws.register.port=8088
ws.register.path=/register

# SSL server port
ssl.port=9443
```

Slika 20: Primer nastavitvene datoteke aplikacije

Pomembne nastavitve so uporabniško ime in geslo za dostop do storitve *AJPES*, za testiranje je priloženo testno uporabniško ime in geslo. Ostale nastavitve spustimo. Če do zunanjega omrežja ne dostopamo direktno je potrebno nastaviti še nastavitve za vmesni strežnik (*angl. proxy*), sicer aplikacija ne bo mogla dostopati do *AJPES* storitve.

Poleg teh nastavitev je potrebno nastaviti še naslov, kjer bo dostopna spletna storitev (vrste *SOAP 1.1*) za dostop do registra. Privzeto je naslov strežnika *localhost*, kar je primerno za lokalno testiranje. Spremenimo lahko tudi vrata (*angl. port*) ter pa pot (*angl. path*). Te nastavitve bo aplikacija sama prebrala in jih na koncu izpisala na osnovni strani aplikacije, kakor je možno videti v prikazu uporabniškega vmesnika v naslednjem poglavju.

Zadnja nastavitev so *SSL* strežniška vrata, ki morajo ustrezati vratom, na katerih je nastavljen varen promet na spletni strežnik. Privzeta vrednost je 8443, lahko pa vrednost poljubno nastavimo. Ko smo vse nastavitve ustrezno spremenili lahko datoteko shranimo in ponovno oz. na novo zaženemo aplikacijski strežnik, če pa je potrebno, aplikacijo ponovno namestimo (*angl. deploy*) nanj.

6 Pregled implementacije s primeri uporabniškega vmesnika

V tem podpoglavju bomo poizkusili bolj podrobno razdelati vse možne scenarije iz diagrama primerov uporabe, s tem pa bomo tudi pokrili vse tipične primere uporabe, kot tudi prikazali naš informacijski sistem iz stališča, kot je viden končnim uporabnikom. Poleg tega nam to služi tudi kot pregled implementacije oziroma lahko spremljamo, katere funkcionalnosti smo implementirali in na kakšen način, brez da bi se spustili na najnižji nivo izvorne kode, čeprav je bralcu tudi ta na voljo ob tej diplomski nalogi.

Zanima nas torej sistem, kot ga vidi končni uporabnik. Sistem, ki ga vidi se praktično ne razlikuje bistveno od katerekoli druge spletne strani , uporabnik namreč nima vpogleda v to, kateri del spletne strani se je generiral dinamično, kateri del pa je statičen. Da bi najbolje predstavili ta vidik, bomo uporabili kar posnetke samih strani, v obliki statične slike. Temu se bo dodal še kratek opis scenarija.

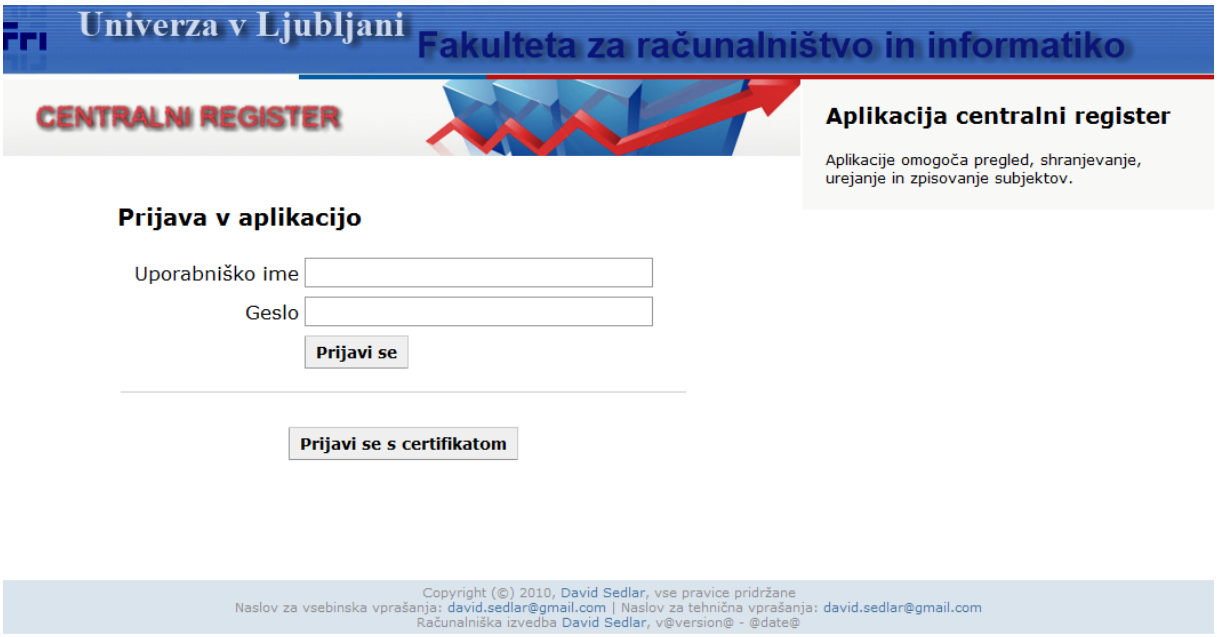
6.1 Prijava v sistem

Prvi primer uporabe, ki je pravzaprav predpogoj vsem ostalim (razen pregleda osnovne strani in dostopa do registra), je prijava v sistem (Slika 21). Tudi sicer je v večini informacijskih sistemov prisoten tak ali podoben način nadzora dostopa. Sistemi, ki upravljajo z nadzorom dostopa so lahko čisto enostavni in predstavljajo samo majhen del sistema. Imamo pa tudi povsem ločene in izredno kompleksne sisteme upravljanja, ki lahko nadzorujejo tudi veliko število aplikacij oziroma drugih informacijskih sistemov.

Ena od uporabniških zahtev je tudi:

- Prijava v uporabniški vmesnik naj temelji na uporabniškem imenu in geslu ali digitalnem potrdilu.

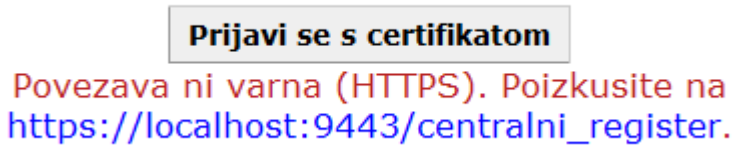
To pomeni, da je potrebno minimalno podpreti vsaj to funkcionalnost. Ker uporabniške zahteve niso specifične glede implementacije, se ta del prepusti izvajalci. Podrobnosti implementacije so sicer vidne v sami izvorni kodi aplikacije.



Slika 21: Obrazec za prijavo v sistem

V aplikacijo se torej prijavljamo preko uporabniškega imena ali gesla ter tudi preko certifikata. Prijava z uporabniškim imenom in geslom je standardna in najbolj razširjena metoda, ki je vidna v več spletnih aplikacijah. Uporabnik vnese uporabniško ime in geslo, ki mu je bilo dodeljeno, nato pa sistem preveri, če tak uporabnik v sistemu obstaja. Če uporabnik obstaja, sistem prebere tudi njegove uporabniške pravice, tako da lahko izve, do katerih delov aplikacije ima uporabnik dostop.

Za prijavo preko certifikata je potrebna varna povezava do strežnika, nekaj o aplikacijskem strežniku smo povedali že v prejšnjih poglavjih, več o nastavitvah pa je moč najti v dokumentaciji strežnika samega. Če varna povezava ni prisotna, nas aplikacija sama opozori na to (Slika 22), podan pa je tudi naslov oziroma vstopna točka, kjer lahko do strežnika vstopamo varno. Ker smo pojem varno tukaj večkrat uporabili naj bo jasno, da gre za povezavo preko *HTTPS Secure* protokola [3].



Slika 22: Opozorilo pri prijavi v sistem s certifikatom

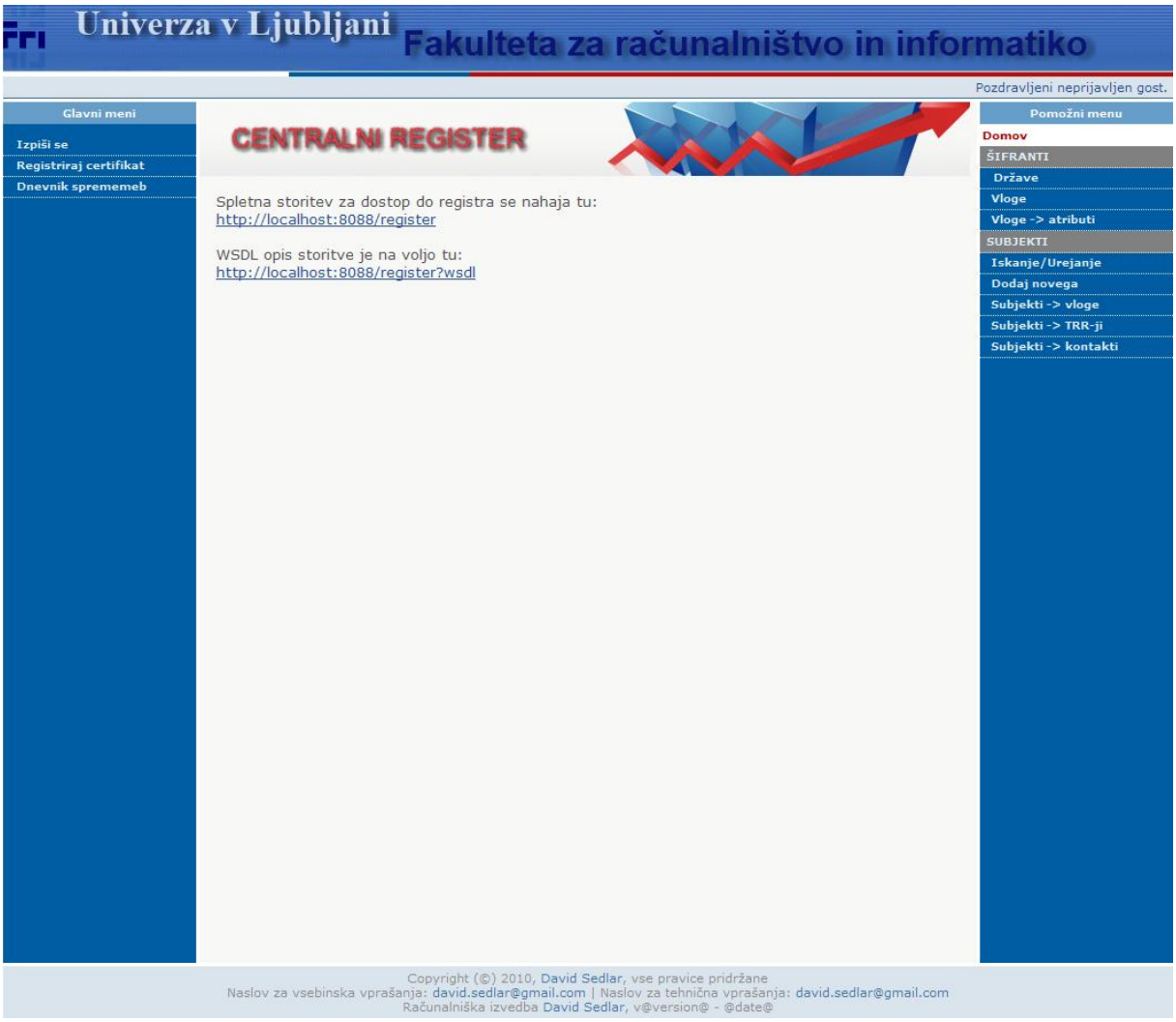
Vprašanje, ki se pojavi je, ali je vsak certifikat dober za dostop do aplikacije. Certifikati, ki so podprti, oziroma katerim je dostop dovoljen so že vnaprej določeni z nastavitvijo samega stražnika. Podprti so na primer certifikati, ki so pri nas precej tipični, kar pomeni, da je njihov overitelj znan in zaupanja vreden. V tem primeru so to overitelji Sigen in Sigov [10], Halcom, Poštarca.

6.2 Osnovna stran

Osnovna stran nam prikaže podatke, ki jih aplikacije potrebujejo za dostop do spletne storitve za dostop do registra (Slika 23). Osnovna stran je dostopna brez predhodne prijave, za vso ostalo funkcionalnost pa se je potrebno prijaviti. Ta princip je implementiran predvsem zato, da pred registracijo ali pa prijavo damo uporabniku na voljo nekaj uporabnih informacij, na primer o samem delovanju aplikacije, njeni vsebini in podobno.

V našem primeru se na tem mestu nahajajo, kot je bilo omenjeno že prej, podatki, ki jih ostale aplikacije potrebujejo za dostop do registra. Ker se dostop do registra vrši preko protokola standardnih spletnih storitev (*SOAP*), je v ta namen tu prikazan URL naslov te storitve. Kako je storitev implementirana in ostale tehnične podrobnosti nas tu ne zanimajo, o komponentah za povezovanje na spletne storitve je več napisanega tudi že v prejšnjih poglavjih. Naj le omenimo to, da je opis spletne storitve sestavljen iz dveh delov, in sicer:

1. Naslov same spletne storitve
2. Naslov *WSDL* dokumenta, ki za zgornjo spletno storitev v standardizirani obliki poda vse parametre, ki jih aplikacija potrebuje za komunikacijo s storitvijo. Se pravi sem spadajo vse metode, ki jih storitev ponuja, parametri teh metod in vrnjen rezultat, tipi teh parametrov kot tudi tipi vseh podprtih parametrov itd.



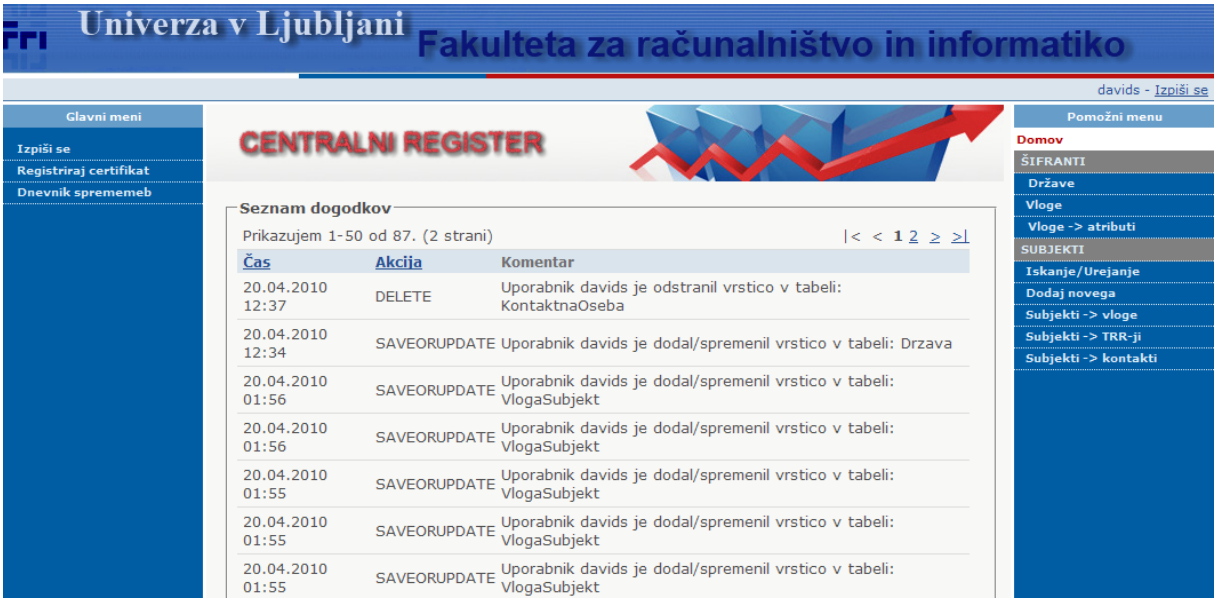
Slika 23: Osnovna stran

6.3 Dnevnik sprememb

Dnevnik sprememb nam omogoča pregled nad vsemi spremembami v registru, se pravi čas spremembe, kateri uporabnik je izvedel spremembo in za kak tip spremembe gre (Slika 24). Podatke lahko izvozimo v več formatov (*Excel, Word, PDF*). Ta stran bi bila v neki realni aplikaciji navadnim uporabniškim vlogam skrita, saj je njen namen zgolj nadzorovanje nad delovanjem aplikacije. Zakaj se nekdo odloči za nadzor aplikacije je seveda odvisno od njegovih potreb, vendar naročniki velikokrat postavijo pred razvijalca zahtevo, da hočejo imeti nek centralen nadzor nad tem, kdo aplikacijo uporablja, na kakšen način, kako uporaba aplikacije vpliva na podatke v podatkovni bazi in podobno.

Zaradi preprostosti smo se v našem primeru odločili narediti samo nek osnoven pregled nad spremembami v podatkovni bazi, zabeleži se torej čas transakcije, vrsta transakcije (ali gre za vnos, izbris ali posodobitev vrstice), poleg tega pa se kot komentar zabeleži tudi uporabnik, ki je to spremembo sprožil. Seveda kot vse druge podatke, tudi te shranjujemo v podatkovno

bazo, ni pa to edini način, ki se za ta namen lahko uporabi. Velikokrat hočemo imeti zabeležene podatke o nadzoru tudi v primeru odpovedi podatkovne baze ali pa pri težavah s povezavo na le to. Tedaj nam prav pride tudi neke vrste lokalno beleženje sprememb. Ta aplikacija v ta namen uporablja dodatno beleženje večine parametrov delovanja aplikacije, ki se nato na aplikacijske strežniku shrani kot tekstovna datoteka.



Slika 24: Dnevnik sprememb

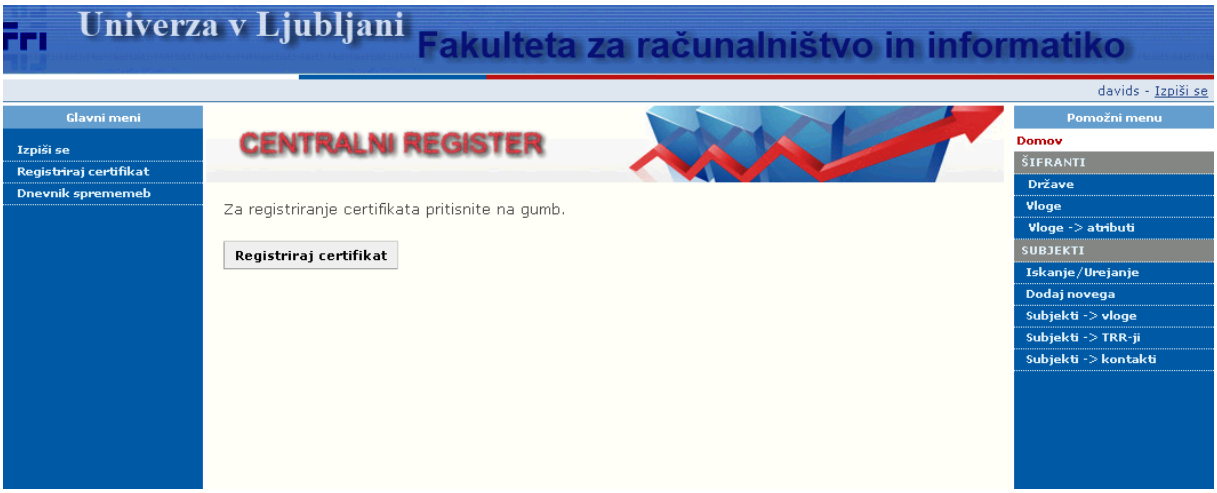
6.4 Registracija certifikata

Preden se lahko s certifikatom prijavljamo v aplikacijo, je potrebno certifikat registrirati, kar tudi pomeni, da se moramo za ta postopek najprej prijaviti v sistem z dodeljenim uporabniškim imenom in geslom. Ko se certifikat enkrat registrira, ga naslednjič ni potrebno več, razen če želimo vstopati s kakšnim drugim certifikatom.

Sama stran je karseda enostavna, saj vsebuje le en sam gumb, ki uporabniku omogoča registracijo svojega certifikata v sistem (Slika 25). Večina poslovne logike na tej strani je torej v ozadju. Uporabnik, ki je seznanjen z delovanjem javnih in privatnih ključev bo najbrž znal povedati, da brskalnik strežniku vedno pošlje javni del certifikata, kar je zelo pomembno za varnost. To pa seveda tudi pomeni, da moramo ta del certifikata (oziroma njegov izvleček) nekam shraniti, da lahko potem uporabnika samega identificiramo kot pravega. Za ta primer se ponovno uporabi podatkovna baza, kjer je po registraciji certifikata shranjen uporabnikov javni del tega certifikata, ta se nato uporabi za nadaljnjo primerjavo. Se pravi, če uporabnik zopet pride na stran z istim certifikatom, ga bo sistem prepoznal, povezal z njegovimi uporabniškimi podatki (tu je mišljena predvsem uporabniška vloga) ter spustil naprej.

To seveda ni edina možna rešitev uporabe certifikatov za vstop v neko spletno aplikacijo. Ena izmed možnosti je na primer ta, da se dovoljeni certifikati nastavijo že na nivoju aplikacijskega strežnika, v tako imenovanih varnih shrambah *(angl. truststore*), je pa tu velik problem dinamičnega upravljanja s temi certifikati, medtem ko aplikacija sama že teče.

Tu bi le še opozorili, da je, kot smo to že omenili prej, za uspešno registracijo certifikata in tudi dostop do stran is certifikatom, potrebno do strani dostopati preko varne povezave, na kar nas sistem samodejno opozori.

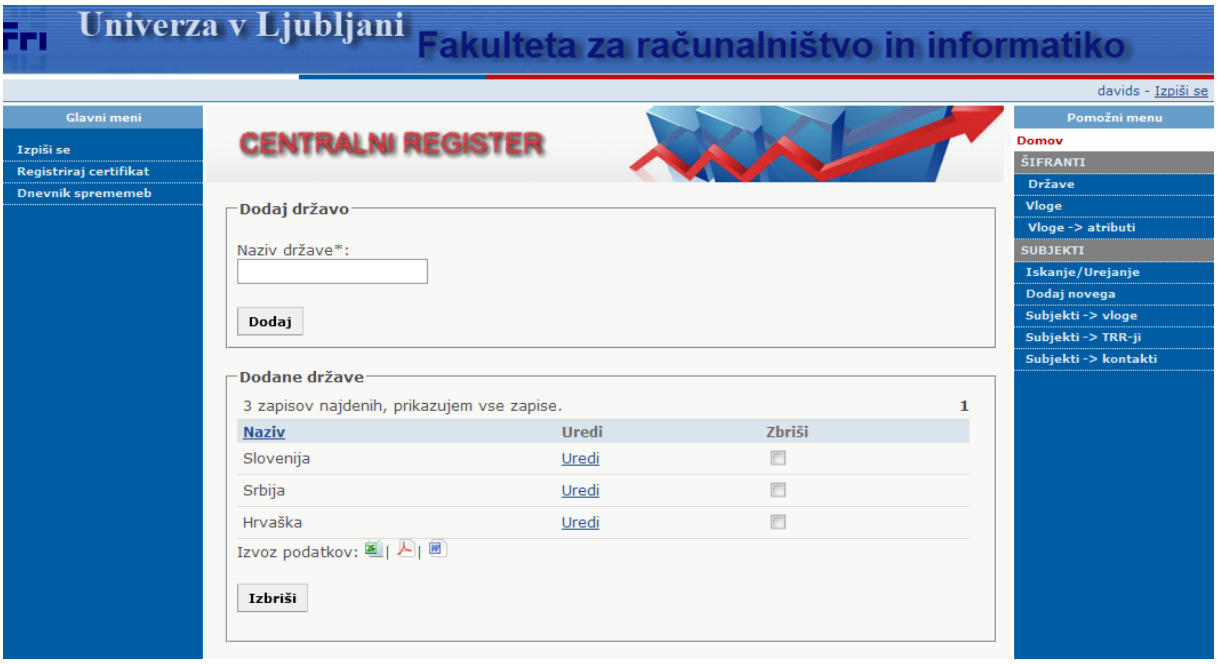


Slika 25: Registracija certifikata

6.5 Šifrant držav

Na tem obrazcu (Slika 26) lahko dodajamo ali urejamo države, ki jih lahko kasneje izbiramo pri urejanju oz. dodajanju subjektov. Podatke lahko zopet izvozimo v različnih formatih. Taka stran je tipična za večino aplikacij, ki se navezujejo na relacijsko podatkovno bazo. Cilj podatkovnega modela je namreč racionalizacija hrambe podatkov, kar pomeni, da se pogosto uporabljene vrednosti shranijo kot šifranti, na te vrednosti pa se potem nekje sklicujemo. Šifrantom bi lahko rekli tudi zaloge vrednosti, saj nam dajo na izbiro dovoljene vrednosti, med katerimi lahko izbiramo.

V našem primeru bi lahko mogoče celo izpustili korak izdelovanja obrazca za države, saj je to eden izmed šifrantov, ki se načeloma od začetnih vrednosti ne bo dosti spreminjal. Je pa še vedno lahko za naročnika tak obrazec uporaben, če se odloči, da bo recimo imel podprte samo nekatere od celotne množice držav.



Slika 26: Šifrant držav

Pri vnosnih obrazcih, kot je zgornji je potrebno predvsem paziti na vnosne kontrole, tu imamo sicer zelo preprosto kontrolo, ki ob kliku na gumb »Dodaj« samo preveri, če je polje »Naziv države« izpolnjeno pravilno, oziroma da ni prazno (Slika 27). Če namreč tega ne bi preverjali že na aplikacijski plasti, bi do napake prišlo pri shranjevanju v podatkovni bazi, če bi bil ta atribut označen kot obvezen. Take kontrole so zelo priporočljiv del vsake aplikacije, kjer imamo vnosna polja. Obrazec nam omogoča tudi urejanje že dodanih vrednosti, prav tako pa tudi brisanje obstoječih vrednosti.

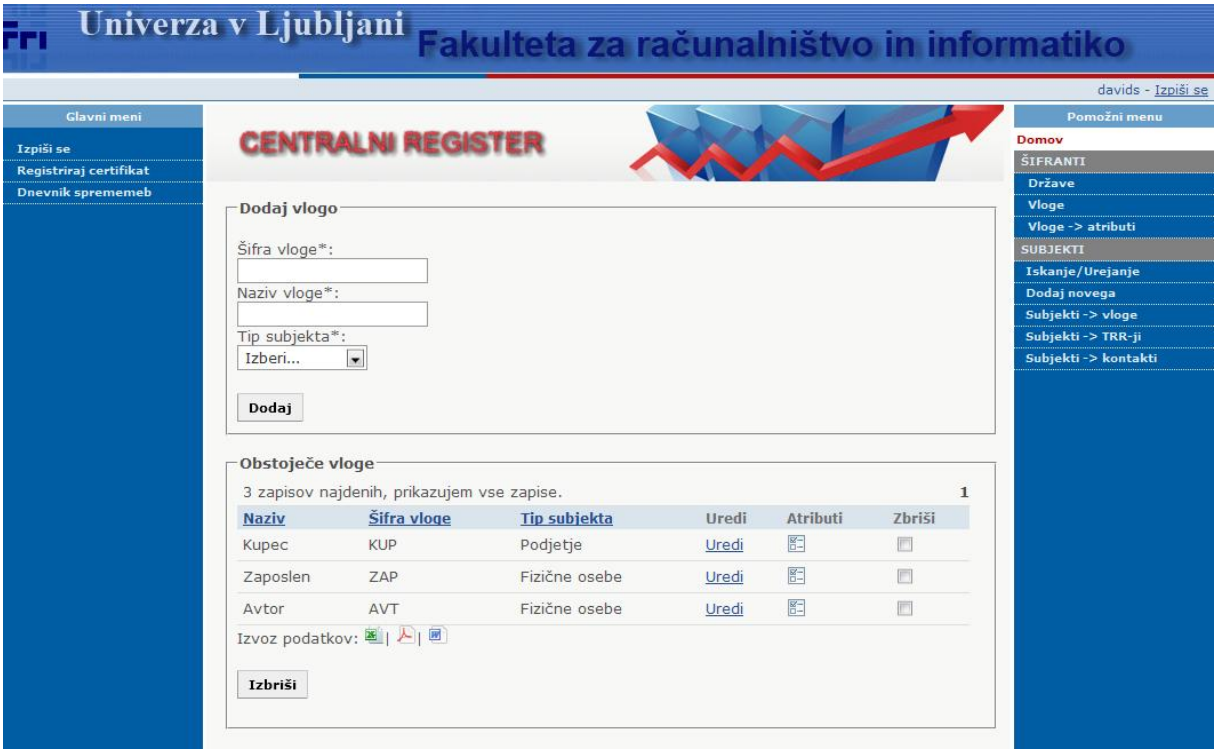
Slika 27: Opozorilo pri napačnih vhodnih podatkih

6.6 Urejanje/dodajanje vlog

Obrazec nam omogoča urejanje in dodajanje vlog, v katerih lahko nastopajo subjekti (Slika 28). Vlogam določimo tudi tip subjekta, se pravi ali gre za fizično ali pravno osebo. Šifra vloge je kratko ime, ki se uporablja za identifikacijo, naziv pa polno ime. Tip subjekta je v našem primeru tudi šifrant, vendar ker lahko zavzema samo dve vrednosti, zanj nismo naredili posebnega obrazca. Podobno kot pri prejšnji vlogi je tudi tu prisotna kontrola vnosa, in sicer

ker so vsa tri vnosna polja obvezna, nas mora aplikacija ustrezno obvestiti, če katerega pustimo praznega oz. v primeru izbire tipa subjekta, če ne izberemo nobenega tipa.

To je prvi obrazec, kjer lahko malo bolj nazorno nakažemo tudi delovanje seznama že dodanih vlog (z uporabo knjižnice *Displaytag*). Lahko bi sicer imeli model, kjer sta strani za urejanje ter pregledovanje že dodanih vlog ločeni, vendar se izkaže, da je taka kombinirana verzija obrazca uporabnikom precej prijazna, saj so rezultati urejanja oz. dodajana vidni takoj. Poleg dejanskih vnesenih vrednosti imamo na tabeli še tri zadeve. To sta, podobno kot pri ostalih obrazcih, gumba za urejanje ter brisanje že obstoječih vrednosti, imamo pa tudi gumb za dodajanje atributov že obstoječim vlogam, kar pa nas pripelje na naslednji obrazec, ki je opisan v naslednjem podpoglavju.

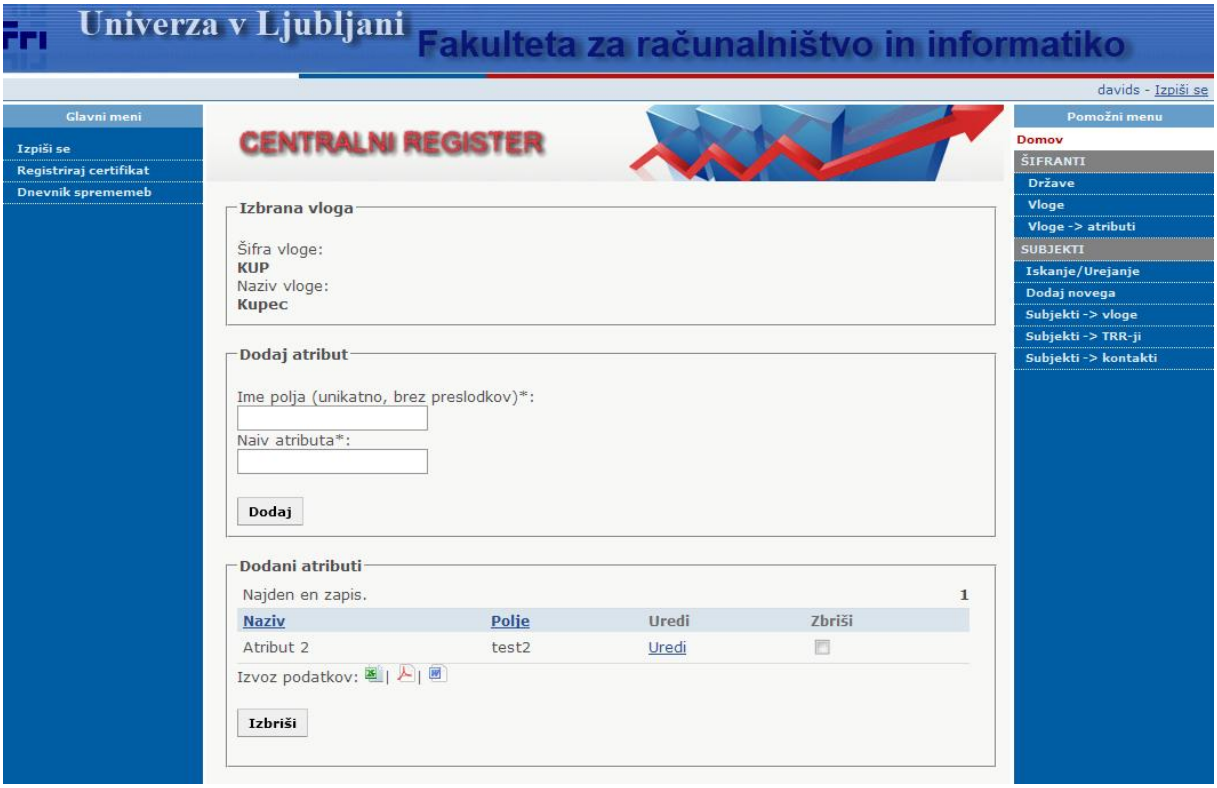


Slika 28: Urejanje/dodajanje vlog

6.7 Dodajanje/urejanje atributom vlogam

Obrazec omogoča določanje dodatnih atributnih polj za določeno vlogo (Slika 29). Vrednosti teh atributov bomo lahko nato določili pri urejanju subjektov. Ime polja je unikatno ime atributa, zato niso dopuščeni duplikati. Ta obrazec izvira iz uporabniških zahtev, ki pravijo, da subjektu glede na vlogo pripadajo dodatni atributi. To pomeni, da so ti atributi odvisni od vloge oziroma se nanjo nanašajo, zato je ta obrazec pravzaprav nadaljevanje prejšnjega obrazca za urejanje vlog, kjer izberemo vlogo, za katero želimo dodajati oziroma urejati atribute.

Zopet vršimo kontrolo nad vhodnimi podatki, predvsem pa poleg že standardne kontrole o obveznosti parametra tukaj dodamo še kontrolo za preverjanje imena polja. Ime polja namreč identificira ta dodatni atribut v podatkovni bazi in se uporablja tudi na obrazcu za urejanje subjektov. Če aplikacija že obstoječe ime polja najde v podatkovni bazi, nas na to tudi opozori. Ker moramo podpreti polja različnih tipov je ta dodaten atribut kasneje predstavljen kot navadno tekstovno vnosno polje.



Slika 29: Dodajanje/urejanje atributov za vloge

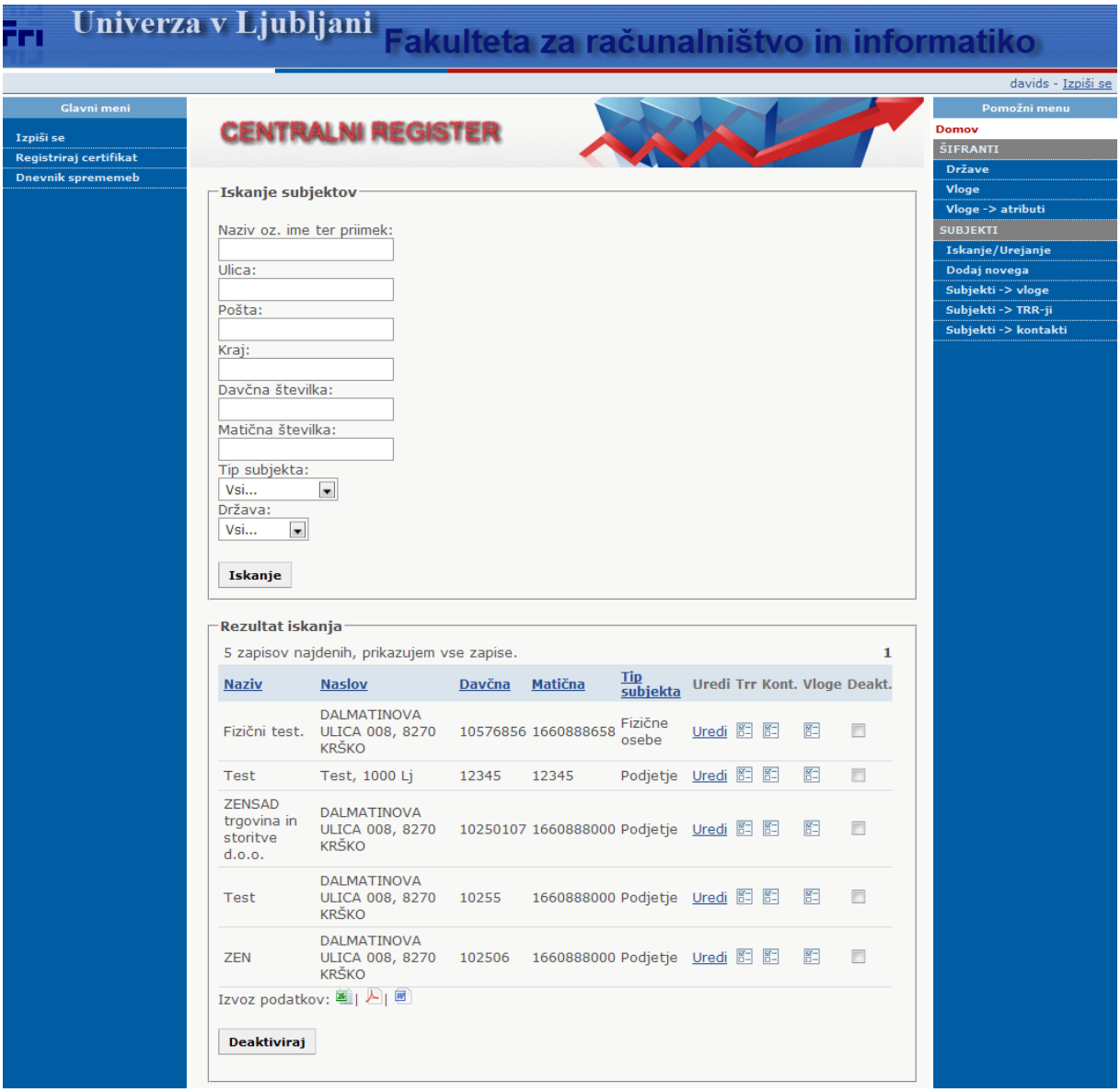
6.8 Iskanje po subjektih

Obrazec za iskanje po subjektih (Slika 30) omogoča klasično iskanje po več kriterijih, ki pa so neobvezni. Privzeto se ob odprtju obrazca išče po vseh subjektih, razen če ne določimo kriterijev. Iskanje po kriterijih ne razlikuje med velikimi in malimi črkami, prav tako pa išče samo po delih besede.

Obrazec nam služi tudi za dostop do drugih podobrazcev, ki nam omogočajo delo s subjekti. Kot ostali obrazci tudi ta omogoča izvoz v več formatov. Ta obrazec je torej nek vstopni obrazec za obdelovanje subjektov, preko katerega po izbiri že obstoječega subjekta lahko nadaljujemo z urejanjem, brisanjem, dodajanjem tekočih računov ter kontaktov.

Poleg tega pa ima obrazec kot tak tudi pomembno samostojno vlogo, saj je iskanje po veliki količini podatkov v vseh aplikacijah kritičnega pomena, seveda to še posebej velja za aplikacije, ki podatke prebirajo iz podatkovnih baz. Načelo iskanja je, da se čim manj operacij

izvede na strani poslovne logike, oziroma da poskušamo celotni filter za iskanje izvesti že na sami podatkovni bazi ob izvajanju poizvedb.

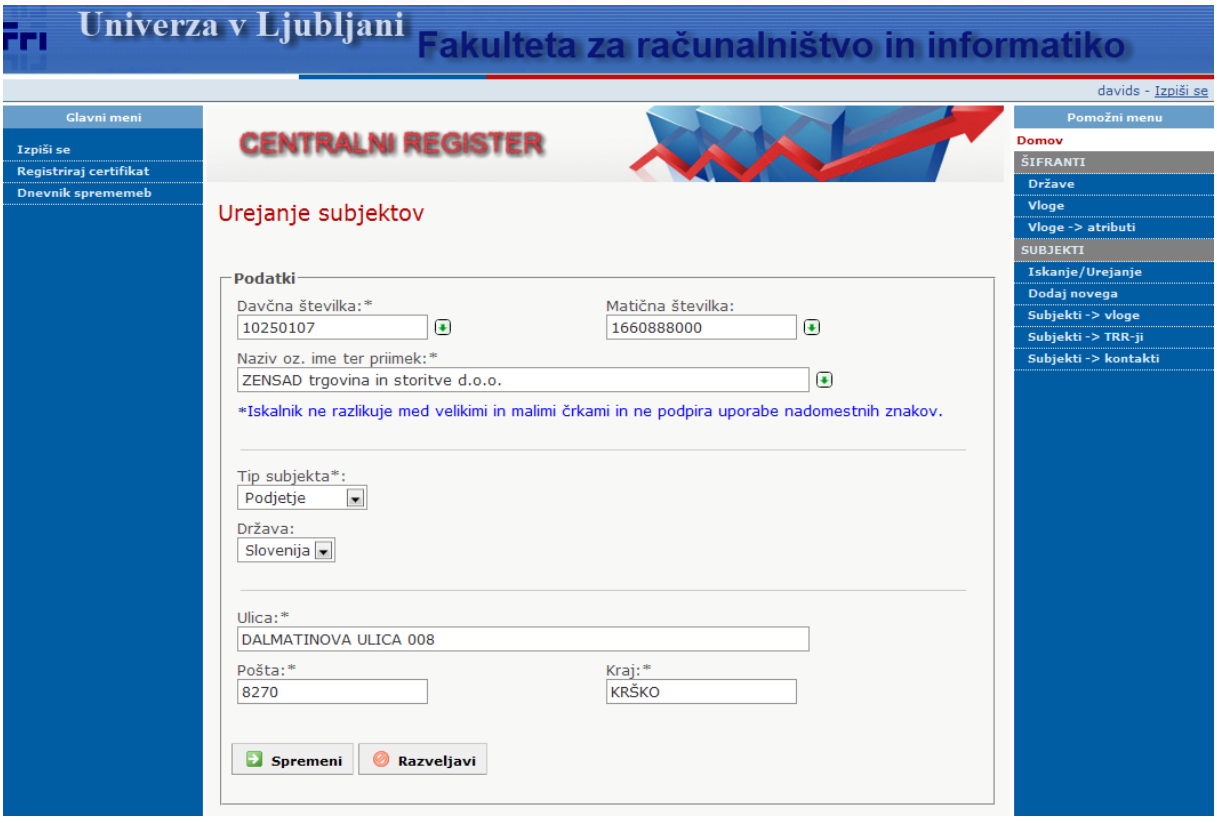


Slika 30: Iskanje po subjektih

6.9 Urejanje ter dodajanje subjektov

To je osrednji obrazec aplikacije (Slika 31), ki nam omogoča dodajanje oz. urejanje subjektov. Obrazec uporablja povezavo do *AJPES*-a, kar nam omogoča, da se lahko večino polj napolni avtomatsko preko klica spletnega servisa. Iščemo lahko po davčni številki, matični številki ter nazivu podjetja.

Poleg povezave z *AJPES-om* omogoča tudi klasično ročno urejanje/dodajanje subjektov, določanje tipa subjekta in države ter tudi določanje vrednosti dodatnim atributom, ki so določeni z dodeljenimi vlogami. Tudi pri ročnem dodajanju se izvedejo nekatere kontrole, na primer za pravne subjekte se preveri, ali davčna obstaja (preko povezave z *AJPES-om*), če podjetje s to davčno že obstaja v bazi. Za fizične osebe pa se podobno preveri, ali ta oseba že obstaja v registru. Večina omenjenih kontrol izhaja seveda iz uporabniških zahtev.



Slika 31: Dodajanje in urejanje subjektov

Centralni del tega obrazca je seveda povezava na *AJPES* preko njihove spletne storitve [11], katere tehnične podrobnosti in implementacija so bolje vidne iz same izvorne kode in tudi uporabljenih virov, zato tu ne bomo zašli v podrobnosti. Na kratko bomo le pokazali, kako obrazec deluje in kako se ga uporablja. Pri obrazcu je vseeno, ali gre za nov ali obstoječ objekt, razlika se pokaže šele ob shranjevanju sprememb, ki se nato v podatkovno bazo shranijo kot nov zapis ali pa se samo obstoječi zapis popravi.

Preko *AJPES* servisa lahko iščemo po treh parametrih (možno je seveda tudi po drugih, le da ti niso vključeni v aplikacijo). To so davčna številka, matična številka in pa naziv podjetja,

oziroma ime in priimek, če gre za samostojnega podjetnika. Pri davčni in matični številki gre predvsem za iskanje vrste je zadetek/ni zadetka, kar pomeni da servis ne išče po korenu številke. To iskanje je najbolj primerno takrat, ko že imamo na voljo ta podatek in bi radi preko njega pridobili še ostale podatke o pravni osebi. Primer iskanja po teh dveh parametrov je prikazan spodaj (Slika 32).

Slika 32: Primer iskanja po davčni številki

Obrazec nam vrne rezultat našega iskanja, v tem primeru seveda natanko en zadetek. Ko zadetek izberemo s klikom nanj se vsa polja avtomatsko napolnijo, kar je tudi namen pridobivanja podatkov preko tega servisa. Če pa iščemo subjekte po imenu, pa je možno tudi iskanje po korenu besede, tako da nam aplikacija ustrezno vrne več rezultatov, med katerimi lahko izbiramo. Servis omogoča vračanje največ stotih (100) zadetkov. V primeru, da je zadetkov več kot 100, nas aplikacija na to opozori in v takem primeru moramo iskati z drugačnimi parametri, v primeru da želenega subjekta ni med zadetku. Na spodnji sliki (Slika 33) lahko vidimo, da iskanje po besedi »dars« vrne 6 rezultatov, med katerimi so tudi subjekti, ki imajo besedo »dars« v korenu imena. Podobno kot v prejšnjem primeru se podatki samodejni izpolnijo po kliku na želen rezultat.

Slika 33: Iskanje subjekta po imenu

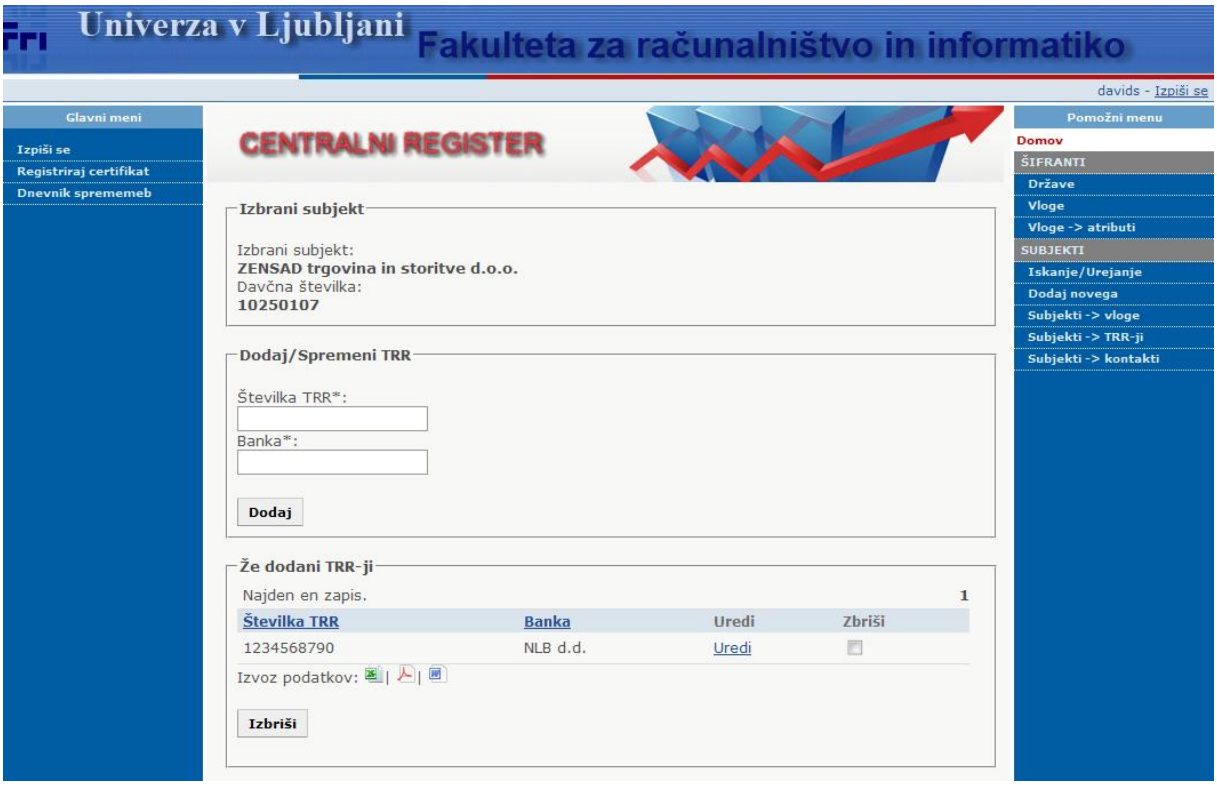
Poseben primer so subjekti, kateri so že bili vneseni v naš sistem in katerim smo kasneje dodelili neko vlogo. V tem primeru ima lahko subjekt dodatne atribute, katerim lahko na tem mestu določimo vrednost. Za primer vzemimo nek subjekt, ki ima vlogo zaposlenega v podjetju (je torej fizična oseba). Ko tako osebo urejamo, so nam vidni tudi vsi dodatni atributi, ki sovpadajo s posamezno vlogo, v spodnjem primeru (Slika 34) je ta dodatni atribut delovna doba, ki mu lahko podamo neko vrednost.

Slika 34: Primer izpolnjevanja dodatnih atributov subjektom

6.10 Dodajanje/urejanje tekočih računov za subjekte

Obrazec za dodajanje in urejanje tekočih računov za subjekte (Slika 35) nam omogoča, kot je razvidno iz imena, dodajanje in urejanje tekočih računov za subjekte. Vsak subjekt ima lahko več tekočih računov. Podobno kot na ostalih obrazcih imamo tudi tu preprosto kontrolo, ki preverja ali so vsi podatki, ki so obvezni, tudi pravilno vneseni. Obvezna podatka na tem obrazcu sta številka tekočega računa ter banka, pri kateri je ta račun odprt. Kot pri prejšnjih seznamih lahko tudi tu že vnesene podatke urejamo, dodajamo ali pa brišemo.

Obrazec izhaja iz uporabniških zahtev, vendar ni težko videti, da neko podjetje, ki vodi tak seznam partnerjev potrebuje tudi podatke o njihovih tekočih računih za nemoteno poslovanje. Kompleksnejši in namenski programi za vodenje poslovanja tu omogočajo še nemalo ostalih možnosti, a mi se moramo zaradi bistva te diplomske omejiti zgolj na neko podmnožico te funkcionalnosti.

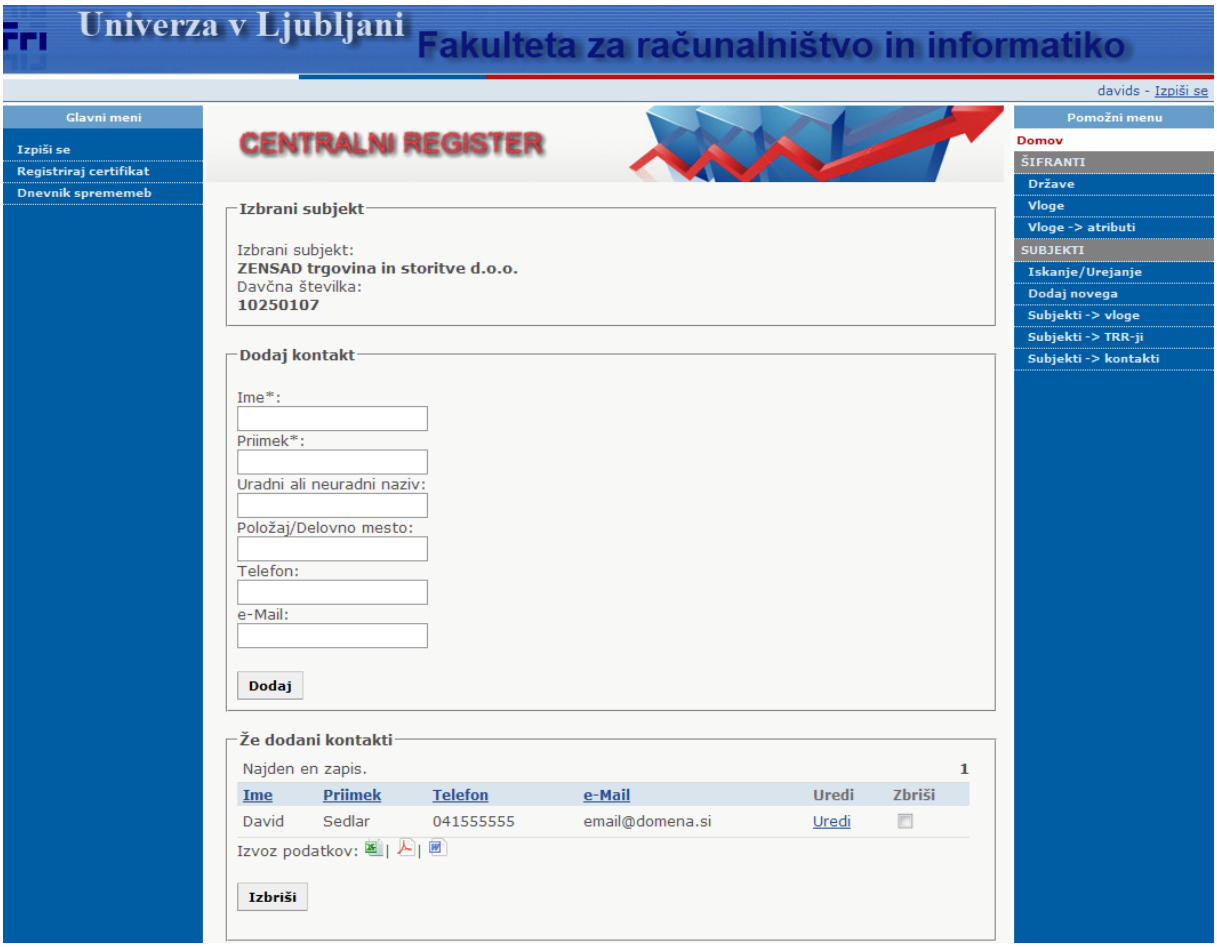


Slika 35: Dodajanje/urejanje tekočih računov za subjekte

6.11 Dodajanje/urejanje kontaktov subjektom

Obrazec spodaj (Slika 36) nam omogoča, da za nek subjekt določimo več kontaktov. Obvezni polji sta ime ter priimek, določimo pa lahko tudi opcijska polja, kot so uradni naziv, položaj/delovno mesto, e-mail, telefonska številka. Sam obrazec po funkcionalnosti ne izstopa kaj dosti od že prejšnjih, zato tukaj zadostuje samo kratek opis. Kot pri prejšnjih seznamih lahko tudi tu že vnesene podatke urejamo, dodajamo ali pa brišemo, lahko pa jih tudi izvozimo kot seznam.

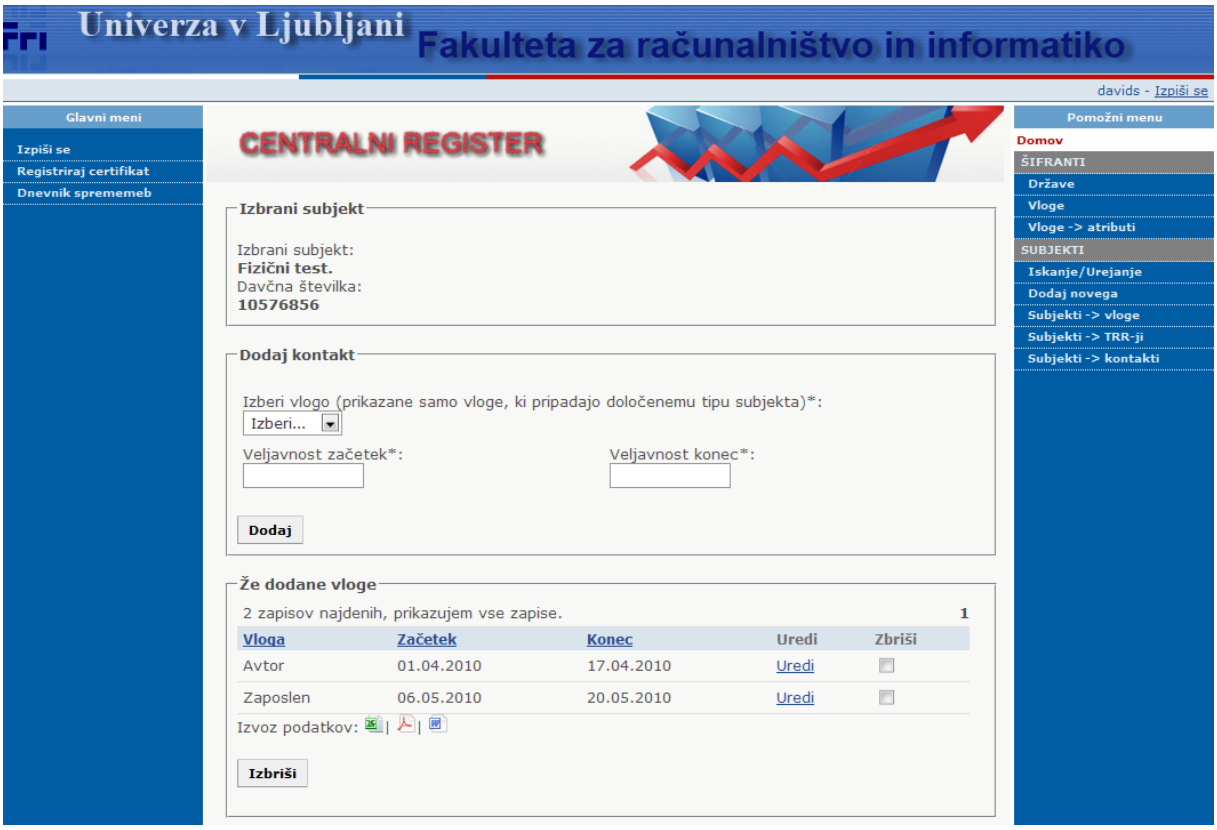
Ena izmed možnih izboljšav na tem mestu bi bila, da se kontakti ali pa zakoniti zastopniki nekega podjetja avtomatsko prenesejo iz *AJPES* spletne storitve, tako bi si prihranili nekaj dela pri vnašanju le teh, vendar je seveda potrebno tudi pretehtati čas, ki bi bil potreben za implementacijo tega.



Slika 36: Urejanje/dodajanje kontaktov za subjekt

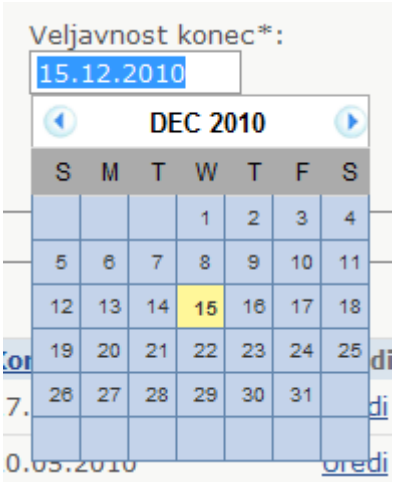
6.12 Urejanje/dodajanje vlog subjektov

Obrazec (Slika 37) nam omogoča, da subjektom dodelimo eno ali več vlog, k s mo jih že prej vnesli s pomočjo obrazca za vnos vlog. Uporabniku lahko določimo samo vloge, ki sovpadajo s tipom subjekta (fizične osebe ali podjetja). Se pravi, podjetju na primer ne moremo dodeliti vloge »Zaposleni«, medtem ko fizični osebi ne moremo dodeliti vloge »Dobavitelj«. To je prav tako ena od uporabniških zahtev, ki smo jo morali pri implementaciji upoštevati. Vloge so pomembne za določanje dodatnih atributov, katerim lahko pri urejanju subjektov dodelimo vrednosti, pa tudi zaradi preglednosti in kategorizacije subjektov. V nadaljnjem razvoju bi se tako lahko na primer odločili, da za posamezno vlogo dodamo nek nov del aplikacije, ki pa za ostale vloge ne bi bil aktualen (na primer urejanje bonitet pravnih oseb).



Slika 37: Dodajanje/urejanje vlog subjektom

Vlogam lahko določimo začetni ali končni datum, pri vnosu nam je v pomoč vgrajeni koledar, ki se prikaže ob pomiku v polje (Slika 38). Pri fizičnih subjektih se izvede tudi preverjanje, ali katera vloga časovno sovpada z že prejšnjimi in aplikacija nas v takem primeru na to opozori. Pri poslovnih subjektih se kontrola ne izvaja. Tudi ta kontrola izhaja iz uporabniških zahtev, je pa tudi smiselna, saj fizična oseba (na primer zaposleni v podjetju) redkokdaj nastopa v več kot eni vlogi.



Slika 38: Primer vgrajena koledarja za pomoč pri vnosu

6.13 Nudenje podatkov registra ostalim aplikacijam

To je eden izmed aspektov te spletne aplikacije, ki jo naredi zelo uporabno tudi za delo z ostalimi aplikacijami oziroma za bolj avtomatizirane postopke. Namreč podjetje mora stremeti k temu, da svoje procese čim bolj optimizira in s tem tudi avtomatizira. Kar pa se tiče samega scenarija te storitve, je ta malo bolj abstrakten, ker nimamo nobenega obrazca, da bi si bralec to lažje predstavljal. Navaden uporabnik bo na glavni strani aplikacije videl naslednje podatke (Slika 39).

Slika 39: Podatki za dostop do spletne storitve nudenja podatkov

Ti podatki so potrebni zato, da lahko druga aplikacija uporablja našo storitev. Če prvi naslov odpremo z brskalnikom, nam to ne da želenega rezultata, ampak le skoraj prazno stran z nekim opisom napake (Slika 40), ki pa uporabniku ne pove dosti, če že prej ne ve, da gre za spletno storitev.

Slika 40: Primer sporočila, ki ga dobimo ob odprtju v brskalniku

To je seveda narejeno namensko, kajti za komunikacijo s spletno storitvijo moramo komunicirati s spletno storitvijo preko protokola *SOAP*, ta pa predvideva za komunikacijo obliko sporočila v *XML* formatu. Šele ko storitvi v pravilni obliki pošljemo zahtevo, dobimo od nje tudi ustrezen odgovor. Če pa odpremo z brskalnikom drugi naslov, ki vsebuje *WSDL* dokument, dobimo prav tako odgovor v *XML* obliki, delček tega je prikazan spodaj (Slika 41).

```
<portType name="RegisterService">
    <operation name="vrniSubjekte" parameterOrder="arg0 arg1 arg2 arg3 arg4 arg5">
        <input message="tns:vrniSubjekte"></input>
        <output message="tns:vrniSubjekteResponse"></output>
    </operation>
    <operation name="vrniVseSubjekte">
        <input message="tns:vrniVseSubjekte"></input>
        <output message="tns:vrniVseSubjekteResponse"></output>
    </operation>
</portType>
```

Slika 41: Izsek iz dokumenta WSDL, ki opisuje spletno storitev

Za obdelavo tega dokumenta obstaja precej orodij in vseh različicah programskih jezikov, zato navadno interpretacijo prepustimo tem orodjem. Namreč, ko spletna storitev postane obširna in z veliko možnimi operacijami, postane tak dokument neberljiv za nekoga, ki poskuša podrobnosti razbrati ročno. Iz zgornjega izseka lahko na primer bralec razbere, da spletna storitev omogoča dve operaciji, in sicer:

- operacija vrniSubjekte, ki ima 6 vhodnih parametrov in nam omogoča iskanje po subjektih
- operacija vrniVseSubjkete, ki nima vhodnih parametrov in vrača vse subjekte v podjetju

7 Zaključek in sklepne ugotovitve

V zaključku bi se radi posvetili pregledu narejenega in dodali še kak komentar na vse skupaj. Izhajali bomo kar iz cilja, ki smo si ga zadali na začetku. Ta je bil zgraditi neko fleksibilno ogrodje, sestavljeno iz več manjših celot ali komponent, ob tem pa tudi predstaviti bralcu eno izmed mnogih alternativ, ki jih lahko uporabimo za gradnjo centraliziranih informacijskih sistemov. Ob tem smo za potrebe lažjega razumevanja in tudi kasnejšega lažjega nadgrajevanja razvili tudi preprost spletni informacijski sistem, preko katerega smo skušali bralcu približati vse uporabljene tehnologije.

Naše mnenje je, da smo vsaj v veliki večini uspeli izpolniti te zastavljene cilje in odpraviti nekatere probleme, ki izhajajo iz same motivacije, rezultat pa je informacijski sistem, ki ga lahko na kratko povzamemo z naslednjimi lastnostmi:

- Aplikacija je spletne narave in kot taka omogoča namestitev na enem računalniku, dostop pa je možen več odjemalcem
- Aplikacija je grajena v programskem jeziku Java (1.6), kompatibilna pa je tudi s prejšnjo verzijo 1.5 (kjer pa so potrebne dodatne knjižnice za uporabo spletne storitve, *JAX-WS RI* 2.1.x).
- Zasnova je neodvisna od platforme, aplikacija pa preizkušeno deluje na aplikacijskem strežniku *Apache Tomcat* verzije 6.0 ali novejših, neuradno pa tudi na *Oracle AS* verzije 10gR3 in novejših ter ostali podobni *Java Servlet* strežniki, ki podpirajo *JRE* 1.6.
- Zaščita z uporabo uporabniških imen in gesel ali pa z uporabo certifikatov ter podpora več uporabnikom
- Aplikacija omogoča varno povezavo od uporabnika do strežnika z uporabo certifikatov ter *SSL*
- Implementacija spletnih storitev za dostop do *AJPES* registra, vsi parametri za dostop se lahko poljubno nastavijo
- Postopek za preverjanje in razčlembo prejetih podatkov iz *AJPES*
- Avtomatska pretvorba baznih podatkov iz registra v podatke, ki so kompatibilni za dostop preko spletne storitve.
- Preko spletne storitve je omogočen dostop do vseh subjektov ali pa preko kriterijev do podmnožice.
- Aplikacija je precej fleksibilna in omogoča nastavitev večine parametrov, na primer podatke za povezovanje, podatke o certifikatih in podobno, nastavljiv je tudi mehanizem za integracijo s podatkovno bazo preko povezave *JDBC* in *Hibernate* ogrodja.

Omenili bi še nekaj malenkosti s področja prispevka, ki ga ta diplomska naloga daje bralcu in tudi sami znanosti. Kot bolj praktično usmerjena naloga že v začetku ni bil cilj izumiti nekaj

novega ali nekaj, kar bi se v znanosti informacijske tehnologije smatralo kot inovativno. Dandanes je zaradi vse večje razširjenosti tehnologij in globalizacije tudi vedno težje priti do idej, ki bile inovativne, vseeno pa se v znanosti dogajajo premiki, le da morda niso tako vidni, kot je na primer viden nek pomemben dosežek v gradbeništvu, ki je kot neka fizična zgradba vsem na očeh. Prispevki, ki jih daje ta naloga izhajajo že iz samih motivacij, ki smo jih opisali na začetku. Ena od omenjenih stvari je bila, da so nekatere tehnologije premalo ali pa tudi preveč dokumentirane in jih zato razvijalci s težavo uporabijo v svojih sistemih. Vsaj del tega bremena smo v tej nalogi rešili, se pravi s primerom implementacije in tudi dokumentiranim pristopom smo morebitnemu razvijalcu dali možnost, da naš sistem poljubno nadgrajuje, brez da bi moral pred tem iti skozi vse tegobe, ki navadno pripadajo razvoju novega sistema. To še ne pomeni, da tega znanja ne bo potreboval, je pa precej zoženo področje, iz katerega rabi te informacije jemati.

Drugi prispevek, ki prav tako izhaja iz prvotne motivacije pa je dejstvo, da se je z izdelavo te naloge vsaj nek majhen del znanja prenesel tudi iz smeri gospodarstva v bolj teoretične in akademske vode. Vsekakor primanjkuje teh prispevkov, še bolj pa primanjkuje volje na obeh straneh. Po eni strani imamo v praksi veliko uspešnih projektov in rešitev, ki bi s prenosom na akademski svet postali tudi učni primeri za bodoče inženirje. Po drugi strani pa se tudi v svetu teorije pojavljajo zelo dobre rešitve, ki bi lahko zelo pripomogle k reševanju pravih problemov, a ponavadi ne pridejo niti do faze implementacije. V tem smislu je torej ta naloga dala možnost preliva nekaterih znanj, če pa bo stvar na koncu res izkoriščena pa je težko vedeti. Dejstvo namreč je, da se diplomske naloge ponavadi ne smatra kot neke dosežke znanosti in velikokrat ostanejo samo na policah. Dobra stvar digitalne dobe pa je, da se sedaj tudi arhivi diplomskih nalog počasi premikajo v smer, da so na voljo vedno več ljudem, na primer preko interneta.

Omenimo pa še nekaj glede možnih izboljšav in predlogov, ki bi jih lahko implementirali. Ena izmed teh bi bila na primer boljša izraba povezave na sistem *AJPES*. Trenutno se povezava na ta sistem uporablja več ali manj izključno za pridobitev osnovnih podatkov o nekem poslovnem subjektu. To bi lahko razširili tako, da bi poleg tega pridobivali tudi ostale podatke, kot so na primer podatke o družbenikih in zastopnikih, podatke o strukturi podjetja in hčerinskih podjetij, podatke o bilančnih stanjih in podobno.

Druga možnost za izboljšavo se kaže v tem, da trenutno aplikacija daje podatke na voljo ostalim aplikacijam preko navadne spletne storitve po protokolu *SOAP*. To bi lahko razširili tako, da bi bil možen dostop veliko večji množici aplikacij kot sedaj. Dodali smo sicer že nekatere izvoze podatkov, vendar so ti omejeni na posamezne obrazce in tabele. Te izvoze bi lahko tako razširili tudi na izvoz celotnih podatkov o subjektih. Tudi te izpisi, ki jih generiramo, so precej enostavni, tako da bi jih lahko z uporabo ustreznih dodatnih knjižnic izboljšali, kar bi nam dalo tudi možnost, da izpise oblikujemo v bolj pregledno obliko, kot je sedaj (tabelarična). Še zadnja pripomba glede dostopa podatkov pa je varnost dostopa. Trenutno je zaščita omogočena samo za fizične uporabnike, ki dostopajo do aplikacije, to funkcionalnost bi bilo potrebno razširiti tudi za dostope drugih aplikacij preko spletne storitve, na primer preko preverjanja *IP* naslovov ali pa certifikatov aplikacij.

Kazalo slik

Literatura in viri

[1] Sinan Si Alhir. (2000) Understanding Use Case Modeling. Dostopno na: http://www.methodsandtools.com/archive/archive.php?id=24

[2] Gary Anthes. (2001) The Learning Curve. Dostopno na: http://www.computerworld.com/s/article/61762/The_Learning_Curve

[3] Apache Software Foundation. (2010) Apache Tomcat 6.0, SSL Configuration HOW-TO. Dostopno na: http://tomcat.apache.org/tomcat-6.0-doc/ssl-howto.html

[4] Stephanie Bodoff. (2002) Java Servlet Technology. Dostopno na: http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html

[5] Michael Cymerman. (2000) Automate your build process using Java and Ant. Dostopno na: http://www.javaworld.com/javaworld/jw-10-2000/jw-1020-ant.html

[6] Ian Jacobs. (2004) URIs, Addressability, and the use of HTTP GET and POST. Dostopno na: http://www.w3.org/2001/tag/doc/whenToUseGet.html

[7] Java-sources.net. (2010) Open Source Software in Java. Dostopno na: http://java-source.net/

[8] Marjan Krisper, *EMRIS: Enotna Metodologija Razvoja Informacijskih, 4. zvezek*. Ljubljana: Založba FRI, 2000.

[9] Reenskaug, Trygve. (2008) MVC XEROX PARC 1978-79. Dostopno na: http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html

[10] Ministrstvo za javno upravo RS. (2003) Profili kvalificiranih digitalnih potrdil in registra preklicanih potrdil SIGEN-CA in SIGOV-CA. Dostopno na: http://www.si-ca.si/dokumenti/priporocila-Digitalna-potrdila-v1.1.pdf

[11] Agencija RS za javnopravne evidence in storitve. (2010) wsPrsInfo Opis servisa za razvijalce, verzija 0.7. Dostopno na: http://www.ajpes.si/doc/AJPES/Za_razvijalce/wsPRSInfo_Opis_servisa_za_razvijalce.doc

[12] Sun Microsystems, Oracle. (2002) Java BluePrints, Model-View-Controller. Dostopno na: http://www.oracle.com/technetwork/java/mvc-detailed-136062.html

[13] W3C. (2009) W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. Dostopno na: http://www.w3.org/TR/xmlschema11-1/