Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Domen Čebulj

# Razvoj večslojnega informacijskega sistema z uporabo Java EE in WildFly na primeru gasilskega društva

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: prof. dr. Matjaž Branko Jurič

Ljubljana, 2013

*Besedilo je oblikovano z urejevalnikom besedil LaTeX.*

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

*Tržaška 25*
*1001 Ljubljana, Slovenija*
*telefon: 01 476 84 11*
*01 476 83 87*
*faks: 01 426 46 47*
*01 476 87 11*
*www.fri.uni-lj.si*
*e-mail: dekanat@fri.uni-lj.si*

Št. naloge: 00071 / 2013
Datum: 4.4.2013

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

Kandidat: **DOMEN ČEBULJ**

Naslov: **RAZVOJ VEČSLOJNEGA INFORMACIJSKEGA SISTEMA Z UPORABO JAVA EE IN WILDFLY NA PRIMERU GASILSKEGA DRUŠTVA**
**MULTI-TIER INFORMATION SYSTEM DEVELOPMENT USING JAVA EE AND WILDFLY ON A FIRE DEPARTMENT USE CASE**

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

Analizirajte platformo Java EE in identificirajte ključne tehnologije za razvoj večslojnih informacijskih sistemov. Identificirajte aplikacijske strežnike ter se spoznajte s strežnikom WildFly. Proučite tehnologije JDBC, JPA, JavaMail, Servlet, JSP, JSF, EJB, JMS in JAAS ter definirajte arhitekturo sistema. Pri tem uporabite vzorec MVC. Opišite ključne vidike implementacije informacijskega sistema na primeru gasilskega društva.

Mentor:

prof. dr. Matjaž B. Jurič

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

# Izjava o avtorstvu diplomskega dela

Spodaj podpisani Domen Čebulj, z vpisno številko **63100193**, sem avtor diplomskega dela z naslovom:

*Razvoj večslojnega informacijskega sistema z uporabo Java EE in WildFly na primeru gasilskega društva*

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Matjaža Branka Juriča,
- 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 2. decembra 2013 Podpis avtorja:

*Anga une.*

*Zahvaljujem se mentorju prof. dr. Matjažu Branku Juriču, ki me je sprejel pod svoje mentorstvo ter mi dajal koristne napotke in nasvete pri izdelavi diplomskega dela. Zahvaljujem se podjetju Četrta pot, ki mi je pomagala pri študiju. Prav tako se zahvaljujem g. Marku Bogataju, univ. dipl. lit. komp., ki je lektoriral moje diplomsko delo. Posebna zahvala gre mojim staršem, sestri Ani, punci Alenki ter prijateljem, ki so me tekom študija moralno podpirali in mi pomagali.*

# Kazalo

# Povzetek

V diplomskem delu so predstavljene metode in potek dela za izdelavo informacijskega sistema, ki služi za pomoč pri vodenju administrativnih zadev, katere zajemajo področje pošte v gasilskih društvih. Za izdelavo aplikacije smo pripravili načrt informacijskega sistema, zaključili pa z implementacijo v programskem jeziku Java EE. Poleg pripadajočih poslovnih komponent smo uporabili dodatne knjižnice, podatkovno bazo MySQL in aplikacijski strežnik WildFly. Nastali sistem omogoča registracijo in prijavo uporabnikov, dodajanje ter povezovanje društev med več uporabniki ter administracijo uporabnikov in dodelovanje vlog. Aplikacija tako nudi možnost vodenja prispele in poslane pošte pri društvih, kategorizacijo pošte, vodenje podjetij in povezovanje več uporabnikov med društvi ter izdelavo končnega letnega seznama pošte. Aplikacija je bila narejena z namenom nudenja lažjega in učinkovitejšega vodenja društvenega delovodnika, s katerim nadomestimo že zastareli ročni vnos pošte.

## Ključne besede

*Java EE, WildFly, JSF, JPA, MySQL, JasperSoft iReport, gasilci, vodenje administrativnih zadev*

# Abstract

The goal of this thesis is to develop information system for the mail administrative matters management in the fire department. To develop application we had prepared information system architecture and implement this architecture in programming language Java EE.We have used associated business components, additional libraries, MySQL database and application server WildFly. The information system enables user's registration and login, different fire departments management and connecting these fire departments to users, administration of the users and roles. The additional features of the information system are effective management of the incoming and outgoing mail, mail categorization, governance of the companies, collaboration of the users between different fire departments and creation of annual mail list. The application was developed with the goal of supporting easier and efficient management of working journal, which could replace the legacy manual mail management.

## Key words

*Java EE, WildFly, JSF, JPA, MySQL, JasperSoft iReport, firefighters, management of administrative matters*

# Poglavje 1

# Uvod

V Sloveniji je prostovoljno gasilstvo zelo razvito, kar nam pove tudi podatek, da vsako leto število gasilcev narašča [3]. Po podatkih naj bi bilo v Sloveniji trenutno okoli 150.000 gasilcev, od tega slabih 1000 poklicnih [2]. Poleg tega, da gasilci pomagajo pri gašenju požarov in drugih naravnih nesrečah, morajo navsezadnje upoštevati tudi zakone o gasilskih društvih, ki predvsem zahtevajo urejeno administracijo za vodenje le-teh.

V diplomskem delu je predstavljen informacijski sistem za pomoč pri vodenju administrativnih zadev, ki zajemajo področje pošte v gasilskih društvih. Pripravili smo načrt informacijskega sistema in končali z implementacijo v programskem jeziku Java EE. Uporabljene so bile tudi dodatne knjižnice in podatkovna baza MySQL ter aplikacijski strežnik WildFly.

Ker sem sam član Prostovoljnega gasilskega društva Olševek in zasedam mesto tajnika v upravnem odboru, dodobra poznam problematiko administrativnih zadev in ostale papirologije, ki doletevajo gasilska društva po celotni Sloveniji.

Ravno zato, ker pravila in zakoni velevajo, naj se vsa prejeta in poslana pošta - pisemska ali elektronska - vodi v tako imenovanem *delovodniku*, posebni "knjigi", kjer je zabeležen datum prejema/pošiljanja pošte, podjetje od katerega je pošta, oziroma komu je namenjena, kratka zadeva ter še nekatere druge manj pomembne zadeve, je bil naš namen izdelati sodobno elektronsko

spletno aplikacijo, s katero bi nadomestili stare zvezke in papirje in bi tako ta del preselili v območje elektronske tehnologije, kot je to storjeno že za vodenje evidenc članov in opreme (spletna aplikacija Vulkan) in za vodenje intervencij (spletna aplikacija Spin).

Osnovni cilj diplomskega dela je izdelati informacijski sistem, ki upošteva naslednje točke:

- **Spletna aplikacija**
  Sistem mora biti na spletu, kar omogoča dostop in uporabo v vsakem trenutku in kjerkoli.

- **En uporabnik, več društev**
  Omogočati mora, da lahko z enim uporabniškim računom vodimo več društev naenkrat.

- **Nabor modulov**
  Osnovni nabor modulov mora zadoščati vnosu in pregledu prispele in poslane pošte, kategorizacijo pošte, vodenje podjetij in izdelavo končnega poročila.

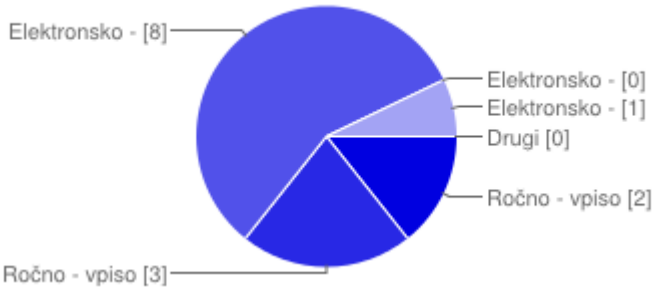
- **Prijazen vmesnik in enostaven za uporabo**
  Sistem mora imeti prijazen uporabniški vmesnik in mora biti enostaven za uporabo. Poleg tega mora omogočiti, da se uporabniki lahko kadarkoli obrnejo po pomoč administratorju aplikacije.

- **Brezplačna uporaba**
  Sistem mora biti brezplačen za uporabo, saj gasilska društva nimajo prilivov, s katerimi bi plačevali namensko programsko opremo.

Pred izdelavo in pisanjem diplomskega dela smo se odločili izvesti anketo o uporabi delovodnika v društvih Gasilske zveze Kokra. Prvi podatki so bili zanimivi, kot je razvidno iz slike 1.1, kjer dobrih 35 odstotkov (5 društev) uporablja še liste papirja oziroma knjigo za vpisovanje pošte. Zbrani podatki so nam bili v dodatno spodbudo za izdelavo elektronske namenske aplikacije,

**Na kakšen način vodite delovodnik v vašem društvu?**

| | | |
|---|---|---|
| Ročno - vpisovanje v zvezek, list papirja | **2** | 14 % |
| Ročno - vpisovanje v zato namenjeno knjigo | **3** | 21 % |
| Elektronsko - v preglednice (npr. MS Excel, OO/LO Calc, Lotus) | **8** | 57 % |
| Elektronsko - posebaj namenjene aplikacije | **0** | 0 % |
| Elektronsko - preko interneta (npr. Google Preglednica) | **1** | 7 % |
| Drugi | **0** | 0 % |




Slika 1.1: Odgovori na anketno vprašanje o uporabi delovodnika v društvih GZ Kokra.

ki bi olajšala in posodobila zastareli način vodenja delovodnika v gasilskih društvih.

V drugem poglavju diplomskega dela bomo opisali uporabljene tehnologije in programe, ki smo jih uporabili pri izdelavi in so nam bile v pomoč pri razvoju. Poleg tega poglavje opisuje tudi dodatne knjižnice. V tretjem poglavju bomo predstavili poslovne komponente, s katerimi smo se srečevali pri razvoju ter njihovo medsebojno povezanost. V četrtem poglavju je predstavljena implementacija in zanimivi vidiki razvoja sistema. Tu je opisan tudi posamezen del implementacije; od samega načrtovanja podatkovne baze do končnega izdelka pripravljenega za uporabo.

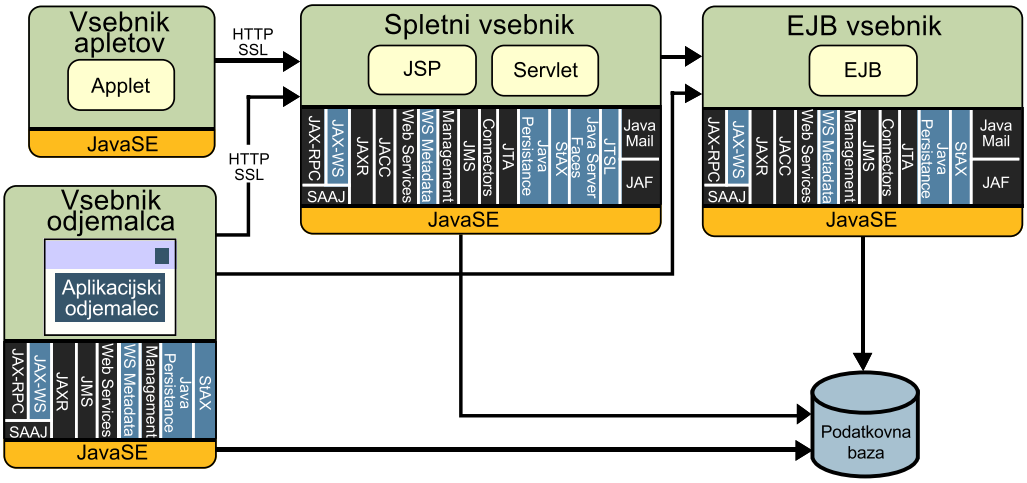
# Poglavje 2

# Opis uporabljenih tehnologij in programov

Za izdelavo katerekoli vrste aplikacij potrebujemo ustrezno programsko opremo in izbrane tehnologije. Slednje so na kratko opisane v tekočem poglavju, v katerem opišemo tudi programe, ki smo jih uporabili pri izdelavi.

## 2.1 Programski jezik Java

Java je sodoben, objektno usmerjen, prenosljiv programski jezik, ki ga je razvil James Gosling v podjetju Sun Microsystems leta 1995 [4]. Jezik Java je zelo razširjen, saj ga najdemo na nekaterih področjih našega življenja. Nahaja se denimo v vgrajenih napravah (ang. Embedded System), mobilnih telefonih (npr. Android) ter na spletnih strežnikih, kjer se poganja naprednejša verzija Jave, tj. Java EE (Java Enterprise Edition). Slednjo bomo uporabili kot primarni jezik za gradnjo naše aplikacije. Naj omenimo še, da je Java brezplačna za uporabo in prenosljiva med najbolj znanimi operacijskimi sistemi.




Slika 2.1: Vsebniki v Java EE.

### 2.1.1 Java EE

Java EE je razširitev standardne Jave SE (Java Standard Edition). Je skupek programskih paketov in knjižnic (ang. Application Programming Interface - API), ki omogočajo gradnjo večjih poslovno informacijskih rešitev, od omrežnih aplikacij do spletnih servisov. Z njeno pomočjo si olajšamo razvoj distribuiranih ter večnivojskih strežniških aplikacij, ki so odporne na odpoved posameznih komponent [8].

Java EE se tako deli na štiri vrste komponent, le-te pa vsebujejo vsaka svoj vsebnik (ang. Container), kjer se koda izvaja. Slika 2.1 (vir [9]) prikazuje omenjene komponente in pripadajoče vsebnike ter storitve, ki se izvajajo pod vsakim vsebnikom.

Kot zanimivost naj omenimo, da je v letošnjem letu izšla že sedma različica, ki nosi ime Java EE 7, medtem se pripravlja že osma različica, ki bo temeljila predvsem na aplikacijah v oblačni arhitekturi (ang. Cloud Computing).

## 2.2 Podatkovna baza MySQL

Za shranjevanje podatkov pri spletnih in namiznih aplikacijah potrebujemo ustrezno podatkovno bazo. Ena izmed brezplačnih in najbolj pogosto uporabljenih je MySQL podatkovna baza. MySQL je sistem za upravljanje s podatkovnimi bazami, ki temeljijo na relacijski podatkovni bazi s povpraševalnim jezikom SQL (Structured Query Language). Ravno tako je odprtokoden in prenosljiv med več operacijskimi sistemi ter je najbolj razširjen za preproste spletne aplikacije. Trenutno se nahaja v različici 5.6. in omogoča vse osnovne in naprednejše operacije. Z zadnjo verzijo je pridobil tudi hiarhične poizvedbe.

Po podatkih naj bi za svoje shranjevanje MySQL uporabljali tudi naslednja podjetja: Wikipedia, Facebook, Twitter, Flicker, YouTube in delno Google [7].

## 2.3 Aplikacijski strežniki

Če želimo našo aplikacijo, ki je napisana v Java EE, poganjati, potrebujemo ustrezni aplikacijski strežnik. Aplikacijski strežniki omogočajo izvajanje Java EE vsebnikov in njihovih storitev. Pri tem moramo paziti, da izberemo pravilnega oziroma, da je strežnik certificiran za poganjanje Java EE kot celote (ang. Full Certified) in ne samo spletnega vsebnika (ang. Web Certified).

Na izbiro imamo nekaj polno certificiranih aplikacijskih strežnikov:

- WebSphere Application Server
- WebLogic server
- TomEE
- WildFly/JBoss AS
- GlassFish

Eden izmed trenutno polno certificiranih aplikacijskih strežnikov za Java EE 7 je brezplačni Oraclov Glassfish (verzija 4.0) [5]. Poleg prej omenjenega aplikacijskega strežnika velja omeniti tudi JBoss AS 7, ki je med boljšimi brezplačnimi aplikacijskimi strežniki. Sedaj se je preimenoval v WildFly 8, ki je trenutno še v beta fazi. Ta naj bi bil drugi polno certificirani aplikacijski strežnik za Java EE 7. Uradno naj bi izšel konec novembra 2013.

### 2.3.1 WildFly

WildFly je aplikacijski strežnik za poganjanje vsebnikov jezika Java EE. Kot smo omenili, je trenutno še v beta različici, vendar je dovolj stabilen, da lahko sam razvoj premaknemo nanj. Omogoča vse Java EE 7 funkcionalnosti.

Spodaj so naštete nekatere izmed njih, ki jih bomo uporabili pri realizaciji naše aplikacije:

- Java Mail
- Java Servlet
- JSF - Java Server Faces
- JSP - Java Server Pages
- EJB - Enterprise Java Beans
- JPA - Java Persistence API
- JTA - Java Transaction API
- JMS - Java Message Service
- JDBC - Java Database Connectivity
- JNDI - Java Naming and Directory Interface
- JAAS - Java Authentication and Authorization Service

## 2.4 Označevalni jezik HTML in stilski jezik CSS

HTML (Hyper Text Markup Language) je označevalni jezik za opisovanje strukture in vsebine dokumentov na svetovnem spletu. Z njegovo pomočjo definiramo strukturo strani oziroma aplikacije za prikaz. Z zadnjo različico standarda, HTML 5 je pridobil nekaj izboljšav, denimo integracijo multimedije, slikarsko površino (ang. Canvas), več atributov za oblikovanje ter nove elemente za boljšo semantiko spleta.

Poleg oblikovanja s HTML poznamo tudi kaskadno stilsko predlogo (ang. Cascading Style Sheets - CSS), ki skrbi za način predstavitve HTML dokumenta. Z njeno pomočjo definiramo vizualni videz dokumentov spletnih aplikacij in drugih strani. Omogoča ločevanje dela razvijalca od oblikovalca ter enoten stil spletne strani oziroma aplikacije.

## 2.5 Java Script

Za dinamiko na spletni strani poskrbi Java Script, ki je namenjen dinamičnosti na spletni strani in se izvaja v brskalniku odjemalca. Pogosto se uporablja t.i. AJAX (Asynchronous Java Script and XML), ki je namenjen asinhroni komunikaciji med odjemalcem in strežnikom z XML (Extensible Markup Language) dokumenti. Znani paket za delo z Java Script-om je knjižnica jQuery, ki vsebuje AJAX klice, JSON (Java Script Object Notation) klice, različne funkcije, animacije ter druge funkcionalnosti, kot so preverjanje pravilnosti podatkov na strani odjemalca, asinhrono klicanje oddaljenega strežnika in obdelava podatkov.

## 2.6 Ostale knjižnice za javansko okolje

Pri samem programiranju aplikacij je pomembno, da čim manj odkrivamo "toplo vodo". Pri uporabi že napisanih knjižnic si pri tem prihranimo čas.

Paziti moramo le, da njihova licenca dovoljuje uporabo knjižnic za namen, kjer jo želimo uporabiti.

### 2.6.1 JasperSoft iReport

Orodje iReport podjetja JasperSoft je program namenjen izdelavi dinamičnih poročil in dokumentov v javanskem okolju. Z njegovo pomočjo lahko dinamično kreiramo dokumente, ki vsebujejo tabele, slike, dinamično besedilo, HTML besedilo, podporočila, grafe itn. Vse kar potrebujemo za njegovo delovanje so predloge poročil (kreirane z orodjem iReport), ustrezne podatke iz podatkovne baze ter javanske knjižnice. Na sliki 2.2 je prikazan postopek za izdelavo dinamičnega poročila.

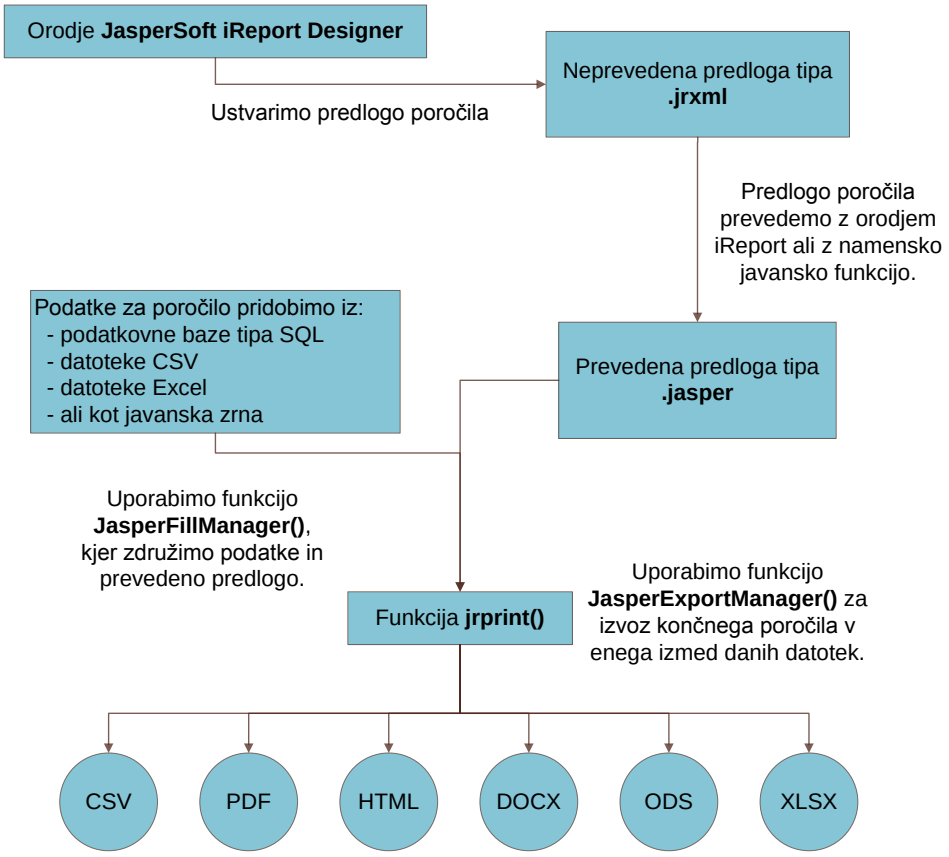
### 2.6.2 Primefaces

Za lažje prikazovanje podatkov v okolju JSF (Java Server Faces) bomo uporabili knjižnico Primefaces, saj bomo z njeno pomočjo pridobili nekaj naprednih komponent, predvsem za prikazovanje podatkov pri končnem odjemalcu. Knjižnica omogoča lažje delo z AJAX klici, odpravlja tudi nekatere slabosti JSF okolja, podpira HTML 5 in ima vgrajenih nekaj CSS predlog za prikaz. Poleg tega temelji na razširjeni Java Script knjižnici jQuery in jQuery UI.

## 2.7 Uporabljeni programi pri razvoju

Pri razvoju bomo uporabili brezplačna orodja. Z njimi si bomo pomagali razviti naš informacijski sistem od glave do nog. V največji meri bosta uporabljeni naslednji orodji:

- Eclipse
- MySQL Workbench




Slika 2.2: Prikaz ustvarjanja poročila.

### 2.7.1 Razvojno okolje Eclipse

Eclipse IDE (Integrated Development Environment) je integrirano razvojno okolje, ki omogoča razvoj aplikacij, gradnjo aplikacij (ang. Build), razhroščevanje kode (ang. Debuging) in še mnogo več. Pri tem nam je v veliko pomoč ponujanje predlaganih ukazov (ang. Auto Completion), obarvanje kode za lažje sledenje, odlično razhroščevanje kode itn. Samo orodje Eclipse se deli na več podvej - odvisno, kateri programski jezik izberemo. Za našo izdelavo bomo uporabili Eclipse IDE for Java EE Developers.

#### 2.7.1.1 Nadzor različice kode

Pri razvoju aplikacije bomo uporabljali program za nadzor različice programske opreme in sicer SVN (Apache Subversion). Z njegovo pomočjo bomo nadzirali kodo za nazaj, vedno pa bomo imeli pri roki rezervno kopijo prejšnje kode.

### 2.7.2 Orodje MySQL Workbench

V pomoč pri načrtovanju in implementaciji entitetno-relacijskega modela bomo uporabili orodje podjetja Oracle - MySQL Workbench. Z njegovo pomočjo bomo kreirali podatkovni model za našo aplikacijo ter jo sinhronizirano posodabljali neposredno na podatkovnem strežniku MySQL.

# Poglavje 3

# Arhitektura in načrt informacijskega sistema

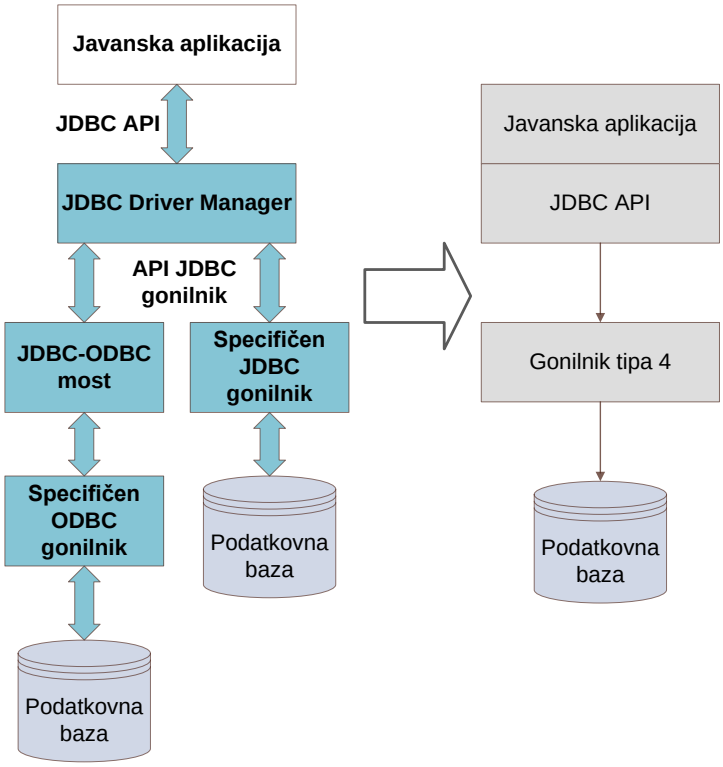
## 3.1 Opis poslovnih komponent

Večina programskih jezikov, ki so namenjeni izdelavi poslovnih aplikacij, je sestavljena iz poslovnih komponent, ki sodelujejo pri poganjanju celotnega informacijskega sistema. Pri naši implementaciji smo uporabili poslovne komponente, ki so naštete in na kratko opisane spodaj.

### 3.1.1 JDBC in JPA

JDBC (Java Database Connectivity) je standardna javanska knjižnica, ki omogoča delo z relacijskimi podatkovnimi bazami. Omogoča vzpostavljanje in zapiranje povezave na bazo, izvajanje SQL povpraševanj in vračanje strukturiranih podatkov. Nahaja se v Javi SE in je trenutno v verziji 4.0 [10]. Slika 3.1 prikazuje, kako je bil osnovno načrtovan JDBC ter kakšna je sedanja implementacija v verziji JDBC 4.0.

JDBC je predpomnilnik povezav na podatkovni bazi, ki jih upravlja upravljalec na aplikacijskem strežniku oziroma samostoječa aplikacija. Z njim zmanjšamo večkratno ustvarjanje povezave (operacije za vzpostavljanje so potratne), saj se dostop do povezave deli. Slabost verzije 4.0 je v tem, da




Slika 3.1: Primer osnovnega delovanja JDBC in JDBC 4.0.

potrebujemo za vsako verzijo podatkovnega strežnika svojo knjižnico za povezovanje.

Eden izmed drugih načinov dostopa do podatkovne baze je tudi JPA (Java Persistence API). JPA je ORM (Object Relational Mapping) ogrodje, ki preslikava navadne javanske razrede (ang. Plan Old Java Object - POJO) v relacije (tabele in stolpce) na podatkovnem strežniku. JPA ima nekaj prednosti [8]:

- Enostavnost pri upravljanju transakcij.
- Samodejno kreiranje razredov tabel.
- Preproste nastavitve.
- Ni potrebno pisanje osnovnih SQL stavkov.
- Boljša optimizacija in uporaba predpomnilnika.
- Avtomatizrana preslikava med objekti in relacijskimi tabelami.

Poznamo več ponudnikov JPA rešitev. Najbolj znani so *OpenJPA*, *Oracle TopLink*, *Hibernate* in *EclipseLink*.

Za ustrezno delovanje JPA v naši aplikaciji bomo morali pravilno nastaviti aplikacijski strežnik. V nastavitvah bomo nastavili strežnik, uporabniško ime ter geslo do podatkovnega strežnika in poimenovali povezavo, tako da bomo lahko preko storitve JNDI (Java Naming and Directory Interface) pridobili dani vir.

Kreirali bomo novi JPA projekt, v katerega bomo dodali datoteko *persistence.xml* in v njej določili JNDI vir do naše podatkovne povezave, ter razrede, ki se bodo samodejno prevedli v SQL transakcijo ob klicu podatkovnega strežnika.

```
// Vsebina nastavitev na spletnem strezniku - standalone.xml
...
```

```
<datasource jta="true"
    jndi-name="java:jboss/datasources/TajnikRazvojDS"
    pool-name="TajnikRazvojDS" enabled="true" use-ccm="true">
    <connection-url>jdbc:mysql://localhost:3306/baza</connection-url>
    <driver>mysql</driver>
    <security>
        <user-name>uporabnik</user-name>
        <password>geslo</password>
    </security>
    <statement>
        <prepared-statement-cache-size>100</prepared-statement-cache-size>
        <share-prepared-statements>true</share-prepared-statements>
    </statement>
</datasource>
...

// Datoteka persistence.xml
<persistence>
    <persistence-unit name="MeteorJPA" transaction-type="JTA">
        <jta-data-source>java:jboss/datasources/TajnikRazvojDS</jta-data-source>
        <class>eu.tajnik.jpa.entities.UserPrincipal</class>
        ...
    </persistence-unit>
</persistence>

// Datoteka UserPrincipal.java (POJO)
@Entity
@Table(name="UserPrincipal")
public class UserPrincipal implements Serializable {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private int                     id;
```

```
    @Column(name = "password")
        private String                password;

    @Column(name = "username")
    private String                    username;

    @Column(name = "status")
        @Enumerated(EnumType.STRING)
        private UserPrincipalStatusType status;

        @ManyToOne
        @JoinColumn(name = "selected\_unit\_id")
        private Unit                  unit;

        @ManyToOne
        @JoinColumn(name = "user\_id")
        private User                  user;

    // Ustrezne SET in GET metode.
    ...
}
```

Entitetni razred (primer *UserPrincipal*) je navaden javanski razred z dodatnimi anotacijami. Anotacija *@Entity* nad imenonom razreda pove, da gre za entiteto, ki se v bazi preslika v tabelo, navedeno v anotaciji *@Table(name="IME_TABELE")*. Vsaka entiteta mora imeti atribute, ki se v bazi preslikajo v stolpce. Atributi so označeni z anotacijami *@Id* (atribut predstavlja primarni ključ), *@Column* predstavlja stolpec, anotaciji *@ManyToOne* in *@JoinColumn* predstavljata tuji ključ med izbranim stolpcem in drugo entiteto v bazi (v našem primeru lastnost *unit* predstavlja entiteto *Unit*).

### 3.1.2 Java Mail API

Java Mail API je knjižnica programskih funkcij za pošiljanje in sprejemanje elektronskih sporočil iz okolja Java. Privzeto je knjižnica vključena v Java EE, lahko se uporablja kot dodatna knjižnica na okolju Java SE. Podpira tri najpogostejše protokole, in sicer SMTP, POP3 in IMAP [11].

Za pošiljanje (in prejemanje) pošte potrebujemo *sejo* (ang. Mail Session), ki jo najlažje pridobimo tako, da aplikacijski strežnik pravilno nastavimo. Pri nastavitvah moramo nastaviti uporabniško ime, geslo ter strežnik, kakor je prikazano spodaj. Sejo bomo pridobili z JNDI virom, ki ga nastavimo pri nastavitvah *mail-session*.

```
// standalone.xml
<mail-session jndi-name="java:jboss/mail/TajnikMail">
    <smtp-server ssl="true" outbound-socket-binding-ref="GmailSMTP">
        <login name="uporabnisko.ime" password="geslo"/>
    </smtp-server>
</mail-session>

<outbound-socket-binding name="GmailSMTP">
    <remote-destination host="smtp.gmail.com" port="465"/>
</outbound-socket-binding>

// Javansko zrno
@Resource(mappedName = "java:jboss/mail/TajnikMail")
private Session        mailSession;

public void sendEmail(Email email) {
    MimeMessage msg = new MimeMessage(mailSession);

    msg.setFrom(new InternetAddress(email.getSender()));
    msg.setSubject(email.getSubject());
    msg.setContent(email.getContent());
    msg.setSentDate(email.getSendDate());
```

```
    msg.setRecipients(RecipientType.TO, email.getRecipientTO());

    Transport.send(msg);
}
```

V programski kodi, kjer želimo dostopati do omenjene seje, preprosto pokličemo vir z anotacijo *@Resource*, ki nam vrne dekleriran vir s sejo. Sejo nato uporabimo pri pošiljanju (sprejemanju) elektronske pošte.

### 3.1.3 Servlet, JSP in JSF

*Servlet* je javanski razred, ki razširja zmožnosti strežnika. Tako lahko *servlet* odgovarja na katerokoli poslano zahtevo. Običajno se uporablja spletni strežnik in s tem zahteva HTTP (ang. HTTP Request) in odgovor HTTP (ang. HTTP Response). Sam *servlet* se izvaja v spletnem vsebniku na strani strežnika. Pri tem se uporablja podtip servleta *HTTPServlet*, ki omogoča dinamično kreiranje prikaza vsebine končnemu uporabniku. *HTTPServletRequest* vsebuje trenutno zahtevo uporabnika. V njej se običajno nahajajo piškotki, ki so del dane strani, aktivne seje in vsi ostali podatki o uporabniku in strežniku. Nasprotno pa *HTTPServletResponse* vsebuje odgovor na uporabnikovo zahtevo. Tipično so to novo nastavljeni piškotki, seje ali drugi podatki, ki jih želimo vrniti uporabniku.

Naprednejša tehnologija, ki temelji na servletih, se imenuje JSP (Java Server Pages). Prednosti JSP so v tem, da lažje urejamo statično in dinamično spletno vsebino in da se nahaja v svoji datoteki tipa *.jsp*. Tako je lažje kombinirati HTML kodo in CSS pravila, saj slednje ločimo od javanske kode, ki se še vedno nahaja v svojih razredih. Zavedati se moramo, da se ob prvi zahtevi strežniške strani le-ta prevede v *servlet*. Pri JSP se začne uporaba javanskih zrn (ang. Java Beans). To so navadni javanski razredi, ki vsebujejo zasebne lastnosti in ustrezne SET in GET metode. Razred mora še implementirati vmesnik *Serializable* ter imeti privzeti konstruktor. Znotraj JSP strani preko posebnih značk (<jsp:useBean>, <jsp:getProperty>,

<jsp:setProperty>) lahko nastavljamo lastnosti javanskih zrn in s tem naredimo prikaz podatkov še bolj dinamičen. Poleg klicanja javanskih zrn preko posebnih značk se uporablja tudi *EL* (Expression Language), ki poenostavi klicanje in nastavljanje lastnosti javanskih zrn. Primer EL je naslednji klic *#imeRazreda.imeLastnosti*.

Tretja tehnologija predstavitve podatkov končnemu uporabniku se imenuje *JSF*. JSF je ogrodje za izdelavo spletnih aplikacij in temelji na vzorcu model-pogled-kontroler (ang. Model-View-Controller - MVC) [13]. Verzija 2.0 privzeto uporablja *Facelets* ogrodje za spletne predloge (ang. Web Template), medtem ko je prejšnja verzija JSF uporabljala JSP za ta namen. JSF tako vsebuje množico komponent za gradnjo uporabniškega vmesnika. Te komponente so posebne HTML značke, ki se uporabljajo za definicijo v predlogi in se tekom izvajanja prevedejo v skupek HTML, CSS in Java Script kode. Poleg same strani, ki je običajno shranjena s končnico *.xhtml* se lastnosti JSF nastavi tudi v datoteki *web.xml* in *faces-config.xml*.

Skupaj z JSF se uporabljajo upraviteljska zrna (ang. Managed Beans). To so razredi, ki so podobni razredom pri JSP tehnologiji. Razlika je v tem, da imajo upraviteljska zrna nad imenom razreda običajno dodani dve anotaciji. Prva anotacija *@ManagedBean* nam pove, da gre za upraviteljsko zrno, ki ga lahko kličemo neposredno z JSF strani z uporabo EL značk. Druga anotacija označuje doseg zrna, ki nam pove, koliko časa je zrno aktivno oziroma kdaj se izvede novi primerek zrna. Najbolj pogosti dosegi so:

- *@RequestScoped* - Doseg zahteve, ki velja v času zahteve HTTP.
- *@ViewScoped* - Doseg pogleda, ki velja doker se nahajamo na isti strani.
- *@SessionScoped* - Doseg seje, ki velja dokler je seja veljavna (kontroliramo začetek in konec).
- *@AplicationScoped* - Doseg velja skozi celotno življensko obdobje aplikacije.

Znotraj upraviteljskega zrna lahko pokličemo tudi drugo zrno, in sicer z anotacijo *@ManagedProperty* in ustrezno SET metodo. Ta vrne podatke

danega zrna (v primeru, da je zrno veljavno in v dosegu), sicer nam vrne prazno zrno.

Spodnji primer prikazuje predlogo JSF in ustrezno upraviteljsko zrno.

```
// JSF primer predloge za prijavo v sistem
<h:form>
    <p:outputLabel value="#{msg.username}" />
    <p:inputText id="username" name="username"
        value="#{loginBean.username}" required="true"
        requiredMessage="#{msg.required_field}" />

    <p:outputLabel value="#{msg.password}" />
    <p:password id="password" name="password"
        value="#{loginBean.password}" required="true"
        requiredMessage="#{msg.required_field}" />

    <p:commandButton value="#{msg.login}"
        action="#{loginBean.login()}" ajax="false" />
</h:form>

// Upraviteljsko zrno z dosegom "zahtevka"
@ManagedBean(name = "loginBean")
@RequestScoped
public class LoginBean implements Serialzable {
    // Spremenljivke
    private String          username;
    private String          password;

    @ManagedProperty(value = "#{currentUser}")
    private CurrentUser     currentUser;

    public void login() {
        // Preverimo uporabnika preko JAAS.
    }
```

```
    // Ustrezne SET in GET metode.
}
```

### 3.1.4 EJB

Javanska strežniška zrna (ang. Enterprise Java Beans - EJB) so strežniške komponente za modularno izgradnjo poslovno informacijskih sistemov [8]. Izvajajo se znotraj lastnega vsebnika (ang. EJB Container).

V zadnji verziji EJB (EJB 3.0) sta na razpolago dva tipa strežniških zrn:

- *Sejna zrna (ang. Session Beans)* so zrna, ki opravljajo poslovne operacije in transkacije z bazo podatkov.
- *Sporočilna zrna (ang. Message Driven Beans - MDBs)* so zrna, ki so asinhrona in odgovarjajo na zunanje dogodke (tipično iz vrste ali teme).

Sejna zrna se delijo še na tri podtipe:

- *@Stateless* - Ne vzdržujejo stanja in so najpogostejša pri generičnih opravilih, saj ne ohranjajo stanja med interakcijo z uporabniki.
- *@Stateful* - Vdržujejo stanje tako, da ima vsak uporabnik svoje zrno in je običajno pogojeno z uporabnikovo sejo prijave.
- *@Singleton* - Zrno, ki ima stanje od prvega klica zrna do odstranjevanja aplikacije iz strežnika.

Za klicanje strežniških zrn iz drugih ali lokalnih sistemov poznamo dva načina:

- *Lokalni klici* - Klici, ki se izvedejo znotraj istega aplikacijskega strežnika in zahtevajo lokalni vmesnik.
- *Oddaljeni klici* - Klici, ki se izvedejo na oddaljenem strežniku ali aplikaciji in zahtevajo oddaljeni vmesnik.

V naši aplikaciji uporabljamo samo lokalne klice in zato le lokalne vmesnike. Lokalni vmesnik je abstraktni javanski razred, ki vsebuje definicijo metod in funkcij. Strežniško zrno ima nalogo razširitve vmesnika, kjer implementira dane metode in funkcije.

### 3.1.5 JMS

Javanski sporočilni sistem (ang. Java Message Service - JMS) je del sistema MOM (Message Oriented Middleware), ki skrbi za pošiljanje sporočil med dvema ali več odjemalci. Zasnovan je kot standard, ki dovoljuje komponentam, ki so del Java EE, da ustvarjajo, pošiljajo, sprejemajo in berejo sporočila (objekte). Dovoljuje sporočanje med več različnimi komponentami distribuiranega sistema z namenom zanesljivosti in asinhronske komunikacije [14].

Vsebuje sedem standardnih komponent:

- *Ponudnik (ang. JMS Provider)* - Implementacija JMS vmesnika, ki skrbi za povezavo z MOM sistemom.
- *Odjemalec (ang. JMS Client)* - Komponenta, ki pošilja ali sprejema sporočila.
- *Proizvajalec (ang. JMS Producer/Publisher)* - JMS proizvajalec, ki skrbi za kreiranje in pošiljanje sporočil.
- *Porabnik (ang. JMS Consumer/Subscriber)* - JMS porabnik, ki prejema in bere sporočila.
- *Sporočilo (ang. JMS Message)* - Sporočila, ki so običajno javanski objekti tipa POJO in se prenašajo med JMS odjemalci.
- *Vrsta (ang. JMS Queue)* - Vrsta, ki vsebuje sporočila in čaka na branje enega odjemalca; ko se sporočilo prebere se le to odstrani iz vrste.
- *Tema (ang. JMS Topic)* - Mehanizem, ki dovoljuje branje sporočil uporabnikom, ki so prijavljeni na določeno temo.

Za primer si oglejmo vrsto za pošiljanje elektronskih sporočil. Vrsto (tudi temo) je potrebno najprej ustvariti v aplikacijskem strežniku, nato pa kreirati sporočilno zrno.

Sporočilno zrno mora vsebovati anotacijo *@MessageDriven* in mora razširjati razred *MessageListener*. S tem pridobimo metodo *onMessage*, ki se kliče vedno ob prispelem sporočilu. Metoda deluje tako, da ob pridobljenem sporočilu izvede določeno kodo (običajno sporočilo posreduje naprej poslovni metodi), počaka odgovor in nato označi sporočilo kot prebrano ter ga odstrani iz vrste. Vse skupaj se ponavlja dokler se aplikacija nahaja na strežniku.

```
// standalone.xml
<jms-destinations>
    <jms-queue name="MailsQueue">
        <entry name="java:jboss/queue/MailsQueue"/>
    </jms-queue>
</jms-destinations>

// Sporocilno zrno, ki se odzove na prispeto sporocilo iz vrste.
@MessageDriven(
    name = "EmailQueue", activationConfig = {
        @ActivationConfigProperty(propertyName = "destinationType",
                                  propertyValue = "javax.jms.Queue"),
        @ActivationConfigProperty(propertyName = "destination",
                                  propertyValue =
                                      "java:jboss/queue/MailsQueue"),
        @ActivationConfigProperty(propertyName = "acknowledgeMode",
                                  propertyValue = "Auto-acknowledge")
                                      })
public class EmailQueue implements MessageListener {
    public void onMessage(Message message) {
        ObjectMessage sporocilo = (ObjectMessage) message;
        Email email = (Email) sporocilo.getObject();
        // Klicanje poslovne metode za posiljanje e-sporocila.
```

```
    }
}
```

### 3.1.6 Časovne storitve

Časovna storitev (ang. Timer Service) je del javanskih strežniških zrn in omogoča nadzor klicev poslovnih metod [8]. Z njeno pomočjo lahko upravljamo časovni nadzor nad zrnom. Poznamo dve delitvi časovnih storitev, in sicer *interval klica* ter *čas poteka intervala*. V naši aplikaciji bomo uporabili *interval klica*, s katerim vsakoletno ob koncu leta kreiramo ustrezne podatke. Dano metodo bomo označili kot *@Schedule*, kar pomeni, da gre za koledarsko osnovano časovno storitev. Metodo, ki bo neodvisna od uporabnika in za svoje delovanje ne potrebuje seje, bomo ustvarili v strežniškem zrnu tipa *@Singleton*. Primer zrna:

```
@Singleton
public class PripravaPodatkovObNovemLetu {
    @Schedule(dayOfMonth = "1", month = "12", hour = "3", minute =
        "00", year="*")
    private void kreiranjeStevcevZaModule() {
        // Izvorna koda funkcije.
    }
}
```

### 3.1.7 JAAS

JAAS (Java Authentication And Authorization Service) je implementacija osnovnega standardnega avtorizacijskega modula in je del standardne različice Java SE. Glavni namen JAAS je ločitev preverjanja uporabnikov in urejanja le-teh, tako da so uporabniki neodvisno urejeni s strani drugega sistema. JAAS ne beleži, od kje koda izvira in kdo je podpisnik izvajalne kode, temveč vsebuje podatek o tem, kdo trenutno poganja kodo [15].

Polna funkcionalnost storitve JAAS je odvisna od implementacije ponudnika storitve. Vmesnik, ki upravlja s storivijo JAAS, mora vsebovati:

- Opis identitete (ang. *Principal*).
- Množico vlog (ang. *Subject*).
- Prijavni modul, ki preverja identiteto in vrača množico vlog.
- Storitev, ki preverja ali ima prijavljeni uporabnik dodeljeno vlogo.

Prijavni moduli (ang. Login Modules) so lahko različni in odvisni od ponudnika storitve. Poznamo naslednje tipične tipe prijavnih modulov:

- Bazno preverjanje uporabnika (ang. Database Server Login Module).
- LDAP preverjanje uporabnika (ang. Ldap Login Module).
- Preverjanje na podlagi tekstovnih datotek (ang. Users Roles Login Module).
- Preverjanje s certifikati (ang. Base Cert Login Module).

Pri naši implementaciji varnostnega mehanizma bomo uporabili bazni modul. Oglejmo si primer, kako bomo to storili na aplikacijskem strežniku WildFly.

Najprej bomo v konfiguracijsko datoteko strežnika (v našem primeru *standalone.xml*) vpisali ustrezne podatke. Značka *security-realm* nam pove, naj za preverjanje istovetnosti uporabimo varnostno domeno *TajnikJAAS*, ki jo definiramo z značko *security-domain*. Pri tem lahko še dodamo, da želimo podatke predpolniti in s tem prihraniti dodatne zahtevke med aplikacijskim in podatkovnim strežnikom. Pri definiciji varnostne domene podamo naslednje opcije (spodaj se nahaja primer nastavitev):

- dsJndiName - JNDI vir do definicije bazne povezave.
- principalsQuery - Stavek SQL, ki vrne zakodirano geslo in s katerim preverimo ali je uporabnik v bazi.

- rolesQuery - Stavek SQL, ki vrne seznam vlog na podlagi uporabniškega imena.
- hashAlgorithm - Tip algoritma s katerim je geslo shranjeno (uporabili smo SHA-512).
- hashEncoding - Tip kodiranja.
- unauthenticatedIdentity - Ime uporabnika, če le-ta ni prijavljen.

```
// Datoteka: standalone.xml
...
<security-realm name="RealmTajnikJAAS">
    <authentication>
        <jaas name="TajnikJAAS"/>
    </authentication>
</security-realm>
...
<security-domain name="TajnikJAAS" cache-type="default">
    <authentication>
        <login-module
            code="eu.tajnik.jaas.TajnikDatabaseLoginModule"
            flag="required">
            <module-option name="dsJndiName"
                value="java:jboss/datasources/TajnikRazvojDS"/>
            <module-option name="principalsQuery" value="SELECT
                password FROM UserPrincipal WHERE username=? AND
                status='ACTIVATED'"/>
            <module-option name="rolesQuery" value="SELECT role,
                'Roles' FROM vi_username_roles WHERE username=?"/>
            <module-option name="hashAlgorithm" value="SHA-512"/>
            <module-option name="hashEncoding" value="hex"/>
            <module-option name="unauthenticatedIdentity"
                value="guest"/>
        </login-module>
```

```
    </authentication>
</security-domain>
...
```

Poleg preverjanja istovetnosti uporabnika pri prijavi je možno uporabiti tudi posebne varnostne anotacije pri izvajanju strežniških zrn skupaj z varnostnimi vlogami. Anotacijo, ki jo želimo uporabiti, lahko napišemo nad izbrano poslovno metodo ali nad celotnim razredom oziroma zrnom - odvisno, kaj želimo zaščititi s pripisano vlogo [8]. Primeri anotacij so zapisani v tabeli 3.1.

| **Anotacija** | **Uporaba** | **Razlaga** |
|---|---|---|
| @RunAs | Nad celotnim zrnom. | Zrno se bo izvedlo pod dano vlogo. |
| @DeclareRoles | Nad celotnim zrnom. | Določa, katere vloge se bodo uporabile v zrnu. |
| @RolesAllowed | Nad celotnim zrnom in metodo. | Določa, katere vloge lahko izvedejo dani del aplikacije. |
| @DenyAll | Nad celotnim zrnom. | Preprečuje izvajanje vsem vlogam. |
| @PermitAll | Nad celotnim zrnom in metodo. | Dovoljuje izvajanje vsem vlogam. |

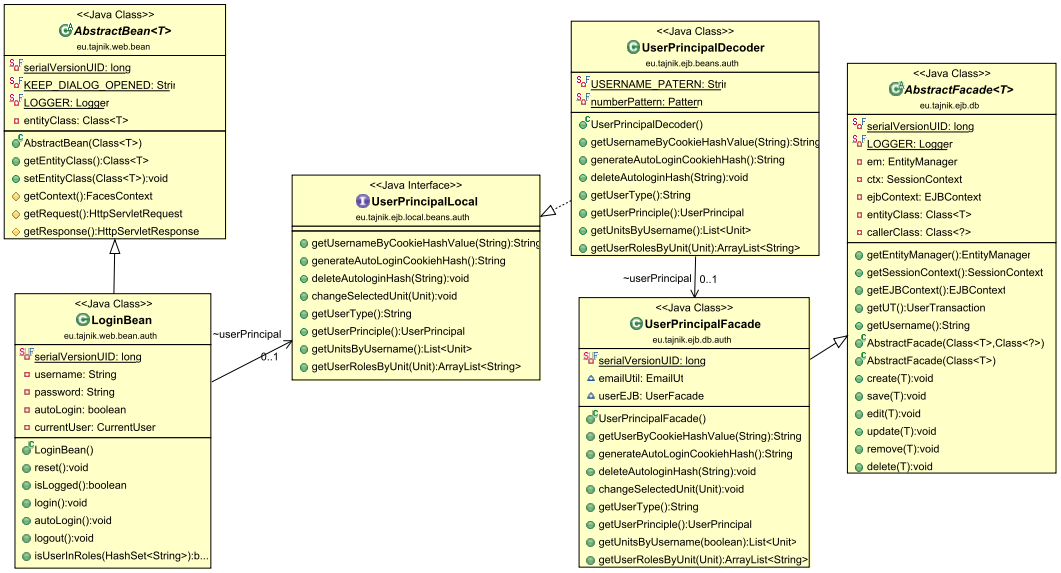
Tabela 3.1: Varnostne anotacije.

Za boljši pregled, kako poteka prijava uporabnika v našem sistemu, si oglejte diagram poteka na sliki 4.10.

## 3.2 Povezanost komponent

Povezanost komponent je v poslovnem sistemu zelo pomembna. Brez nje ne bi imeli več neodvisnih komponent, ki se preko vmesnikov povezujejo in naredijo celoten sistem zanesljiv in varen.

Na sliki 3.2 lahko vidimo povezanost večih komponent aplikacijskega strežnika. Leva stran prikazuje abstraktni razred *AbstractBean* in razred *LoginBean*, ki je izpeljan iz abstraktnega razreda. Oba razreda se nahajata v




Slika 3.2: Primer povezanosti upraviteljskega in strežniškega zrna.

spletnem vsebniku. Razred *LoginBean* je upraviteljsko zrno, ki se uporablja za prikaz podatkov na spletni strani.

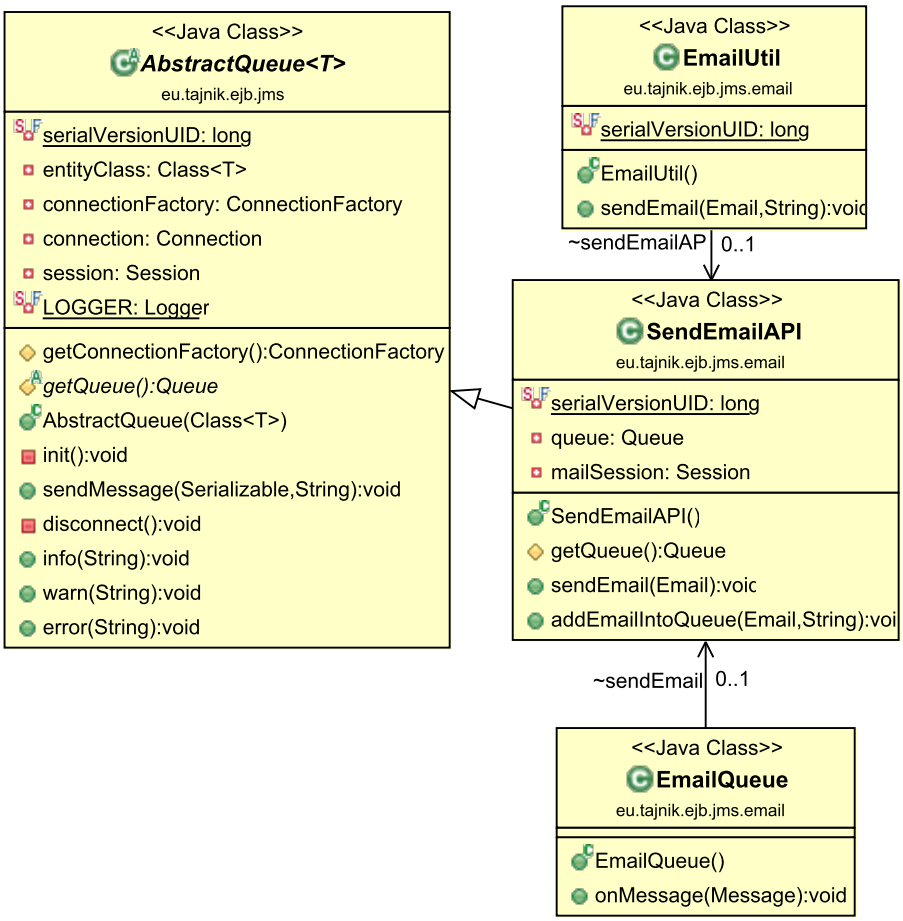
Na sredini se nahaja lokalni vmesnik med spletnim in EJB vsebnikom, ki zagotavlja prenos podatkov v obe smeri. Pri upraviteljskem zrnu se lokalni vmesnik navede kot odvisnost (ang. Dependency Injection) z uporabo anotacije *@EJB*, kar zagotovalja, da je vir ob klicu vedno na voljo.

Desna stran predstavlja strežniška zrna v EJB vsebniku. Strežniško zrno *UserPrincipalDecoder* implementira lokalni vmesnik *UserPrincipalLocal* in vse njegove metode. V tem razredu preverimo parametre in jih po potrebi popravimo oziroma dopolnimo. Ko so parametri in entitete pravilne, pokličemo strežniško zrno *UserPrincipalFacade* preko EJB odvisnosti. Slednji razširja abstraktni razred *AbstractFacade*, ki vsebuje najpogostejše uporabljene metode za delo z JPA entitetami. Primeri teh metod so: *ustvari entiteto (ang. Persist)*, *posodobi entiteto (ang. Merge)* ter *briši entiteto (ang. Remove)*. Tip entitete (*UserPrincipal*) podamo kot razred pri razširjanju abstraktnega razreda *AbstractFacade*.

Oglejmo si še primer na sliki 3.3, kjer je prikazana povezanost sporočilnega in strežniškega zrna ter abstraktnega vmesnika za delo z vrsto.

Na desni strani slike imamo predstavljen abstraktni razred *AbstractQueue*, kjer je kot tip T naveden generični razred, ki predstavlja vrsto. Razred ima definirane skupne metode in lastnosti, ki pripadajo vsem vrstam. Primeri skupnih metod so pošiljanje sporočil v vrsto ter povezovanje in zapiranje povezave nad vrsto. Za implementacijo abstraktnih metod v prejšnjem razredu smo uporabili razred *SendEmailAPI*. Uporablja se z namenom, da entitetni razred (v spodnjem primeru razred *Email*) vstavi na začetek vrste z metodo *addEmailIntoQueue*.

Ko sporočila prispejo v vrsto, jih metoda *onMessage* (sporočilnega zrna *EmailQueue*) zaporedoma prebira in posreduje naprej strežniškemu zrnu, da stori v naprej definirano operacijo. V našem primeru smo uporabili razred *SendEmailAPI* in metodo *sendEmail*, ki pošlje elektronsko sporočilo na naslove, definirane v entiteti *Email*. Za uporabo posredovanja sporočila v vrsto se uporablja razred *EmailUtil*, ki preveri parametre razreda *Email*, predno se slednja entiteta vstavi v vrsto.




Slika 3.3: Primer povezanosti strežniškega sporočilnega zrna.

# Poglavje 4

# Implementacija informacijskega sistema

Implementacija našega informacijskega sistema se je začela s pripravo diagrama primera uporabe. Nadeljevala se je s kreiranjem entitetno-relacijskega diagrama (ang. Entity Relationship Diagram - E-R diagram) in z izvozom SQL kode, s katero smo izdelali tabele na našem podatkovnem strežniku MySQL. Za tem smo se lotili programiranja v Java EE in končali s celovito informacijsko rešitvijo za društva.

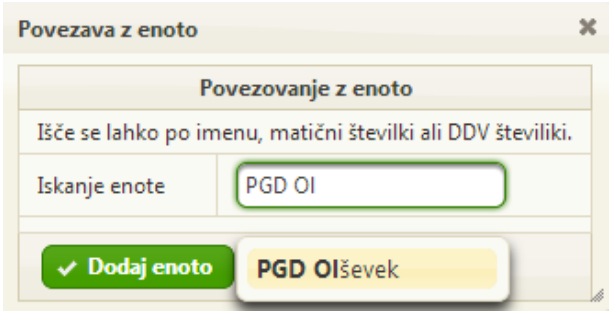
## 4.1 Opis aplikacije

Aplikacija (poimenovali smo jo Meteor) je celovita informacijska rešitev za vodenje gasilskih društev. Sistem trenutno ponuja registracijo in prijavo uporabnikov (slika 4.1), dodajanje in povezovanje enot oziroma društev med več uporabniki (slika 4.2), administracijo uporabnikov in dodeljevanje vlog. Kot smo omenili, lahko z njeno pomočjo z enim uporabniškim računom hkrati upravljamo več društev naenkrat, pri čemer imamo pri posameznem društvu različne pravice, odvisno, katere nam administratorji dodelijo.

Trenutno so na voljo trije moduli (slika 4.3):

- Delovodnik


Slika 4.1: Prijava v aplikacijo.

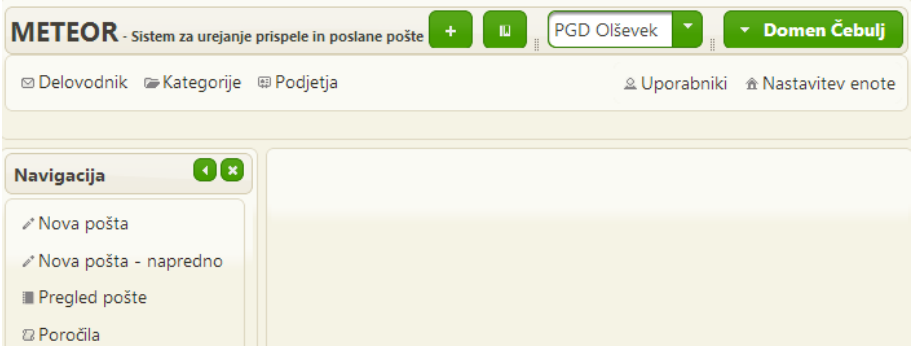



Slika 4.2: Iskanje in povezovanje z enoto.

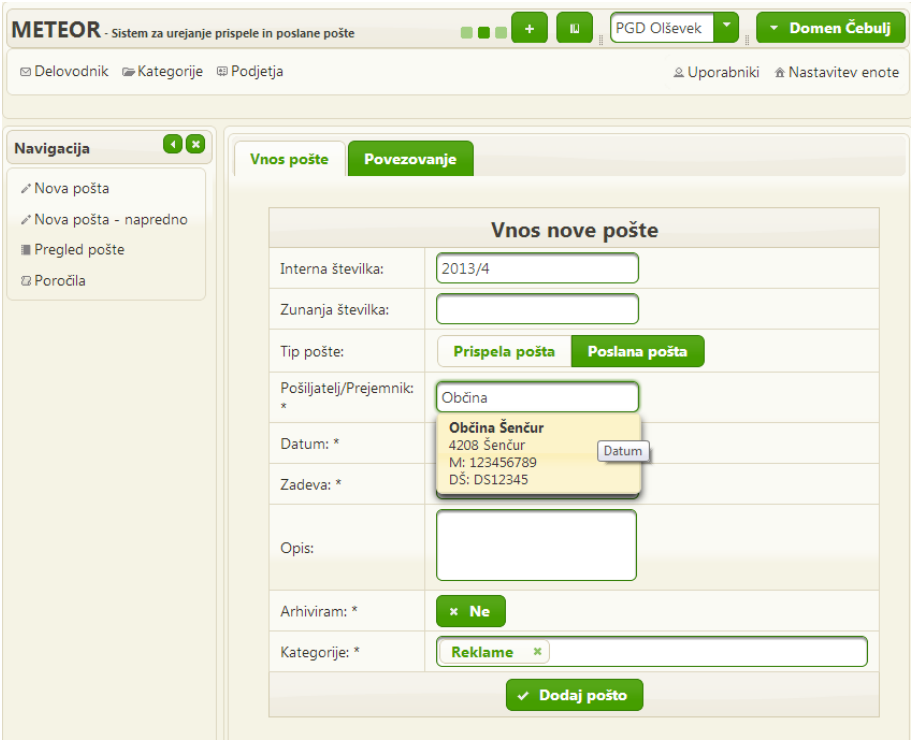
- Kategorije
- Podjetja

Modul *Delovodnik* omogoča vpisovanje prispele in poslane pošte vseh tipov (navadna pisemska pošiljka, elektronska pošta, fax sporočilo itn.), razvrščanje pošte po kategorijah, pregled in urejanje pošte ter izpis seznama pošte glede na podano časovno obdobje. Vnos pošte prikazuje slika 4.4.

Modul *Kategorije* je namenjen razvrščanju pošte v kategorije, kjer lahko eno pošto razdelimo v več kategorij. Kategorijo lahko ustvarimo v modulu *Kategorije* ali ob kreiranju nove pošte (če seveda kategorija še ne obstaja in imamo ustrezne pravice). Enako velja tudi za modul *Podjetja*, kjer vpisujemo podjetja, od katerih smo pošto pridobili oziroma komu pošto pošiljamo. Oba modula *Kategorije* in *Podjetja* sta vzporedno povezana z modulom *Delovodnik*.




Slika 4.3: Prikaz menijskih točk v aplikaciji.




Slika 4.4: Vnos pošte.


Slika 4.5: Dodeljevanje pravic uporabniku.

Pri vsakem modulu je na voljo celovit pregled nad izbranimi podatki. Ravno tako je glede na vloge omogočeno vnašanje, urejanje in brisanje podatkov. Pri modulu *Delovodnik* je na voljo še sekcija za kreiranje poročila, pri katerem se kreira poročilo iz iReport predloge.

Administratorji društva (običajno osebe, ki registrirajo društvo) imajo poleg vlog, ki si jih dodelijo, še naslednje vloge: lahko urejajo enoto, potrjujejo uporabnike in dodeljujejo/odstranjujejo vloge ostalim članom društva (slika 4.5).

Aplikacija je bila izdelana po metodi model-pogled-kontroler. S tem smo ločili prikaz strani od podatkov ter od poslovne logike. Za prikaz strani oziroma pogleda smo uporabili JSF skupaj z upraviteljskimi zrni, za model podatkov smo uporabili entitete JPA, za poslovno logiko javanska strežniška zrna.

## 4.2 Diagram primera uporabe

Na začetku projekta smo naredili poenostavljen diagram primera uporabe naše aplikacije (slika 4.6).

Ob obisku strani našega sistema se mora uporabnik najprej registrirati, zatem potrditi registracijo in nato prijaviti v sistem. Po prijavi se uporabnik odloči, ali bo kreiral novo društvo, ali se bo povezal z obstoječim društvom.

Ob kreiranju novega društva prijavljeni uporabnik pridobi vse trenutno mogoče vloge nad novo ustvarjenim društvom ter dodatno vlogo, ki ga označuje kot *administratorja društva*. S to vlogo lahko uporabnikom, ki bodo zaprosili za vstop v njegovo društvo, dodeli ustrezne vloge ali jih zavrne.

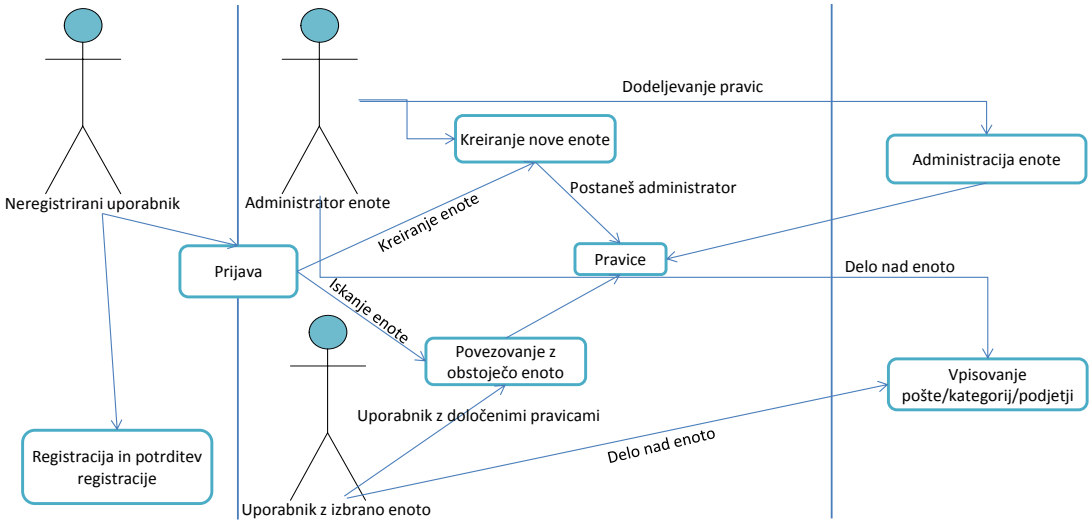
Ob povezovanju uporabnika v že obstoječe društvo ga mora administrator odobriti ter mu dodeliti ustrezne vloge, npr. dovoljenje za pisanje v delovodnik, dovoljenje za kreiranje kategorij itd.

Ko ima prijavljeni uporabnik izbrano društvo ter na njem ustrezne vloge, se lahko loti dela - vpisovanje pošte v delovodnik, urejanje kategorij itn.

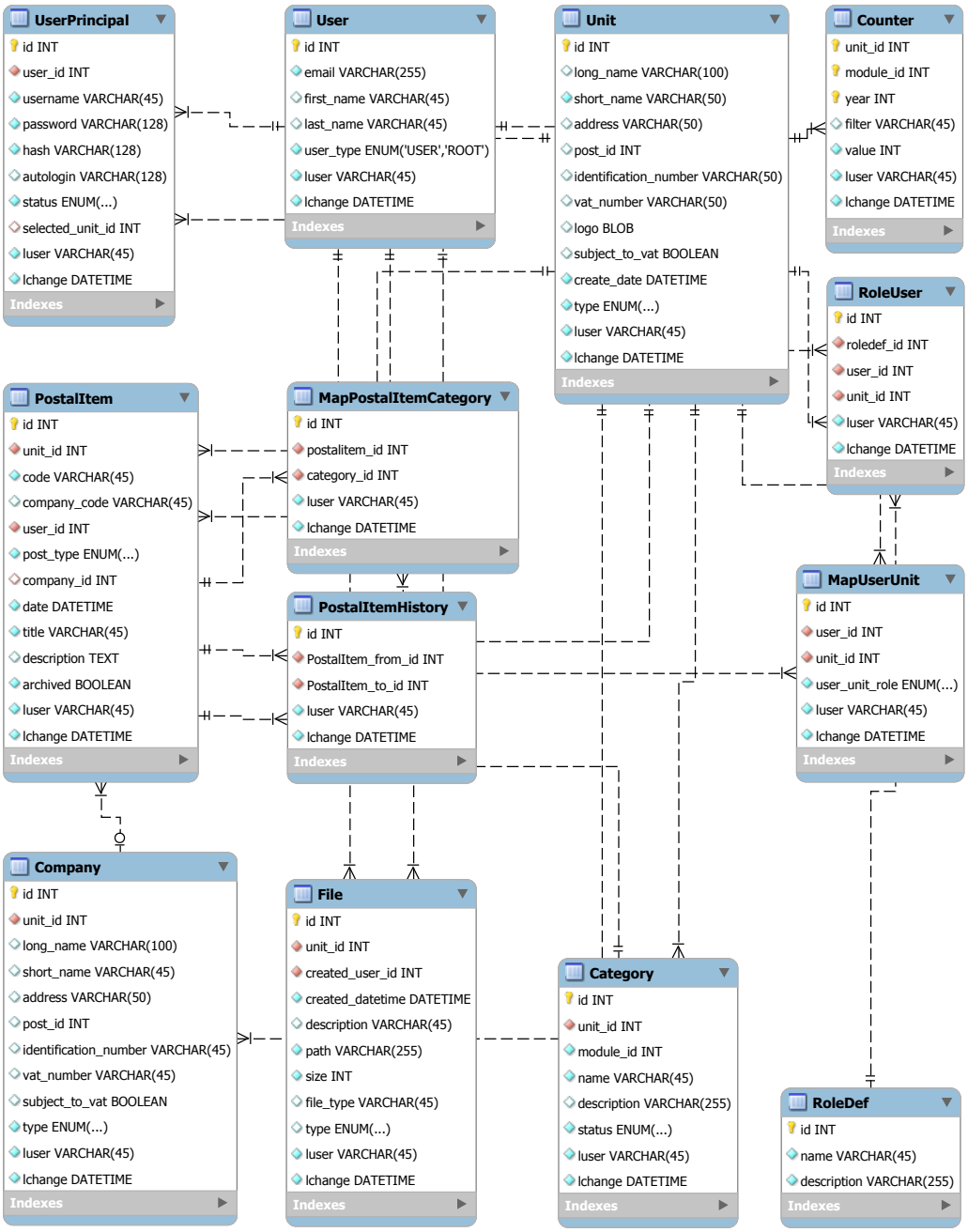
## 4.3 Entitetno-relacijski diagram

Sledilo je ustvarjanje E-R diagrama za podatkovno bazo z orodjem MySQL Workbench. Diagram nam služi kot logična predstavitev podatkovnega modela, ki naj bi bila razumljiva programerjem pri izdelavi aplikacij. Na sliki 4.7 je prikazan konceptualni model našega sistema, ki temelji na primeru uporabe.

Na začetku izdelave diagrama imamo tabelo **UserPrincipal**, ki vsebuje polja *username*, *password*, *status* in *user_id*. Za prva tri polja je jasno, kaj predstavljajo (uporabniško ime, geslo in status uporabnika - aktiven, neaktiven). Medtem ko zadnje polje predstavlja povezavo s tabelo **User**, ki vsebuje dodatne atribute za prijavljeno osebo (npr. elektronski naslov, ime in priimek). Za prijavo v sistem potrebujemo tako aktivirano uporabniško ime kot tudi geslo.




Slika 4.6: Diagram primera uporabe.




Slika 4.7: E-R diagram aplikacije.

Za vpis društva se uporabljata tabeli **Unit** in **MapUserUnit**. Ob kreiranju nove enote oziroma društva se slednje zapiše v prvo tabelo, v drugo se doda povezava med novo nastalo enoto in uporabnikom. Ta povezava nam pove, v kakšni zvezi je uporabnik do dane enote.
Poznamo naslednja stanja:

- *ADMINISTRATOR* - Uporabnik je administrator enote (običajno tisti, ki je enoto ustvaril).
- *USER* - Uporabnik je splošen uporabnik v izbrani enoti, kateremu je potrebno dodati dodatne vloge.
- *PENDING* - Uporabnik čaka, da ga administrator potrdi za uporabo njegove enote.
- *DELETED* - Uporabnik je bil izbrisan iz enote.

Za seznam vlog modulov in dodeljevanje vlog posameznem uporabniku nad izbrano enoto se uporabljata tabeli **RoleDef** in **RoleUser**. V prvi so naštete vse možne vloge in opis le-teh, v drugi pa so vloge dodeljene uporabniku in enoti.

Glavna tabela modula *Delovodnik* je **PostalItem**. V njej so zapisani vsi potrebni podatki za vpisovanje prispele in poslane pošte. V relaciji z dano tabelo sta tabeli **Category** in **Company**. V prvi je seznam vseh kategorij, uporabljenih v dani enoti, v drugi tabeli seznam podjetij, ki so bila dodeljena na prispelo oziroma poslano pošto. Za relacijo M:M (mnogo proti mnogo) pri kategorijah in pošti se uporablja tabela **MapPostalItemCategory**. Ta nam omogoča, da na posamezno vpisano pošto dodamo N kategorij.

Vse ostale tabele niso tako pomembne in si jih bralec lahko ogleda iz slike 4.7. Naj dodamo še, da se pri vsaki tabeli uporabljata še dve polji, in sicer *lchange* in *luser*, ki se spremenita ob vsakem spreminjanju podatkov. V prvo polje se zapiše uporabniško ime uporabnika, v drugo datum in čas spremembe.

## 4.4 Arhitektura MVC

Model-pogled-kontroler je programsko-aplikacijska arhitektura, ki skrbi, da so podatki, grafične predloge in poslovna logika medsebojno ločeni. MVC se deli na tri dele:

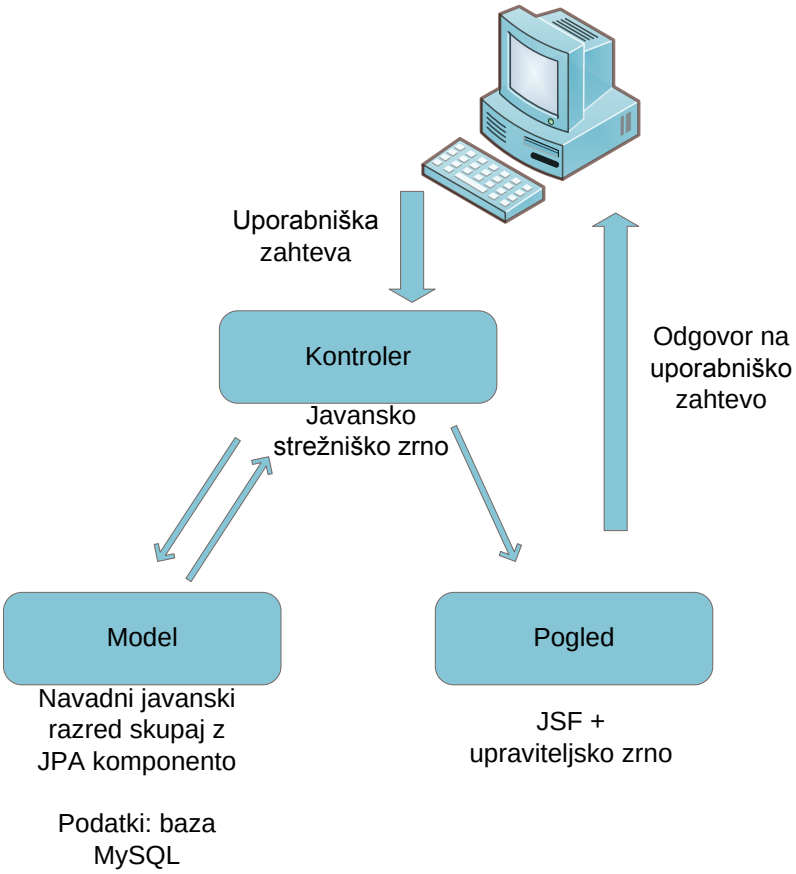
- Model (ang. Model)
- Pogled (ang. View)
- Kontroler (ang. Controller)

Model predstavlja podatke iz podatkovne baze ter morebitne omejitve in dodatne lastnosti, vezane na podatke. Pogled je način prikaza podatkov končnemu uporabniku, kontroler pa predstavlja logiko, ki skrbi za medsebojno komunikacijo med uporabnikom in podatki [17].

Pri implementaciji smo upoštevali arhitekturo MVC. Kot model podatkov smo uporabili navaden javanski razred skupaj s komponento JPA, ki sta omogočila pridobivanje oziroma vnos podatkov v podatkovno bazo. Za pogled smo uporabili komponento JSF skupaj z upraviteljskimi zrni, ki so skrbeli za prikaz podatkov uporabniku. Za interakcijo med uporabniki in podatki je poskrbel kontroler, ki smo ga implementirali z uporabo javanskih strežniških zrn.

Proces MVC, ki je prikazan na sliki 4.8, poteka tako, da najprej pridobimo uporabniško zahtevo, ta nato pride do strežniških zrn (kontroler), v katerem se zahteva obdela. Po zahtevane podatke se odpravi v podatkovno bazo preko JPA komponente (model), ki vrne podatke strežniškemu zrnu oziroma kontrolerju. Kontroler nato poskrbi, da se podatki pošljejo naprej do pogleda (upraviteljskemu zrnu in zatem strani JSF). Stran se posreduje naprej uporabniku kot odgovor na njegovo zahtevo.

Podrobnejši pogled na zasnovo razredov nam prikazuje slika 4.9. Model predstavlja razred *RoleUser*, ki vsebuje podatek o uporabniških pravicah. Pogled nam predstavlja razred upraviteljskega zrna *UserUnitRole* in pripa-




Slika 4.8: Arhitektura MVC.

dajočo datoteko JSF (prikazana spodaj), ki kliče upraviteljsko zrno. Končni videz je prikazan na sliki 4.5.

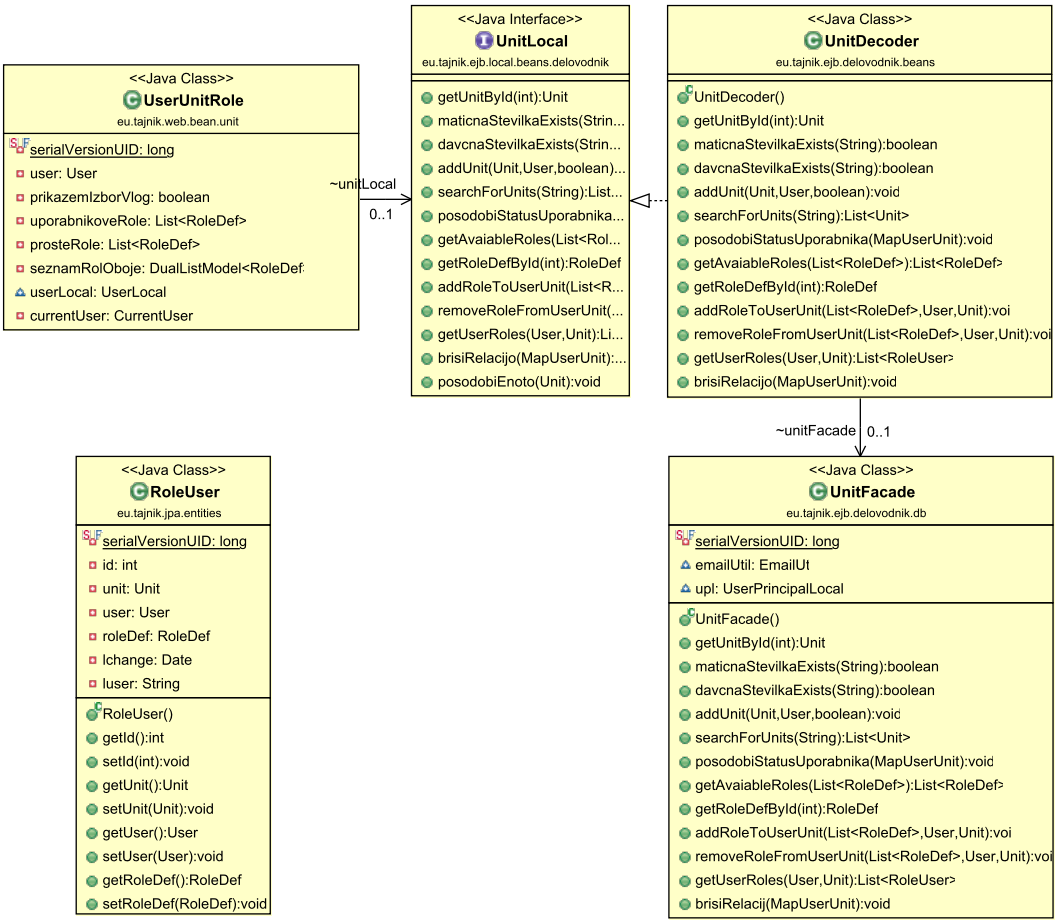
```
<p:outputPanel id="izbiraRol" styleClass="izbiraRol"
    layout="block">
<h:outputText value="Izbrani uporabnik:"
    rendered="#{userUnitRole.prikazemIzborVlog}" />
<b> <h:outputText escape="false" id="sUser"
        value="#{userUnitRole.user.imepriimek()}"
        rendered="#{userUnitRole.prikazemIzborVlog}" />
</b>

<p:pickList id="seznamRoleId" styleClass="tipa"
    value="#{userUnitRole.seznamRolOboje}" var="rola"
    itemValue="#{rola}" itemLabel="#{rola.name}"
    converter="rolaConverter" showCheckbox="true"
    rendered="#{userUnitRole.prikazemIzborVlog}"
        showSourceFilter="true"
    showTargetFilter="true" filterMatchMode="contains"
    addLabel="Dodaj izbrane" addAllLabel="Dodaj vse"
    removeAllLabel="Odstrani vse" removeLabel="Odstrani izbrane"
    label="Dodajanje vlog">

    <f:facet name="sourceCaption">Seznam nedodeljenih vlog</f:facet>
    <f:facet name="targetCaption">Seznam vlog na
        uporabniku</f:facet>

    <p:ajax event="transfer" listener="#{userUnitRole.prenesiVloge}"
        update="@widgetVar(message) @(.top, .zgornjiMeniModulov)" />

    <p:column style="width:100%;white-space:
        pre;">#{rola.name}&#10;#{rola.description}</p:column>
</p:pickList>
</p:outputPanel>
```




Slika 4.9: Primeri razredov za potrebe MVC v aplikaciji.

Kontroler je v naši aplikaciji javansko strežnško zrno (*UnitDecoder* in *UnitFacade*), ki obravnava uporabniške zahteve in uporabniku vrača zahtevane podatke.

Oglejmo si primer uporabniške zahteve. Uporabnik pride na stran za urejanje uporabniških pravic. Ob izbiri uporabnika se zahteva izpis njegovih dodeljenih pravic. Spletna zahteva potuje preko upraviteljskega zrna *UserUnitRole* in metode *getUserUnitRoles* do lokalnega vmesnika (*UnitLocal*). Slednji posreduje zahtevo do strežniškega zrna *UnitDecoder* (kontroler), kjer se kliče funkcija *getUserRoles*. Funkcija vrne podatke, katere pridobi na pod-

lagi parametrov in s klicem JPA entitete *RoleUser* (model). JPA klic vrne podatke EJB vsebniku, le-ta pa nazaj upraviteljskemu zrnu. Za prikaz teh poskrbi JSF stran (pogled).

## 4.5 Zanimivi vidiki implementacije

Pri programiranju aplikacije smo naleteli na zanimive "težave", ki smo jih uspeli uspešno rešiti, medtem ko smo nekatere rešili s preoblikovanjem zasnovane ideje. Nekaj zanimivih rešitev opisuje to podpoglavje.

### 4.5.1 Prijava uporabnika in dodeljevanje uporabniških pravic

Pri prijavi uporabnika smo želeli implementirati t.i. samodejno prijavo na podlagi piškotka (ang. Auto Login With Cookie). Težava je bila v uporabljeni storitvi JAAS, ki privzeto ne ponuja želene funkcionalnosti.

Težavo smo rešili z implementacijo lastnega prijavnega modula, ki deluje skupaj s storitvijo JAAS. Naredili smo novi razred, ki smo ga razširili z obstoječim JAAS modulom (razred *DatabaseServerLoginModule*). V novem razredu smo povozili nadrejeno metodo *validatePassword* za preverjanje enakosti vnešenega gesla in gesla, pridobljenega iz baze podatkov. Naša metoda sedaj deluje tako, da če pridobi šifrirano geslo, ki je enako konstanti *AUTO_LOGIN_PASS*, potem vemo, da se uporabnik želi prijaviti samodejno in s to kombinacijo vrnemo veljavno geslo, sicer se kliče nadrejena metoda za preverjanje enakosti gesla. Ko se uporabnika dokončno preveri, mu storitev JAAS dodeli vloge iz baze podatkov glede na zadnje izbrano društvo iz prejšnje prijave. Bolj natančen prikaz prijave opisuje diagram poteka na sliki 4.10.

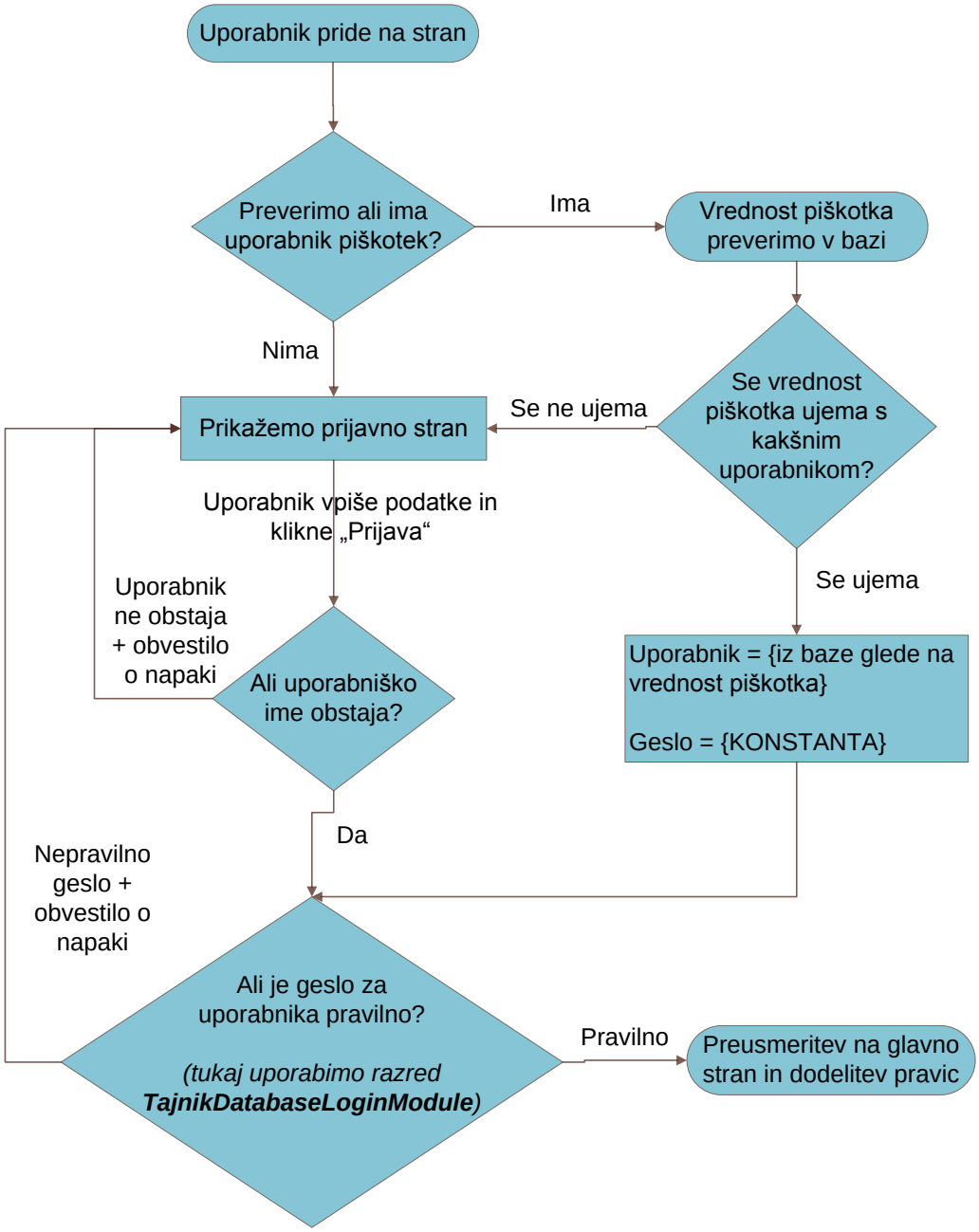
```
public class TajnikDatabaseLoginModule extends
    DatabaseServerLoginModule {
    private static final String AUTO\_LOGIN\_PASS = "KONSTANTA";
```

```
    @Override
    protected boolean validatePassword
        (String inputPassword, String expectedPassword) {
            if (inputPassword.equalsIgnoreCase(
                    PasswordUtil.toSHA512(AUTO\_LOGIN\_PASS))) {
                return true;
            }
        return super.validatePassword(inputPassword,
            expectedPassword);
    }
}
```

### 4.5.2 Spletni filter

Pri vseh spletnih aplikacijah, ki ponujajo registracijo in prijavo, je potrebno zagotoviti, da do zaščitenih spletnih strani dostopajo le prijavljeni uporabniki. V ta namen smo naredili posebni javanski razred označen kot spletni filter (anotacija *@WebFilter*). V tem filtru preverimo, ali je oseba, ki zahteva neko stran prijavljena ali ne. Če je oseba prijavljena, pustimo njeno zahtevo pri miru, sicer uporabnika preusmerimo na stran za prijavo. Ker smo v naši aplikaciji uporabili storitev JAAS, lahko uporabnika enostavno preverimo z metodo *getUserPrincipal()* iz razreda *HttpServletRequest*, ki se pošilja ob vsaki zahtevi.

```
@WebFilter(urlPatterns = { "*" })
public class LoginFilter implements Filter {
    // ...
    public void doFilter(ServletRequest request, ServletResponse
        response, FilterChain chain) throws IOException,
        ServletException {
        HttpServletRequest req = (HttpServletRequest) request;
        // Preverimo prijavo => ali lahko pridobimo uporabnisko ime?
```




Slika 4.10: Diagram poteka prijave uporabnika s storitvijo JAAS.

```
        if (req.getUserPrincipal().getName() == null) {
            // Preusmeritev za prijavo.
        } else {
            // Smo prijavljeni, procesiramo zahtevo naprej.
            chain.doFilter(request, response);
        }
    }
    // ...
}
```

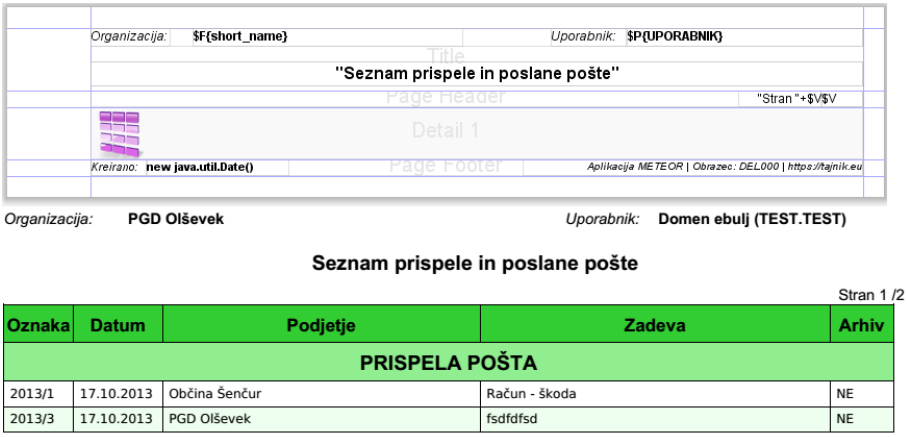
### 4.5.3 Izdelava končnega poročila

Za izdelavo končnega poročila smo si pomagali s knjižnicami programa iReport in s predlogo, ki smo jo naredili v orodju iReport. Delovanje je opisano na sliki 2.2, razliko med predlogo in končnim izgledom poročila prikazuje slika 4.11.

Pri implementaciji izdelave poročila smo upoštevali, da se posamezno poročilo lahko kreira od nekaj sekund ter tja do nekaj minut (bolj kompleksna poročila z večimi podporočili). Zaradi slednjega smo uvedli vrsto z uporabo javanskega sporočilnega sistema. S tem smo dosegli, da se zahteva za kreiranje poročila postavi v vrsto in se izdela šele takrat, ko je na vrsti - torej po principu "prvi pride, prvi melje". Dosegli smo, da uporabniku ni potrebno čakati na odgovor, da se ponudi izdelano poročilo, ampak se bo le-to kasneje pojavilo v tabeli poročil.

### 4.5.4 Časovna priprava podatkov

Pri uporabi aplikacije se pri dodajanju pošte ustvari t.i. števec za vsako društvo. Naloga števca je, da zaporedoma poda številko pošte z ustrezno masko (npr. 2013/1, 2013/5, 2013/7). Prva številka predstavlja tekoče leto, zadnja številka pove zaporedno številko pošte. Ker je potrebno števce vsakoletno kreirati, smo ustvarili časovno funkcijo oziroma časovno storitev, ki se




Slika 4.11: Prikaz predloge poročila (zgoraj) in končne različice.

kliče vsako leto na prvi dan v mesecu decembru. Pri tem kreira vse možne števce, tako da so za naslednje leto le-ti že pripravljeni in se uporabniku ni potrebno dodatno naprezati. Še en primer časovne funkcije je brisanje poročil ob koncu meseca. Ker so poročila dinamična in hitro kreirana, vsak zadnji dan v mesecu izbrišemo nastala poročila in tako prihranimo prostor na trdem disku.

```
@Schedule(dayOfMonth = "1", month = "12", hour = "3", minute =
    "00", year="*")
private void kreiranjeStevcevZaModule() {
    // Klicanje metode iz EJB.
}

@Schedule(dayOfMonth = "last", month = "*", hour = "3", minute =
    "00", year="*")
private void brisanjePorocilnihDatotek() {
    // Klicanje metode iz EJB.
}
```

# Poglavje 5

# Zaključek

V diplomskem delu smo predstavili informacijski sistem za pomoč pri vodenju administrativnih zadev, ki zajemajo področje pošte v gasilskih društvih. Osredotočili smo se na razvoj aplikacije, ki omogoča učinkovito vodenje prispele in poslane pošte pri društvih, kategorizacijo pošte, vodenje podjetij in povezovanje več uporabnikov med društvi ter izdelavo končnega letnega seznama pošte.

Pri implementaciji smo uporabili javanske poslovne komponente. Ena izmed njih je JPA. Za ustrezno delovanje le-te smo nastavili strežnik, uporabniško ime in geslo do podatkovnega strežnika. Uporabljena je bila tudi komponenta JAAS, ki je poskrbela za avtorizacijo in avtentikacijo uporabnikov. Uporabljene so bile še druge poslovne komponente. Za povezanost komponent so poskrbeli lokalni vmesniki, ki so se nahajali v sredini med spletnim in EJB vsebnikom in zagotavljali prenos podatkov v obe smeri. Aplikacija je bila izdelana po metodi MVC, s čimer smo ločili prikaz strani podatkov od poslovne logike.

Nastala aplikacija je celovita informacijska rešitev za vodenje gasilskih društev. Sistem ponuja registracijo in prijavo uporabnikov, dodajanje in povezovanje enot oziroma društev med več uporabniki ter administracijo uporabnikov in dodeljevanje vlog. Trenutno imamo na razpolago tri tipe modulov, ki nam omogočajo uporabo aplikacije in ti so: *Delovodnik*, *Kategorije* in

*Podjetja*. Aplikacija je nadomestila že zastareli (ročni) vpis pošte v elektronskega. Obenem je pripomogla tudi k hitrejšemu vpisovanju in pregledovanju pošte.

Celotno aplikacijo bi se dalo še izboljšati oziroma nadgraditi. Uporabniški vmesnik aplikacije, bi lahko izboljšali tako, da bi izboljšali razporeditev modulov glede na nastavitve uporabnika. Ravno tako bi bilo možno nadgraditi aplikacijo še z dodatnimi moduli. Eden izmed njih bi bil modul *Hidranti*, kjer bi gasilci vodili stanje hidrantnega omrežja v svojem okolišu.

# Slike

# Tabele

# Literatura


[1] B. Šuligoj, "Na paradi kar 4500 slovenskih gasilcev". Dostopno na: http://www.delo.si/novice/slovenija/na-paradi-kar-4500-slovenskih-gasilcev.html

[2] (2013) Gasilska zveza Slovenije. Dostopno na: http://www.gasilec.net/

[3] (2013) Gasilec. Dostopno na: http://sl.wikipedia.org/wiki/Gasilec

[4] (2013) Java (software platform). Dostopno na: http://en.wikipedia.org/wiki/Java_(software_platform)

[5] (2013) Java Platform, Enterprise Edition. Dostopno na: http://en.wikipedia.org/wiki/Java_EE

[6] (2013) Why Facebook Uses MySQL for Timeline. Dostopno na: http://mashable.com/2011/12/15/facebook-timeline-mysql/

[7] (2013) MySQL. Dostopno na: http://en.wikipedia.org/wiki/MySQL

[8] M. B. Jurič, gradivo predmeta "Postopki razvoja programske opreme", 2012

[9] M. B. Jurič, Novosti Java EE 7 in pogled v oblak. Dostopno na: http://www.soa.si/presentations/JavaEE_7.pdf, 2012

[10] (2013) JDBC. Dostopno na: http://en.wikipedia.org/wiki/Java_Database_Connectivity

[11] (2013) JavaMail. Dostopno na: http://en.wikipedia.org/wiki/JavaMail

[12] (2013) Java Servlet. Dostopno na: http://en.wikipedia.org/wiki/Java_Servlet

[13] M. B. Jurič, gradivo predmeta "Računalniške storitve v oblaku", 2012

[14] (2013) Java Message Service. Dostopno na: http://en.wikipedia.org/wiki/Java_Message_Service

[15] (2013) Java Authentication and Authorization Service Dostopno na: http://en.wikipedia.org/wiki/Java_Authentication_and_Authorization_Service

[16] (2013) File upload doesn't work. Dostopno na: https://issues.jboss.org/browse/WFLY-2329

[17] (2013) Model–view–controller. Dostopno na: http://en.wikipedia.org/wiki/Model–view–controller