



UNIVERZITET SINGIDUNUM
FAKULTET ZA INFORMATIKU I RAČUNARSTVO
SAVREMENE INFORMACIONE TEHNOLOGIJE

**Veb aplikacija za rezervaciju
prostorija**

- Master rad -

Mentor: Prof. dr Aleksandar Jevremović

Student: Nikola Rančić

Br. indeksa: 2014/410023

Prof. dr Aleksandar Jevremović _____

Prof. dr Mladen Veinović _____

Prof. dr Milan Milosavljević _____

Beograd, 2015.

Sažetak

U ovom radu opisane su tehnologije koje su korišćene za razvoj aplikacije za onlajn rezervaciju prostorija unutar jedne organizacije. Osnovna funkcionalnost aplikacije je pretraga i rezervacija prostorija po određenim kriterijumima u cilju organizacije nekog događaja. Aplikacija je projektovana u programskom jeziku Java u skladu sa *MVC* arhitekturom i korišćenjem *MySQL* baze podataka.

U radu su opisani osnovni principi korišćenih tehnologija i data je kratka istorija razvoja. U ovom radu opisan je model, arhitektura, realizacija i slučajevi korišćenja veb aplikacije za rezervaciju prostorija. Predstavljeni su osnovni principi *MVC* arhitekture, način na koji je on primenjen i navedeni su primeri iz same aplikacije, dati su primeri iz baze podataka kao i primeri osnovnih principa zaštite i slučajeva korišćenja.

Ključne reči: aplikacija, rezervacija, prostorija, događaj, *online*, *web*, *Java*, *MVC*, *NetBeans*, *MySQL*, *JavaScript*, *jQuery*, klasa, servlet, *JSP*.

Abstract

This paper describes technologies that are used to develop application for online room booking within an organization. The basic functionality of the application is searching and making reservations for rooms according to certain criteria in order to organize an event. The application is designed in Java programming language in accordance with the *MVC* architecture and using *MySQL* database.

The paper describes the basic principles of the technologies used and give a short history of those technologies. In this paper we described the model, architecture, implementation and use cases of web application for online room reservation. We introduced the basic principles of *MVC* architecture and the way it is applied, we listed examples of the application itself and examples from the database as well as examples of basic protection principles.

Key words: application, reservation, room, event, online, web, Java, MVC, NetBeans, MySQL, JavaScript, jQuery, class, servlet, JSP.

SADRŽAJ

1	Uvod.....	- 5 -
2	Metodologija istraživačkog rada.....	- 6 -
2.1	Obrazloženje rada i predmet istraživanja	- 6 -
2.2	Cilj istraživanja.....	- 6 -
2.3	Hipotetički okvir	- 6 -
2.4	Metode i tok istraživanja	- 7 -
3	Projektni zadatak	- 8 -
3.1	Tipovi korisnika.....	- 8 -
3.2	Funkcionalni zahtevi.....	- 9 -
4	Izbor tehnologija za realizaciju aplikacije.....	- 11 -
4.1	Desktop aplikacije.....	- 11 -
4.2	Veb aplikacije	- 12 -
4.2.1	Veb	- 12 -
5	Izbor programskog jezika	- 13 -
5.1	Java	- 13 -
5.2	PHP	- 14 -
6	HTML.....	- 15 -
6.1	Document Object Model (DOM)	- 16 -
6.2	HTML u aplikaciji.....	- 16 -
7	Kaskadni stilovi CSS	- 16 -
7.1	Istorija	- 17 -
7.2	Primena kaskadnog stila.....	- 17 -
8	JavaScript.....	- 19 -
8.1	jQuery.....	- 20 -
8.2	Asynchronous JavaScript and XML (AJAX)	- 21 -
9	Java Server Pages (JSP) i Servleti	- 22 -
9.1	Java Server Pages (JSP)	- 22 -
9.2	Servleti	- 22 -

10 Baze podataka.....	- 24 -
10.1.1 Savremeni sistemi baza podataka	- 24 -
10.1.2 Relacioni model.....	- 25 -
10.1.3 Sistemi za upravljanje bazama podataka.....	- 25 -
10.1.4 SQL	- 26 -
10.1.5 Povezivanje aplikacije sa bazom podataka	- 27 -
11 Model – View – Controler (MVC).....	- 28 -
12 Realizacija.....	- 30 -
12.1 Model	- 30 -
12.1.1 Primer klase: Klasa Korisnik.....	- 31 -
12.2 Prezentacioni deo (View).....	- 46 -
12.2.1 Primer JSPa: administrator.jsp	- 47 -
12.3 Kontroler (<i>Controller</i>).....	- 48 -
12.4 Paket util.....	- 52 -
12.5 Baza podataka.....	- 54 -
12.6 Zaštita.....	- 56 -
12.6.1 Kontrola pristupa.....	- 56 -
12.6.2 Šifrovanje lozinke.....	- 57 -
12.6.3 SQL injekcija (<i>SQL injection</i>)	- 58 -
13 Slučajevi korišćenja.....	- 59 -
13.1 Dodavanje novog korisnika.....	- 59 -
13.2 Prijavljivanje na sistem	- 59 -
13.3 Pregled prostorija, raspored i rezervacija.....	- 59 -
13.4 Premeštanje po prostorijama	- 60 -
14 Zaključak.....	- 62 -
Literatura	- 63 -

1 Uvod

Upravljanje prostornim resursima i vremenom predstavlja problem većini preduzeća i organizacija pri planiranju raznovrsnih događaja.

- Rasporede zauzetosti prostorija uglavnom kreira jedna osoba pa to zahteva previše vremena.
- Tehničke karakteristike potrebne za realizaciju događaja moraju biti unapred definisane.
- Svaka promena događaja uglavnom znači promenu i ostalih događaja.
- Nemogućnost odabira termina koji organizatoru najviše odgovara.
- Problemi oko otkazivanja događaja.
- Nedostupnost rasporeda, raspored se mora slati svima.
- Problemi oko dodavanja novih i izmene postojećih događaja za period za koji raspored već postoji.

Upotrebom odgovarajućih savremenih računarskih tehnologija moguće je značajno olakšati i pojednostaviti vremensku i tehničku iskorišćenost prostorija pa samim tim i olakšati organizaciju događaja :

- Raspored je dostupan svima koji imaju prvo rezervacije (korisnicima), nije potrebno slanje rasporeda i izmena svakome ponaosob.
- Korisnici mogu sami da biraju prostorije na osnovu tehničkih karateristika.
- Odabir termina je moguć samo ako je termin slobodan.
- Mogućnost odobravanja, odbijanja, izmene i otkazivanja rezervacija.
- Čuvanje podataka o tome ko je, kada i koju prostoriju koristio i u koju svrhu.

2 Metodologija istraživačkog rada

2.1 Obrazloženje rada i predmet istraživanja

U modernom svetu obrazovne ustanove, većina kompanija kao i pojedinci organizuju razne edukativne aktivnosti (seminare, radionice, kurseve, obuke itd). Jedan od najvećih izazova prilikom organizacije ovakvih događaja jeste pronalaženje termina i sala sa adekvatnom opremom i kapacitetom koje određeni događaj zahteva kao i vođenje evidencije o svim događajima jedne organizacije, zgrade, prostorije ili osobe.

U cilju realizacije aplikacije koja će služiti za rezervaciju prostorija na osnovu određenih kriterijuma dat je pregled i poređenje tehnologija koje će se koristiti.

2.2 Cilj istraživanja

Cilj ovog rada i same aplikacije je da olakša organizaciju događaja i korišćenje slobodnih prostorija na osnovu rasporeda zauzetosti, odabranog termina i da omogući bolju iskorišćenost prostora na osnovu specifičnih zahteva.

Na osnovu postojeće aplikacije za rezervaciju prostorija u okviru jedne visokoškolske ustanove prikazane su popularne tehnologije za razvoj savremene aplikacije, data su poređenja sa istorijskim osvrtom na razvoj tih tehnologija i predstavljen je način i postupak razvoja veb aplikacije korišćenjem modernih tehnologija i MVC arhitekture.

2.3 Hipotetički okvir

Opšta hipoteza: Cilj i svrha razvoja računara i računarskih tehnologija jeste olakšanje posla i života ljudima. Uvođenjem računarskih mreža i pojavom interneta ljudima je omogućena laka komunikacija i saradnja sa jednog na drugi kraj sveta i time su pojednostavljeni svakodnevni poslovi.

Posebna hipoteza: Korišćenjem modernih računarskih tehnologija i alata možemo automatizovati i pojednostaviti svakodnevne aktivnosti. Primenom ovih tehnologija na problem rezervacije odgovarajuće prostorije za održavanje raznih aktivnosti možemo značajno olakšati samu rezervaciju i čuvati sve relevantne podatke.

2.4 Metode i tok istraživanja

Pre početka projektovanja i realizacije sistema, potrebno je upoznati se sa osnovnim zahtevima i funkcionalnostima. Ovaj postupak se odvija u više faza koje na kraju kao rezultat daju konačni proizvod, aplikaciju:

1. Prikupljanje podataka o procesu organizacije događaja.
2. Istraživanje potreba i strukture organizacije.
3. Prikupljanje literature o računarskim tehnologijama.
4. Izdvajanje relevantnih tehnologija koje će se koristiti.
5. Razvoj veb aplikacije za rezervaciju prostorija.
6. Pisanje dokumentacije i korisničkog uputstva.

3 Projektni zadatak

Aplikacija mora da bude dostupna svim nastavnicima koji imaju kreirane korisničke naloge, da omogući prijavljivanje korisnika i spreči neovlašćeni pristup sistemu, da omogući pregled rasporeda i rezervaciju prostorija korisnicima, mora da koristi bazu podataka za čuvanje i manipulaciju podacima i da omogući administraciju naloga i celog sistema.

Pre početka razvoja baze podataka i samog kodovanja potrebno je precizno definisati osnovne funkcionalnosti same aplikacije i dokumentovati ih.

3.1 Tipovi korisnika

Aplikacija treba da ima tri vrste korisnika u zavisnosti od privilegija koje im treba omogućiti:

1. Administrator

- Može da dodaje nove lokacije, zgrade, prostorije, nastavnike, zvanja, volontere, moderatore i vrši izmenu i brisanje istih.
- Ima pregled svih prostorija
- Ima pregled celokupnog rasporeda aktivnosti i mogućnost štampanja rasporeda po danu ili nedelji za odabranu prostoriju.
- Može da vrši obradu pristiglih rezervacija (odobrava ili odbija).
- Ima uvid u sve rezervacije (nove i odobrene).
- Ima mogućnost rezervacije bez potrebe za odobrenjem.
- Može da vrši izmenu rezervacije (vreme, dan, prostorija, naziv itd.).
- Ima mogućnost pozivanja volontera na određeni događaj.
- Ima mogućnost slanja poruke volonterima i nastavnicima.

2. Moderator

- Ima pregled svih prostorija
- Ima pregled celokupnog rasporeda aktivnosti i mogućnost štampanja rasporeda po danu ili nedelji za odabranu prostoriju.
- Može da vrši obradu pristiglih rezervacija (odobrava ili odbija).
- Ima uvid u sve rezervacije (nove i odobrene).
- Ima mogućnost rezervacije bez potrebe za odobrenjem.
- Može da vrši izmenu rezervacije (vreme, dan, prostorija, naziv itd.).
- Ima mogućnost pozivanja volontera na određeni događaj.
- Ima mogućnost slanja poruke volonterima i nastavnicima.
- Može da promeni svoju adresu elektronske pošte i lozinku.

3. Nastavnik

- Ima pregled svih prostorija
- Ima pregled celokupnog rasporeda aktivnosti i mogućnost štampanja rasporeda po danu ili nedelji za odabranu prostoriju.
- Ima uvid u sve svoje rezervacije (nove i odobrene).
- Ima mogućnost rezervacije.
- Može da promeni svoju adresu elektronske pošte i lozinku.
- Ima pregled svih volontera i mogućnost slanja poruke volonteru.

3.2 **Funkcionalni zahtevi**

- Organizacija može poslovati u više zemalja, gradova i na više adresa pa se s toga mora voditi evidencija o svim zgradama, prostorijama i lokacijama na kojima se one nalaze.
- Za svaku prostoriju potrebno je navesti bitne osobine kao što su naziv, kapacitet, broj računara i u kojoj zgradi se prostorija nalazi.
- Za svaku prostoriju potrebno je prikazati zauzetost na osnovu odabranog datuma (prikazuju se samo događaji koji su prethodno odobreni).
- Svaki korisnik može da rezerviše prostoriju u terminu koji je slobodan.

- Prilikom rezervacije korisnik unosi podatke o tipu aktivnosti (naziv, tip rezervacije, očekivani broj studenata, potreban broj demonstratora i opis koji nije obavezan).
- Ako je rezervaciju izvršio nastavnik, ona se šalje moderator na odobrenje, a ako je rezervaciju izvršio moderator ili administrator ona je automatski odobrena.
- Za svaku novu rezervaciju koju izvrši nastavnik potrebno je elektronskom poštom obavestiti moderatora.
- Nakon što moderator ili administrator potvrdi ili odbije rezervaciju, potrebno je korisniku koji je izvršio rezervaciju poslati obaveštenje elektronskom poštom.
- Ukoliko rezervacija koju je izvršio nastavnik još uvek nije obrađena, nastavnik može da obriše svoju rezervaciju.

4 Izbor tehnologija za realizaciju aplikacije

Prilikom izbora tehnologija najbitniji stvar je definisanje strukture i arhitekture aplikacije. Izbor tehnologija se razlikuje u zavisnosti da li će aplikacija biti realizovana kao desktop varijanta koja mora da se dostavi svakom računaru na kojem će se aplikacija koristiti ili kao aplikacija dostupna putem mreže kojoj se pristupa preko pretraživača sa bilo kog računara koji ima pristup toj mreži (web aplikacija).

4.1 Desktop aplikacije

Desktop aplikacije su vrsta aplikacija koje je potrebno preneti a u velikom broju slučajeva i instalirati na svakom računaru posebno. Ako dođe do promene nekog dela programa, na svakom računaru pojedinačno koji koristi ovu aplikaciju mora se izvršiti ažuriranje ili instalacija nove verzije aplikacije.

Situacija se dodatno komplikuje ukoliko aplikacija radi sa podacima na lokalnom računaru. Ovo bi značilo da je osoba koja radi sa programom vezana za rad isključivo na svom računaru ili bi morala da sa sobom nosi sve podatke koji su joj potrebni za rad. Drugi bitan problem jeste tačnost i čuvanje istih podataka na više različitih mesta. Zamislite situaciju u kojoj više osoba koristi program za evidenciju robe jednog magacina na više računara. Ako na jednom računaru ubeležimo neku količinu robe koja je izašla iz magacina a na drugom robu koja je došla u magacin da li ćemo imati tačan podatak o tome koliko robe imamo u magacinu? Naravno da ne i zato se ovakvi sistemi danas slabo i koriste (osim za neke specifične namene i uređaje).

Postoje i desktop aplikacije koje koriste udaljene ili centralizovane baze podataka koje nalaze na serverima pa time olakšavaju pristup podacima i prevazilaze problem dostupnosti i tačnosti podataka. Današnje desktop aplikacije u najvećem broju slučajeva koriste ovakav pristup.

I pored prevazilaženja problema sa podacima, veliki problem i dalje predstavlja dostupnost aplikacije i potreba za instalacijom i ažuriranjem. U prošlosti su gotovo sve aplikacije bile ovog tipa, međutim, razvojem globalne svetske mreže i unapređenjem računarske moći i tehnologija sve više aplikacija se realizuju kao web aplikacije.

4.2 Veb aplikacije

4.2.1 Veb

Veb je reč koja se danas koristi kao zamena za globalnu svetsku mrežu (*World Wide Web* ili WWW). Veb je nastao u *CERN*ovoj laboratoriji u Švajcarskoj. Tim Berns Li je 1989. godine napisao prvi klijentski i serverski kod a prva veb stranica je objavljena 1991. godine. [4]

Zbog veoma brzog rasta i usavršavanja veba 1994. godine osnovana je organizacija *World Wide Web Consortium (W3C)* u cilju standardizacije. [4]

Prvi sadržaji na vebu bile su statičke tekstualne strane. Razvojem i unapređivanjem tehnologija menjale su se i potrebe i zahtevi korisnika. Danas na vebu prevladavaju dinamički sadržaji poznatiji kao veb aplikacije. [6]

Razlika između ranijih statičkih i današnjih dinamičkih sadržaja je u samoj aktivnosti korisnika i sadržaju koji se prikazuje i generiše. Korisnik može da uredi izgled i sadržaj koji mu se prikazuje na osnovu svojih potreba i sklonosti. Osim pregleda sadržaja, korisnik može i da vrši mnoge druge aktivnosti. Može da skladišti podatke, pretražuje materijale, šalje i prima elektronsku poštu, radi na raznim dokumentima i datotekama i sl. Najveća prednost veb aplikacija u poređenju sa desktop aplikacijama jeste mogućnost pristupa sa bilo koje lokacije koja ima pristup internetu. Veb aplikacijama se pristupa putem mreže korišćenjem pretraživača bez potrebe za instalacijom na klijentskoj strani. [6] Ukoliko dođe do neke izmene na samoj aplikaciji, dovoljno je da se ta promena implementira na jednom mestu, na serveru. Problemi kao što su tačnost i pristupačnost podacima koji se javljaju kod desktop aplikacija, kod veb aplikacija su prevaziđeni. Najpopularnije veb aplikacije danas su aplikacije za korišćenje elektronske pošte.

U zavisnosti od toga da li će aplikacija biti realizovana kao desktop, veb aplikacija ili kao aplikacija za neki specifični uređaj zavisi i izbor alata i tehnologija. Aplikacija za rezervaciju prostorija koja se navodi kao primer realizovana je kao veb aplikacija.

5 Izbor programskog jezika

Dva trenutno najpopularnija i najzastupljenija programska jezika za realizaciju veb aplikacija su *PHP* i *Java*.

5.1 *Java*

Tim "*Green Team*" pod vođstvom Džejsma Goslinga je 1991. godine započeo rad na novom programskom jeziku koji bi trebao da bude jednostavniji od jezika *C* i *C++* i koji bi imao sposobnost izvršavanja na različitim uređajima. Kao rezultat rada ovog tima 1995. godine objavljena je prva zvanična verzija *Jave*. [16]

Java je objektno orijentisani programski jezik visokog nivoa. Da bi se programski kod napisan na jeziku visokog nivoa izvršio na računaru, taj kod mora da se prevede na mašinski jezik procesora. Proces prevođenja na mašinski jezik obavlja prevodilac ili kompajler pa se često i ceo proces naziva kompajliranje. Izvršavanje *java* koda se razlikuje od izvršavanja koda drugih programskih jezika. *Java* istovremeno prevodi i izvršava red po red koda zahvaljujući interpretatoru. Interpretator je program koji uzima sledeću naredbu jezika visokog nivoa, određuje koje mašinske instrukcije treba da se izvrše i izvršava ih. Za razliku od ostalih jezika visokog nivoa, *Java* kod se ne prevodi na mašinski jezik stvarnog procesora već na mašinski jezik *Java* Virtuelne Mašine (JVM) koji se zove *Java* bajtkod. Glavna svrha *Javine* virtuelne mašine je da se *Java* kod može izvršavati na različitim uređajima sa različitim tipovima procesora. Dodavanjem brojnih komponenti i unapređivanjem *Jave*, *Java* je prerasla u platformu. [1, 16]

Java platforma je podeljena u tri celine u zavisnosti od namene programa:

- *Java Standard Edition (Java SE)* za desktop aplikacije
- *Java Micro Edition (Java ME)* za uređaje sa ograničenim resursima
- *Java Enterprise Edition (Java EE)* za distribuirane aplikacije

[1]

5.2 *PHP*

PHP je prvenstveno skript jezik za izradu dinamičkih veb aplikacija koji je prošao kroz dosta izmena od svog izvornog oblika.

Rasmus Lerdorf, je 1994. godine koristeći programski jezik C napravio skup alata po imenu *PHP/FI* za potrebe praćenja svog sajta. *PHP* je prvobitna skraćenica od "alati za ličnu prezentaciju" ili na engleskom "*Personal Home Page Tools*". Kasnije je ovaj alat postao dostupan korisnicima širom interneta. Kada je uvideo da su potrebe korisnika porasle, Rasmus je proširio osnovne funkcionalnosti i omogućio korišćenje baze podataka. Uvidevši da je *PHP* postao previše robustan za održavanje od strane jednog čoveka i da je dalji napredak spor, juna 1995. godine Rasmus je objavio izvorni kod *PHPa* koji je time dostupan programerima širom sveta u cilju unapređenja i ispravljanja grešaka. [31]

PHP je pretrpeo dosta izmena i prošao kroz nekoliko verzija. Već 1997. godine izlazi verzija *PHP 2.0* a ubrzo zatim i *PHP 3.0* koja je zvanično objavljena 1998. godine. *PHP 3.0* kreirali programeri iz Izraela, Zev Suraski i Endi Gutsman na osnovu *PHP 2.0* ali su ga dosta izmenili kada su otkrili veliki broj propusta. Od ove verzije *PHP* skraćenica se odnosi na "predprocesor hiperteksta" ili na engleskom "*Hypertext Preprocessor*". U želji da još više unaprede *PHP* i omoguće rad na većim projektima, Zev i Endi su odlučili da izmene celo jezgro *PHPa* i osnovali su kompaniju *Zend Technologies* koja nadgleda razvoj *PHPa*. Verzija *PHP 4.0*, zvanično je objavljena 2000. godine. *PHP 5* verzija je objavljena u julu 2004. godine i koristi jezgro *Zend* mašine 2.0. Najveće unapređenje u verziji 5 se ogleda u poboljšanju pristupa kod objektno orijentisanom programiranju. [31]

Kao i Java, i *PHP* koristi interpretator za izvršavanje svog koda. Postoje dve varijante korišćenja interpretatora za *PHP*:

1. Interpretator kao spoljašnja aplikacija
2. Interpretator kao modul veb-servisa

Aplikacija za rezervaciju prostorija koja je i predmet ovog rada izrađena je u programskom jeziku Java koristeći servlete i *JSP (Java Server Pages)* koji su deo celine *Java EE*.

6 HTML

U cilju standardizovanog i lakšeg prikazaja sadržaja na internetu 1991. godine nastao je jezik za označavanje hiperteksta (*HyperText Markup Language*) *HTML*. Do svog današnjeg oblika i verzije *HTML* je prošao kroz nekoliko verzija. [11]

Verzije *HTML*-a:

- *HTML* 1991. godina
- *HTML+* 1993. godina
- *HTML 2.0* 1995. godina
- *HTML 3.2* 1997. godina
- *HTML 4.0* 1997. godina
- *HTML 4.01* 1999. godina
- *HTML 5* 2008. godina

[11]

HTML dokument ima ekstenziju .html ili .htm i u njemu se nalaze tekst i oznake (tagovi). Ove oznake služe kao uputstvo koje određuje kako će se navedeni sadržaj prikazivati u pretraživaču.

Svaki *HTML* dokument počinje oznakom <html> koja predstavlja početak dela koji treba da se prikaže a završava se oznakom </html> koja označava kraj dokumenta. Ostale oznake i sadržaj se navode između ovih oznaka. Oznake <head> i </head> služe za definisanje osnovnih osobina dokumenta i sadržaj koji se ovde nalazi se ne prikazuje. Najčešće se na ovom mestu vrši povezivanje sa dokumentima i resursima koji nisu deo samog *HTML* dokumenta kao što su *CSS* dokumenti, *JavaScript* biblioteke i drugi. U okviru ovih oznaka se definiše i naziv prozora pretraživača koji se navodi između oznaka <title> i </title>.

Sadržaj koji se prikazuje u pretraživaču se definiše između oznaka <body> i </body>. Unutar ovih oznaka se pored teksta za prikaz definiše i način prikazivanja korišćenjem mnogobrojnih oznaka kao što su <div></div>, <p></p>, <textarea></textarea>, <form></form> i druge.

Od verzije 5.0 uvedeno je dosta novih oznaka u cilju lakšeg kreiranja dokumenata, bolje organizacije, lakšeg dodavanja sadržaja i radi bolje preglednosti. Neki od novih oznaka su `<article></article>`, `<section></section>`, `<video></video>`, `<header></header>`, `<footer></footer>` i drugi. [12, 27]



Slika 1: Izgled HTML dokumenta

6.1 Document Object Model (DOM)

Document Object Model (DOM) je API za *HTML* i *XML* dokumente koji je nastao iz dinamičkog *HTMLa*. Njegova uloga je da definiše logičku strukturu, način pristupa i manipulaciju *HTMLa* i *XMLa*. [21]

HTML se koristi kao prezentacioni sloj u *Web* aplikacijama, pa se pojavljuje i u *JSP*, *ASP*, *PHP* aplikacijama u zavisnosti od serverske tehnologije. [3]

6.2 HTML u aplikaciji

Kako je aplikacija za rezervaciju prostorija realizovana pomoću *JSP* tehnologije tako se i prezentacioni sloj, *HTML*, nalazi unutar *JSP* strana. Osim *HTML* tagova i sadržaja, na *JSP* stranama u aplikaciji su korišćeni i skripteti i *JavaScript* funkcije.

7 Kaskadni stilovi CSS

Izgled *HTML* elemenata nije unapred definisan. Većina oznaka nema automatski definisanu veličinu, boju, font i druge elemente koji čine izgled jednog elementa. Da bi se

postigao željeni izgled, dizajn, potrebno je definisati stil korišćenjem kaskadnih stilova *CSS (Cascading Style Sheet)*.

7.1 Istorija

Da bi se omogućila promena izgleda *HTML* elemenata 1996. godine objavljena je prva verzija kaskadnih stilova *CSS 1* od strane organizacije *World Wide Web Consortium (W3C)*. Uvođenjem kaskadnih stilova omogućena je promena boja, fontova, margina, dimenzija, pozicioniranje i druge stvari. U narednim godina *CSS* je doživeo širenje i unapređenje pa samim tim postoji i više verzija:

- *CSS 1* 1996. godina
- *CSS 2* 1998. godina
- *CSS 2.1*
- *CSS 3* (aktuelni stil, sastavni deo *HTMLa 5*) 1999. godina

[14]

7.2 Primena kaskadnog stila

Kaskadni stil se može primeniti na više načina. Jedan od načina je da se oznaci direktno dodeli izgled koristeći atribut *style* u kome se navode atributi izgleda.

```
<div style="border: solid black 1px; background-color: paleturquoise; height: 500px;">
</div>
```

Slika 2: Dodeljivanje stila *div* element u samoj oznaci

Drugi način je kreiranje posebnog dokumenta u kojem će se nalaziti samo *CSS* kod i koji ima ekstenziju *.css*. Da bi odredili kojem elementu se dodeljuje stil moramo da znamo *id*, klasu ili samu oznaku tog elementa. Ukoliko stil primenjujemo korišćenjem identifikatora elementa (*id*), prvo moramo da navedemo znak “#” iza kojeg sledi identifikator. Ako koristimo klasu, umesto znaka “#” koristi se znak “.”; a ako koristimo samu oznaku nije potrebno navoditi bilo šta isred nje. [15]

```
#ID{
width: 55%;
height: 17%;
margin-left: 23%;
margin-right: 23%;
margin-top: 1%;
border: solid 2px black;
}

.KLASA{
max-height: 20%;
min-height: 20%;
cursor: pointer;
}

OZNAKA{
height: 98%;
width: 99%;
background-color: #d3d3d3;
}
```

Slika 3: Primer *css* dokumenta

Ako koristimo poseban document za stil, onda moramo uključiti (“linkovati”) taj document u *HTML* dokumentu kako bi se stil primenio. Uključivanje stila u *HTML* dokument se vrši navođenjem putanje CSS dokumenta u oznaci link.

```
<head>
  <title>NASLOV</title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link type="text/css" rel="stylesheet" href="profil.css" />
  <link rel="stylesheet" href="http://code.jquery.com/ui/1.10.2/themes/smoothness/jquery-ui.css" />
</head>
```

Slika 4: Povezivanje css dokumenta u HTML dokumentu

8 JavaScript

Juna 1995. godine Brendan Ajh je napravio *JavaScript* koji je tada nosio naziv *Mocha*. Tokom svog razvoja *JavaScript* je više puta menjao svoje ime pa je bio poznat i pod imenom *LiveScript*.

JavaScript (JS) je skript programski jezik koji se nalazi u gotovo svakoj veb aplikaciji. Kod *JavaScripta* se, za razliku od Java jezika, izvršava na klijentskoj strani u pretraživaču. Osnovna svrha JSa je da učini veb stranicu dinamičnom, odnosno da se izvršavanjem neke korisničke akcije poziva odgovarajuća funkcija.

Funkcije *JavaScripta* se pišu unutar oznaka `<script>` `</script>` ili u posebnom dokumentu sa ekstenzijom `.js`. Svaka funkcija počinje rezervisanom rečju `function` iza koje se nalazi ime funkcije, zatim idu zagrade za argumente (ove zagrade su obavezne bilo da postoje argumenti ili ne) i na kraju otvorena i zatvorena vitičasta zagrada (blok) između kojih se piše kod.

Pozivanje *JavaScript* funkcija se može uraditi na više načina. Neki od najčešće korišćenih su klikom na neki objekat, prilikom učitavanja, prilikom promene nekog parametra ili nekom drugom akcijom. [19]



```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <script>
      function sumaBrojeva(x,y) {
        z = x + y;
        alert('x + y = ' + z);
      }
    </script>
    <input type="button" value="Suma" onclick="sumaBrojeva(3,2)"/>
  </body>
</html>
```

Slika 5: Primer JavaScript funkcije za sabiranje dva broja u HTML dokumentu

8.1 *jQuery*

jQuery je *JavaScript* biblioteka nastala 2005. godine a zvanično je objavljena januara 2006. godine. [34] Ova biblioteka je nastala sa ciljem da se znatno olakša pisanje *JavaScript* funkcija bez predugačkih selektora i da olakša komunikaciju između *JavaScript* funkcija i drugih jezika. [20]

jQuery funkcije se, kao i *JavaScript* funkcije, pišu unutar oznaka `<script></script>`, a sama *jQuery* biblioteka se dodaje dodavanjem atributa `src` i putanje do biblioteke.



```
<head>
  <title>jQuery</title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <script src="//code.jquery.com/ui/1.11.2/jquery-ui.js"></script>
  <link rel="stylesheet" href="//code.jquery.com/ui/1.11.2/themes/smoothness/jquery-ui.css">
</head>
```

Slika 6: Uključivanje *jQuery* biblioteke u HTML dokumentu

U dokumentu je lako primetiti korišćenje *jQuery* funkcija zbog znaka dolar (\$) koji označava da se radi o *jQuery* biblioteci. Umesto znaka dolar može se napisati `jquery` i funkcija će funkcionisati potpuno identično. Nakon dolara slede zagradu u kojima se upisuje selektor a zatim funkcija koja se poziva ili definiše.

U *jQuery* biblioteci se nalaze elementi za:

- *HTML/DOM* manipulaciju
- *CSS* manipulaciju
- Efekte i animaciju
- *AJAX*

Zbog svoje jednostavnosti i funkcionalnosti *jQuery* biblioteka je i danas u masovnoj upotrebi pa je koriste i neke od najvećih kompanija kao što su *Google*, *IBM*, *Microsoft* i mnoge druge. Gotovo svi današnji pretraživači imaju ugrađenu podršku za *jQuery*. [20]

8.2 *Asynchronous JavaScript and XML (AJAX)*

AJAX (Asynchronous JavaScript and XML) predstavlja tehnologiju za asinhrono osvežavanje veb stranica. Ukoliko veb aplikacija radi bez ajaxa onda bi za svaku aktivnost korisnika koja ima komunikaciju sa serverom korisnik morao da čeka odgovor sa servera i ponovno učitavanje stranica kako bi nastavio sa radom. Ako koristimo *AJAX*, samo delovi stranice na koje je korisnička aktivnost imala uticaja bi se izmenili dok bi korisnik moga da nastavi sa radom na ostatku strane. Od ovakvog asinhronog načina osvežavanja stranica i potiče deo naziva *AJAXa*. [35]

Učitavanjem stranice na klijentskoj strani učitava se i *JavaScript* mašina (engine). Ovaj engine salje samo važne podatke o promenama serveru u *XML* formatu. Server vraća odgovor samo sa podacima koji treba da se izmene takođe u *XML* formatu nakon čega mašina (engine) izvršava pristigle izmene. [35]

AJAX predstavlja jednu od osnovnih tehnologija složenih Internet veb 2.0 aplikacija i podržan je od strane gotovo svih frejmworka. [3]

9 *Java Server Pages (JSP) i Servleti*

Java Server Pages (JSP) i *Servleti* su sastavni deo *Java EE* celine. Njihova uloga je da omoguće jednostavno i brzo kreiranje dinamičkih veb aplikacija.

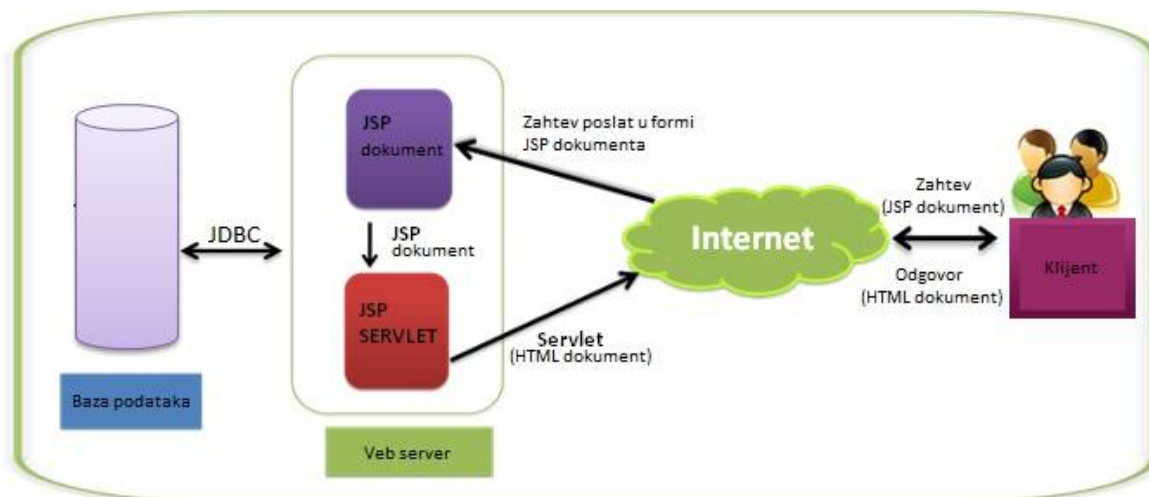
9.1 *Java Server Pages (JSP)*

Kao i *HTML* dokumenti i *JSP* predstavlja grafički interfejs u *Java* veb aplikacijama. Ova tehnologija omogućava jednostavno i brzo kreiranje dinamičkih strana koje se prikazuju korisniku. Za razliku od *HTML* dokumenata u *JSPu* se pored *HTML* tagova i sadržaja može pisati i *Java* kod koji se tada naziva skriptlet. Skriptleti se pišu između oznaka `<% %>`. Uvođenjem *ELa* (*expression language*) olakšano je korišćenje programskog koda u *JSP* dokumentima, međutim skriptleti su i dalje u upotrebi. Osim *HTMLa* i *Java* koda u *JSPu* se mogu pisati *JavaScript* funkcije, *XML*, *EL* itd. Za komunikaciju između *JSPa* i servera zaduženi su *servleti*. [5]

9.2 *Servleti*

Servleti su *Java* klase koje se nalaze na serverskoj strani. *Servleti* su zaduženi za obradu zahteva (*request*) koji su pristigli sa klijentske strane, dinamičko generisanje sadržaja i slanje odgovora klijentu (*response*). Na serverskoj strani može da se nalazi više različitih *servleta*. Odluka o tome koji *servlet* će preuzeti obradu donosi se na osnovu pristiglog zahteva. Svaki *servlet* mora da implementira interfejs *Servlet*. Korišćenjem klase *HttpServlet* dodaju se dodatni metodi kao što su *doGet* i *doPost*. Interfejsi i klase koje proširuju *servlete* i njihove funkcionalnosti nalaze se u paketu *javax.servlet*.

Kao podrška *servletima* koriste se serveri kao što su *Tomcat* i *GlassFish*. [5]



Slika 7: Prijem zahteva, obrada i slanje odgovora korišćenjem JSPa i Servleta
(preuzeto sa sajta <http://www.java4study.com/JSP/JSP2.HTML>)

10 Baze podataka

Prvi sistemi i programi koji su se izvršavali na računaru bili su zasnovani na sistemu datoteka u koje su unosili, čitali i menjali podatke. Sistem datoteka je bio veoma nepouzdan zbog čestih grešaka koje su se javljale prilikom rada sa podacima. Pored nepouzdanosti, ovi sistemi su imali i veliki broj ograničenja, programi su bili zavisni od podataka, sam razvoj sistema je bio veoma složen a javljala se i velika redudansa podataka pa je samim tim bilo naporno održavati ovakav sistem. [2]

10.1.1 Savremeni sistemi baza podataka

Razvojem tehnologija, računara i računarskih programa javljala se sve veća potreba za efikasnijim načinom upravljanja podacima i za razdvajanjem podataka od samog programa. [2]

Usled nedostataka sistema datoteka šezdesetih godina dvadesetog veka javljaju se prvi sistemi baza podataka. Sistemi datoteka su i dalje bili u upotrebi pošto su se baze podataka koristile za složenije projekte kao što je bilo sletanje Apola na Mesec. Osamdesetih godina dvadesetog veka dolazi do razvoja relacionih sistema baza podataka i prevazilaženja brojnih ograničenja. Zahvaljujući relacionom modelu baze podataka su osamdesetih godina dvadesetog veka doživele veliku komercionalnu upotrebu.

Danas, kada kažemo baza podataka to se odnosi na ceo sistem za upravljanje logički povezanih podataka. Takvi sistemi omogućavaju unos, čitanje i izmenu podataka, kontrolišu prava pristupa korisnika, ograničavaju i onemogućavaju pogrešan unos itd. Baza podataka se prilagođava potrebama korisnika, potrebama sistema ili potrebama organizacija. U današnjim bazama podataka pored teksta se sve više čuvaju i drugi dokumenti kao što su slike, video zapisi, tonski zapisi itd. [2, 8]

Postoje četiri osnovna modela baza podataka:

- Hijerarhijski model
- Mrežni model
- Relacioni model
- Objektni model

[2]

10.1.2 Relacioni model

Sam naziv ovog modela potiče od načina organizacije podataka. Podaci su organizovani po tabelama koje se drugačije nazivaju relacije. Relacije predstavljaju entitete i sastoje se od zapisa (redova) i atributa (kolona). Relacione baze podataka se zasnivaju na strogom matematičkom modelu koji omogućava jednostavno čitanje, izmenu i skladištenje podataka. [2]

U relacionim bazama podataka svaki zapis mora da bude jedinstven. Jedinstvenost se postiže određivanjem jedinstvenog identifikatora koji se naziva primarni ključ. Primarni ključ može biti jedan atribut ili kombinacija više atributa. [2]

Svaka tabela predstavlja jedan entitet pa kako entiteti mogu biti povezani tako se povezuju i tabele. Da bi povezali dve tabele potreban nam je strani ključ (*Foreign Key, FK*). Strani ili spoljašnji ključ je atribut čiji je domen vrednosti zadat nad domenom primarnog ključa neke tabele. Kod povezivanja tabela mora se voditi računa o referencijalnom integritetu. Referencijalni integritet je ograničenje koje ne dozvoljava da se unese pogrešna vrednost, odnosno da se kao strani ključ unese vrednost koja nije primarni ključ tabele sa kojom je tabela povezana. Nepravilnom izmenom ili brisanjem zapisa može doći do narušavanja referencijalnog integriteta ali moderne baze podataka uglavnom same vode računa o tome. [2]

10.1.3 Sistemi za upravljanje bazama podataka

Sistemi za upravljanje bazama podataka (*Database Management System, DBMS*) su softverski sistemi koji posebno projektovani da obezbede jednostavan i siguran rad sa bazama podataka. Osnovne funkcionalnosti ovakvih sistema su kreiranje, izmena, pregled i održavanje baza podataka. *DBMS* deluje kao posrednik između aplikacije koja koristi bazu podataka i same baze a poseduju i mehanizme za zaštitu podataka. Jedan od osnovnih načina zaštite jeste dodeljivanje privilegija korisnicima u zavisnosti od uloge koju imaju u sistemu. [2, 8]

10.1.4 SQL

Structured Query Language (SQL) je relacioni upitni jezik, koji je standardizovala organizacija ANSI. Prva verzija ovog jezika nastala je sedamdesetih godina dvadesetog veka od strane Donalda Čemberlena i Rejmonda Bojsa u IBMu i nosila je ime *SEQUEL* (*Structured English Query Language*). Današnja *Oracle* kompanija je 1979. godine predstavila novu verziju koja nosi ime *SQL* i koja je bila namenjena za komercijalnu upotrebu. Godine 1986. *SQL* je usvojen kao standard i prva verzija je poznata pod nazivom *SQL-86*. Složenost *SQL*-a je rasla dodavanjem novih funkcionalnosti, naročito uvođenjem koncepta objektna tehnologije i mehanizama kao sto su trigeri 1999. godine. Ovaj jezik služi za kreiranje tabela, veza između tabela i za manipulaciju podacima. [2]

10.1.4.1 SQL naredbe

U zavisnosti od svrhe, sve SQL naredbe se mogu podeliti u tri grupe:

1. Osnovne naredbe za definiciju (*Data Definition Language, DDL*):

- *CREATE DATABASE* imeBaze; (Kreiranje nove baze)
- *CREATE imeTabele { imeKolone1 tipKolone1 (veličina)*
imeKolone2 tipKolone2 (veličina)
imeKolone3 tipKolone3 (veličina)
}; (Kreiranje nove tabele)

[9]

2. Osnovne naredbe za manipulaciju (*Data Manipulation Language, DML*):

UPDATE imeTabele

SET imeKolone = vrednost

WHERE imeKolone = nekaVrednost; (Ažuriranje vrednosti)

INSERT INTO imeTabele (imeKolone1,imeKolone2,...)

VALUES (vrednost1,vrednost2,...); (Novi zapis u tabeli)

SELECT (imeKolone ili viseKolona) FROM imeTabele;

(Čitanje podataka iz tabele) [9]

3. Osnovne naredbe za kontrolu pristupa (*Data Control Language, DCL*):

GRANT privilegije
ON tabela (tabele)
[WITH GRANT OPTION] (Dodeljivanje prava)

REVOKE privilege_name
ON tabela(tabele+)
FROM korisnik; (Uklanjanje prava)

[36]

10.1.5 Povezivanje aplikacije sa bazom podataka

Aplikacija i baza su dva odvojena sistema, u aplikaciji se nalazi poslovna logika a podaci u bazi. Da bi ova dva sistema mogla da komuniciraju moraju predhodno da se povežu. Za povezivanje *Java* aplikacije sa bazom koristi se *JDBC* drajver koji je deo *JavaEE* specifikacije. Osim drajvera potrebno je uneti i parametre, korisničko ime i lozinku, za pristup bazi. [10]

Osnovne klase *JDBC*a i njihove uloge:

- *java.sql.DriverManager*:

Učitava drajver i pruža podršku za uspostavljanje nove konekcije

- *java.sql.Connection*:

Predstavlja konekciju sa određenom bazom

- *java.sql.Statement* :

Predstavlja kontejner za izvršavanje *SQL* naredbi u okviru postojeće konekcije.

java.sql.Statement je interfejs sa dva važna podtipa:

java.sql.PreparedStatement

Služi za izvršavanje prekompajlirane *SQL* naredbe

java.sql.CallableStatement

Služi za pozivanje stored procedure u okviru baze podataka

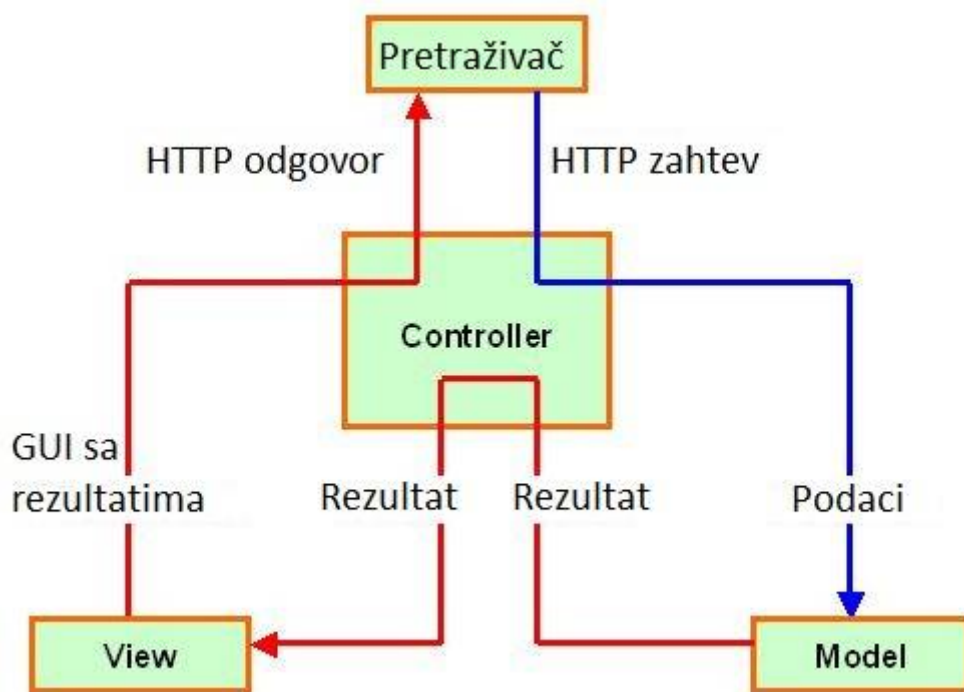
- *java.sql.ResultSet* :

Kontroliše pristup rezultatima dobijenim izvršavanjem *SQL* naredbe

[3]

11 Model – View – Controller (MVC)

Kod ranijih, desktop i veb, aplikacija ceo sistem je bio jedna celina. Poslovna logika i podaci nisu bili razdvojeni pa je projektovanje, razvoj, održavanje i menjanje aplikacija bio težak i dugotrajan proces. Kako bi se olakšao razvoj i održavanje aplikacija došlo se na ideju da se kretira model po kojem di aplikacija bila podeljena u tri celine. Prva celina služila bi za rad sa bazom podataka i kao poslovna logika (*Model*), drugi deo bio bi zadužen za prezentaciju rezultata korisnicima (*View*) a treći deo bi obrađivao pristigle zahteve i i omogućavao da aplikacija funkcioniše kao jedna celina (*Controller*).



Slika 8: Komunikacija između slojeva MVC modela

[\[http://www.j2mesalsa.com/struts/mvc.php/\]](http://www.j2mesalsa.com/struts/mvc.php/)

Kompanija *Xerox* je prva kompanija koja je primenila *MVC* model i to u biblioteci klasa jezika *Smalltalk-80*. I ako je najpre zamišljen kao koncept za desktop aplikacije, *MVC* je kasnije prilagođen za veb aplikacije pa je i danas jedna od najčešće korišćenih arhitektura. [25]

Prvi zvanični rad u kojem je opisana *MVC* arhitektura izašao je 1988. godine i nosio je naziv "*A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk -80*" a autori su bili Glen Krasner i Stiven Poup. [26]

MVC se sastoji od tri, funkcionalno odvojena dela od kojih svaki ima posebnu ulogu. Delovi MVC modela su :

- *Model* (Model)
- *View* (Prezentacioni sloj)
- *Controller* (Kontroler)

[25]

12 Realizacija

Aplikacija za rezervaciju prostorija je realizovana po *MVC* modelu u Java programskom jeziku korišćenjem *JSP* i servlet tehnologija (*Java EE*). Aplikacije je rađena u *NetBeans* 8.0.2 okruženju i korišćenjem *MySQL* servera baza podataka.

12.1 Model

Model predstavlja deo aplikacije koji je zadužen za rad sa podacim i za poslovnu logiku. U veb aplikaciji za rezervaciju prostorija model predstavljaju Java klase grupisane u paketu *model*.



Slika 9: Klase iz aplikacije za rezervaciju prostorija u paketu *model*

Sve klase pisane su u skladu sa objektno orijentisanom paradigmom i svaka klasa je zatvorenog tipa, izvršena je enkapsulacija. Enkapsulacija je način definisanja članova klase pri kojem se ne dozvoljava direktan pristup članovima van same klase u cilju sakrivanja funkcionalnih detalja. Sakrivanje se postiže definisanjem člana klase kao privatnog člana korišćenjem službene reči *private*. Privatni članovi se mogu koristiti samo unutar klase u okviru koje su definisani. Ukoliko je neki atribut definisan kao privatni član njemu ne može da se pristupi direktno van klase. Privatnim atributima se van klase pristupa korišćenjem getera i setera. Geteri i Seteri su metode koje služe za pristup privatnim atributima, geteri (*get*) služe za vraćanje vrednosti atributa dok seteri (*set*) služi za postavljanje vrednosti atributa. Nazivi ovih metoda po nepisanom pravilu počinju sa *get* ili *set* a u nastavku je ime atributa na koji se odnose (*getIme()*, *setIme("Milan")*).

U ovim klasama se nalaze atributi, konstruktori i metode. U skoro svakoj klasi definisane su *CRUD* metode i metode za obradu podataka koje predstavljaju poslovnu logiku. *CRUD* (*Create Read Update Delete*) predstavlja anagram koji se odnosi na operacije sa podacima iz baze podataka.

12.1.1 Primer klase: Klasa Korisnik

Klasa Korisnik je najvažnija klasa u aplikaciji jer bez nje korisnici ne bi postojali i sama aplikacija ne bi imala smisla. Pošto smo definisali da postoje tri tipa korisnika, administrator, moderator i nastavnik, u ovoj klasi se nalaze metode koje se odnose na sve tri vrste korisnika a u zavisnosti od tipa određeno je koji korisnik koje metode može da poziva.

12.1.1.1 Atributi

U klasi Korisnik se nalaze sledeći atributi:

- id - jedinstveni identifikator tipa Integer iz baze podataka dobijen autoincrement funkcijom.
- korisnickoIme – korisničko ime koje korisnik koristi pri prijavljivanju na sistem tipa String.
- Lozinka – lozinka za prijavljivanje na sistem tipa String.
- ime – ime korisnika tipa String.
- prezime – prezime korisnika tipa String.
- email – elektronska adresa korisnika (E-mail) na koju stižu obaveštenja o rezervacijama i tipa je String.
- statusID – služi da se odredi da li je korisnik aktivan ili ne. Tipa je Integer.
- tipKorisnikaID – označava da li je korisnik administrator, moderator ili nastavnik. Tip ovog atributa je Integer.
- zvanjeID – određuje koje zvanje korisnik ima u ustanovi u kojoj radi i tipa je Integer.

Svaki od ovih atributa je privatni član klase i svakom atributu pristup je omogućen preko getera i setera.

12.1.1.2 Metode

Klasa *Korisnik* pored getera i setera sadrži i sledeće metode:

- *public Korisnik()* – prazan konstruktor
- *public Korisnik(String korisnickoIme, String lozinka, String ime, String prezime, String email, Integer statusID, Integer tipKorisnikaID, Integer zvanjeID)* – konstruktor za dodelu vrednosti na osnovu prosleđenih argumenata.
- *public Korisnik(ResultSet rs)* – Konstruktor za dodelu vrednosti na osnovu rezultata dobijenih upitom nad bazom podataka.
- *public static boolean dodajKorisnika(String ime, String prezime, String korisnickoIme, String lozinka, String email, Integer tip_korisnika_id, Integer status_korisnika_id, Integer zvanje_id)* – za dodavanje (kreiranje) novog korisnika sa zadatim argumentima.
- *public static Korisnik login(String korisnickoIme, String lozinka)* – za proveru argumenata i pronalaženje korisnika sa zadatim korisničkim imenom i lozinkom pri prijavljivanju na sistem.
- *public String tipKorisnika(Integer id)* – vraća tip korisnika na osnovu prosleđenog identifikatora.
- *public Boolean korisnickoImePostoji(String korisnickoIme)* – prilikom dodavanja novog korisnika proverava da li već postoji korisnik sa tim korisničkim imenom pošto ne mogu postojati dva ili više korisnika sa istim korisničkim imenom.
- *public static Korisnik nadjiKorisnikaPoID(Integer id)* – pronalazi korisnika u bazi na osnovu identifikatora.
- *public static Boolean nadjiKorisnickiEmail(String email)* – proverava prilikom dodavanja novog korisnika da li je adresa elektronske pošte već u upotrebi.
- *public static boolean izmeniKorisnika(Integer id, String ime, String prezime, String korisnickoIme, String email, Integer tip_korisnika_id, Integer status_korisnika_id, Integer zvanje_id)* – menja podatke o korisniku na osnovu prosleđenih argumenata.

- *public List<Korisnik> sviKorisniciPoTipu(Integer t)* – vraća listu svih korisnika za određeni tip (sve administratore, sve moderatore, sve nastavnike).
- *public static List<Korisnik> sviKorisnici()* – vraća listu svih korisnika iz baze podataka.
- *public static List<Korisnik> sviKorisniciBezAdmina()* – vraća listu svih korisnika koji nisu administratori.
- *public static boolean obrisiKorisnika(Integer id)* – briše korisnika sa zadatim identifikatorom iz baze podataka.
- *public static boolean izmeniLozinku(String lozinka, Integer id)* – korisniku sa prosleđenim identifikatorom se postavlja prosleđena lozinka kao nova.
- *public static boolean izmeniStatus(Integer statusId, Integer id)* – služi za promenu statusa korisnika (aktivan ili neaktivan).
- *public static List<Korisnik> sviAktivniKorisnici()* – vraća listu svih aktivnih korisnika.
- *public static List<Korisnik> sviDeaktiviraniKorisnici()* – vraća listu svih korisnika koji nisu aktivni.
- *public static List<Korisnik> sviAdministratoriModeratori()* – rezultat je lista svih administratora i moderatora.
- *public static List<Korisnik> sviModeratori()* – vraća spisak svih moderatora.
- *public static Korisnik korisnikPoId(List<Korisnik> korisnici, int korisnikId)* – pronalazi korisnika u prosleđenoj listi na osnovu identifikatora.

```
package model;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import javax.websocket.Decoder;
import org.apache.commons.codec.digest.DigestUtils;

public class Korisnik {

    private Integer id;
    private String korisnickoIme;
    private String lozinka;
    private String ime;
    private String prezime;
    private String email;
    private Integer statusID;
    private Integer tipKorisnikaID;
    private Integer zvanjeID;

    public Korisnik() {
    }

    public Korisnik(String korisnickoIme, String lozinka, String ime, String prezime,
        String email, Integer statusID, Integer tipKorisnikaID, Integer zvanjeID) {
        this.korisnickoIme = korisnickoIme;
        this.lozinka = lozinka;
        this.ime = ime;
        this.prezime = prezime;
        this.email = email;
        this.statusID = statusID;
        this.tipKorisnikaID = tipKorisnikaID;
        this.zvanjeID = zvanjeID;
    }

    public Korisnik(ResultSet rs) {
        try {
            this.id = rs.getInt("id");
            this.ime = rs.getString("ime");
            this.prezime = rs.getString("prezime");
            this.korisnickoIme = rs.getString("korisnicko_ime");
            this.lozinka = rs.getString("lozinka");
        }
    }
}
```

```
        this.email = rs.getString("email");
        this.statusID = rs.getInt("status_korisnika_id");
        this.tipKorisnikaID = rs.getInt("tip_korisnika_id");
        this.zvanjeID = rs.getInt("zvanje_id");
    } catch (Exception e) {
        this.id = 0;
        this.ime = "";
        this.prezime = "";
        this.korisnickoIme = "";
        this.lozinka = "";
        this.email = "";
        this.statusID = 0;
        this.tipKorisnikaID = 0;
        this.zvanjeID = 0;
    }
}

public Integer getId() {
    return id;
}

public void setId(Integer id) {
    this.id = id;
}

public String getKorisnickoIme() {
    return korisnickoIme;
}

public void setKorisnickoIme(String korisnickoIme) {
    this.korisnickoIme = korisnickoIme;
}

public String getLozinka() {
    return lozinka;
}

public void setLozinka(String lozinka) {
    this.lozinka = lozinka;
}

public String getIme() {
    return ime;
}
```

```
public void setTime(String ime) {  
    this.ime = ime;  
}  
  
public String getPrezime() {  
    return prezime;  
}  
  
public void setPrezime(String prezime) {  
    this.prezime = prezime;  
}  
  
public String getEmail() {  
    return email;  
}  
  
public void setEmail(String email) {  
    this.email = email;  
}  
  
public Integer getStatusID() {  
    return statusID;  
}  
  
public void setStatusID(Integer statusID) {  
    this.statusID = statusID;  
}  
  
public Integer getZvanjeID() {  
    return zvanjeID;  
}  
  
public void setZvanjeID(Integer zvanjeID) {  
    this.zvanjeID = zvanjeID;  
}  
  
public Integer getTipKorisnika() {  
    return tipKorisnikaID;  
}  
  
public void setTipKorisnika(Integer tipKorisnika) {  
    this.tipKorisnikaID = tipKorisnika;  
}
```

```
public static boolean dodajKorisnika(String ime, String prezime, String korisnickoIme, String lozinka,
    String email, Integer tip_korisnika_id, Integer status_korisnika_id, Integer zvanje_id) {
    try {
        String SQL = "INSERT INTO korisnik (korisnicko_ime, lozinka, ime, prezime, "
            + "email, tip_korisnika_id, status_korisnika_id, zvanje_id) VALUES (?, ?, ?, ?, ?, ?, ?, ?) ";
        Connection con = util.ConnectionManager.getConnetcion();
        PreparedStatement ps = con.prepareStatement(SQL);
        ps.setString(1, korisnickoIme);
        ps.setString(2, DigestUtils.sha1Hex(lozinka));
        ps.setString(3, ime);
        ps.setString(4, prezime);
        ps.setString(5, email);
        ps.setInt(6, tip_korisnika_id);
        ps.setInt(7, status_korisnika_id);
        ps.setInt(8, zvanje_id);
        ps.executeUpdate();
        ps.close();
    } catch (Exception e) {
        return false;
    } finally {
        util.ConnectionManager.closeConnection();
    }
    return true;
}
```

```
public static Korisnik login(String korisnickoIme, String lozinka) {
    Korisnik k;
    try {
        String tipIStatus = "";
        String SQL = "SELECT * FROM korisnik WHERE korisnicko_ime=? AND lozinka=?";
        Connection con = util.ConnectionManager.getConnetcion();
        PreparedStatement ps = con.prepareStatement(SQL);
        ps.setString(1, korisnickoIme);
        ps.setString(2, DigestUtils.sha1Hex(lozinka));
        ResultSet rs = ps.executeQuery();

        if (rs.next()) {
            k = new Korisnik(rs);
        } else {
            k = null;
        }
        ps.close();
    } catch (Exception e) {
        k = null;
    }
}
```

```
    } finally {  
        util.ConnectionManager.closeConnection();  
    }  
    return k;  
}  
  
///  
public String tipKorisnika(Integer id) {  
    String tip;  
    try {  
        String SQL = "SELECT * FROM tip_korisnika WHERE id=?";  
        Connection con = util.ConnectionManager.getConnection();  
        PreparedStatement ps = con.prepareStatement(SQL);  
        ps.setInt(1, id);  
        ResultSet rs = ps.executeQuery();  
        if (rs.next()) {  
            tip = rs.getString("naziv");  
        } else {  
            tip = null;  
        }  
        ps.close();  
        rs.close();  
    } catch (Exception e) {  
        tip = "";  
    } finally {  
        util.ConnectionManager.closeConnection(); return tip;  
    }  
  
public Boolean korisnickoImePostoji(String korisnickoIme) {  
    Boolean postoji = false;  
    try {  
        String SQL = "SELECT * FROM korisnik WHERE korisnicko_ime=?";  
        Connection con = util.ConnectionManager.getConnection();  
        PreparedStatement ps = con.prepareStatement(SQL);  
        ps.setString(1, korisnickoIme);  
        ResultSet rs = ps.executeQuery();  
  
        if (rs.next()) {  
            postoji = true;  
        } else {  
            postoji = false;  
        }  
        ps.close();  
        rs.close();  
    } catch (Exception e) {  
    } finally {  
        util.ConnectionManager.closeConnection();  
    }  
}
```

```
    }  
    return postoji;  
}  
  
public static Korisnik nadjiKorisnikaPoID(Integer id) {  
    Korisnik k = new Korisnik();  
    try {  
        String SQL = "SELECT * FROM korisnik WHERE id=?";  
        PreparedStatement ps = util.ConnectionManager.getConnection().prepareStatement(SQL);  
        ps.setInt(1, id);  
        ResultSet rs = ps.executeQuery();  
        if (rs.next()) {  
            k = new Korisnik(rs);  
        }  
        ps.close();  
        rs.close();  
    } catch (Exception e) {  
        k = null;  
    } finally {  
        util.ConnectionManager.closeConnection();  
    }  
    return k;  
}  
  
public static Boolean nadjiKorisnickoIme(String korisnickoIme) {  
    Boolean provera = false;  
    try {  
        String SQL = "SELECT * FROM korisnik WHERE korisnicko_ime=?";  
        Connection con = util.ConnectionManager.getConnection();  
        PreparedStatement ps = con.prepareStatement(SQL);  
        ps.setString(1, korisnickoIme);  
        ResultSet rs = ps.executeQuery();  
        if (rs.next()) {  
            provera = true;  
        }  
        rs.close();  
    } catch (Exception e) {  
    } finally {  
        util.ConnectionManager.closeConnection();  
    }  
    return provera;  
}  
  
public static Boolean nadjiKorisnickiEmail(String email) {  
    Boolean provera = false;
```

```

    try {
        String SQL = "SELECT * FROM korisnik WHERE email=?";
        Connection con = util.ConnectionManager.getConneccion();
        PreparedStatement ps = con.prepareStatement(SQL);
        ps.setString(1, email);
        ResultSet rs = ps.executeQuery();
        if (rs.next()) {
            provera = true;
        }
    } catch (Exception e) {
    } finally {
        util.ConnectionManager.closeConnection();
    }
    return provera;
}

public static boolean izmeniKorisnika(Integer id, String ime, String prezime, String korisnickoIme,
    String email, Integer tip_korisnika_id, Integer status_korisnika_id, Integer zvanje_id) {
    boolean t = true;
    try {
        String sql = "UPDATE korisnik SET korisnicko_ime=?,ime=?,prezime=?,email=?"
            + "tip_korisnika_id=?,status_korisnika_id=?,zvanje_id=? WHERE id=?";
        PreparedStatement ps = util.ConnectionManager.getConneccion().prepareStatement(sql);
        ps.setString(1, korisnickoIme);
        ps.setString(2, ime);
        ps.setString(3, prezime);
        ps.setString(4, email);
        ps.setInt(5, tip_korisnika_id);
        ps.setInt(6, status_korisnika_id);
        ps.setInt(7, zvanje_id);
        ps.setInt(8, id);
        ps.executeUpdate();
    } catch (Exception e) {
        t = false;
    } finally {
        util.ConnectionManager.closeConnection();
    }
    return t;
}

public List<Korisnik> sviKorisniciPoTipu(Integer t) {
    Korisnik k;
    List<Korisnik> korisnici = new ArrayList<Korisnik>();
    try {
        String sql = " SELECT * FROM korisnik WHERE tip=?";
        PreparedStatement ps = util.ConnectionManager.getConneccion().prepareStatement(sql);

```

```
        ResultSet rs = ps.executeQuery(sql);
        ps.setInt(1, t);
        while (rs.next()) {
            k = new Korisnik(rs);
            korisnici.add(k);
        }
        ps.close();
        rs.close();
    } catch (Exception e) {
        korisnici = null;
    } finally {
        util.ConnectionManager.closeConnection();
    }
    return korisnici;
}

public static List<Korisnik> sviKorisnici() {
    List<Korisnik> korisnici = new ArrayList<Korisnik>();
    Korisnik k;
    try {
        String sql = " SELECT * FROM korisnik";
        PreparedStatement ps = util.ConnectionManager.getConnctcion().prepareStatement(sql);
        ResultSet rs = ps.executeQuery(sql);
        while (rs.next()) {
            k = new Korisnik(rs);
            korisnici.add(k);
        }
        ps.close();
        rs.close();
    } catch (Exception e) {
        korisnici = null;
    } finally {
        util.ConnectionManager.closeConnection();
    }
    return korisnici;
}

public static List<Korisnik> sviKorisniciBezAdmina() {
    List<Korisnik> korisnici = new ArrayList<Korisnik>();
    Korisnik k;
    try {
        String sql = " SELECT * FROM korisnik WHERE tip_korisnika_id!=1;";
        PreparedStatement ps = util.ConnectionManager.getConnctcion().prepareStatement(sql);
        ResultSet rs = ps.executeQuery(sql);
        while (rs.next()) {
            k = new Korisnik(rs);
```

```
        korisnici.add(k);
    }
    ps.close();
    rs.close();
} catch (Exception e) {
    korisnici = null;
} finally {
    util.ConnectionManager.closeConnection();
}
return korisnici;
}

public static boolean obrisiKorisnika(Integer id) {
    boolean t = true;
    try {
        String sql = "DELETE FROM korisnik WHERE id=?";
        PreparedStatement ps = util.ConnectionManager.getConnection().prepareStatement(sql);
        ps.setInt(1, id);
        ps.executeUpdate();
    } catch (Exception e) {
        t = false;
    } finally {
        util.ConnectionManager.closeConnection();
    }
    return t;
}

public static boolean izmeniLozinku(String lozinka, Integer id) {
    boolean t = true;
    try {
        String sql = "UPDATE korisnik SET lozinka=? WHERE id=?";
        PreparedStatement ps = util.ConnectionManager.getConnection().prepareStatement(sql);
        ps.setString(1, DigestUtils.sha1Hex(lozinka));
        ps.setInt(2, id);
        ps.executeUpdate();
        ps.close();
    } catch (Exception e) {
        t = false;
    } finally {
        util.ConnectionManager.closeConnection();
    }
    return t;
}

public static boolean izmeniStatus(Integer statusId, Integer id) {
    boolean t = true;
```

```
try {
    String sql = "UPDATE korisnik SET status_korisnika_id=? WHERE id=?";
    PreparedStatement ps = util.ConnectionManager.getConnetcion().prepareStatement(sql);
    ps.setInt(1, statusId);
    ps.setInt(2, id);
    ps.executeUpdate();
    ps.close();
} catch (Exception e) {
    t = false;
} finally {
    util.ConnectionManager.closeConnection();
}return t;}

public static List<Korisnik> sviAktivniKorisnici() {
    List<Korisnik> korisnici = new ArrayList<Korisnik>();
    Korisnik k;
    try {
        String sql = " SELECT * FROM korisnik WHERE status_korisnika_id=1;";
        PreparedStatement ps = util.ConnectionManager.getConnetcion().prepareStatement(sql);
        ResultSet rs = ps.executeQuery(sql);
        while (rs.next()) {
            k = new Korisnik(rs);
            korisnici.add(k);
        }
        ps.close();
        rs.close();
    } catch (Exception e) {
        korisnici = null;
    } finally {
        util.ConnectionManager.closeConnection();
    }
    return korisnici;
}

public static List<Korisnik> sviDeaktiviraniKorisnici() {
    List<Korisnik> korisnici = new ArrayList<Korisnik>();
    Korisnik k;
    try {
        String sql = " SELECT * FROM korisnik WHERE status_korisnika_id=2;";
        PreparedStatement ps = util.ConnectionManager.getConnetcion().prepareStatement(sql);
        ResultSet rs = ps.executeQuery(sql);
        while (rs.next()) {
            k = new Korisnik(rs);
            korisnici.add(k);
        }
        ps.close();
    }
```

```
        rs.close();
    } catch (Exception e) {
        korisnici = null;
    } finally {
        util.ConnectionManager.closeConnection();
    }
    return korisnici;
}

public static List<Korisnik> sviAdministratoriModeratori() {
    Korisnik k;
    List<Korisnik> korisnici = new ArrayList<Korisnik>();
    try {
        String sql = " SELECT * FROM korisnik WHERE tip_korisnika_id = 1 OR tip_korisnika_id = 3;";
        PreparedStatement ps = util.ConnectionManager.getConnection().prepareStatement(sql);
        ResultSet rs = ps.executeQuery(sql);
        while (rs.next()) {
            k = new Korisnik(rs);
            korisnici.add(k);
        }
        ps.close();
        rs.close();
    } catch (Exception e) {
        korisnici = null;
    } finally {
        util.ConnectionManager.closeConnection();
    }
    return korisnici;
}

public static List<Korisnik> sviModeratori() {
    Korisnik k;
    List<Korisnik> korisnici = new ArrayList<Korisnik>();
    try {
        String sql = " SELECT * FROM korisnik WHERE tip_korisnika_id = 3;";
        PreparedStatement ps = util.ConnectionManager.getConnection().prepareStatement(sql);
        ResultSet rs = ps.executeQuery(sql);
        while (rs.next()) {
            k = new Korisnik(rs);
            korisnici.add(k);
        }
        ps.close();
        rs.close();
    } catch (Exception e) {
        korisnici = null;
    } finally {
```

```
        util.ConnectionManager.closeConnection();
    }
    return korisnici;
}

public static Korisnik korisnikPoId(List<Korisnik> korisnici, int korisnikId) {
    for (Korisnik ko : korisnici) {
        if (ko.getId() == korisnikId) {
            return ko;
        }
    }
    Korisnik k = new Korisnik();
    return k;
}
}
```

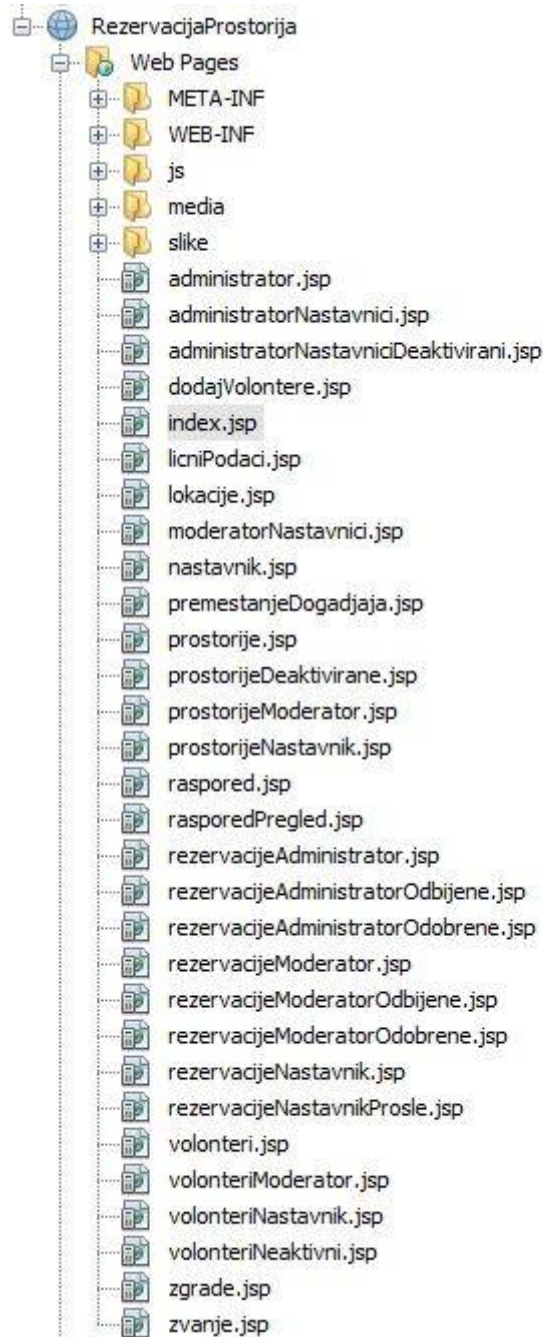
Slika 10: Klasa Korisnik iz aplikacije za rezervaciju prostorija

U aplikaciji, u klasi model, se pored klase Korisnik nalaze i klase:

- Lokacija
- Prijavljivanje
- Prostorija
- Rezervacija
- RezervacijaVolonter
- StatusKorisnika
- StatusRezervacije
- Tip
- TipProstorije
- TipRezervacije
- Volonter
- VremeRezervacije
- Zgrada
- Zvanje

12.2 Prezentacioni deo (View)

Prezentacioni sloj u aplikaciji je realizovan korišćenjem *JSP*a. U *JSP* dokumentima se osim *HTML* oznaka, *JavaScripta* i *jQuerya* nalaze i skripteti. *JSP* dokumenti su u okviru aplikacije organizovani u datoteci *Web Pages* u kojoj se nalaza i *jQuery* biblioteke, *CSS* i *JS* dokumenti kao i neke slike.



Slika 11: Datoteka *Web Pages* u kojoj se nalaze *JSP* dokumenti iz aplikacije

12.2.1 Primer JSPa: administrator.jsp

```

<!--
Document   : administrator
Created on : 26-Nov-2014, 22:43:03
Author    : Nikola
-->

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<link type="text/css" rel="stylesheet" href="profil.css" />
</head>
<body>

<% if (request.getAttribute("poruka") != null) {%><script>alert('<%=request.getAttribute("poruka")%>');</script><%}%>

<% if (session.getAttribute("ID") != null && session.getAttribute("Tip").equals("admin")) {%>
<div id="kontejner">
<form method="post" action="CommandServlet?cmd=logout">
<input type="submit" value="Logout" style="float: right;" />
</form>

<div id="meni">
<table id="meniTabela">
<tr>
<td id="meniOpcija" onClick="document.location.href = 'rezervacijeAdministrator.jsp'">Rezervacije</td>
<td id="meniOpcija" onClick="document.location.href = 'administratorNastavnici.jsp'">Nastavnici</td>
<td id="meniOpcija" onClick="document.location.href = 'volonteri.jsp'">Volonteri</td>
<td id="meniOpcija" onClick="document.location.href = 'prostorije.jsp'">Prostorije</td>
<td id="meniOpcija" onClick="premesti()">Premestanje po prostorijama</td>
<td id="meniOpcija" onClick="document.location.href = 'zgrade.jsp'">Zgrade</td>
<td id="meniOpcija" onClick="document.location.href = 'lokacije.jsp'">Lokacije</td>
<td id="meniOpcija" onClick="document.location.href = 'zvanje.jsp'" style="border: none;">Zvanje</td>
</tr>
</table>

</div>

<div id="prikaz">

</div>

<script src="//code.jquery.com/jquery-1.10.2.js"></script>
<link rel="stylesheet" href="//code.jquery.com/ui/1.11.2/themes/smoothness/jquery-ui.css">
<script src="//code.jquery.com/ui/1.11.2/jquery-ui.js"></script>

```

```
<script>
    function premesti() {
        pomeranje = document.createElement('div');
        $('<form style="display: none;" method="post" action="CommandServlet?cmd=proslediNaPromenu"><input \n\
        |type="submit" value="Potvrdi" style="float: right;"></form>').submit();
    }
    ;
</script>
</div>
<%> else {<%> <script>
    alert('Nemate prava pristupa!');
    window.location.href = "index.jsp";
</script> <%>
</body>
</html>
```

Slika 12: Primer JSP dokumenta iz aplikacije (administrator.jsp)

12.3 Kontroler (Controller)

U aplikaciji za rezervaciju prostorija kontroler je predstavljen uz pomoć servleta. Kontroler vrši koordinaciju između slojeva.[48] Glavni servlet u aplikaciji je *CommandServlet.java* i zahvaljujući njemu se donosi odluka koja operacija će se izvršiti. Odluka se donosi na osnovu pristiglog zahteva i naredbe višestrukog grananja (*switch*).

```
try {
    HttpSession session = request.getSession();
    String komanda = request.getParameter("cmd");
```

Slika 13: Preuzimanje parametra cmd iz pristiglog zahteva radi određivanja akcije koja treba da se izvrši

```
if (session.getAttribute("ID") != null || komanda.equals("login")) {  
    switch (komanda) {  
        case "login":  
            login(request, response);  
            break;  
        case "logout":  
            logout(request, response);  
            break;  
        case "dodajZvanje":  
            dodajZvanje(request, response);  
            break;  
        case "svaZvanja":  
            svaZvanja(request, response);  
            break;  
        case "izmeniZvanje":  
            izmeniZvanje(request, response);  
            break;  
        case "dodajKorisnika":  
            dodajKorisnika(request, response);  
            break;  
        case "obrisiKorisnika":  
            obrisiKorisnika(request, response);  
            break;  
        case "izmeniKorisnika":  
            izmeniKorisnika(request, response);  
            break;  
        case "izmeniLozinku":  
            izmeniLozinku(request, response);  
            break;  
        case "dodajVolontera":  
            dodajVolontera(request, response);  
            break;  
        case "izmeniVolontera":  
            izmeniVolontera(request, response);  
            break;  
        case "obrisiVolontera":  
            obrisiVolontera(request, response);  
            break;  
        case "dodajLokaciju":  
            dodajLokaciju(request, response);  
            break;  
        case "izmeniLokaciju":  
            izmeniLokaciju(request, response);  
            break;  
        case "dodajZgradu":  
            dodajZgradu(request, response);  
    }  
}
```

```
        break;
    case "izmeniZgradu":
        izmeniZgradu(request, response);
        break;
    case "obrisiProstoriju":
        obrisiProstoriju(request, response);
        break;
    case "dodajProstoriju":
        dodajProstoriju(request, response);
        break;
    case "izmeniProstoriju":
        izmeniProstoriju(request, response);
        break;
    case "posaljiMail":
        posaljiMail(request, response);
        break;
    case "izmeniSvojuLozinkuNastavnik":
        izmeniSvojuLozinkuNastavnikModerator(request, response);
        break;
    case "izmeniSvojePodatkeNastavnik":
        izmeniSvojePodatkeNastavnik(request, response);
        break;
    case "izmeniRezervaciju":
        izmeniRezervaciju(request, response);
        break;
    case "prikaziRaspored":
        prikaziRaspored(request, response);
        break;
    case "izmeniStatusKorisnika":
        izmeniStatusKorisnika(request, response);
        break;
    case "izmeniLozinkuModerator":
        izmeniLozinkuModerator(request, response);
        break;
    case "izmeniRezervacijuModerator":
        izmeniRezervacijuModerator(request, response);
        break;
    case "idiNaRezervaciju":
        idiNaRezervaciju(request, response);
        break;

    case "proslediNaDodajVolontere":
        proslediNaDodajVlontere(request, response);
        break;
```

```
        case "pozoviVolontere":
            pozoviVolontere(request, response);
            break;
        case "rasporedPregled":
            rasporedPregled(request, response);
            break;
        case "stampaj":
            stampa(request, response);
            break;
        case "deleteEvent":
            deleteEvent(request, response);
            break;
        case "izmeniKorisnikaForma":
            izmeniKorisnikaForma(request, response);
            break;
        case "izmeniZgraduForma":
            izmeniZgraduForma(request, response);
            break;
        case "izmeniProstorijuForma":
            izmeniProstorijuForma(request, response);
            break;
        case "promeniProstoriju":
            promeniProstoriju(request, response);
            break;
        case "proslediNaPromenu":
            proslediNaPromenu(request, response);
            break;
    }
} else {
    request.setAttribute("poruka", "Greska 0000! Morate biti ulogovani!");
    RequestDispatcher rd = getServletContext().getRequestDispatcher("/index.jsp");
    rd.forward(request, response);
}
} catch (Exception e) {
    request.setAttribute("poruka", "Greska 000!");
    RequestDispatcher rd = getServletContext().getRequestDispatcher("/index.jsp");
    rd.forward(request, response);
}
```

Slika 14: Naredba višestrukog grananja (switch) iz *CommandServlet* za određivanje akcije koja treba da se izvrši na osnovu *cmd* parametra iz zahteva

Pored servleta *CommandServlet* u aplikaciji se koristi i servlet *KalendarServlet* koji je zadužen za rad sa rezervacijama (kreiranje, brisanje, izmena) u okviru kalendara. Većina akcija za koje je zadužen ovaj servlet poziva se prilikom prevlačenja događaja (*drag and drop*).

```

public void prikaziRaspored(HttpServletRequest request, HttpServletResponse response) throws MessagingException, ServletException, IOException {
    try {
        String prostorijaId = request.getParameter("prostorijaId");
        request.setAttribute("prostorijaId", prostorijaId);
    } catch (Exception e) {
        request.setAttribute("poruka", "Greska 025!");
    }
    RequestDispatcher rd = getServletContext().getRequestDispatcher("/raspored.jsp");
    rd.forward(request, response);
}

public void izmeniStatusKorisnika(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    try {
        Integer statusId = Integer.parseInt(request.getParameter("status"));
        Integer korisnikId = Integer.parseInt(request.getParameter("id"));
        Korisnik.izmeniStatus(statusId, korisnikId);
    } catch (Exception e) {
        request.setAttribute("poruka", "Greska 026!");
    }
    RequestDispatcher rd = getServletContext().getRequestDispatcher("/moderatorNastavnici.jsp");
    rd.forward(request, response);
}

```

Slika 15: Primer metoda iz servleta *CommandServlet* koja se pozivaju na osnovu pristiglog parametra *cmd*

12.4 Paket util

Paket *util* je pomoćni paket u okviru kojeg se nalaze klasa *ConnectionManager.java*, klasa *SendEmail* i dokument *db.properties*.

Klasa *ConnectionManager* je zadužena za uspostavljanje i raskidanje veze sa bazom podataka. Klasa sadrži atribut *conn* klasnog tipa *Connection*, metod *getConnection()* za uspostavljanje veze i metod *closeConnection()* za raskidanje veze.

Klasa *SendEmail* je sadrži metod *sendingEmail* koja služi za slanje elektronske pošte. Da bi pošta bila uspešno poslata potrebno je podesiti adresu servera, korisničko ime i lozinku za elektronsku poštu sa koje će poruka biti poslata i adresu primaoca.

U dokumentu *db.properties* se nalaze parametri za pristup bazi podataka. Da bi pristupili bazi neophodno je da navedemo korisničko ime, lozinku i adresu (*url*) baze kojoj želimo da pristupimo.



Slika 16: Paket *util* iz aplikacije za rezervaciju prostorija

```
package util;

import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ConnectionManager {

    private static Connection conn=null;

    public static Connection getConnetcion(){
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Properties prop = new Properties();
            InputStream in = ConnectionManager.class.getResourceAsStream("db.properties");
            prop.load(in);
            String username = prop.getProperty("username");
            String url = prop.getProperty("url");
            String password = prop.getProperty("password");

            conn = DriverManager.getConnection(url, username, password);

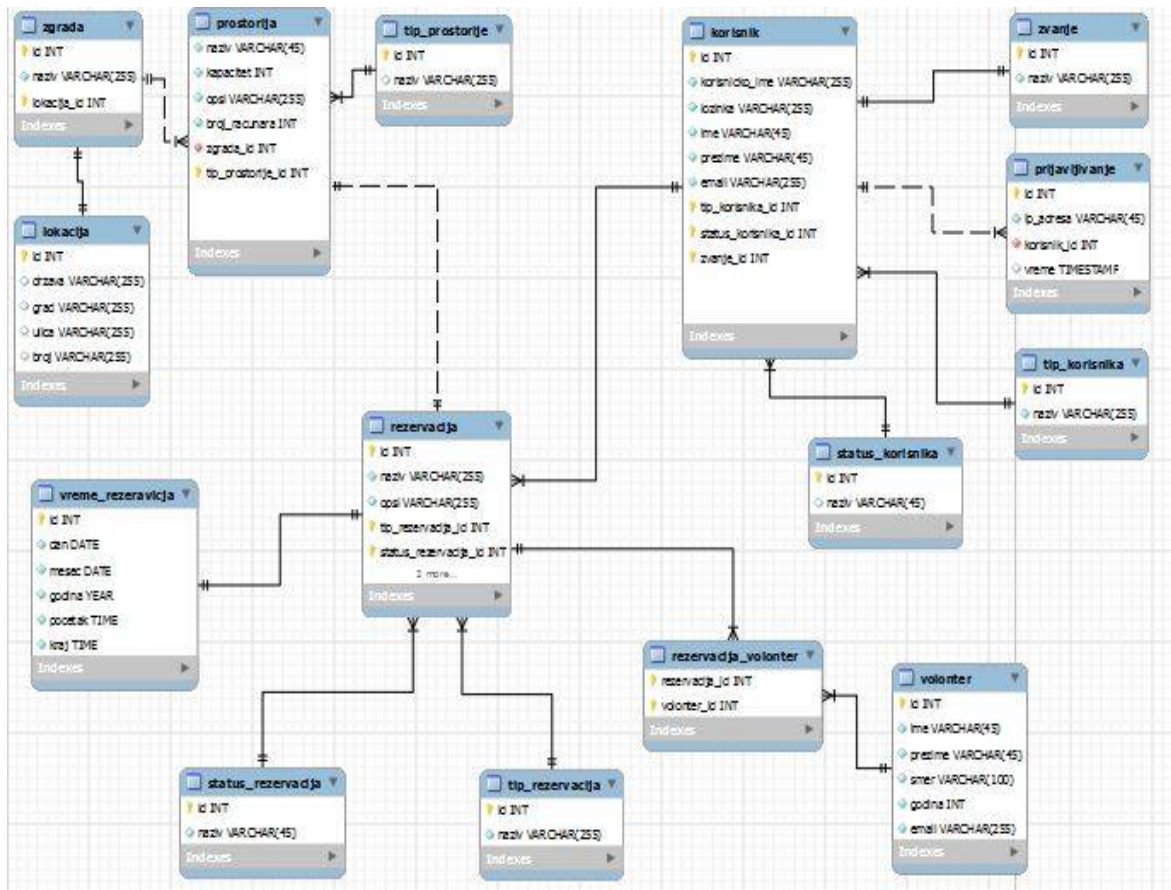
        } catch (ClassNotFoundException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
        return conn;
    }

    public static void closeConnection(){
        if(conn!=null){
            try {
                conn.close();
            } catch (SQLException ex) {
                ex.printStackTrace();
            }
        }
    }
}
```

Slika 17: Klasa ConnectionManager iz aplikacije za rezervaciju prostoriya

12.5 Baza podataka

Za potrebe aplikacije korišćena je relacionala baza podataka koja se nalazi na *MySQL* serveru. Šema baze je napravljena u alatu *WorkBench 5.2 CE* a zatim je izvršavanjem *SQL* dokumenta prebačena na server.



Slika 18: ERD baze podataka iz aplikacije

SQL kod za kreiranje tabele korisnik i dodavanje ograničenja:

```
CREATE TABLE `korisnici` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `korisnicko_ime` varchar(255) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT NULL,
  `lozinka` varchar(255) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT NULL,
  `ime` varchar(45) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT NULL,
  `prezime` varchar(45) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT NULL,
  `email` varchar(255) CHARACTER SET latin1 COLLATE latin1_swedish_ci NOT NULL,
  `tip_korisnika_id` int(11) NOT NULL,
  `status_korisnika_id` int(11) NOT NULL,
  `zvanje_id` int(11) NOT NULL,
  PRIMARY KEY (`id`, `tip_korisnika_id`, `status_korisnika_id`, `zvanje_id`, `korisnicko_ime`),
  FOREIGN KEY (`status_korisnika_id`) REFERENCES `status_korisnika` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  FOREIGN KEY (`tip_korisnika_id`) REFERENCES `tip_korisnika` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  FOREIGN KEY (`zvanje_id`) REFERENCES `zvanje` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  INDEX `fk_korisnik_tip_korisnika1_idx` (`tip_korisnika_id`) USING BTREE,
  INDEX `fk_korisnik_status_korisnika1_idx` (`status_korisnika_id`) USING BTREE,
  INDEX `fk_korisnik_zvanje1_idx` (`zvanje_id`) USING BTREE
)
```

id	korisnicko_ime	lozinka	ime	prezime	email	tip_korisnika_id	status_korisnika_id	zvanje_id
1	admin	d033e22ae348aeb5660fc214f	Nikola	Rancic	nikola.rancic@hotmail.com	1	1	3
18	milic	67451f243b6d09beb3bf4696	Milan	Milic	milic.m@hotmail.com	2	1	3
19	nesa	46cde443605c2185cf90e7ee7	Nenad	Nesic	nesa@yahoo.com	3	1	1
20	milan	c5983e484db0b621516387b3	Milan	Milanovic	milan@gmail.com	2	2	1

Slika 19: Tabela korisnik iz aplikacije za rezervaciju prostoriya

12.6 Zaštita

U današnjem svetu su sve učestaliji napadi na veb aplikacije i prezentacije. U cilju sprečavanja napada svaka aplikacija ili prezentacija koja se nalazi na vebu trebala bi da ima bar osnovne mehanizme zaštite. Aplikacija za rezervaciju prostorija za korišćenje zahteva korisničko ime i lozinku i time ograničava pristup podacima i samoj aplikaciji. Neoprezno bi bilo čuvati lozinke u izvornom obliku pa se zato koriste sažetci (heš vrednosti, digitalni otisci lozinke) ili “posoljeni” sažetci (*salted hash*). Neophodno je sprečiti i neovlašćeni pristup i upotrebu podataka iz baze kao i onemogućiti *SQL* injekciju.

12.6.1 Kontrola pristupa

Više korisnička aplikacija, kakva je i aplikacija za rezervaciju prostorija, imaju različite vrste korisnika. Nekim tipovima korisnicima je potrebno ograničiti kojim podacima mogu da pristupaju, šta sa njima mogu da rade ali i kojim delovima sistema mogu da pristupe dok je pristup sistemu bez korisničkog naloga nemoguć. Aplikacija za rezervaciju prostorija ima tri tip korisnika koji se razlikuju po ulogama u sistemu. Administrator ima apsolutnu kontrolu nad aplikacijom i dostupne su mu sve mogućnosti, moderator ima manja ovlašćenja od administratora dok nastavnik ima najmanja ovlašćenja pa samim tim i najmanji broj opcija u meniju. Tip korisnika se čuva u bazi podataka u tabeli korisnik i na osnovu tog parametra se generiše i prikazuje odgovarajući korisnički interfejs a administrator može da menja privilegije korisnicima.

Pored pristupa samoj aplikaciji i baza podataka je zaštićena korisničkim imenom i lozinkom a pristupne parametre zna samo administrator aplikacije.



Slika 20: Forma za prijaljivanje na sistem iz aplikacije za rezervaciju prostorija

12.6.2 Šifrovanje lozinke

Ukoliko bi došlo do neovlašćenog pristupa bazi podataka neko bi mogao da ukrade korisnička imena i lozinke i da ih kasnije zloupotrebi ili proda nekom trećem licu. Da bi se izbegao ovakav scenario lozinke se ne čuvaju u bazi kao otvoreni tekst (*plain text*) već kao sažetak (*hash*) te lozinke a u češćim slučajevima i kao “posoljeni” sažetak. U aplikaciji za rezervaciju prostorija korišćen je *SHA1* algoritam za dobijanje sažetka. Primenom *SHA1* (*Secure Hash Algorithm*) kriptološkog algoritma na ulaznu vrednost dobijamo 160o bitni heksadecimalni broj kao lozinku koja se nalazi u bazi. Ako dođe do upada u sistem i krađe podataka, korisničkih imena i lozinki, napadač ne bi imao upotrebljive informacije. [30]

SHA1 kao kriptološki algoritam objavljen je 1995. Godine i standardizovan je od strane NISTa (Nacionalni Institut za Standarde i Tehnologiju SAD). [31]

```
ps.setString(2, DigestUtils.sha1Hex(lozinka));
```

Slika 21: Primer korišćenja *SHA1* algoritma u aplikaciji

12.6.3 SQL injekcija (SQL injection)

SQL injekcija je tip napada na bazu podataka aplikacije umetanjem SQL koda. Ako SQL koji se izvršava nad bazom zahteva ulazne parametre od korisnika, može doći do injekcije nedozvoljenog SQL koda. Osoba koja vrši napad može da, ukoliko je to programskim kodom dozvoljeno, unese parametar i iza njega ";" i željeni SQL kod koji će se izvršiti. [33]

U aplikaciji je korišćen Java interfejs *PreparedStatement* koji za razliku od interfejsa *Statement* sprečava SQL injekciju. Korišćenjem pripremljenih naredbi navode se parametri koji se prvo formatiraju, a zatim umeću u SQL kod.

```
String SQL = "SELECT * FROM korisnik WHERE korisnicko_ime=? AND lozinka=?";  
Connection con = util.ConnectionManager.getConnetcion();  
PreparedStatement ps = con.prepareStatement(SQL);  
ps.setString(1, korisnickoIme);  
ps.setString(2, DigestUtils.sha1Hex(lozinka));  
ResultSet rs = ps.executeQuery();
```

Slika 22: Primer korišćenja prekompajlirane SQL naredbe (*PreparedStatement*)

13 Slučajevi korišćenja

13.1 Dodavanje novog korisnika

Novog korisnika u sistemu može da doda samo administrator. U meniju se odabere opcija 'Nastavnici' a zatim se pritiskom na dugme 'Dodaj korisnika' pojavljuje forma za unos podataka o korisniku.

13.2 Prijavljivanje na sistem

Da bi korisnik pristupio sistemu mora da poseduje korisničko ime i lozinku. Prilikom učitavanja aplikacije korisniku se prikazuje forma za unos korisničkog imena i lozinke. Nakon unosa podataka, klikom na dugme 'Prijava' ili pritiskom entera serveru se šalju uneti parametri. Ako korisnik sa unetim parametrima postoji, generiše se odgovarajuća strana na osnovu tipa korisnika koja se prikazuje korisniku.

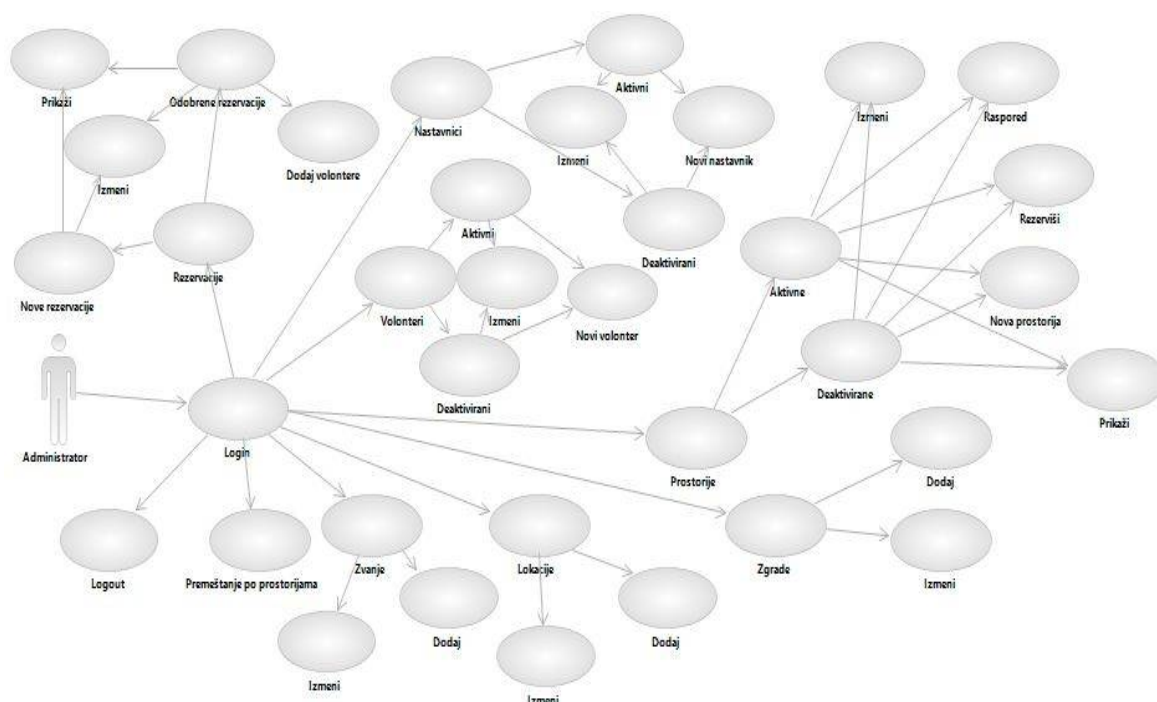
13.3 Pregled prostorija, raspored i rezervacija

Svi tipovi korisnika u svom meniju imaju opciju 'Prostorije'. Izborom ove opcije dobija se spisak svih prostorija koje postoje u sistemu. Za svaku prostoriju je moguće prikazati detaljan opis klikom na dugme 'Prikaži', prikazati raspored zauzetosti prostorija po datumu kao i izvršiti rezervaciju prostorije. Samo administrator ima mogućnost promene informacija o prostoriji, pregled neaktivnih prostorija i dodavanja nove prostorije.

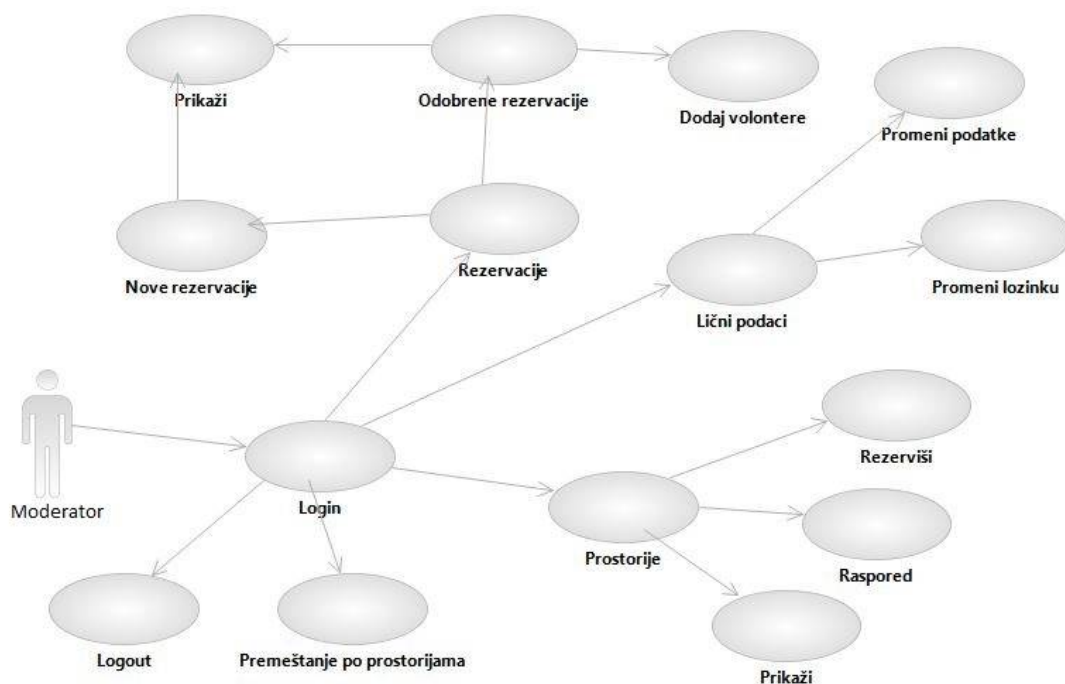
Pritiskom na dugme 'Rezerviši' prikazuje se forma za unos osnovnih podataka o rezervaciji (naziv, tip rezervacije, očekivani broj studenata...). Nakon klika na dugme 'Sačuvaj' prikazuju se kalendar. Odabirom datuma sa kalendara prikazuje se trenutni raspored prostorije za taj datum. Prevlačenjem miša preko odrađenog dana i satnice unosi se nova rezervacija. Ako je korisnik koji unosi rezervaciju administrator ili moderator, rezervacija je automatski odobrena dok je u slučaju da je korisnik nastavnik potrebno odobrenje rezervacije od strane moderatora ili administratora. Korisnik može da obriše svoju rezervaciju ako ona još uvek nije odobrena klikom na dugme 'x' u gornjem desnom uglu rezervacije. Odobrene rezervacije su plave boje dok su one koje čekaju na odobrenje crvene boje.

13.4 Premeštanje po prostorijama

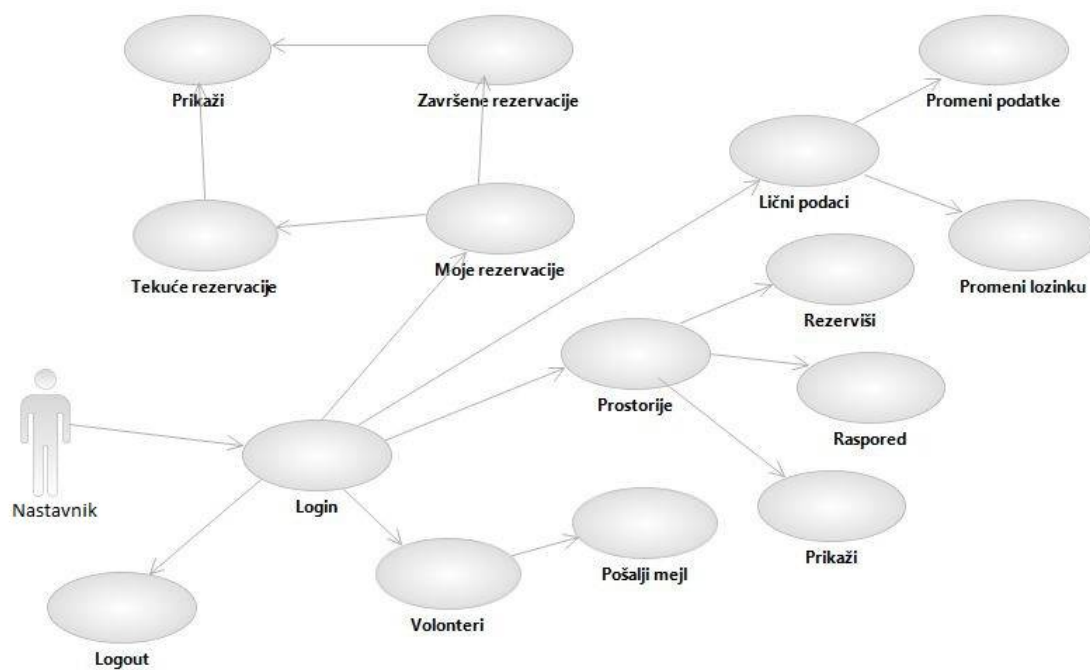
Odabirom opcije 'Premeštanje po prostorijama' koja je dostupna moderatoru i administratoru prikazuju se dva rasporeda (tabele) rezervacija za odabrane datume. Kolone svakog rasporeda predstavljaju satnicu dok redovi predstavljaju prostorije. Prevlačenjem događaja (*drag and drop*) iz jedne ćelije u drugu vrši se premeštanje događaja po prostorijama i/ili satnici.



Slika 23: Dijagram slučajeve korišćenja za administratora



Slika 24: Dijagram slučajeva korišćenja za moderatora



Slika 25: Dijagram slučajeva korišćenja za administratora

14 **Zaključak**

U ovom radu su predstavljeni osnovni korisnički i funkcionalni zahtevi veb aplikacije za rezervaciju prostorija na visokoškolskoj ustanovi. Opisane su i popularne moderne tehnologije, njihove osobine i kratka istorija od kojih je većina korišćena pri samoj realizaciji aplikacije. Takođe su predstavljeni osnovni principi i primeri razvoja jedne moderne veb aplikacije. Opisan je *MVC* model koji je primenjem na aplikaciji za rezervaciju prostorija i navedeni su primeri iz same aplikacije, opisane su dodatne klase koje se koriste, opisana je baza podataka kao i elementarni principi zaštite.

Na osnovu predstavljenih tehnologija, postupaka u realizaciji i primera iz aplikacije za rezervaciju prostorija možemo videti da se upotrebom savremenih računarskih tehnologija mogu napraviti aplikacije koje značajno pojednostavljaju, ubrzavaju i olakšavaju svakodnevne poslovne procese i aktivnosti.

Literatura

- [1] Živković, D., **Osnove Java programiranja**, treće izdanje, Univerzitet Singidunum, Beograd, 2010.
- [2] Veinović, M., Šimić, G., **Uvod u baze podataka**, peto izdanje, Univerzitet Singidunum, Beograd, 2010.
- [3] Branović, I., „*Praktikum programski sistemi*“, materijali sa predavanja, Univerzitet Singidunum, Beograd, 2014.
- [4] <http://www.w3.org/People/Berners-Lee/> [dostupno dana: 02.08.2015.]
- [5] <http://www.oracle.com/technetwork/java/javaee/jsp/index.html>
[dostupno dana: 03.08.2015.]
- [6] <http://docs.oracle.com/javaee/1.3/tutorial/doc/WebApp.html>
[dostupno dana: 03.07.2015.]
- [7] <http://help.eclipse.org/juno/index.jsp?topic=%2Forg.eclipse.wst.webtools.doc.user%2Ftopics%2Fccwebprj.html> [dostupno dana: 03.08.2014.]
- [8] <http://office.microsoft.com/sr-latn-cs/access-help/osnove-baza-podataka-HA010064450.aspx> [dostupno dana: 08.08.2015.]
- [9] <http://www.w3schools.com/sql/> [dostupno dana: 08.08.2015.]
- [10] <http://www.oracle.com/technetwork/java/javase/jdbc/index.html>
[dostupno dana: 08.07.2015.]
- [11] http://www.w3.org/community/webed/wiki/HTML/Training/What_is_HTML
[dostupno dana: 03.08.2015.]
- [12] <http://www.w3.org/community/webed/wiki/HTML/Elements>
[dostupno dana: 06.08.2015.]
- [13] http://www.it.uom.gr/teaching/distributedSite/platoAcadiau_ca/Slides/JSP%20101%20Introduction%20to%20JavaServer%20Pages.htm
[dostupno dana: 05.07.2014.]
- [14] <http://www.w3.org/Style/LieBos2e/history/> [dostupno dana: 09.08.2015.]
- [15] http://www.w3schools.com/css/css_howto.asp [dostupno dana: 03.08.2015.]
- [16] <http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html> [dostupno dana: 05.08.2015.]
- [17] <http://www.oracle.com/technetwork/java/index-jsp-135475.html> [dostupno dana: 10.08.2015.]
- [18] <http://www.java4study.com/JSP/JSP2.HTML> [dostupno dana: 10.08.2015.]
- [19] https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript
[dostupno dana: 09.08.2015.]
- [20] http://www.w3schools.com/JQuery/jquery_intro.asp [dostupno dana: 09.08.2015.]
- [21] <http://www.w3.org/TR/WD-DOM/introduction.html> [dostupno dana: 03.08.2015.]

- [22] <http://www.oracle.com/technetwork/java/javaee/jsp/index.html>
[dostupno dana: 30.08.2015.]
- [23] <http://www.oracle.com/technetwork/java/javaee/jsp/index.html>
[dostupno dana: 30.08.2015.]
- [24] <http://www.j2mesalsa.com/struts/mvc.php/>
[dostupno dana: 01.09.2015.]
- [25] <http://msdn.microsoft.com/en-us/library/ff649643.aspx>
[dostupno dana: 01.09.2015.]
- [26] <https://www.lri.fr/~mbl/ENS/FONDIHM/2013/papers/Krasner-JOOP88.pdf>
[dostupno dana: 01.09.2015.]
- [27] http://www.w3schools.com/html/html5_new_elements.asp
[dostupno dana: 01.09.2015.]
- [28] <http://msdn.microsoft.com/en-us/library/ff649643.aspx>
[dostupno dana: 03.09.2015.]
- [29] <http://www.it-modul.rs/05/2013/mvc-web-patern/>
[dostupno dana: 03.09.2015.]
- [30] <http://tools.ietf.org/html/rfc3174>
[dostupno dana: 04.09.2015.]
- [31] <http://search.cpan.org/~gaas/Digest-SHA1-2.13/SHA1.pm>
[dostupno dana: 05.08.2015.]
- [32] <http://php.net/manual/en/history.php.php>
[dostupno dana: 05.08.2015.]
- [33] http://www.w3schools.com/sql/sql_injection.asp
[dostupno dana: 07.08.2015.]
- [34] <https://jquery.org/history/>
[dostupno dana: 07.08.2015.]
- [35] <http://www.phpasks.com/articles/historyajax.html>
[dostupno dana: 15.08.2015.]
- [36] <http://beginner-sql-tutorial.com/sql-grant-revoke-privileges-roles.htm>
[dostupno dana: 23.08.2015.]
- [37] <http://www.it-modul.rs/05/2013/mvc-web-patern/>
[dostupno dana: 26.08.2015.]