



UNIVERZITET U ZENICI

Politehnički fakultet

Softversko inženjerstvo

Napredne tehnike programiranja

Harun Smriko

**Idejno rješenje za razvoj aplikacije u Java okruženju primjenom JFrame**

*Diplomski rad*

Mentor:

Van. Prof. dr. Nevzudin Buzadžija

Zenica, 2021.

## **BIBLIOGRAFSKA KARTICA RADA**

**NAUČNO PODRUČJE RADA:** Tehničke nauke

**NAUČNO POLJE RADA:** Softversko inženjerstvo

**NAUČNA GRANA RADA:** Napredne tehnike programiranja

**USTANOVA U KOJOJ JE IZRAĐEN RAD:** Univerzitet u Zenici, Politehnički fakultet u Zenici

**MENTOR RADA:** Van. Prof. dr. Nevzudin Buzadija

**DATUM ODBRANE RADA:**

**ČLANOVI KOMISIJE ZA ODBRANU:**

## **IZJAVA**

Izjavljujem da sam diplomski rad pod naslovom „Idejno rješenje za razvoj aplikacije u Java okruženju primjenom JFrame“ izradio samostalno pod mentorstvom Van. Prof. dr. Nevzudina Buzadije. U radu sam koristio literaturu koja je navedena na kraju diplomskog rada. Tuđe spoznaje, zaključke, stavove, teorije i zakonitosti koje sam izravno ili parafrizirajući naveo u diplomskom radu na uobičajan, standardan način sam povezao sa fusnotama i korištenim bibliografskim jedinicama.

Student:

Harun Smriko

## SAŽETAK

Svjedoci smo pojave sve većeg broja informacionih sistema. Informacioni sistemi se kreiraju u cilju olakšanja procesa koji su podrazumijevani u okviru nekog servisa. U slučaju ovog rada, informacioni sistem je sproveden u vidu desktop aplikacije, kreirane u programskom jeziku Java, korištenjem JFrame formi. JFrame forme predstavljaju izuzetno jednostavan i efikasan način dizajniranja aplikacija, gdje je, uz pomoć NetBeans okruženja, moguće upravljanje elementima u prozoru „klikom miša“, što uključuje njihovo pozicioniranje na ekranu, funkcionalnost koju okidaju nakon određenog *event*-a (događaja) i sl. Za bazu podataka korišten je MySQL, u sklopu WampServera.

**Ključne riječi:** informacioni sistem, desktop aplikacija, Java, JFrame forma, MySQL

## ABSTRACT

We are witnessing the emergence of an increasing number of information systems. Information systems are created in order to facilitate the processes that are implied within a service. In the case of this work, the information system is implemented in the form of a desktop application, created in the Java programming language, using the JFrame forms. JFrame forms are an extremely simple and efficient way of designing applications, where it is possible to control the elements in the window with a "mouse click", with the usage of NetBeans environment, which includes their positioning on the screen, the functionality they trigger after a certain event etc. MySQL was used for the database, as part of WampServer.

**Keywords:** information system, desktop application, Java, JFrame form, MySQL

# SADRŽAJ

1. UVOD .....	6
2. DESKTOP APLIKACIJE .....	7
3. JAVA .....	7
3.1. Osnovne karakteristike Java programskog jezika .....	9
3.2. Java Swing .....	10
3.2.1. JFrame .....	10
3.3. NetBeans .....	11
3.3.1. NetBeans GUI Design Tool .....	12
4. IMPLEMENTACIJA .....	15
4.1. Analiza zahtjeva .....	16
4.2. Analiza i dizajn sistema .....	18
4.2.1. Use Case dijagrami .....	19
4.2.2. Class dijagram .....	23
4.2.3. ER – dijagram .....	24
4.3. Kreiranje aplikacije .....	25
4.3.1. Admin page .....	31
4.3.2. Zaposlenik .....	37
4.3.3. Korisnik .....	44
4.4. Testiranje .....	58
4.5. Planovi za budućnost .....	58
5. ZAKLJUČAK .....	61
6. LITERATURA .....	62

## 1. UVOD

Pandemija virusa COVID-19 uveliko je utjecala na eksponencijalan porast korištenja informacionih sistema za razne servise, u raznim sferama poslovanja, ali i života. S tim u vezi, postoji i veliki porast u potražnji novih aplikacija i informacionih sistema, a samim tim i porast broja istih.

Cilj ovog rada jeste predstavljanje idejnog rješenja za informacioni sistem za biblioteke, na nekom određenom nivou. Ova tema je odabrana zbog toga što se smatra izuzetno zahvalnom za „scope“ diplomskog rada, sa mnogo potencijalnih problema za rješavanje.

Informacioni sistem je kreiran u formi dektop aplikacije, kreirane u programskom jeziku Java, tačnije upotrebom JFrame formi. Aplikacija je koncipirana i kreirana na taj način da je jednostavna i razumljiva (*user-friendly*) za svakog potencijalnog korisnika iste, te ne treba mnogo vremena za spoznavanje svih funkcionalnosti aplikacije i načina na koje se iste realizuju.

Cilj ove aplikacije jeste olakšavanje bibliotekarskih procesa i uspostavljanje jednostavne mreže knjiga i biblioteka kroz koju bi korisnici mogli veoma jednostavno pretraživati i filtrirati knjige. Ultimativni cilj jesu jednostavna i funkcionalna aplikacija, i zadovoljni korisnici aplikacije.

## 2. DESKTOP APLIKACIJE

Desktop aplikacija je aplikacija koja samostalno može da se izvodi na računaru ili laptop uređaju. Zahtijeva neki operativni sistem za pokretanje, bio to Windows, Linux, MacOS, UNIX... Primjeri desktop aplikacije mogu nam biti Office, Adobe Acrobat Reader, MySQL, GitHub itd. Da bi koristili desktop aplikaciju prvo je moramo instalirati lokalno na računar. Svojim izvođenjem aplikacije koriste resurse, prvenstveno snagu procesora, RAM memorije, kao i prostor na disku (HDD/SDD).

Desktop aplikacije su robusnije nego web aplikacije. Za razliku od desktop aplikacije, web aplikacija balansira s potrebama za resursima. Jedan dio uzima sa lokalnog računara, dok će drugi dio resursa uzeti sa servera s kojim komunicira u tom trenutku. Ako je aplikacija složenija, sa više mogućnosti i jačom kontrolom za korisnika, svakako je bolje koristiti desktop aplikaciju. Produktivnost web aplikacija ne može se mjeriti s produktivnošću desktop aplikacija.

Neke manje složene desktop aplikacije mogu raditi bez potrebe da se prethodno instaliraju. Tijekom instalacije, „wizard“ (eng. „čarobnjak“) za instalaciju će uraditi sve potrebne pripreme i promjene na sistemu koje su potrebne da bi kasnije aplikacija ispravno radila. To znači da će se kopirati potrebne datoteke na određena mjesta na disku, da će se uraditi konfiguracija sistema, snimiti potrebni procesi, i dodijeliti određene permisije za korisnika i sistem.

Tijekom instalacije, u slučaju da se radi o komercijalnoj aplikaciji, moguće je da će čarobnjak tražiti aktiviranje licence. To može biti unos ključa za aplikaciju, ili neki drugi oblik aktivacije. Pored toga, tijekom instalacije korisnik mora prihvatiti uvjete korištenja, i moguće neke dodatne uvjete za korištenje aplikacije koju instalira.

## 3. JAVA

Java je objektno-orijentisani jezik u kojem svaki program posjeduje najmanje jednu klasu. Njegov razvoj započela je tvrtka „Sun Microsystems“ (danas kompanija po vlasništvom Oracle-a) pod vodstvom Jamesa Goslinga, dok prva verzija izlazi 1995. godine. Programski jezik Java poznat je po tome što ne ovisi o platformi na kojoj se izvodi. Kôd napisan u Javi može se izvršavati na svakom okruženju koje posjeduje *Java Virtual Machine* (JVM). Za

razvoj programa napisanih u Javi potrebno je imati postavljenu Java razvojnu opremu (engl. *Java Development Kit*). Ovaj jezik je prvobitno nosio naziv „OAK”, ali je on 1995. godine promijenjen u „Java”.



*Slika 1. Sjedište Oracle-a<sup>1</sup>*

Većinu svoje sintakse Java nasljeđuje iz jezika C++. Za razliku od C++, u Javi je teže programirati u funkcionalnom stilu; Java je više prilagođena za objektno-orientisani stil. Kôd napisan u Javi nalazi se u klasama, pa čak i glavna izvršna metoda *main* mora biti unutar klase. Klasa je temeljni dio objektno-orientisanog programiranja u Javi i sastoji se od atributa i metoda. Svaki podatak u javi je objekat, a izuzeci su primitivni tipovi podataka (npr. *integer*, *float*, *boolean*, *character*), koji nisu objekti zbog brzine izvođenja. Java, za razliku od C++, ne podržava preopterećenje operatora, višestruko nasljeđivanje, pokazivače, ali najveća razlika je da se upravljanje memorije u Javi rješava pomoću „sakupljača smeća“ (engl. *Garbage collector*). Sakupljač smeća brine se o alokaciji i delokaciji memorije, i tako olakšava posao programerima. U programskim jezicima poput C i C++, potrebno je

---

<sup>1</sup> [https://sco.wikipedia.org/wiki/Oracle\\_Corporation](https://sco.wikipedia.org/wiki/Oracle_Corporation)



eksplicitno navesti kada se memorija nekog objekta treba otpustiti. Javin sakupljač smeća to radi automatski, pa je teže ostvariti curenje memorije (engl. *memory leak*).



*Slika 2. Java logo<sup>2</sup>*

### **3.1.Osnovne karakteristike Java programskog jezika**

Java je objektno-orijentisan programski jezik, te se temelji na klasama. Bez postojanja barem jedne klase (*main*), program kreiran u Javi nemoguće je pokrenuti. Kôd u Javi se ne izvršava u realnom vremenu, već ga je prije pokretanja potrebno prevesti, što znači da je Java kompajlerski, a ne interpreterski jezik. Međutim, kôd se ne prevodi odmah u mašinski kôd za JVM, već u tzv. *byte-code* međuformat. S obzirom na činjenicu da je Java virtualna mašina (JVM) integrisana u gotovo svakom operativnom sistemu, može se reći da su aplikacije kreirane u Javi u potpunosti platformski nezavisne.

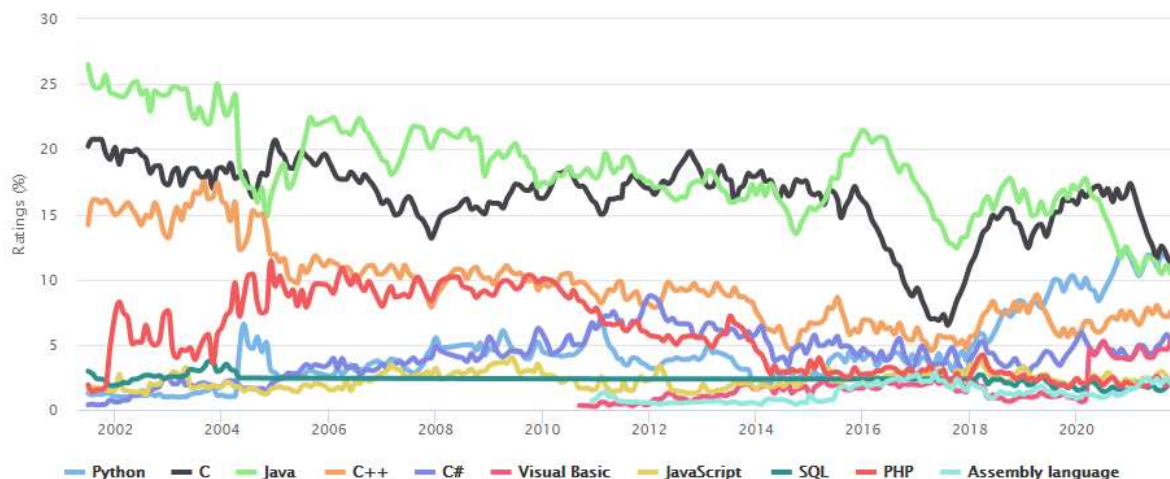
Prema istraživanjima u 2021. godini, uprkos činjenici da postoji već 26 godina, Java je i dalje veoma popularan programski jezik, te je 3. jezik po popularnosti (oktobar 2021. godine).<sup>3</sup> U nekim veoma razvijenim zemljama, kao što su Njemačka, Kina, te Sjeverna Koreja, Java je najkorišteniji programski jezik.<sup>4</sup>

---

<sup>2</sup> [https://en.wikipedia.org/wiki/Java\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Java_(programming_language))

<sup>3</sup> <https://www.tiobe.com/tiobe-index/>

<sup>4</sup> <https://www.jetbrains.com/lp/devecosystem-2021/java/>



Slika 3. Grafik popularnosti programskih jezika u 2021. godini<sup>3</sup>

### 3.2. Java Swing

Java Swing predstavlja tzv. *widget toolkit* za programski jezik Java. To je skup kontrola za jednostavan razvoj grafičkog interfejsa za aplikacije kreirane u Javi. Swing je dio tzv. *Java Foundation Classes* (JFC). Kreiran je u potpunosti u Javi, te je neovisan o platformi, za razliku od AWT (*Abstract Windowing Toolkit*), na osnovu čijeg API-ja je i kreiran sam Swing.<sup>5</sup>

Za razliku od AWT-a, Swing je dosta jednostavniji, te obezbeđuje više elemenata, kao što su tabele, liste, tabovi i sl.

#### 3.2.1. JFrame

JFrame je klasa unutar javax.swing paketa. Korištenje ove klase obezbeđuje prikaz potpuno funkcionalnog pozora na ekranu, na kojeg se mogu postavljati elementi Swing ili AWT kontrola, kao što su labele, polja za unos teksta, dugmad i sl.

Skoro svaka Swing aplikacija pokreće se sa JFrame-om. Za razliku od Frame-a, JFrame ima opciju sakrivanja, odnosno zatvaranja prozora, pomoću metode `setDefaultCloseOperation(int)`.<sup>6</sup>

U sklopu projekta kreiranog u sklopu ovog rada, za rad sa JFrame-om korišteno je okruženje NetBeans, koje omogućava veoma jednostavan rad i manipulaciju elementima u prozoru.

<sup>5</sup> <https://www.javatpoint.com/java-swing>

<sup>6</sup> <https://www.edureka.co/blog/java-jframe/>

### 3.3.NetBeans

NetBeans je razvojno okruženje (tzv. *IDE*) za programski jezik Java. On omogućava da se aplikacije razvijaju iz seta modularnih softverskih komponenti zvanih moduli. NetBeans se može pokrenuti na raznim operativnim sistemima kao što Windows, MacOS, Linux, te Solaris. Osim Java aplikacija, NetBeans pruža usluge razvojnog okruženja i za jezike kao što su PHP, C, C++ i JavaScript, putem ekstenzija.



*Slika 4. NetBeans logo<sup>7</sup>*

NetBeans je započeo kao studentski projekat (prvobitno nazvan „Xelfi“) u Češkoj Republici 1996. godine. Cilj je bio da se u Javi napiše Java IDE (integrirano razvojno okruženje) nalik Delphiju. Xelfi je bilo prvo razvojno okruženje za Javu, koje je bilo napisano na Javi, sa prvim predizdanjima u 1997. godini. Xelfi je bio zabavan projekat za razvoj, pogotovo jer je u to vrijeme razvoj okruženja za Javu prostor bio nepoznanica.<sup>8</sup>

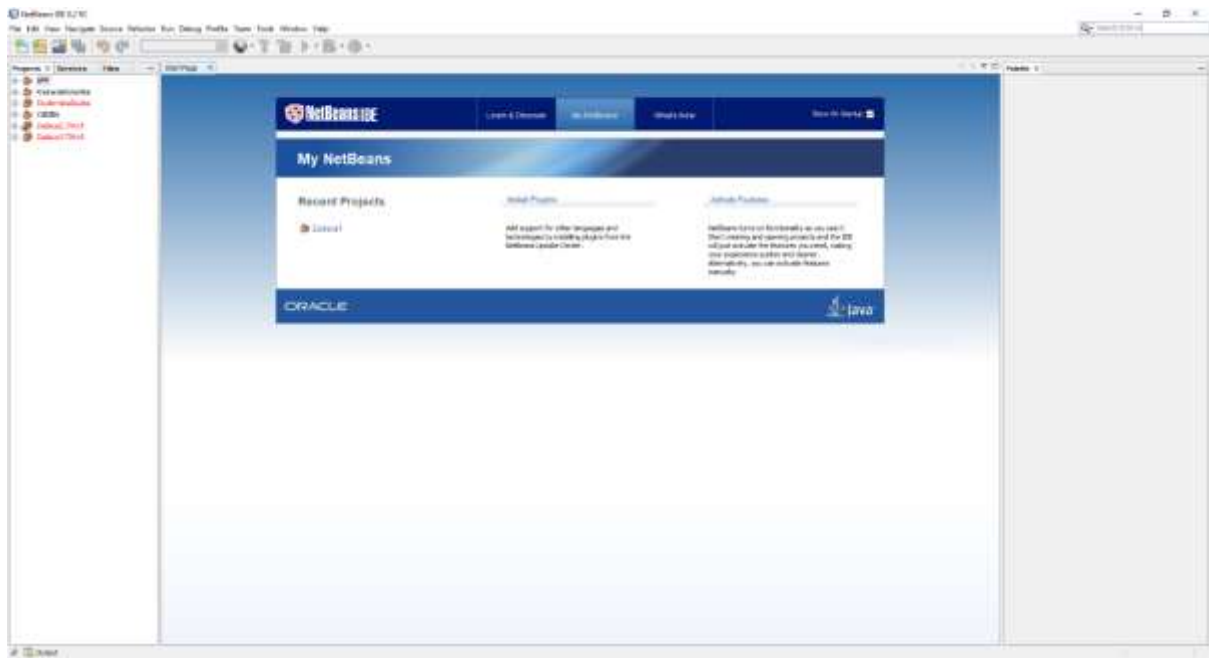
U okviru praktičnog projekta ovog rada korištena je verzija 8.2, preuzeta za zvanične stranice.<sup>9</sup>

---

<sup>7</sup> <https://en.wikipedia.org/wiki/NetBeans>

<sup>8</sup> <https://netbeans.apache.org/about/history.html>

<sup>9</sup> <https://netbeans.org/downloads/8.2/rc/start.html?platform=windows&lang=en&option=all>



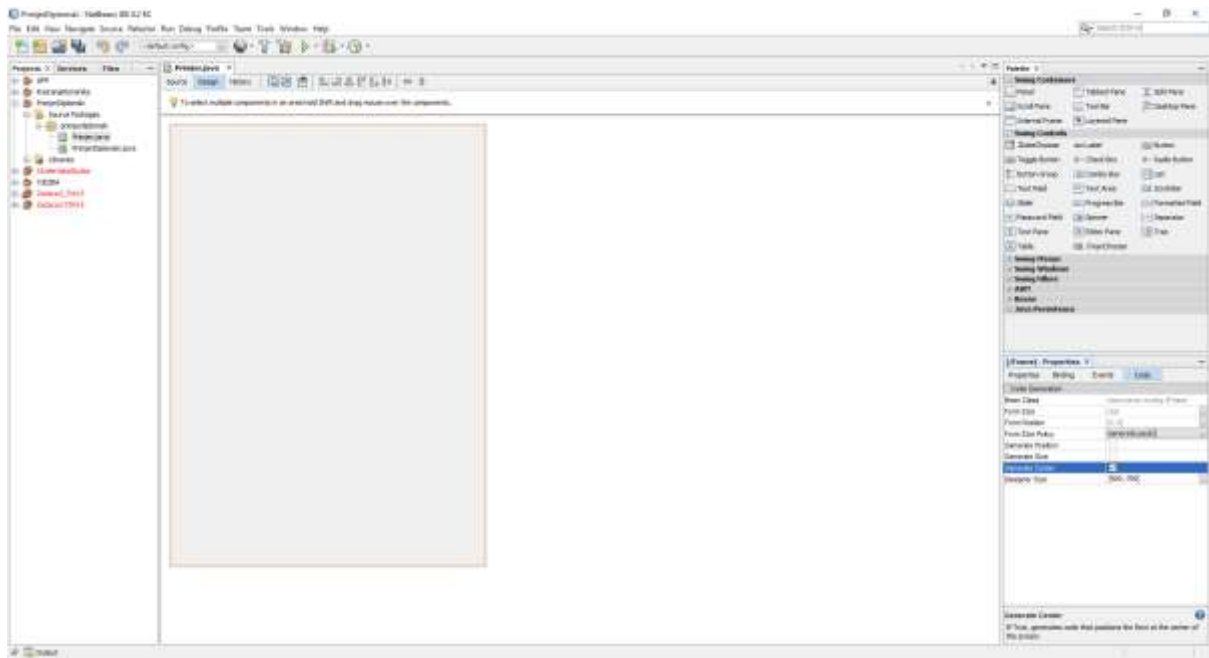
*Slika 5. Početni zaslon NetBeans 8.2 okruženja*

NetBeans okruženje sadrži nekoliko integrisanih modula, a to su NetBeans Profiler, NetBeans JavaScript Editor, te GUI Design Tool, koji nas u ovom slučaju najviše interesuje.

### 3.3.1. NetBeans GUI Design Tool

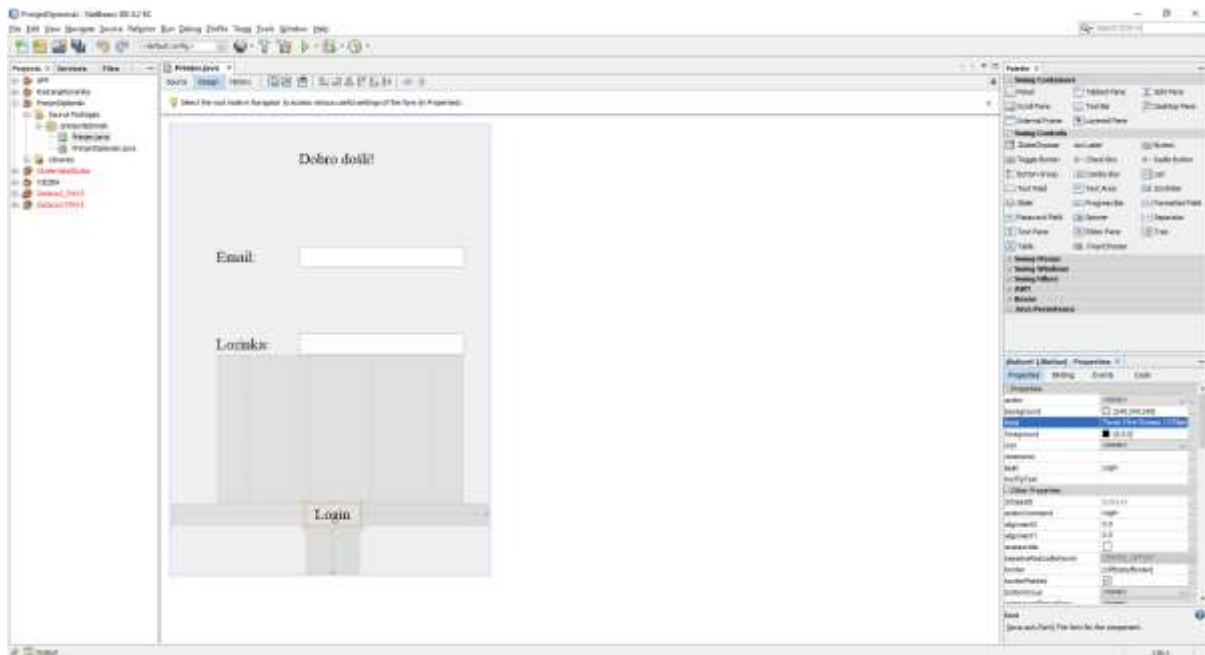
GUI Design Tool u sklopu NetBeans okruženja je modul koji znatno olakšava dizajniranje grafičkog interfejsa u okviru JFrame prozora. On pruža programeru jednostavno upravljanje elementima u prozoru jednostavno „klikom miša“. Kroz kratki primjer ćemo proći kroz osnove ovog modula.

Nakon kreiranja novog projekta „Java Application“, potrebno je kreirati novi JFrame fajl. Na ekranu će biti prikazan „preview“ prozora, nad kojim se mogu vršiti modifikacije kao što su promjena defaultne veličine, pozicioniranje prozora na ekranu pri njegovom otvaranju i sl.



*Slika 6. Prikaz prozora i opcije njegovog modifikovanja u NetBeans okruženju*

U ovom primjeru kreirat ćemo jednu kratnu login formu, korištenjem Swing kontrola. To se postiže veoma jednostavno, korištenjem elemenata iz palete sa desne strane prozora. Koristit ćemo labela za prikaz stacionarnog teksta, polja za unos teksta u svrhu unosa kredencijala, te *submit* dugme. Elementi se veoma jednostavno postavljaju na prozor, jednostavnim klikom na željeni element, te novim klikom na prozor. Elementima se može manipulirati veoma jednostavno unutar prozora, što se ogleda u veoma jednostavnom pozicioniranju elementa u prozoru *drag-and-drop* mehanikom, promjeni veličine elementa, veoma jednostavnom formatiranju teksta elementa, što podrazumijeva promjenu fonta, veličine, boje i sl.



Slika 7. Manipulacija elementima u NetBeansu

Na prozoru prikazanom na slici iznad možemo vidjeti kako je jednostavno manipulirati elementima u prozoru, te da je formu za prijavu moguće napraviti veoma brzo i jednostavno. Ukoliko želimo dodati funkcionalnost na event klika na dugme, sve što trebamo uraditi jeste dvoklik na samo dugme, što će nam automatski u *Source* prikazu klase (prozora) otvoriti funkciju u kojoj možemo kodirati određenu funkcionalnost.

Također, vrijedi istaći da tzv. *event listener* možemo postavljati na bilo koji od elemenata, pa tako naprimjer, u implementacijskom dijelu rada, bit će korišten *element listener* na pojedinim poljima za unos teksta, nakon pritiska dugmeta na tastaturi u slučajevima filtriranja.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}
```

U bloku kôda prikazanom iznad možemo vidjeti funkciju koja se generiše nakon dvoklika na određeni element (u ovom slučaju dugme). Ova funkcija se poziva u slučaju pritiska na dugme u aplikaciji.

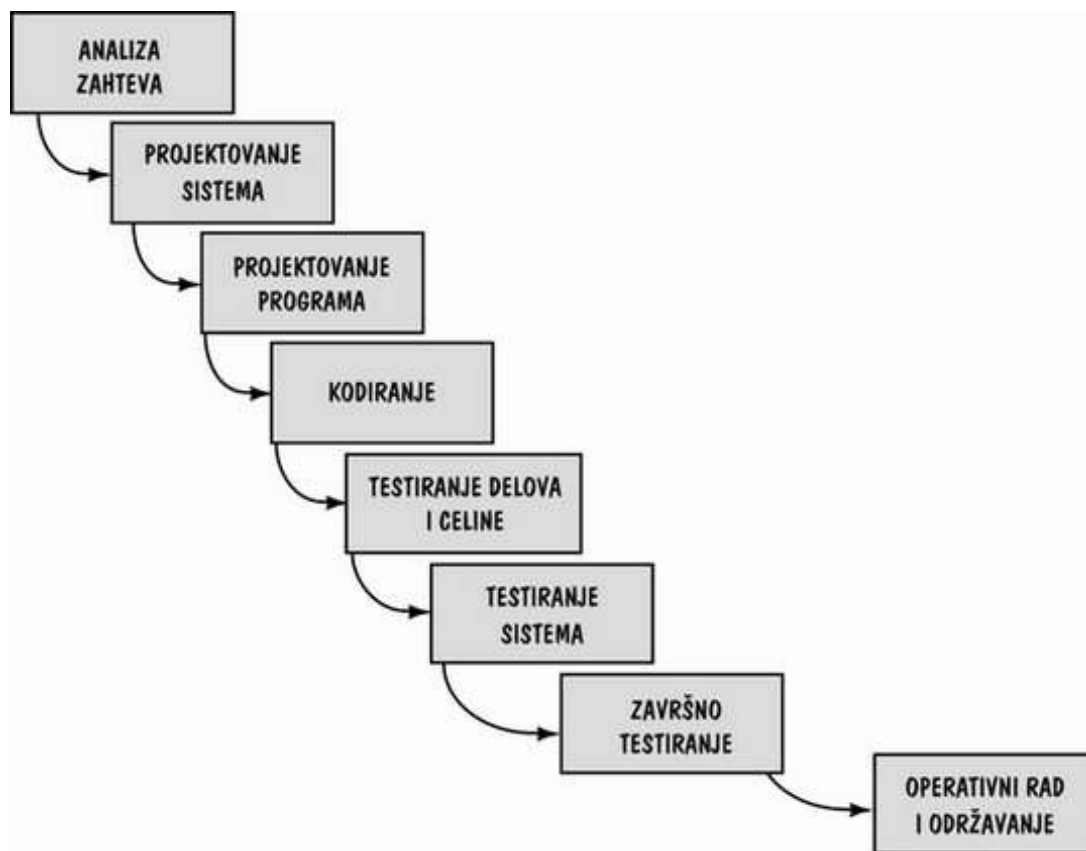
Pored korištenih elemenata, paleta u okviru GUI Design alata sadrži još mnogo elemenata, a neki od njih su tabele, dropdown meniji, liste, polja za unos lozinke (polja za unos teksta sa

skrivenim prikazom unesenih karaktera) itd. Također, moguće je izvršiti i import novih elemenata za unos ili prikaz podataka, preuzimanjem određenih fajlova i njihovim importom unutar samog projekta.

#### 4. IMPLEMENTACIJA

Kao projektni zadatak ovog rada odabrano je kreiranje informacionog sistema za biblioteke na nekom određenom nivou (gradskom, kantonalnom, državnom i sl.). Naziv kreirane aplikacije je „eTeka“.

Pri implementaciji praktičnog dijela ovog rada, korišten je model vodopada. Model vodopada je model u kojem se aplikacija razvija u fazama koje se odvijaju sekvencijalno jedna iza druge. Glavne prednosti razvoja softvera putem modela vodopada su to da je lako pratiti u kojoj fazi se projekat trenutno nalazi, te da omogućava detaljnu razradu i planiranje cjelokupnog procesa razvoja. Faze modela vodopada su prikazane na sljedećoj slici.



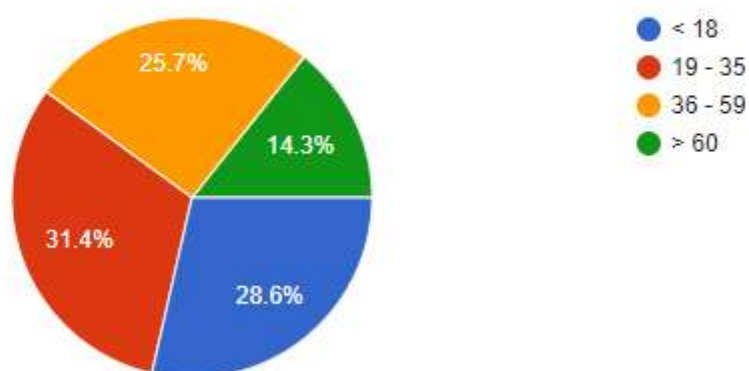
*Slika 8. Faze razvoja softvera u modelu vodopada*

#### 4.1. Analiza zahtjeva

Prva faza razvoja sistema je analiza korisničkih zahtjeva. Ova faza podrazumijeva razne aktivnosti kao što su ispitivanje, te sprovođenje anketa nad potencijalnim korisnicima o tome koje funkcionalnosti bi oni željeli vidjeti u aplikaciji koju bi potencijalno kupili. U sklopu ove faze razvoja projekta, izvršeno je anketiranje potencijalnih korisnika putem platforme Google Forms, a anketi su imali pristup 50 osoba, koje su anketu dobile putem platforme Facebook. Dostavljeno je 35 rezultata ankete (odgovora), a analiza sprovedenog upitnika se nalazi na slikama ispod.

Koja je vaša starosna dob?

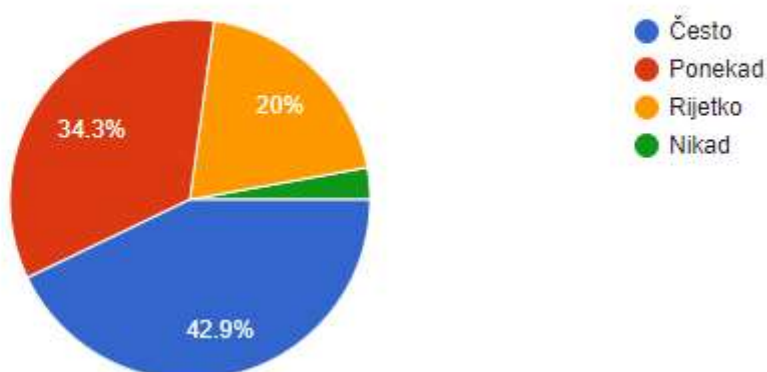
35 responses



Slika 9. Analiza odgovora na prvo pitanje u sprovedenoj anketi

Koliko često koristite usluge biblioteke?

35 responses

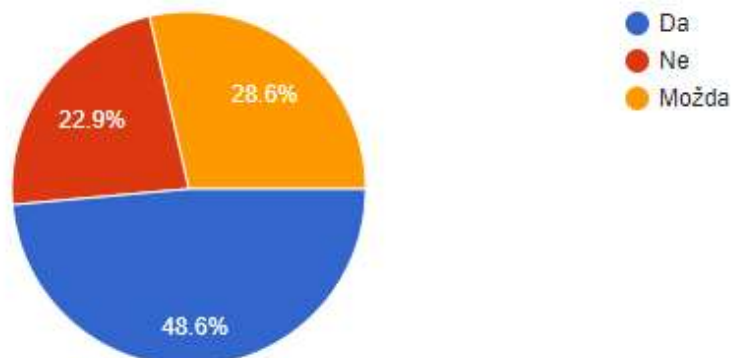


Slika 10. Analiza odgovora na drugo pitanje u sprovedenoj anketi



Da li biste koristili aplikaciju za lakše praćenje izbora knjiga u bibliotekama, njihove dostupnosti kao i detalje praćenja svojih rezervacija?

35 responses



*Slika 11. Analiza odgovora na treće pitanje u sprovedenoj anketi*

Ukoliko Ste odgovorili sa DA ili Možda u prethodnom pitanju, da li postoje neke funkcionalnosti koje biste naročito željeli imati u svojoj aplikaciji?

6 responses

pretraga knjiga

Koliko knjiga ima na stanju

pracenje rezervacije tj period od kad do kad je knjiga kod mene

*Slika 12. Analiza odgovora na četvrto pitanje u sprovedenoj anketi*

Na osnovu sprovedene ankete izvršena je analiza odgovora, nakon koje su ustanovljene sljedeće tražene funkcionalnosti aplikacije:

Za admin korisnika:

- Upravljanje knjigama (što uključuje CRUD – Create, Read, Update, Delete operacije);
- Upravljanje poslovnica (što uključuje CRUD operacije);
- Upravljanje zaposlenicima (što uključuje CRUD operacije);
- Pregled aktivnih i neaktivnih rezervacija, svih korisnika u svim poslovnica;
- Brisanje rezervacija.

Za uposlenika biblioteke:

- Upravljanje korisnicima (što uključuje CRUD operacije);
- Pregled knjiga u poslovnici gdje je zaposlen uposlenik, sa opcijama pretrage i filtriranja;
- Pregled rezervacija u poslovnici uposlenika;
- Pregled računa (account-a) uposlenika;
- Promjena lozinke.

Za korisnika aplikacije:

- Pregled knjiga sa njihovim stanjem u bilo kojoj poslovnici, sa opcijama pretrage i filtriranja;
- Kreiranje rezervacije;
- Uklanjanje postojeće rezervacije;
- Pregled svih rezervacija korisnika, sa detaljima o knjigama, te datumima početka i završetka rezervacija;
- Pregled računa (account-a) korisnika;
- Promjena lozinke.

Na osnovu ustanovljenih funkcionalnosti, dobijenih na osnovu korisničkih zahtjeva, izvršen je prelaz na sljedeću fazu razvoja sistema – analizu i dizajn.

#### **4.2. Analiza i dizajn sistema**

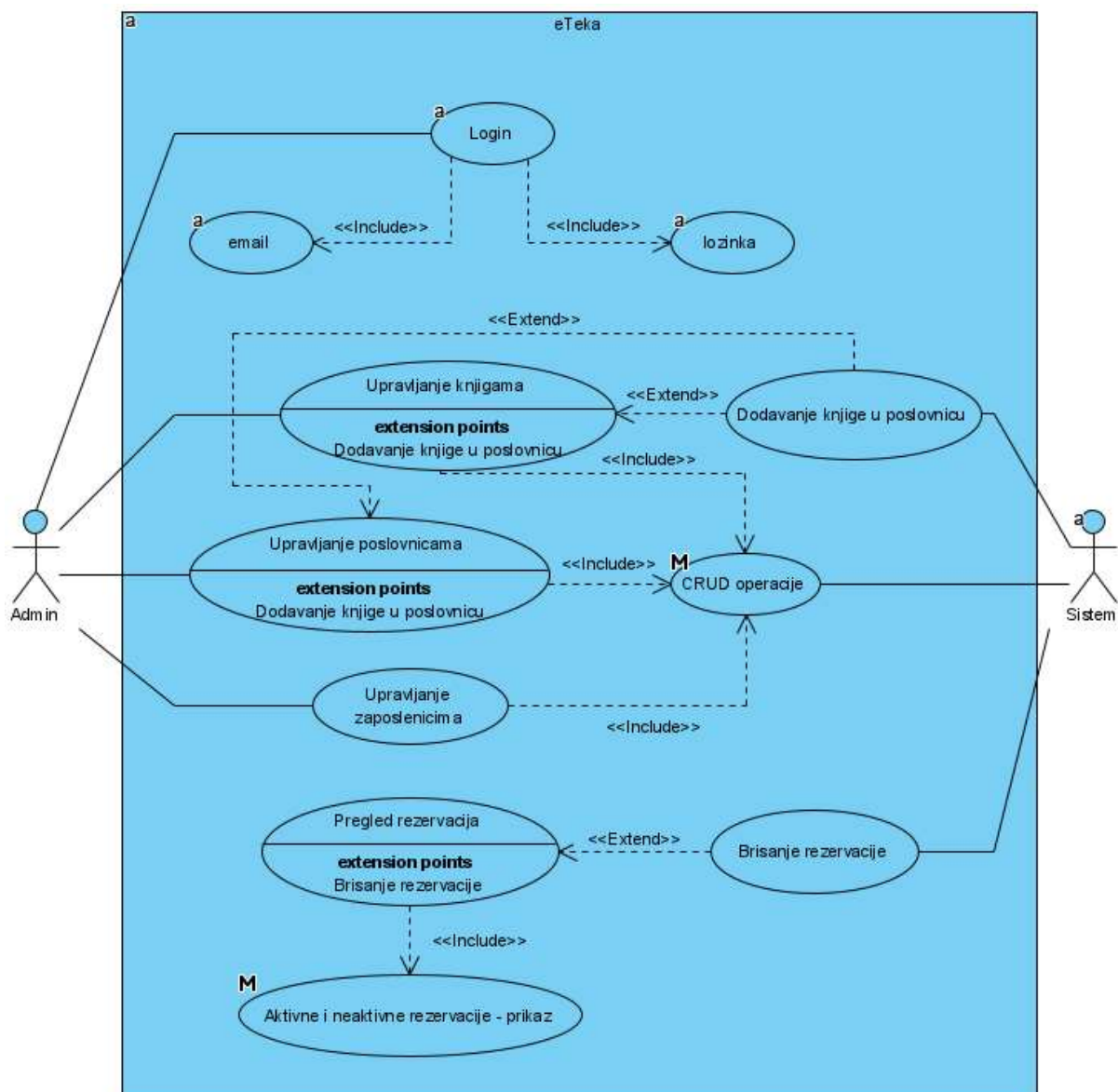
U fazi analize i dizajna sistema, svi objekti i akteri sistema se prikazuju grafički, zajedno sa svim funkcionalnostima. U ovoj fazi se naime vrši projekcija sistema, tj. prikaz međusobno povezanih objekata u sistemu, na način na koji bi trebali funkcionisati u razvijenom sistemu.

Sistemi se mogu predstaviti u formi dijagrama, kojih ima jako mnogo. Za izradu ovog projekta, dijagrami korišteni u fazi analize i dizajna su Use Case dijagram (dijagram slučaja korištenja), Class (klasni) dijagram, te ERD – dijagram (Entity Relationship Diagram).

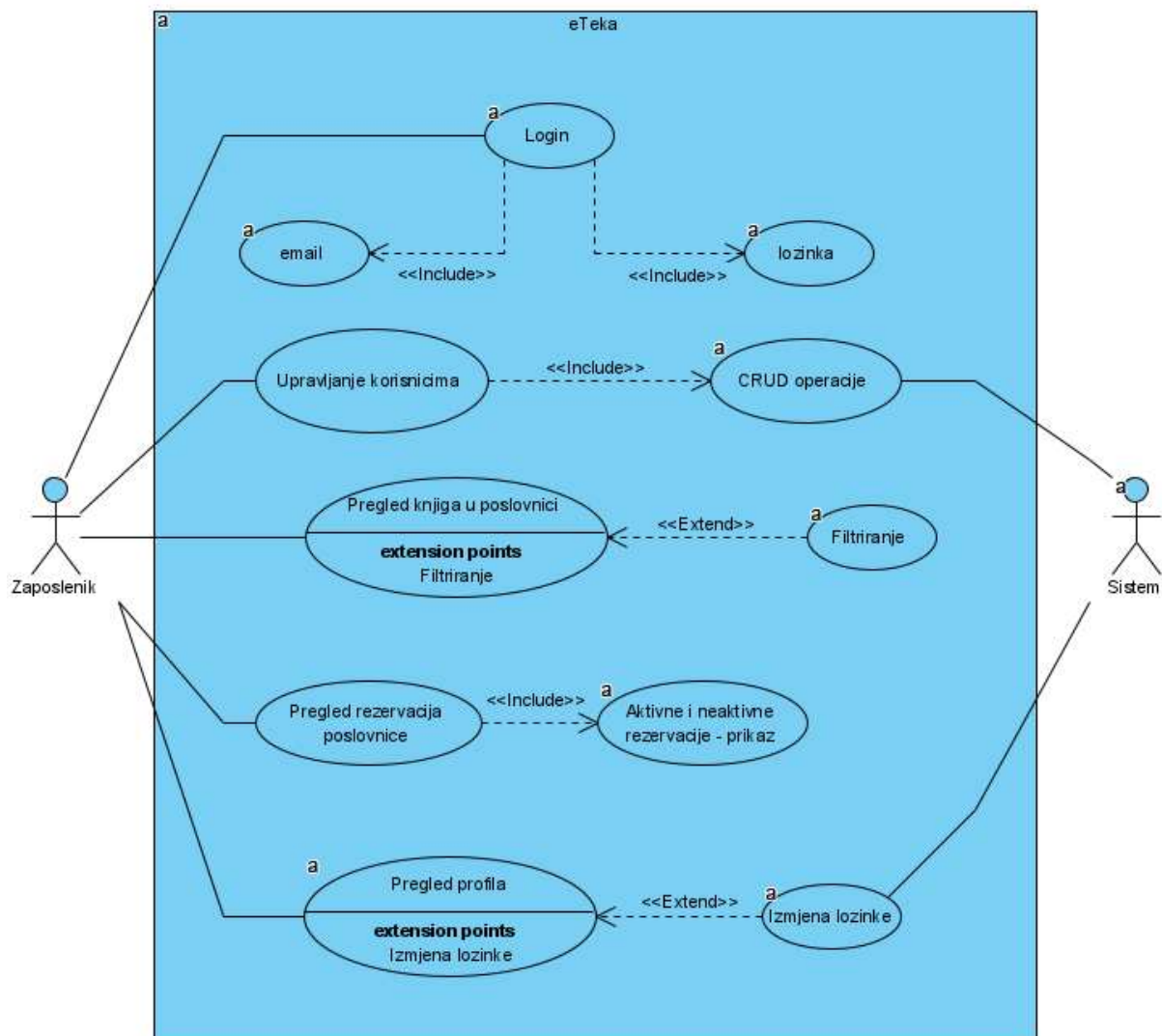
#### 4.2.1. Use Case dijagrami

Use Case dijagram je dijagram koji na slikovit način opisuje funkcionalnosti u sistemu za svakog aktera. Sastoji se od nekoliko glavnih dijelova: sistem, koji je prikazan u vidu četverougla, i koji se obično nalazi između aktera; sa lijeve strane se nalazi akter/akteri čije se funkcionalnosti predstavljaju, dok je na desnoj strani akter koji predstavlja servis, i sa akterom sa lijeve strane vrši interakciju putem tzv. use case-ova (slučajeva korištenja). Use case je osnovna komponenta istoimenog dijagrama, i uglavnom predstavlja funkcionalnost. Obično se nalazi unutar granica sistema. Akteri se sa slučajevima povezuju raznim vrstama veza, a najčešće korištene veze su asocijacija, veza uključivosti, te veza proširenja. Veza asocijacije se koristi uglavnom pri povezivanju aktera sa slučajevima, dok se slučajevi međusobno povezuju sa vezom uključivosti (koristi se kada jedan use case obavezno uključuje drugi), ili vezom proširenja (koristi se kada jedan use case neobavezno uključuje drugi).

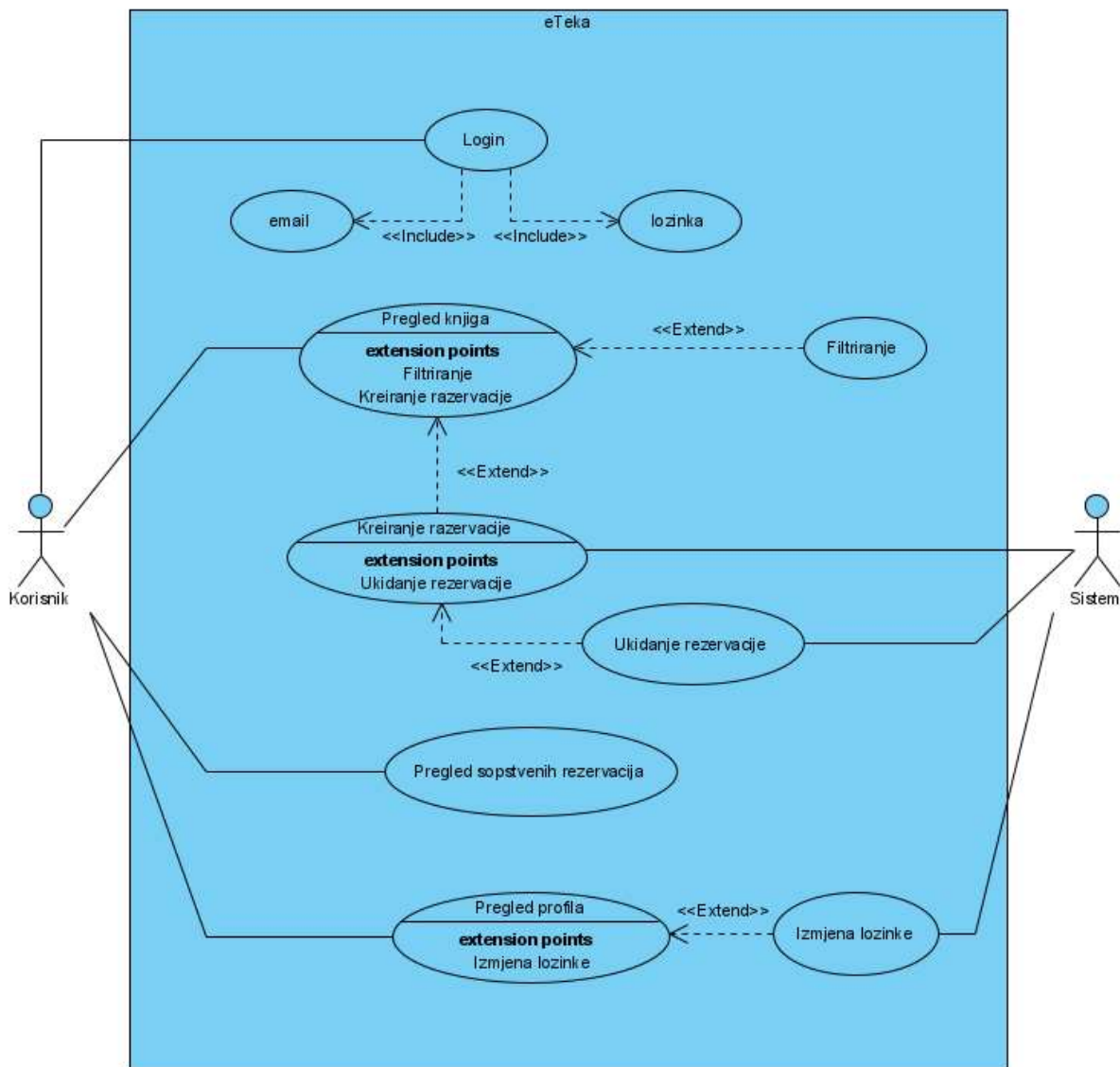
U sklopu ovog rada kreirana su tri use case dijagrama, po jedan za svakog aktera u sistemu (admin uposlenik, korisnik), koji su prikazani u slikama ispod.



Slika 13. Use Case dijagram za admin korisnika



Slika 14. Use Case dijagram za uposlenika

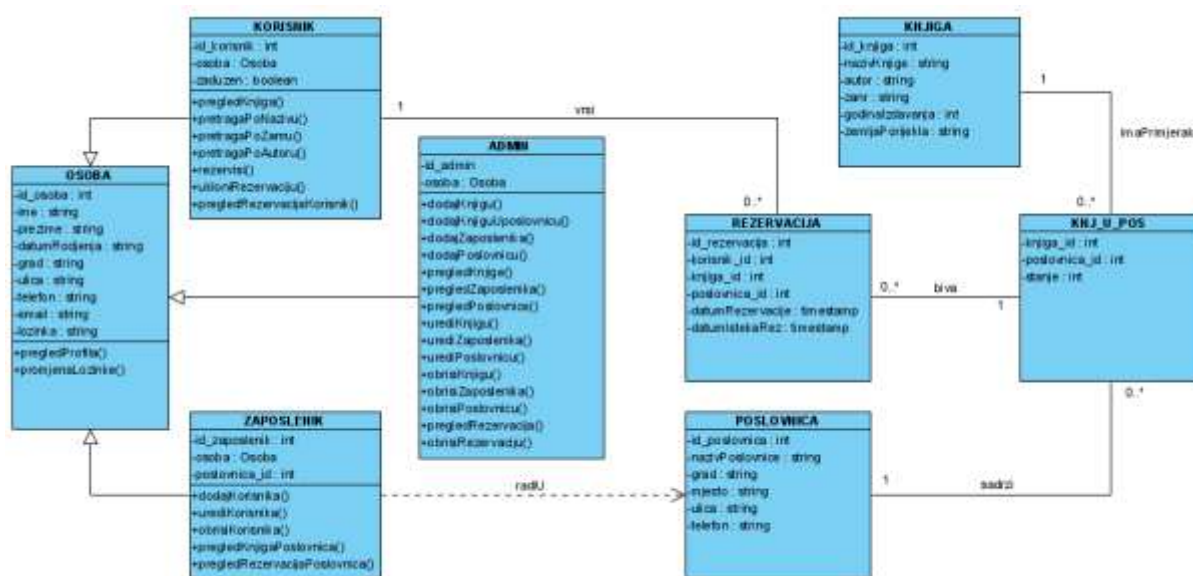


Slika 15. Use Case dijagram za korisnika

Ono što možemo ustanoviti na use case dijagramima jeste da, pored pobrojanih funkcionalnosti, svaki akter ima i zajedničku funkcionalnost logina (prijava) na sistem, putem emaila i lozinke. Također, vrijedi istaći da, u dijagramima za admin korisnika te uposlenika, use case „CRUD operacije“ bi trebao biti raščlanjen na sve operacije, tj. svaka CRUD operacija za svaki objekat bi trebala imati svoj use case zasebno. Međutim, zbog praktičnosti, svedene su na jedan use case, koji je povezan za slučajevima upravljanja, te sa akterom „Sistem“, zbog vršenja promjena unutar same aplikacije.

#### 4.2.2. Class diagram

Class (klasni) dijagram je dijagram koji dosta detaljnije opisuje strukturu samog sistema. Elementi ovog dijagrama su klase, što se i osnovni elementi ovog dijagrama, a predstavljaju zasebne cjeline u sistemu. Okarakterisane su atributima i operacijama. Atributi su elementi koji pobliže opisuju klasu, dok operacije možemo nazvati nekom vrstom događaja, koja se odvija na određeni okidač (*trigger*), i služi za dodavanje, brisanje, učitavanje ili ažuriranje jednog ili više atributa jednog ili više objekata jedne ili više klasa. Klasni dijagram kreiran za ovaj projekat nalazi se na slici ispod.



Slika 16. Class dijagram

Klase se međusobno povezuju raznim vrstama veza, pa tako i, kao kod use case dijagrama, korišteno je više vrsta veza za povezivanje klasa. Osnovni tip veze je asocijacija, koja je prikazana u vidu pune linije. Sastoji se od naziva veze, te kardinalnosti. Kardinalnost veze definiše koliko se instanci jedne klase može povezati sa instancama druge klase.<sup>10</sup> Uzmimo za primjer vezu između klase korisnik, i klase rezervacija. Jedan korisnik u sistemu može da ne izvrši niti jednu rezervaciju, a može da izvrši i samo jednu, ali i više od jedne rezervacije, zbog čega pored klase rezervacija pored veze sa klasom korisnik stoji oznaka **0..\***. Sa druge strane, rezervacija ne postoji ako ne postoji korisnik, što znači da je za rezervaciju sigurno potreban jedan korisnik. Tu istu rezervaciju ne može napraviti niti više od jednog korisnika,

<sup>10</sup> <https://vertabelo.com/blog/cardinality-in-data-modeling/>

samo jedan. Zbog toga, pored klase korisnik, na vezi sa klasom rezervacija, stoji oznaka/broj 1. Jednu rezervaciju može napraviti jedan i samo jedan korisnik.

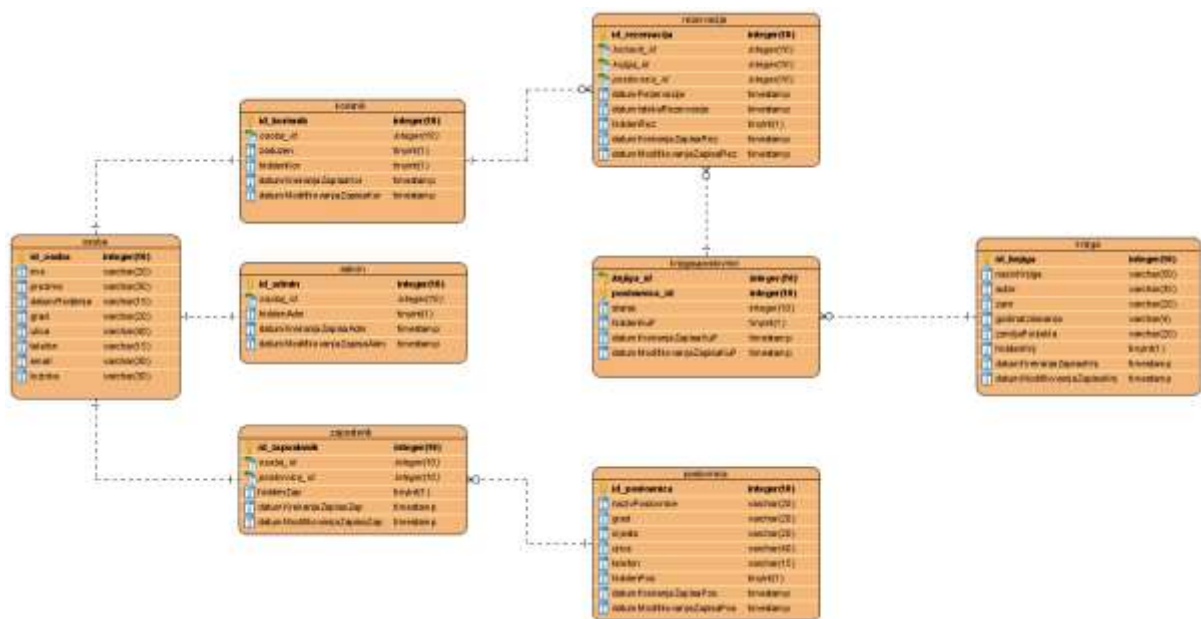
Pored asocijacije, postoje i druge veze za povezivanje klasa, a u ovom projektu korištene su još i generalizacija, te veza ovisnosti. Veza generalizacije predstavlja slikovit prikaz koncepta nasljeđivanja. Ovaj tip veze korišten je između klase Osoba, te klasa Korisnik, Admin i Zaposlenik. Ta veza nam govori da, radi preglednosti, praktičnosti te mogućnosti ponovnog korištenja kôda, attribute i metode jedne klase možemo koristiti u nekoj drugoj klasi/klasama, bez potrebe da više puta vršimo deklaraciju i/ili inicijalizaciju atributa/operacija. Na primjer, svi atributi i operacije klase Osoba su uz pomoć ove veze „naslijeđene“ u tzv. *child* klase, ili podklase Korisnik, Admin i Zaposlenik, gdje će te klase, uz naslijeđene elemente imati i svoje vlastite attribute i metode. Veza generalizacije se predstavlja punom linijom, sa trouglastom strelicom prema klasi iz koje se elementi nasljeđuju.

Veza ovisnosti je vrlo jednostavna; koristi se u slučajevima kada je postojanje jedne klase/instance klase nemoguće bez postojanja druge klase/instance klase. U ovom projektu je korištena za povezivanje klasa Zaposlenik i Poslovnica, jer zaposlenik ne može raditi u poslovnici ako poslovnice nema, ili ista nestane. Označava se isprekidanom linijom sa strelicom okrenutom prema klasi od koje druga klasa ovisi, a sadrži još i naziv veze, u ovom slučaju *radiU*.

#### 4.2.3. ER – dijagram

ER dijagram (Entity Relationship Diagram) je dijagram vrlo sličan klasnom dijagramu. Sadrži entitete, koji su skoro identični klasama, njihove attribute sa tipovima podataka, te veze između tih entiteta koje su „dekorisane“ kardinalnostima. Naime, ER dijagram predstavlja skicu strukture podataka iz baze podataka. Pokazuje kako su entiteti struktuirani, povezani jedni sa drugima, i preko kojih atributa. Svaki entitet ima svoj primarni ključ, koji je označen simbolom ključa odmah pored atributa koji je primarni ključ. Postoje i tzv. strani ključevi (eng. *foreign key*), koji predstavljaju reference primarnih ključeva drugih entiteta u tom samom entitetu. Na sljedećoj slici prikazan je ER dijagram koji je kreiran za ovaj projekat.





Slika 17. ER dijagram

Praksa je da primarni ključ bude neki bročani (integer) *auto-increment* element. Na taj način se osigurava jedinstvenost ključa, te je nemoguć slučaj ponavljanja iste vrijednosti primarnog ključa unutar iste tabele. Također, primarni ključ može činiti jedan, ali i više elemenata (tzv. kompozitni ključ). To možemo vidjeti i ovdje, konkretno unutar entiteta „knjigauposlovnici“, gdje elementi *knjiga\_id* i *poslovnica\_id* čine kompozitni primarni ključ, dok element *knjiga\_id* u isto vrijeme predstavlja i strani ključ koji kao referencu uzimaju primarni ključ iz entiteta Knjiga. Dakle, moguće je da jedan ili više elemenata formiraju i primarni, ali i da imaju funkciju spoljnog (vanjskog) ključa u isto vrijeme.

Na ovom dijagramu se može primijetiti da su u svakoj tabeli dodane po tri kolone (zadnja tri atributa u svakom od entiteta) koje u velikoj mjeri pružaju sigurnost, no o tome će biti govora kasnije.

### 4.3.Kreiranje aplikacije

Nakon uspješne faze analize i dizajna sistema, dobijeni dijagrami mogu poslužiti za kodiranje aplikacije. Pri izradi ove aplikacije korišten je WampServer, koji je preuzet sa zvanične stranice.<sup>11</sup> Kreiranje i struktuiranje baze podataka znatno je olakšano alatom koji dolazi u

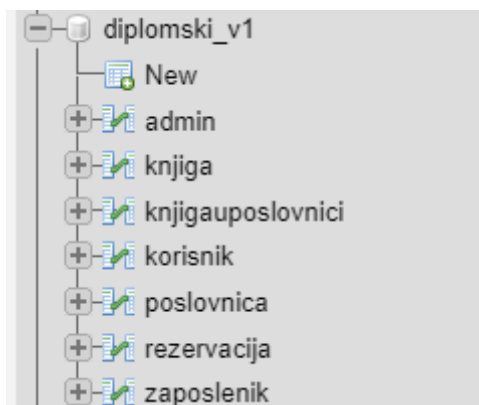
<sup>11</sup> <https://sourceforge.net/projects/wampserver/>

sklopu WampServera, koji se zove *phpMyAdmin*. Ovaj alat uveliko olakšava kreiranje tabela i definisanje atributa, njihovih ograničenja, ali i definisanje primarnih ključeva.



*Slika 18. Kreiranje tabele "knjiga" korištenjem alata phpMyAdmin*

Na slici iznad prikazan je način kreiranja tabele i kolona unutar te tabele. Na ovaj način, kreirane su sve tabele unutar baze podataka.

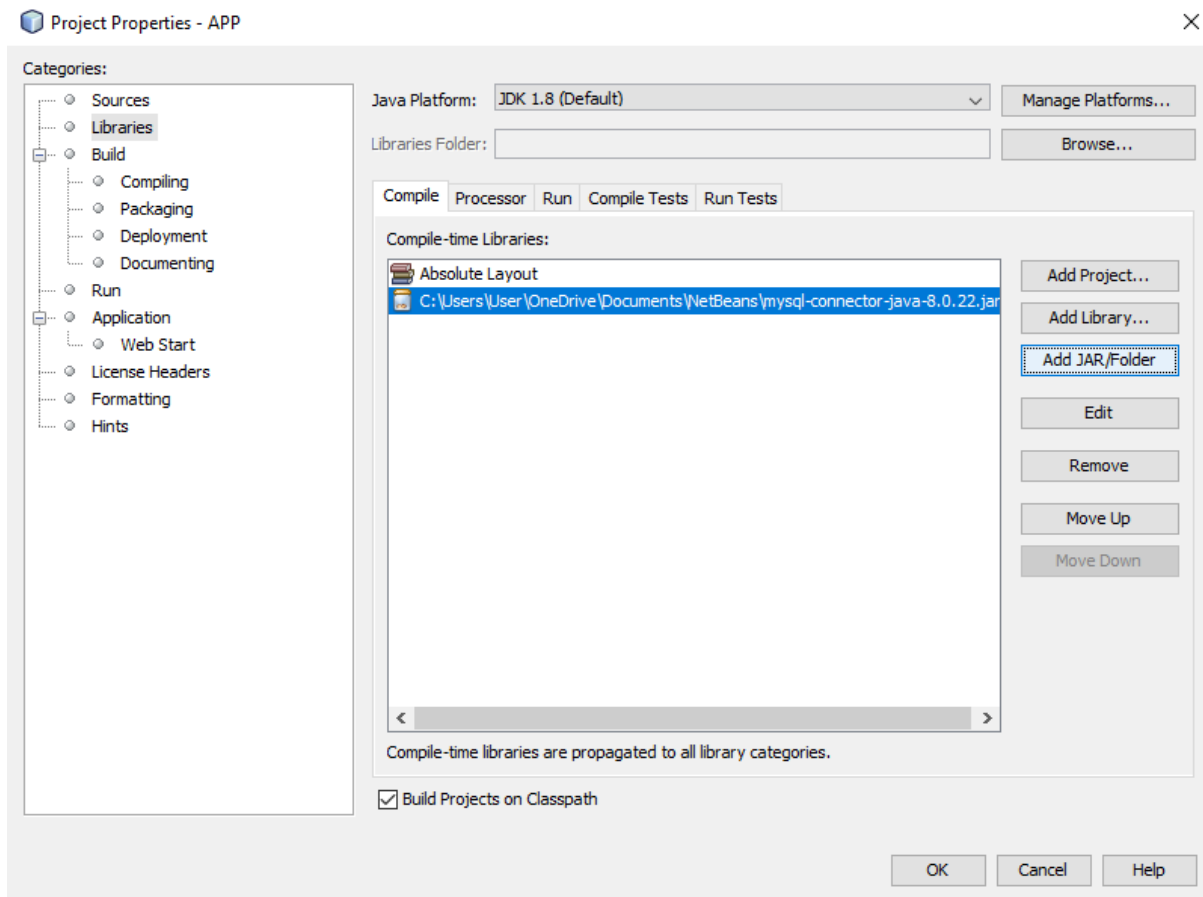


*Slika 19. Prikaz kreiranih tabela u alatu phpMyAdmin*

Nakon kreiranja tabela, njihovih atributa te primarnih ključeva definisani su strani ključevi u okruženju MySQL Workbench, preuzetom sa zvanične stranice.<sup>12</sup> Dakle, baza je uspješno kreirana, i sljedeći korak jeste početak rada u NetBeans okruženju.

Prvi korak u NetBeans okruženju jeste dodavanje MySQL konektora, desnim klikom na kreirani projekat, te dodavanjem JAR file-a konektora. Naziv, tj. verzija konektora korištenog u ovom projektu je „mysql-connector-java-8.0.22“.

<sup>12</sup> <https://dev.mysql.com/downloads/workbench/>



*Slika 20. Import konektora za povezivanje sa bazom podataka u Java projekat*

Sljedeći korak jeste napraviti java klasu u kojoj će biti smješten kôd kojim ćemo projekat povezati sa bazom podataka preko konektora. Kôd je vrlo jednostavan i može se gotovo cio koristiti u više projekata, osim što je potrebno promijeniti naziv baze podataka, ili eventualne kredencijale autorizacije za bazu podataka. U slučaju ovog projekta, klasa, tj. fajl je nazvan Konekcija, i osim kôda za spajanje na bazu, klasa Konekcija će sadržati i sve funkcije koje će biti pozivane u ovoj aplikaciji, a koje zahtjevaju komuniciranje sa bazom, bilo da se radi o metodama koje imaju neku povratnu vrijednost ili o metodama kojima se vrši manipulacija nad podacima.

U bloku kôda prikazanom ispod nalazi se prikaz klase Konekcija, te njen konstruktor, kojim se vrši povezivanje na bazu podataka.

```

public class Konekcija {

    private static final String korisnik = "root";
    private static final String sifra = "";
    private static final String kon =
"jdbc:mysql://localhost:3306/diplomski?serverTimezone=UTC";
    public static Connection veza = null;

    public Konekcija() throws SQLException, ClassNotFoundException {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            veza = DriverManager.getConnection(kon, korisnik, sifra);
        } catch (SQLException e) {
            System.err.println(e);
        }
    }
}

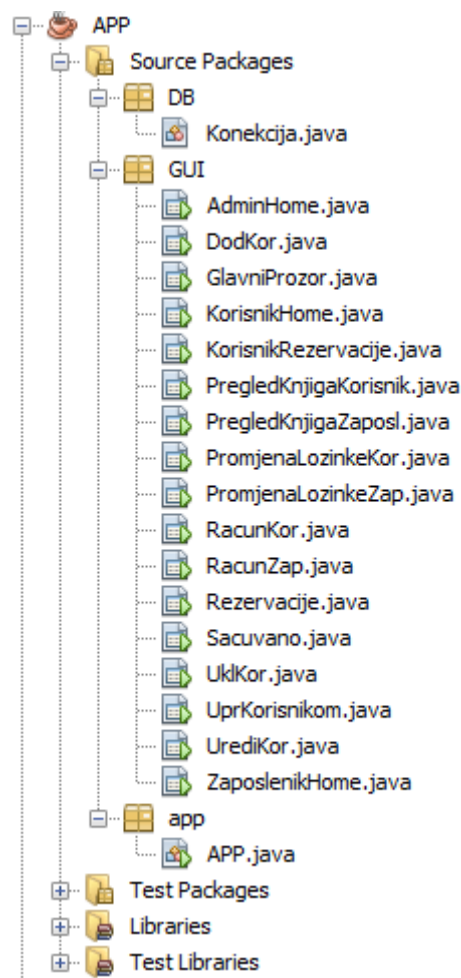
```

Najlakši način testiranja konekcije bi bio dodavanje ispisa unutar *try* bloka sa nekim *string*-om, koji bi se ispisao u konzoli kada bi se konstruktor Konekcija() pozvao u glavnom java fajlu.

Nakon što smo ustanovili da je konekcija Java projekta sa bazom podataka uspješno realizovana, može se početi sa kodiranjem. Radi preglednosti i bolje organizacije projekta, fajlovi su raspoređeni u foldere, koji se dijele po svrsi fajla/fajlova koji se u njima nalaze. Pa tako, fajlovi ove aplikacije su raspoređeni u 3 foldera:

- DB – gdje je smještena klasa Konekcija;
- GUI – gdje su smješteni svi elementi grafičkog sučelja (u ovom slučaju JFrame prozori);
- app – gdje je smješten glavni (main) fajl projekta, mjesto gdje se projekat pokreće.

Treba istaći i da je naziv projekta „APP“, te je zbog toga folder u kome se main klasa nalazi dobio ime generisano na osnovu imena projekta.



Slika 21. Prikaz svih kreiranih klasa u Java projektu

Na slici iznad prikazane su sve klase, tj. fajlovi koji su kreirani u ovom projektu. Pri pokretanju same aplikacije, pokreće se kôd koji se nalazi u klasi „APP.java“.

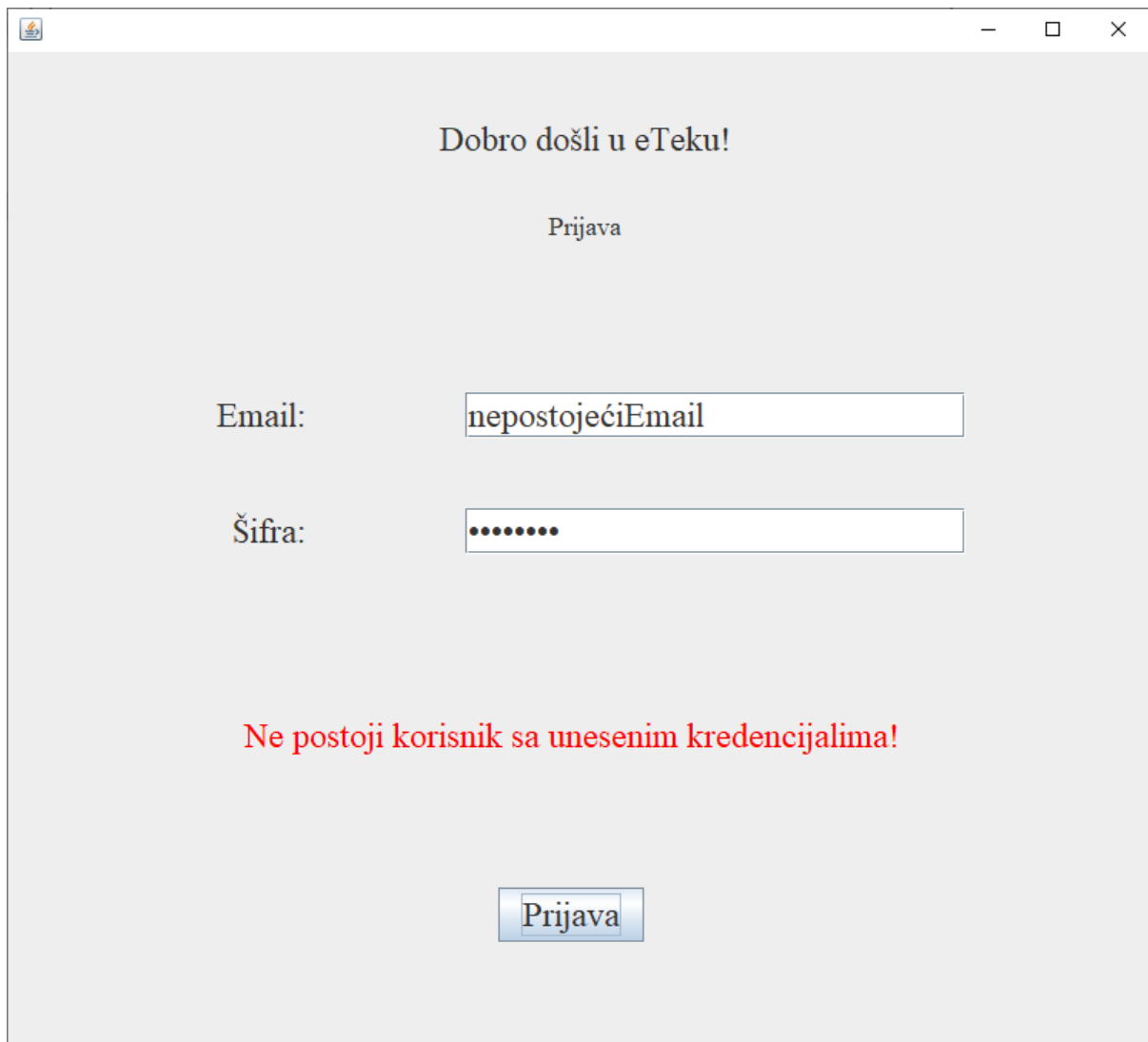
```
package app;

import DB.Konekcija;
import GUI.GlavniProzor;
import java.sql.SQLException;

public class APP {

    public static void main(String[] args) throws SQLException,
ClassNotFoundException {
        GlavniProzor gp = new GlavniProzor();
        gp.setVisible(true);
    }
}
```

U glavnoj klasi programa se ne nalazi ništa osim inicijalizacije objekta klase početnog prozora aplikacije, nazvanog *GlavniProzor*.



*Slika 22. Prikaz glavnog prozora, te pokušaj prijave sa nevalidnim podacima*

Na slici iznad prikazan je glavni prozor, tj. *login page* aplikacije. Nakon unijetih pogrešnih podataka za login i pritisnutog *button*-a za prijavu, ispisuje se poruka za nemoguć daljnji pristup aplikaciji. Međutim, unošenjem validnih kredencijala, nakon prijave se otvara jedan od 3 moguća prozora: prozor za admin korisnika, prozor za uposlenika, ili prozor za korisnika aplikacije.

#### 4.3.1. Admin page

Admin prozor aplikacije eTeka struktuiran je u vidu tab-ova, s tim što su svi tab-ovi koji sadrže CRUD operacije nad nekim objektima dalje struktuirani u podtab-ove. Struktura admin stranice prikazana je na sljedećoj slici:

Upravljanje knjigama Upravljanje uposlenicima Upravljanje poslovnicama Pregled rezervacija Više opcija

Dodaj Dodaj u poslovnicu Pretraži Izmijeni Ukloni

Odaberite knjigu koju dodajete, poslovnicu, te količinu:

Poslovnica:

Knjiga:

NAZIV	AUTOR	ŽANR	Godina izdavanja	Godina izdavanja	Zemlja porijekla
Hamlet	William Shakespeare	Tragedija	1612		Engleska
Na vani Clara Shusterman - mala priča	David Lagercrantz	Thriller	2012		Švedska
Uzavrat: prvi roman	Peter Rabe	Stranica Stranica	2004		Švedska i Norveška
Uzavrat: drugi roman	Henrik Capri	Stranica Stranica	1949		Švedska i Norveška
Uzavrat: treći roman	Henrik Capri	Stranica Stranica	1918		Švedska i Norveška
Pravica	Carlo Collucci	Stranica Stranica	1881		Italija
Hamlet i kralj	William Shakespeare	Tragedija	1567		Engleska

Količina:

Slika 23. Struktura prozora za admin korisnika

Dakle, admin ima sva najveća ovlaštenja, tj. ima pravo na direktan unos, izmjenu ili brisanje podataka iz baze za objekte knjiga, uposlenik, te poslovnica. Međutim, za objekat knjiga, admin može izvršiti i dodatnu operaciju unosa knjige u poslovnicu, na način da iz tabele odabere knjigu, iz padajućeg menija poslovnicu u koju se ta knjiga unosi, te količinu, tj. stanje, odnosno koliko primjeraka te knjige će biti dostupno u odabranoj poslovnici. Ukoliko odabrana knjiga u odabranoj poslovnici već postoji, poziva se druga metoda, koja ima ulogu dodavanja unesene količine primjeraka na već postojeći broj primjeraka te knjige. Sve ovo odvija se nakon pritiska dugmeta „Spremi“, a kôd za čije izvođenje je ovo dugme okidač izgleda ovako:

```

private void btnSubmitKnjiPosActionPerformed(java.awt.event.ActionEvent evt) {
    lblMsgUnosKnjUposl.setText("");
    String poslovnica = cmbPoslovnicaKnj.getSelectedItem().toString();
    ResultSet poslovnicaRS;
    int poslovnica_id = 0;
    int knjigaRed = tblKnjigeKnjPos.getSelectedRow();
    ResultSet knjigaRS;

    String knjiga = (String)
tblKnjigeKnjPos.getModel().getValueAt(knjigaRed, 0);
    int knjiga_id = 0;
    String kolicinaTxt = txtKolicinaKnj.getText();
    int kolicina = Integer.valueOf(kolicinaTxt);
    boolean postoji = false;
    try {
        Konekcija k = new Konekcija();

        poslovnicaRS = k.getCmbID(poslovnica);
        while (poslovnicaRS.next()) {
            poslovnica_id = poslovnicaRS.getInt(1);
        }

        knjigaRS = k.getKnjID(knjiga);
        while (knjigaRS.next()) {
            knjiga_id = knjigaRS.getInt(1);
        }

        ResultSet rezultat = k.KnjigeStanje(poslovnica_id, knjiga_id);
        while (rezultat.next()) {
            int PoslIdBaza = rezultat.getInt("poslovnica_id");
            int knjIdBaza = rezultat.getInt("knjiga_id");
            if (knjIdBaza == knjiga_id && PoslIdBaza == poslovnica_id) {
                postoji = true;
            }
        }
        if (postoji) {
            k.updateStanja(poslovnica_id, knjiga_id, kolicina);
            lblMsgUnosKnjUposl.setText("Uspješno povećano za unesenu
količinu!");
            lblMsgUnosKnjUposl.setForeground(Color.green);
        } else {
            k.unosKnjigeUposlovnicu(poslovnica_id, knjiga_id, kolicina);
            Sacuvano s = new Sacuvano();
            s.setVisible(true);
        }
    } catch (SQLException | ClassNotFoundException ex) {
        System.err.println(ex);
    }
}

```



U bloku kôda prikazanom iznad prikazana je logika unosa određenog broja knjiga u odabranu poslovnicu. U konstruktoru klase AdminHome (tj. pri otvaranju prozora admin stranice) dešava se dohvaćanje podataka za *dropdown* meni u kojem je ispisana lista poslovnica, te za tabelu u kojoj su ispisani detalji o knjigama. Putem funkcija za spašavanje naziva poslovnice koja je odabrana u trenutku pritiska na dugme „Spremi“, te knjige koja je u tom trenutku selektovana, dolazimo do njihovog ID-a kroz interakciju sa bazom podataka, tj. šaljemo nazive knjige i poslovnice, kako bismo izvukli njihov ID. Kada imamo potrebne ID-eve, jedino što ostaje jeste da se pokupi podatak iz tekstualnog polja za količinu primjeraka knjiga. Nakon toga, ispitujemo da li kombinacija odabranih poslovnice i knjige već postoji u bazi; ukoliko ne postoji, dolazi do kreiranja novog zapisa u tabeli „knjigauposlovnici“, kojeg čine dva ID-a (poslovnice i knjige), te unijeto stanje količine primjeraka. Ukoliko pak već postoji unesena kombinacija ID-ova, dolazi samo do update-a stanja, gdje se trenutna količina primjeraka samo povećava za novu unijetu količinu.

Kao što je ranije rečeno, sve funkcije koje vrše interakciju sa bazom podataka nalaze se u klasi Konekcija. Te funkcije su veoma jednostavne, i sastoje se od SQL kôda koji se treba izvršiti na neki *trigger* (okidač), što je u ovom slučaju pritisak na dugme. U sljedećim blokovima kôda prikazani su primjeri funkcija koje se nalaze u klasi „Konekcija“.

```
public void unosKnjigeUposlovnicu(int poslovnica_id, int knjiga_id, int kolicina) throws SQLException {
    Statement upitBaza = (Statement) veza.createStatement();
    String upit = "INSERT INTO knjigauposlovnici (poslovnica_id, knjiga_id, stanje) VALUES ('" + poslovnica_id + "','" + knjiga_id + "','" + kolicina + "')";
    try {
        upitBaza.executeUpdate(upit);
    } catch (SQLException e) {
        System.err.println(e);
    }
}
```

```
public void updateStanja(int poslovnica_id, int knjiga_id, int novaKolicina) throws SQLException {
    Statement upitBaza = (Statement) veza.createStatement();
    String upit = "update knjigauposlovnici set stanje=stanje+" + novaKolicina + " where poslovnica_id=" + poslovnica_id + " and knjiga_id=" + knjiga_id + "";
    try {
        upitBaza.executeUpdate(upit);
    } catch (SQLException e) {
```

```

        System.err.println(e);
    }
}

```

```

public ResultSet ispisKnjiga() throws SQLException {
    Statement upitBaza = (Statement) veza.createStatement();
    ResultSet rezultat = null;
    try {
        rezultat = upitBaza.executeQuery("Select nazivKnjige, autor, zanr,
datumIzdavanja, zemljaPorijekla from knjiga where hiddenKnj = 0 order by 1");
        return rezultat;
    } catch (SQLException e) {
        System.err.println(e);
    }
    return rezultat;
}

```

Dakle, kao što možemo vidjeti, funkcije u klasi Konekcija nisu kompleksne, te samo sadrže upit (*query*) koji se pokreće prilikom okidanja koje dovodi do pozivanja funkcije. Funkcije koje su prikazane u primjerima iznad su funkcije za unos, update i učitavanje podataka, i uglavnom ovako izgledaju njihove strukture.

Međutim, ono što je pomalo neobično a što možemo vidjeti u funkciji za ispis knjiga, jeste pojava atributa *hiddenKnj*. Ovo je jedan od tri atributa koji su spomenuti u poglavlju o analizi i dizajnu, kada smo se dotakli ER dijagrama.

#### 4.3.1.1. Atributi za uspostavljanje sigurnosti

Naime, svaka tabela kreirana je unošenjem svih elemenata (atributa) iz ER dijagrama, što uključuje i sljedeće kolone:

- *hidden*;
- *datumKreiranjaZapisa*;
- *datumModifikovanjaZapisa*.

Svrha ovih atributa svakoj od tabela jeste osiguravanje podataka, što je u ovom slučaju na iznimno visokom nivou. Kolona *hidden* osigurava da podaci nikada ne budu u potpunosti uklonjeni iz baze podataka, već samo da se promijeni atribut *hidden*, koji je tipa *boolean*, iz 0 u 1. Naravno, uspostavljeno je da je vrijednost ovog atributa po default-u 0, te da se mijenja samo ukoliko se izrazi želja nekog od tipa korisnika za brisanjem određenih podataka.

Također, uspostavljeno je da se pri svakom dohvaćanju podataka iz baze uzimaju zapisi čiji atribut *hidden* ima vrijednost 0.

Ovo je izuzetno dobra praksa, koja ima svoju primjenu i u realnoj sferi, gdje je gotovo nezaobilazna. Ključna stvar koja je osigurana konkretno u ovom slučaju jeste da, ukoliko bi se neko sa zlonamjernim motivima uspio ulogovati u sistem putem računa uposlenika, ili još gore admina, koji ima sve privilegije, i pokušao izvršiti brisanje podataka, ne bi uspio u svom pokušaju. Naravno, naizgled bi uspio, jer se podaci više ne prikazuju, ali oni ipak ostaju potpuno netaknuti u bazi, sa samo jednim promijenjenim atributom.

U sljedećem bloku je prikazana funkcija za brisanje knjige iz klase Konekcija, na kojoj možemo vidjeti kako to izgleda praktično:

```
public void brisanjeKnjige(int knjiga_id) throws SQLException {
    Statement upitBaza = (Statement) veza.createStatement();
    String upit = "update knjiga set hiddenKnj = 1 where id_knjiga = " +
knjiga_id + ";
    try {
        upitBaza.executeUpdate(upit);
    } catch (SQLException e) {
        System.err.println(e);
    }
}
```

Preostala dva elementa, tj. atributa, koji su integrisani u svaku tabelu sa stajališta poboljšanja sigurnosti jesu datum kreiranja i datum modifikovanja (posljednjeg modifikovanja) svakog od zapisa. Po default-u, i jedan i drugi atribut imaju vrijednost *CURRENT\_TIMESTAMP* (tip podatka je *timestamp*), što znači da se u momentu kreiranja u ove kolone kreiranog zapisa upisuje trenutno vrijeme, što uključuje datum, te prikaz sati, minuta i sekundi. Atribut *datumKreiranjaZapisa* ostaje nepromijenjen tijekom cijelog života zapisa, dok je atribut *datumModifikovanjaZapisa* sadržan u trigger-u za update, gdje se, nakon bilo kojeg update-a, bilo koje od kolona nekog zapisa, njegova vrijednost mijenja u trenutni timestamp.

Ova praksa je izuzetno korisna kod „neželjenih“, ili slučajnih, ili zlonamjernih update-a izvedenih nad zapisom. Ukoliko se desi da se vrijednost neke kolone promijeni greškom, ili putem nekog zlonamjernog činioca, pomoću ovih kolona možemo ustanoviti da li postoji *backup* podataka koji je spremljen u nekom momentu prije izvršavanja update-a. Kao i atribut *hidden*, ova dva atributa imaju jako široku primjenu u realnom sektoru, jer pružaju novi dodatni stepen jačine zaštite podataka.

Dakle, administrator ima najveće privilegije u sistemu, te zbog toga ima i najviše funkcionalnosti; pored CRUD operacija nad knjigama, uposlenicima i poslovnica, admin korisnik/korisnici ima/imaju pravo i pregleda svih rezervacija, koje se dijele na dva ogranka, aktivne i neaktivne, pri čemu se prikazuju svi detalji o rezervaciji (podaci o korisniku, knjizi, te poslovnici). Također, admin korisnik ima i opciju brisanja rezervacije, bilo da je aktivna, ili neaktivna. Ipak, detaljnije o rezervacijama biće govora u nastavku rada.

Posljednja funkcionalnost admin korisnika je logout, gdje dolazi do zatvaranja admin page-a, te ponovnom otvaranju login page-a.

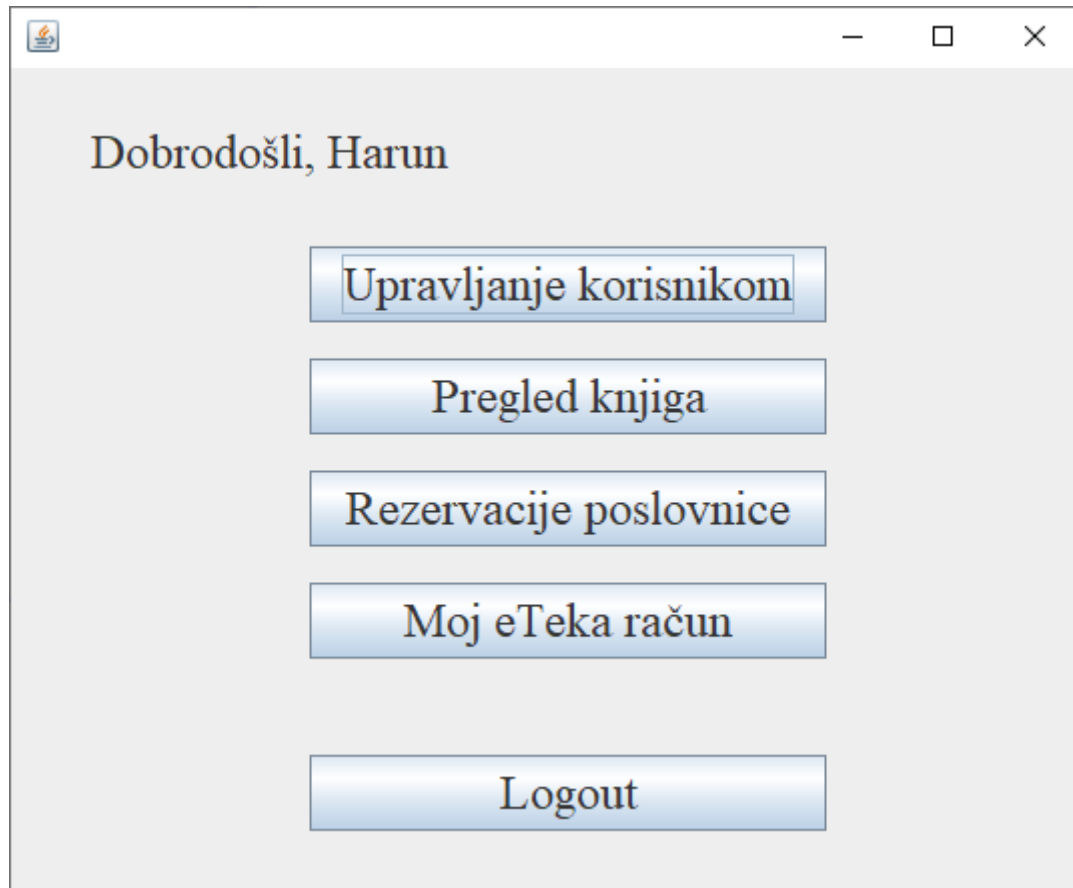
Kôd za ovu aplikaciju je izuzetno obiman, te zbog praktičnih razloga neće biti prikazana i pojašnjena svaka linija kôda, već će se analizirati u manjim blokovima, biranih na osnovu ponovne upotrebljivosti, tj. da se na osnovu pojedinog bloka može objasniti i više slučajeva (funkcionalnosti). Tako, naprimjer, ranije prikazani kôdovi funkcija za unos, ispis, brisanje i update zapisa tabele knjiga, jako su slični onima koji se pozivaju na *trigger* za CRUD operacije nad drugim objektima, kao što su korisnik, uposlenik, poslovnica, rezervacija i sl. Pozivanje preko konstruktora klase Konekcija je uvijek isto, kao i proces dobijanja ID-a preko drugog atributa jednog zapisa.

Ime kotiranja	Posrednik kotiranja	Datum rezervacije	Datum isteka posuđbe	Knjiga	Autor	Poslovnica	Mesto
---------------	---------------------	-------------------	----------------------	--------	-------	------------	-------

Slika 24. Prikaz rezervacija za admin korisnika

#### 4.3.2. Zaposlenik

Unošenjem login kredencijala (emaila i lozinke) u glavnom prozoru koji odgovaraju onima iz tabele „Zaposlenik“ u bazi podataka, otvara se prozor „ZaposlenikHome“, koji predstavlja početni prozor za uposlenika.



*Slika 25. Početni prozor za uposlenika*

Home prozor zaposlenika prikazan je na slici iznad. Odmah na početku vrijedi istaći da, s obzirom na to da sučelja za zaposlenika i korisnika aplikacije nisu organizovani u tab-ovima, kao što je slučaj sa prozorom za admin korisnika, prilikom tranzicije sa prozora na prozor treba prelaziti i informacija o korisniku, tj. njegov jedinstveni identifikator, kako bi se dalje mogle izvršavati operacije nad tim korisnikom. U ovom slučaju taj identifikator je email adresa.

Izvršavanje tranzicije sa prozora na prozor je jako jednostavno impementirati, potrebno je samo napraviti konstruktor koji će prihvatati *string* argument koji bi predstavljao email adresu korisnika, te izvršiti inicijalizaciju atributa u koji će podatak biti pohranjen. Naravno,

pri zatvaranju postojećeg i otvaranju novog prozora, prozor se mora inicijalizovati sa argumentovanim konstruktorom.

```
public class ZaposlenikHome extends javax.swing.JFrame {

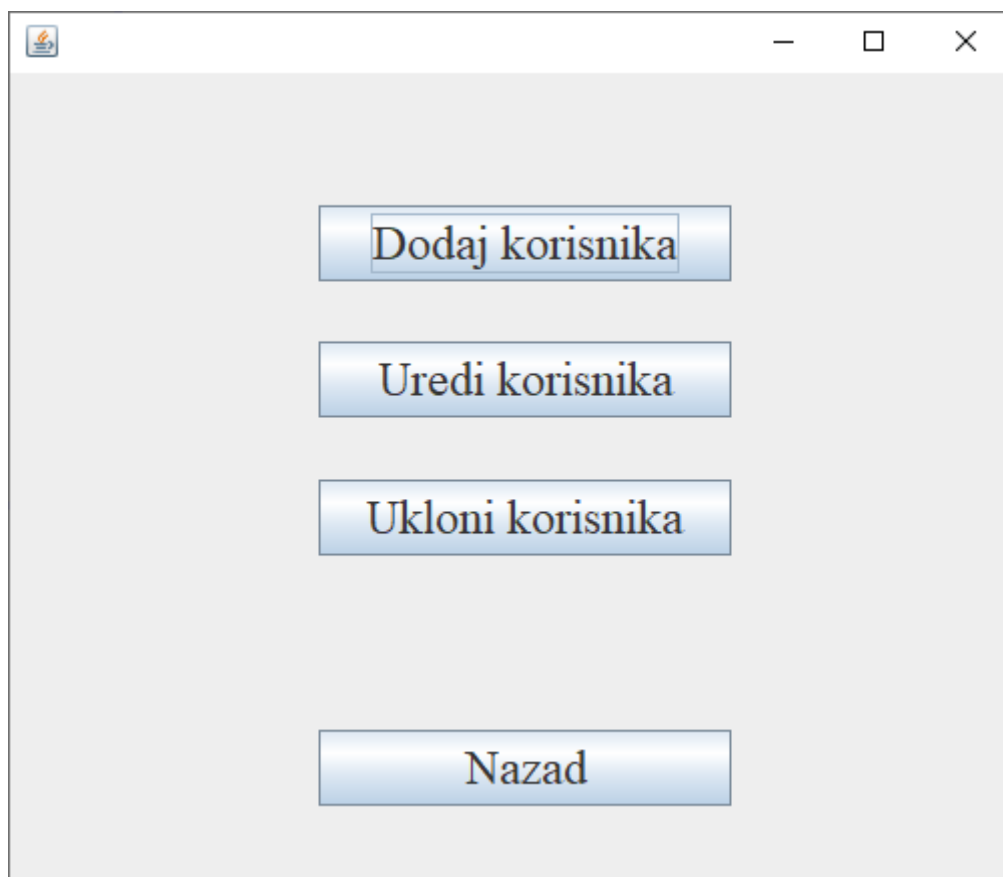
    public ZaposlenikHome() {
        initComponents();
    }

    String adresaUposlenik;

    public ZaposlenikHome(String email) {
        initComponents();
        adresaUposlenik = email;
        String ime = null;
        try {
            Konekcija k = new Konekcija();
            ResultSet rs = k.getImeZap(email);
            while (rs.next()) {
                ime = rs.getString(1);
            }
        } catch (Exception e) {
        }
        jLabel1.setText("Dobrodošli, " + ime);
    }
}
```

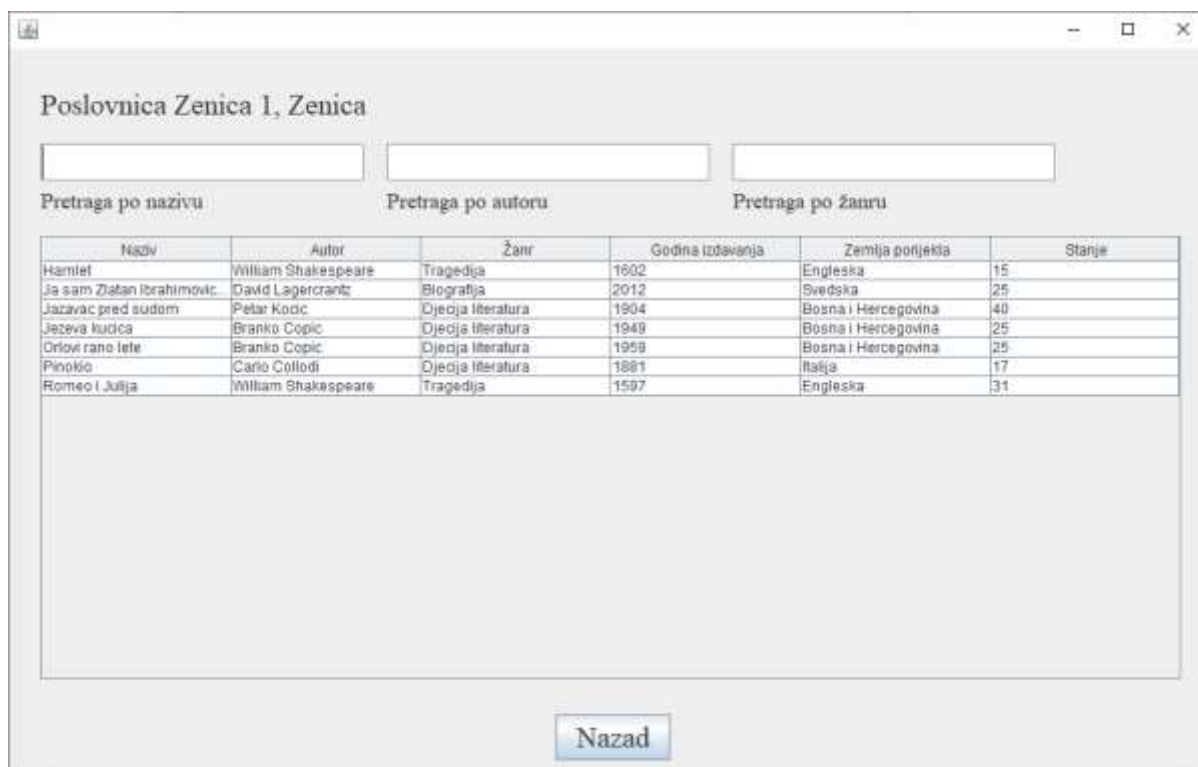
U bloku kôda prikazanom iznad možemo vidjeti kako izgleda konstruktor sa argumentom tipa *string*, koji prihvata email adresu korisnika unesenu u polju za unos teksta u glavnom prozoru. Također, prikazan je i način dohvaćanja imena korisnika na osnovu email adrese; to se vrši putem funkcije u klasi Konekcija, gdje se uz pomoć jednostavnog SQL upita dolazi do imena uposlenika na osnovu email adrese.

Zaposlenik ima funkcionalnosti upravljanja korisnicima, što podrazumijeva operacije dodavanja, uređivanja, te brisanja korisnika iz baze podataka. Funkcije za ove operacije su jako slične operacijama prikazanim ranije u radu (CRUD operacije admin korisnika nad klasom Knjiga). Na sljedećoj slici prikazan je podmeni „Upravljanje korisnikom“:



*Slika 26. Prikaz prozora za upravljanje korisnicima*

Sljedeća funkcionalnost korisnika je pregled knjiga u poslovnici u kojoj je uposlenik stacioniran, što naravno uključuje i višestruko filtriranje; po nazivu knjige, autoru, ili žanru. Podaci o poslovnici u kojoj se nalazi zaposlenik su vrlo jednostavno dostupni, slično kao i podatak o imenu korisnika na početnom zaslonu: funkcijom za učitavanje ID-a poslovnice zaposlenika na osnovu njegove email adrese.



Slika 27. Prikaz knjiga unutar poslovnice za uposlenika

Vrijedi istaći da se filtriranje knjiga, kao i svako drugo filtriranje u ovoj aplikaciji, odvija jako brzo, po puštenoj tipci sa tastature. To se postigne tako što se na polja za unos teksta (koja služe za pretragu po određenom parametru) doda tzv. *event listener*, koji osluškuje kada korisnik podigne prst sa tipke tastature (*Event KeyReleased*). Unutar ovog *event listenera* potrebno je pozvati funkciju za učitavanje zapisa iz baze podataka, čiji atribut naziv/autor/žanr sadrži tekst unesen u odgovarajuće tekstualno polje, te postaviti da se u tabelu ispod unose novi zapisi, koji odgovaraju filtriranom rezultatu.

Blokovi kôda pomoću kojih se ovaj cjelokupan proces ostvaruje prikazani su ispod:

```
private void jTextField1KeyReleased(java.awt.event.KeyEvent evt) {

    DefaultTableModel model = (DefaultTableModel) jTable1.getModel();
    model.setRowCount(0);
    int poslovnica_id = poslovnica;
    String tekst = jTextField1.getText();
    String nazivKnjigeU = "nazivKnjige";
    try {
        Konekcija k = new Konekcija();
        ResultSet rs = k.TraziKnjigu(nazivKnjigeU, tekst, poslovnica_id);
        while (rs.next()) {
```



```

        String naziv = rs.getString("nazivKnjige");
        String autor = rs.getString("autor");
        String zanr = rs.getString("zanr");
        String datumIzdavanja = rs.getString("datumIzdavanja");
        String zemljaPorijekla = rs.getString("zemljaPorijekla");
        String stanje = rs.getString("stanje");
        String podaci[] = {naziv, autor, zanr, datumIzdavanja,
zemljaPorijekla, stanje};

        DefaultTableModel tabelaIspKnj = (DefaultTableModel)
jTable1.getModel();
        tabelaIspKnj.addRow(podaci);
    }
} catch (SQLException | ClassNotFoundException ex) {
    System.err.println(ex);
}
}

```

```

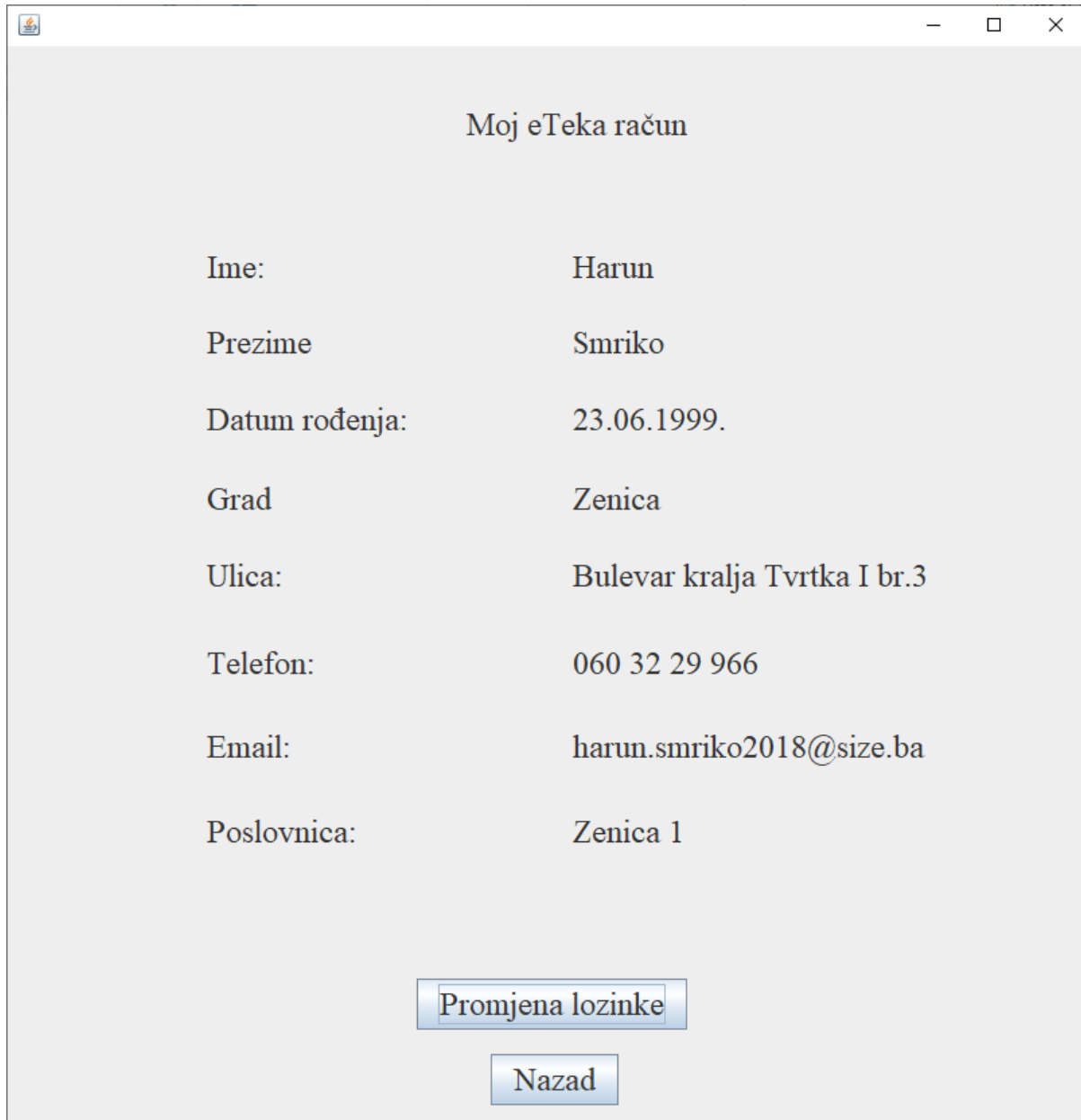
public ResultSet TraziKnjigu(String atribut, String tekst, int poslovnica_id)
throws SQLException {
    Statement upitBaza = (Statement) veza.createStatement();
    ResultSet rezultat = null;
    try {
        rezultat = upitBaza.executeQuery("Select k.nazivKnjige, k.autor,
k.zanr, k.datumIzdavanja, k.zemljaPorijekla, p.nazivPoslovnice, p.grad,
kp.stanje from knjiga k join knjigauposlovnici kp on k.id_knjiga =
kp.knjiga_id join poslovnica p on p.id_poslovnica = kp.poslovnica_id where
kp.poslovnica_id = " + poslovnica_id + " and hiddenKnj = 0 and " + atribut + "
like '%" + tekst + "%' order by 1");
        return rezultat;
    } catch (SQLException e) {
        System.err.println(e);
    }
    return rezultat;
}

```

U prvom bloku se nalazi event listener za tekstualno polje unosa naziva knjige, dok je u drugom bloku prikazana funkcija koja sadrži SQL upit, gdje možemo vidjeti da je implementiran način filtriranja gdje se unesena vrijednost ne mora nalaziti na početku naziva knjige; može i na početku, u sredini, ili na samom kraju naziva knjige.

Preostale dvije funkcionalnosti za uposlenika biblioteke jesu pregled rezervacija samo za biblioteku (poslovnicu) u kojoj je uposlenik stacioniran, te pregled detalja računa. Pregled rezervacija je jako sličan pregledu kojeg ima admin, s tim što pregled uposlenika u obzir

uzima samo poslovnicu uposlenika. Prozor „Moj eTeka račun“ je prozor u kojem su prikazani detalji korisničkog računa uposlenika.



The screenshot shows a window titled "Moj eTeka račun" with a light gray background. It contains a list of user details in a two-column format. At the bottom, there are two buttons: "Promjena lozinke" and "Nazad".

Ime:	Harun
Prezime	Smriko
Datum rođenja:	23.06.1999.
Grad	Zenica
Ulica:	Bulevar kralja Tvrtka I br.3
Telefon:	060 32 29 966
Email:	harun.smriko2018@size.ba
Poslovnica:	Zenica 1

Promjena lozinke

Nazad

*Slika 28. Prikaz detalja o računu (profilu) uposlenika*

U ovom prozoru se ne odvija ništa kompleksno; samo dolazi do učitavanja podataka na način prikazan ranije (putem emaila zaposlenika) te popunjavanja odgovarajućih labela tim podacima. Međutim, uposlenik ima dodatnu funkcionalnost – promjena lozinke. Pritiskom na dugme „Promjena lozinke“ otvara se prozor za promjenu lozinke.

The image shows a graphical user interface window for changing a password. At the top, it says "Unesite novu lozinku:" (Enter new password:). Below this is a text input field containing five dots. Further down, it says "Potvrda unosa:" (Confirmation of input:). Below this is another text input field containing ten dots. A red error message is displayed in the center: "Lozinke se ne poklapaju, molimo pokušajte ponovo" (Passwords do not match, please try again). At the bottom, there are two buttons: "Nazad" (Back) and "Spremi" (Save).

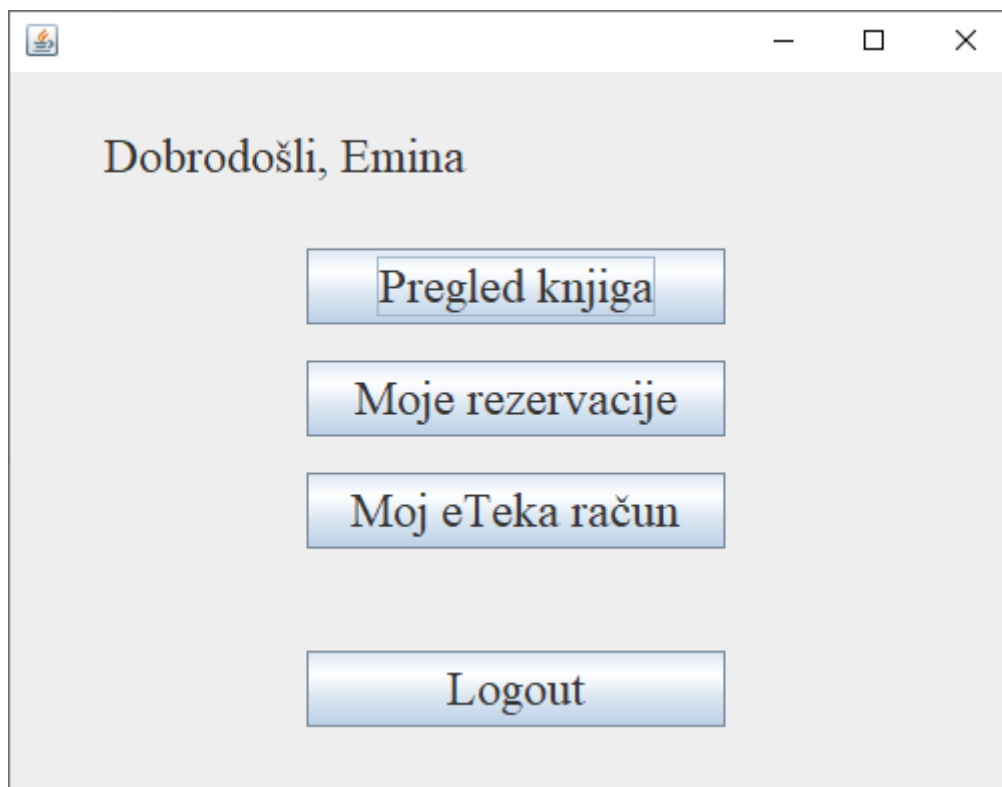
*Slika 29. Prikaz neuspješne promjene lozinke uposlenika*

Na slici iznad prikazan je prozor za promjenu lozinke uposlenika. Ono što je bitno ovdje naglasiti jeste da se pritiskom na dugme „Spremi“ prvenstveno vrše dvije provjere: da li su uneseni podaci u oba polja za unošenje teksta, te da li se podaci koji su uneseni poklapaju. Ukoliko jedan od ova dva uvjeta nije ispunjen, nemoguće je izvršiti promjenu lozinke, te će biti prikazan *error message*, kao u slučaju na slici iznad. Ukoliko su pak uvjeti ispunjeni, iz klase „Konekcija“ se poziva funkcija koja služi za update tabele Zaposlenik, tačnije kolone Lozinka, gdje je nova vrijednost ona unesena u poljima za unos nove lozinke.

Posljednja funkcionalnost zaposlenika je logout, nakon čega se ponovno otvara početni prozor (login forma).

#### 4.3.3. Korisnik

Nakon unosa emaila i lozinke koji se podudaraju sa parom tih podataka u nekom zapisu unutar tabele Korisnik, otvara se novi prozor koji predstavlja početni prozor interfejsa za korisnika aplikacije. Njegov izgled prikazan je na sljedećoj slici:



*Slika 30. Početni prozor korisnika*

S obzirom na to da je interfejs korisnika također realizovan kroz kretanje kroz više prozora, potrebno je, kao i u slučaju uposlenika, spasiti unesenu email adresu iz login forme, prosljeđivati je iz prozora u prozor preko argumentovanih konstruktora, te ju koristiti za dohvaćanje ostalih podataka o korisniku.

Prva funkcionalnost korisnika je pregled korisničkog računa te promjena lozinke korisnika, koja je identična onoj iz interfejsa uposlenika u biblioteci.

Sljedeća funkcionalnost jeste pregled svih knjiga uključujući filtriranje po poslovnica, nazivu, autoru i žanru. Unutar prozora za pregled knjiga, međutim, nalazi se još nekoliko funkcionalnosti, koje na neki način predstavljaju „središte“ aplikacije. To su rezervacija knjige, te ukidanje rezervacije.

Korisnik na prozoru za pregled knjiga može samo vršiti pregled knjiga, ili samo provjeru stanja količine dostupnih primjeraka neke knjige u poslovnici, ili samo pretraživati knjige po npr. autoru, ili žanru. Međutim, klikom na knjigu, te pritiskom na tipku „Rezerviši“, korisnik automatski dobija pravo na posudbu jednog primjerka knjige koju je odabrao, iz poslovnice koju je odabrao.

The screenshot shows a Java Swing window titled "eTeka". It contains a search form with the following fields:

- Poslovnica:** A dropdown menu with "Zenica 1" selected.
- Autor:** An empty text input field.
- Naziv:** A text input field containing the text "je".
- Žanr:** An empty text input field.

Below the search fields is a table with the following data:

Naziv	Autor	Žanr	Godina izdavanja	Zemlja porijekla	Poslovnica	Mjesto	Stanje
Jezova kucica	Biranka Copic	Dječja literatura	1949	Bosna i Hercegovina	Zenica 1	Zenica	25

Below the table, a green message states: "Rezervacija uspješna! Podignite knjigu u odabranoj poslovnici". At the bottom, there are three buttons: "Nazad", "Rezerviši", and "Ukloni rezervaciju".

*Slika 31. Prikaz filtriranih rezultata liste knjiga, te rezervacija odabrane knjige*

Na slici iznad prikazan je postupak rezervisanja knjige. Nakon odabrane poslovnice, knjige, te pritisnute tipke „Rezerviši“, u pozadini aplikacije se odvija mnogo procesa. Krenimo od početka:

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    int knjigaRed = jTable1.getSelectedRow();
    ResultSet knjigaRS;
    String knjiga = (String) jTable1.getModel().getValueAt(knjigaRed, 0);
    int knjiga_id = 0;
    ResultSet korisnikRS;
    int korisnik_id = 0;
    String poslovnica = jComboBox1.getSelectedItem().toString();
```

```

ResultSet poslovnicaRS;
int poslovnica_id = 0;
ResultSet rezervisanRS;
int rezervisan = 0;

try {
    Konekcija k = new Konekcija();
    if (this.jTable1.getSelectionModel().isSelectionEmpty()) {
        jLabel6.setText("Niste odabrali knjigu za rezervisanje!");
        jLabel6.setForeground(Color.red);
    } else {
        knjigaRS = k.getKnjID(knjiga);
        while (knjigaRS.next()) {
            knjiga_id = knjigaRS.getInt(1);
        }
        korisnikRS = k.getKorID(adresaKorisnik);
        while (korisnikRS.next()) {
            korisnik_id = korisnikRS.getInt(1);
        }
        rezervisanRS = k.getRezervisan(adresaKorisnik);
        while (rezervisanRS.next()) {
            rezervisan = rezervisanRS.getInt(1);
        }
        poslovnicaRS = k.getCmbID(poslovnica);
        while (poslovnicaRS.next()) {
            poslovnica_id = poslovnicaRS.getInt(1);
        }

        if (rezervisan == 1) {
            jLabel6.setText("Vratite trenutnu knjigu u biblioteku kako
biste mogli rezervisati novu!");
            jLabel6.setForeground(Color.red);
        } else {
            ResultSet stanjeRS;
            int stanje = 0;
            stanjeRS = k.getStanje(knjiga_id, poslovnica_id);
            while (stanjeRS.next()) {
                stanje = stanjeRS.getInt(1);
            }
            if (stanje < 1) {
                jLabel6.setText("Nema odabrane knjige na stanju u
selektiranoj poslovnici!");
                jLabel6.setForeground(Color.red);
            } else {
                ResultSet timeRS;
                Timestamp time = null;
                timeRS = k.getTwentyDays();
                while (timeRS.next()) {

```

```

        time = timeRS.getTimestamp(1);
    }
    k.unosRezervacije(korisnik_id, knjiga_id, time,
poslovnica_id);
    k.updateRezervisan(adresaKorisnik);
    k.updateStanjaRez(poslovnica_id, knjiga_id);
    jLabel6.setText("Rezervacija uspješna! Podignite
knjigu u odabranoj poslovnici");
    jLabel6.setForeground(Color.green);
    }
    }
}
} catch (SQLException | ClassNotFoundException ex) {
    System.err.println(ex);
}
}
}

```

Prije bilo kakve akcije i pozivanja funkcija, dolazi do validacije da li je korisnik selektirao knjigu. Ukoliko jeste, dolazi do niza ostalih provjera sa stajališta statusa korisnika (da li već ima aktivnu rezervaciju ili ne), te provjere da li postoji slobodan primjerak odabrane knjige na stanju u selektiranoj poslovnici. Naravno, prije te dvije provjere, iz klase Konekcija se dohvaćaju funkcije sa upitima selektovanja podataka (uglavnom iz ID kolona) tabela koje su potrebne za uspostavljanje rezervacije. Ukoliko su svi uvjeti ispunjeni, dolazi do kreiranja rezervacije; poziva se funkcija koja će u tabelu „rezervacija“ izvršiti unos podataka po ID-ovima korisnika, knjige te poslovnice, te tačnom datumu i vremenu isteka rezervacije (u ovom slučaju postavljeno je da rezervacija traje 20 dana). Datum i vrijeme se generišu u bazi automatski korištenjem default vrijednosti *CURRENT\_TIMESTAMP*, dok se računanje 20 dana nakon trenutnog vremena izvršava putem funkcije koja se nalazi u klasi „Konekcija“, *getTwentyDays()*, i koja izgleda ovako:

```

public ResultSet getTwentyDays() throws SQLException {
    Statement upitBaza = (Statement) veza.createStatement();
    ResultSet rezultat = null;
    try {
        rezultat = upitBaza.executeQuery("SELECT
DATE_ADD(CURRENT_TIMESTAMP, interval 20 day)");
        return rezultat;
    } catch (SQLException e) {
        System.err.println(e);
    }
    return rezultat;
}

```

Funkcija koja vrši unos rezervacije u bazu podataka izgleda ovako:

```
public void unosRezervacije(int korisnik_id, int knjiga_id, Timestamp
datumIstekaRezervacije, int poslovnica_id) throws SQLException {
    Statement upitBaza = (Statement) veza.createStatement();
    String upit = "INSERT INTO rezervacija (korisnik_id, knjiga_id,
datumIstekaRezervacije, poslovnica_id) VALUES (" + korisnik_id + "," +
knjiga_id + "," + datumIstekaRezervacije + "," + poslovnica_id + ")";
    try {
        upitBaza.executeUpdate(upit);
    } catch (SQLException e) {
        System.err.println(e);
    }
}
```

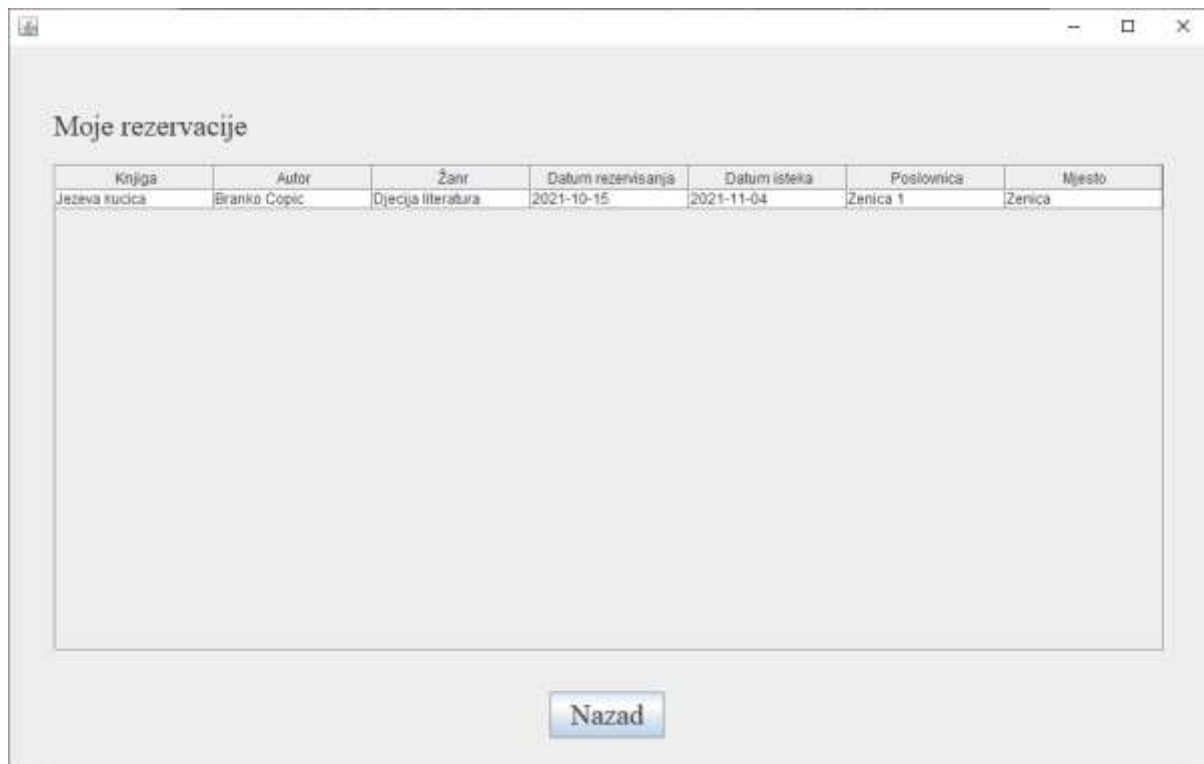
Međutim, pored ove funkcije, izvršavaju se još dvije dodatne funkcije: jedna koja mijenja status korisnika u „zadužen“ (boolean tip podatka), te druga koja smanjuje za jedan ukupnu količinu stanja rezervisane knjige u odgovarajućoj poslovnici. Blokovi kôda dviju navedenih funkcija prikazani su u nastavku:

```
public void updateRezervisan(String email) throws SQLException {
    Statement upitBaza = (Statement) veza.createStatement();
    String upit = "update korisnik set zaduzen = 1 where email = '" +
email + "'";
    try {
        upitBaza.executeUpdate(upit);
    } catch (SQLException e) {
        System.err.println(e);
    }
}
```

```
public void updateStanjaRez(int poslovnica_id, int knjiga_id) throws
SQLException {
    Statement upitBaza = (Statement) veza.createStatement();
    String upit = "update knjigauposlovnici set stanje = stanje-1 where
poslovnica_id=" + poslovnica_id + " and knjiga_id=" + knjiga_id + ";
    try {
        upitBaza.executeUpdate(upit);
    } catch (SQLException e) {
        System.err.println(e);
    }
}
```



Da li je rezervacija uspješno kreirana možemo provjeriti kroz funkcionalnost korisnika – uvid u sve korisnikove rezervacije. To možemo provjeriti povratkom u glavni meni, te odabirom opcije „Moje rezervacije“.



*Slika 32. Prikaz rezervacija korisnika*

Kao što možemo vidjeti, korisnik može vidjeti novokreiranu rezervaciju u prozoru, kao i sve detalje o knjizi, datumu kreiranja i isteka rezervacije, te poslovnici u kojoj je ta rezervacija kreirana. Još jedan način provjere kreiranja rezervacije je povratak na prozor za pregled knjiga, gdje možemo ustanoviti da li je u poslovnici koju smo odabrali količina primjeraka odabrane knjige smanjena za jedan.

**eTeka**

Poslovnica: Zenica 1      Autor: anko

Naziv:       Žanr:

Naziv	Autor	Žanr	Godina izdavanja	Zemlja porijekla	Poslovnica	Mjesto	Stanje
Ježeva kućica	Branko Ćopić	Dječja literatura	1949	Bosna i Hercegovina	Zenica 1	Zenica	24
Orlovi rano leće	Branko Ćopić	Dječja literatura	1959	Bosna i Hercegovina	Zenica 1	Zenica	25

Imate aktivnu rezervaciju, novu možete napraviti nakon isteka, ili prekidom postojeće

Nazad
Rezerviši
Ukloni rezervaciju

*Slika 33. Prikaz knjiga za korisnika u slučaju postojanja aktivne rezervacije*

Ukoliko se vratimo na sliku broj 31, možemo primijetiti da je stanje knjige „Ježeva kućica“ u poslovnici „Zenica 1“ bilo 25 po kreiranju rezervacije, a sada je 24, što nam predstavlja novi dokaz na činjenicu da je rezervacija kreirana. Također, ovdje vrijedi spomenuti i da je konstruktor prozora za pregled knjiga ustanovio da trenutni korisnik ima aktivnu rezervaciju, te da je kreiranje nove nemoguće dok ne dođe do prekida aktivne. Naravno, ukoliko bismo pokušali napraviti novu preko već postojeće rezervacije, ne bismo uspjeli.

Konstruktor ovog prozora jako je obiman. Pored kôdova za punjenje dropdown menija poslovnicama, punjenje tabele knjigama sa stanjem, te promjene podataka u tabeli u ovisnosti od selektirane poslovnice, nalazi se i kratak blok kojim se može ustanoviti porijeklo ispisane poruke na slici 33:

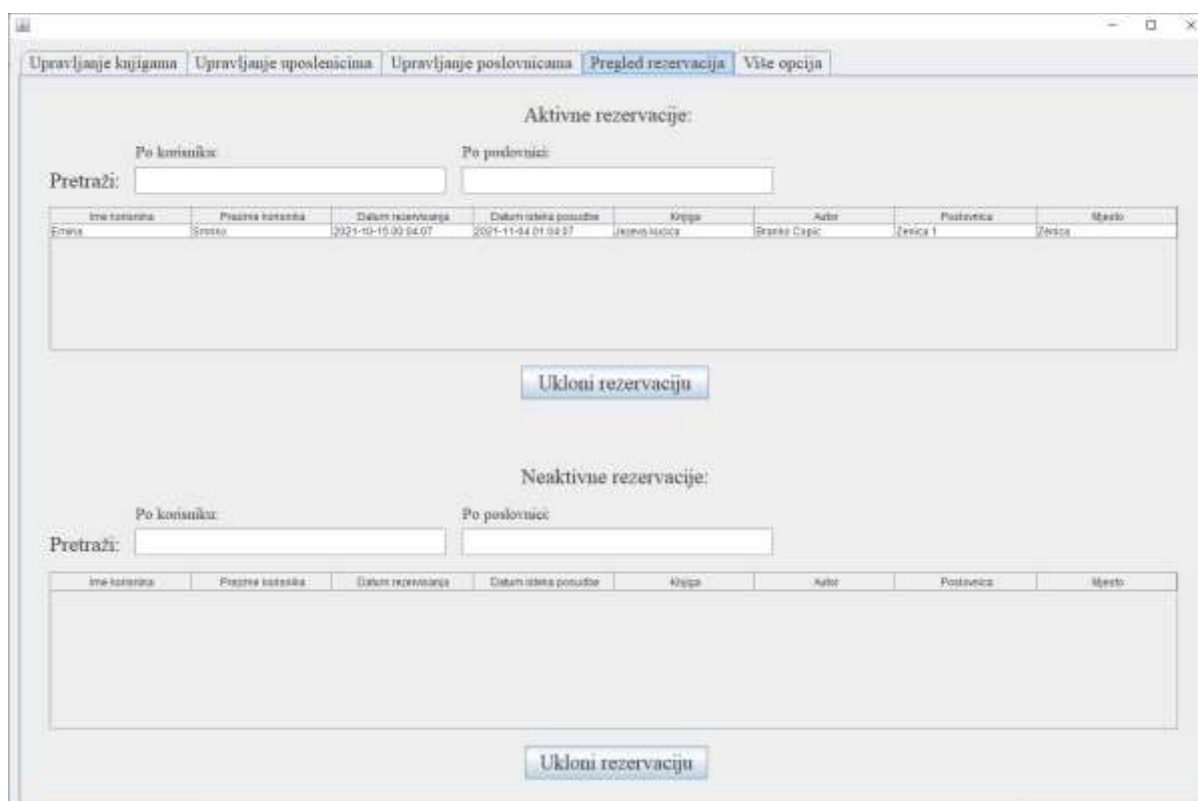
```
rezervisanRS = k.getRezervisan(adresaKorisnik);
while (rezervisanRS.next()) {
    rezervisan = rezervisanRS.getInt(1);
}
```

```

if (rezervisan == 1) {
    jLabel6.setText("Imate aktivnu rezervaciju, novu možete napraviti nakon isteka, ili prekidom postojeće");
    jLabel6.setForeground(Color.blue);
}

```

Prije nego započnemo sa pričom o ukidanju, ili preciznije, prestanku postojanja aktivne rezervacije, još ćemo jednom provjeriti admin page, tačnije tab sa prikazom rezervacija, gdje možemo vidjeti da je kreirana rezervacija unesena i u tabelu prikaza aktivnih rezervacija, sa nešto više detalja sa stajališta preciznog vremena kreiranja i isteka rezervacije.



Slika 34. Kreirana rezervacija prikazana u admin prozoru

Prilikom prekidanja rezervacije od strane korisnika, dešavaju se procesi suprotni onima koji se pozivaju nakon kreiranja rezervacije. Status korisnika se vraća na vrijednost 0, stanje knjige koja je bila zadužena u odabranoj poslovnici se povećava za 1, dok se nad vrijednošću datuma isteka rezervacije vrši update, gdje taj atribut dobija vrijednost *CURRENT\_TIMESTAMP*, tj. egzaktno datum i vrijeme pritiska na tipku „Ukloni rezervaciju“.

**eTeka**

Poslovnica: Zenica 2 Autor:

Naziv:  Žanr:

Naziv	Autor	Žanr	Godina izdavanja	Zemlja porijekla	Poslovnica	Mjesto	Stanje
Hamlet	William Shakespeare	Tragedija	1602	Engleska	Zenica 2	Zenica	14
Ja sam Zlatan Ibrahimović	David Lagercrantz	Biografija	2012	Švedska	Zenica 2	Zenica	9
Jazavac pred sudom	Petar Kocić	Dječja literatura	1904	Bosna i Hercegovina	Zenica 2	Zenica	20
Jezeva kućica	Branko Ćopić	Dječja literatura	1949	Bosna i Hercegovina	Zenica 2	Zenica	20
Orlovi rano lete	Branko Ćopić	Dječja literatura	1959	Bosna i Hercegovina	Zenica 2	Zenica	30
Pinokio	Carlo Collodi	Dječja literatura	1881	Italija	Zenica 2	Zenica	25
Romeo i Julija	William Shakespeare	Tragedija	1597	Engleska	Zenica 2	Zenica	18

Trenutna rezervacija uspješno prekinuta!

Nazad
Rezerviši
Ukloni rezervaciju

*Slika 35. Prekid rezervacije*

**Moje rezervacije**

Knjiga	Autor	Žanr	Datum rezerviranja	Datum isteka	Poslovnica	Mjesto
Jezeva kućica	Branko Ćopić	Dječja literatura	2021-10-15	2021-10-15	Zenica 1	Zenica

Nazad

*Slika 36. Prikaz rezervacija korisnika, izmijenjen datum isteka rezervacije*

Upravljanje knjigama
Upravljanje zaposlenicima
Upravljanje poslovnica
**Pregled rezervacija**
Više opcija

Aktivne rezervacije:

Po korisniku:

Po poslovnicu:

Pretraži:

Ime korisnika	Poslovna jedinica	Datum rezervacije	Datum isteka posuđbe	Knjiga	Autot	Poslovnica	Mjesto
---------------	-------------------	-------------------	----------------------	--------	-------	------------	--------

Ukloni rezervaciju

Neaktivne rezervacije:

Po korisniku:

Po poslovnicu:

Pretraži:

Ime korisnika	Poslovna jedinica	Datum rezervacije	Datum isteka posuđbe	Knjiga	Autot	Poslovnica	Mjesto
Evelina	Strakonja	2021-10-15 01:54:07	2021-10-15 16:05:38	JAKOVA KUČKA	BRANKI ČEPIĆ	ZBROJ 1	Zvonja

Ukloni rezervaciju

Slika 37. Prikaz prekinute rezervacije, koja se sada nalazi u tabeli neaktivnih rezervacija

Sa prethodne tri slike možemo uočiti kako je rezervacija uspješno prikinuta, te kako je dobila status „neaktivne“ rezervacije. Sada korisnik može napraviti novu rezervaciju.

eTeka

Poslovnica: Zenica 1

Autor:

Naziv:

Žanr: djecija literatura

Naziv	Autor	Žanr	Godina izdavanja	Zemlja porijekla	Poslovnica	Mjesto	Stanje
Jazavac pred sudom	Petar Kocić	Djecija literatura	1904	Bosna i Hercegovina	Zenica 1	Zenica	40
Jezeva kucica	Branko Ćopić	Djecija literatura	1948	Bosna i Hercegovina	Zenica 1	Zenica	25
Orlovi rano lete	Branko Ćopić	Djecija literatura	1959	Bosna i Hercegovina	Zenica 1	Zenica	25
Pinokio	Carlo Collodi	Djecija literatura	1981	Italija	Zenica 1	Zenica	17

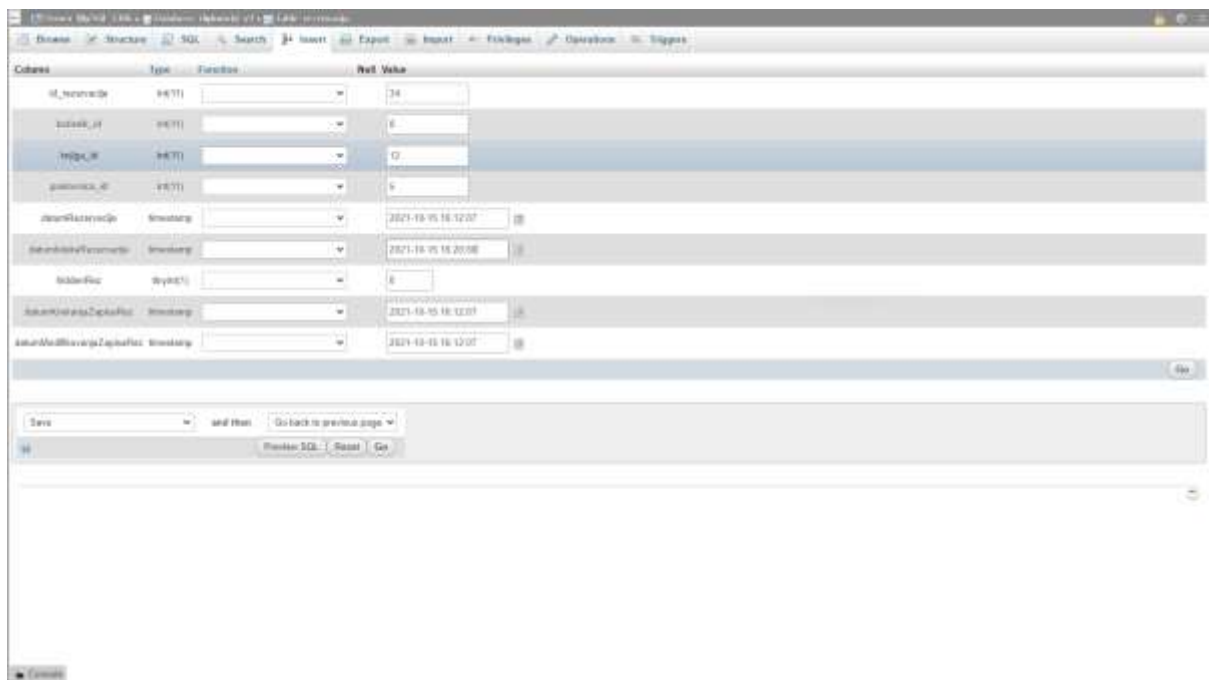
Rezervacija uspješna! Podignite knjigu u odabranoj poslovnici

Nazad
Rezerviši
Ukloni rezervaciju

*Slika 38. Kreiranje nove rezervacije*

Na slici iznad kreirana je nova rezervacija, pomoću koje će biti prikazan drugi način prestanka postojanja aktivne rezervacije – njen istek. Međutim, vrijedi još istaći da je stanje prethodno rezervisane knjige u poslovnici „Zenica 1“ povećano za jedan, kao što možemo vidjeti na slici iznad.

Istek rezervacije ćemo simulirati, tj. izvršiti ćemo edit atributa datuma isteka rezervacije u alatu *phpMyAdmin*, te postaviti timestamp u bliskoj budućnosti.



*Slika 39. Edit atributa datuma isteka rezervacije u phpMyAdmin*

Nakon isteka novounesenog vremena isteka rezervacije, izvršit ćemo provjeru da li je simulacija isteka uspješno izvršena.

**eTeka**

Poslovnica: **Zenica 1** Autor:

Naziv:  Žanr:

Naziv	Autor	Žanr	Godina izdavanja	Zemlja porijekla	Poslovnica	Mjesto	Stanje
Hamlet	William Shakespeare	Tragedija	1602	Engleska	Zenica 1	Zenica	15
Ja sam Zlatan Ibrahimović	David Lagercrantz	Biografija	2012	Švedska	Zenica 1	Zenica	25
Jazavac pred sudom	Petar Kocić	Dječja literatura	1904	Bosna i Hercegovina	Zenica 1	Zenica	40
Jezava kucica	Branko Ćopić	Dječja literatura	1949	Bosna i Hercegovina	Zenica 1	Zenica	25
Orlovi rano lete	Branko Ćopić	Dječja literatura	1959	Bosna i Hercegovina	Zenica 1	Zenica	25
Pinokio	Carlo Collodi	Dječja literatura	1881	Italija	Zenica 1	Zenica	17
Romeo i Julija	William Shakespeare	Tragedija	1597	Engleska	Zenica 1	Zenica	31

Slika 40. Prikaz pregleda knjiga za korisnika

Upravljanje knjigama Upravljanje zaposlenicima Upravljanje poslovnicama **Pregled rezervacija** Više opcija

**Aktivne rezervacije:**

Po korisniku:  Po poslovnici:

Pretraži:

Ime korisnika	Prezime korisnika	Datum rezervacije	Datum isteka posudbe	Knjiga	Autor	Poslovnica	Mjesto
---------------	-------------------	-------------------	----------------------	--------	-------	------------	--------

**Neaktivne rezervacije:**

Po korisniku:  Po poslovnici:

Pretraži:

Ime korisnika	Prezime korisnika	Datum rezervacije	Datum isteka posudbe	Knjiga	Autor	Poslovnica	Mjesto
Ermina	Ermina	2021-10-10 16:12:07	2021-10-15 16:20:00	Jazavac pred sudom	Petar Kocić	Zenica 1	Zenica
Ermina	Ermina	2021-10-10 00:34:37	2021-10-15 16:06:36	Jezava kucica	Branko Ćopić	Zenica 1	Zenica

Slika 41. Prikaz rezervacija za admin korisnika



Na slikama iznad možemo vidjeti da je rezervacija istekla u skladu sa očekivanjima, što se ogleda u stanju knjige „Jazavac pred sudom“ u poslovnici „Zenica 1“, zatim u odsustvu poruke koja se korisniku ispisuje pri pregledu knjiga, ali i prikazu rezervacije u tabeli neaktivnih rezervacija na admin prozoru. Ali šta to u programu osigurava da do ovoga dođe? Odgovor se nalazi u sljedećem bloku kôda:

```
public GlavniProzor() {
    initComponents();
    try {
        Konekcija k = new Konekcija();
        ResultSet rs1 = k.ispisRezervacijaChk();
        while (rs1.next()) {
            int zaduzen = 0;
            int id_knjiga = rs1.getInt("knjiga_id");
            int id_korsinik = rs1.getInt("korisnik_id");
            int id_poslovnica = rs1.getInt("poslovnica_id");
            Timestamp istek = rs1.getTimestamp("datumIstekaRezervacije");
            ResultSet rs2 = k.getRezervisanPoID(id_korsinik);
            while (rs2.next()) {
                zaduzen = rs2.getInt(1);
            }
            if (zaduzen == 1) {
                k.updateOdrijesen(id_korsinik);
                k.updateStanjaOdr(id_poslovnica, id_knjiga);
            }
        }
    } catch (Exception ex) {
        System.err.println(ex);
    }
}
```

U bloku kôda prikazanom iznad vidimo konstruktor glavnog prozora. To znači da pri samom pokretanju aplikacije dolazi do određenih akcija. Prvo se poziva funkcija iz klase Konekcija koja sadrži upit kojim se učitavaju po jedan od zapisa za svakog korisnika iz tabele „rezervacija“, čiji je datum isteka rezervacije najnoviji, ali manji od trenutnog datuma i vremena. Na taj način dobit ćemo podatak o tome koja je najnovija rezervacija koja je istekla. Ukoliko korisnik ima aktivnu rezervaciju, ovaj upit za tog korisnika neće vratiti niti jedan zapis. Tu funkciju možemo vidjeti u sljedećem bloku:

```
public ResultSet ispisRezervacijaChk() throws SQLException {
    Statement upitBaza = (Statement) veza.createStatement();
    ResultSet rezultat = null;
    try {
```

```

        rezultat = upitBaza.executeQuery("Select korisnik_id, knjiga_id,
poslovnica_id, datumIstekaRezervacije From rezervacija r where hiddenRez = 0
and datumIstekaRezervacije = (select max(datumIstekaRezervacije) from
rezervacija where r.korisnik_id = rezervacija.korisnik_id) and
datumIstekaRezervacije < CURRENT_TIMESTAMP");
        return rezultat;
    } catch (SQLException e) {
        System.err.println(e);
    }
    return rezultat;
}

```

Svi zapisi dobijeni iz upita u prethodno prikazanom bloku se ispituju, tj. ispituje se preko ID-a svih korisnika da li je njihov status jednak 1. Ukoliko jeste, pozivaju se funkcije za primjenu statusa nazad u 0, te funkcija koja dodaje na ukupno stanje u biblioteci jednu knjigu, onu iz zapisa dobijenog u upitu iznad.

#### 4.4. Testiranje

Nakon uspješno obavljenog dodavanja svake od funkcionalnosti obavljeno je parcijalno testiranje te funkcionalnosti. Pri kodiranju su ubačena mnoga ograničenja, posebno pri unosu podataka, te je vođeno računa da su sva ograničenja testirana u fazi parcijalnog testiranja.

Nakon kreiranog interfejsa za svakog od aktera u sistemu, vršeno je integralno testiranje, gdje su još jednom testirane sve funkcionalnosti za tog aktera (korisnika) aplikacije. Naravno, svaka uočena greška za sobom je povlačila procedure koje podrazumijevaju momentalan ispravak u kôdu.

Na kraju, poslije kodiranja cjelokupnog sistema, tj. nakon završetka kodiranja i testiranja kôda za svaku partikulu sistema, vršilo se testiranje sistema, tj. završno, finalno testiranje, gdje je testiran sistem kao cjelina, te sve funkcionalnosti aplikacije, te ukoliko je pronađena greška, vršene su odgovarajuće korekcije u kôdu.

#### 4.5. Planovi za budućnost

Aplikacija kreirana u svrhu ovog projekta ima veliki potencijal za upotrebu u stvarnom svijetu. Naravno, uvijek ima prostora za određene popravke, ali i nadogradnje sa stajališta već postojećih ili novih funkcionalnosti, te dizajna aplikacije.

Jedan od glavnih ciljeva u budućnosti razvoja aplikacije „eTeka“ jeste, naravno, priprema aplikacije za funkcionisanje u *Cloud* okruženju. To uključuje procedure kao što su postavljanje baze podataka u tzv. *Docker container*, te postavljanje tog kreiranog *container*-a na Cloud platformu, kao što je npr. *Microsoft Azure*. Hosting baze podataka na cloud-u mogao bi se realizovati na više načina. Jedan od njih je spremanje baze podataka na Azure cloud, koji nudi SQL Server servis, a jedno od rješenja bi mogla biti i primjena virtuelne mašine na kojoj je konfigurisan SQL Sever, a cijela virtuelna mašina pohranjena u cloud-u (Azure).

Sa stajališta sigurnosti, nadogradnja koja bi bila od krucijalnog značaja jeste funkcionalnost validacije prilikom prijave na sistem. Svaki od aktera u sistemu, unutar baze podataka, ima podatke o svojoj email adresi, te broju telefona. Pri kreiranju računa, korisnik (preciznije, akter) bi mogao imati mogućnost izbora validacije putem SMS poruke na uneseni broj telefona, na način da bi mu stigla poruka o nedavnoj prijavi na sistem, dok bi se notifikacija putem email adrese podrazumijevala, što za sobom povlači kreiranje i konfiguraciju servisa za notifikacije, kako putem emaila, tako i putem SMS poruka. Naposljetku, jedna od mogućnosti jeste i implementacija „*Two-factor authentication*“ sistema (naravno, neobavezna i implementirana u zavisnosti od saglasnosti korisnika / aktera), gdje prijava na sistem ne bi bila moguća bez potvrde korisnika u vidu otvaranja linka koji bi bio poslan korisniku na mail prilikom logina.

Činjenica koju treba uzeti u obzir jeste da se u današnje vrijeme sve više i više koriste knjige u elektronskom formatu. Pa tako, funkcionalnost „eKnjiga“ ima jako visok prioritet na „*to-add*“ listi dodatnih funkcionalnosti. Ona bi bila implementirana na način da bi vlasnik (*owner*) aplikacije mogao vršiti upload elektronskih knjiga (npr. u PDF formatu), podrazumijevajući da na njih ima sva odgovarajuća prava (sa pravnog aspekta), te bi svaka knjiga imala svoj link, za koji bi korisnik dobio permisiju na određeni broj dana, u skladu sa zahtjevom za posudbom određene elektronske knjige. Planirano je da ovaj sistem bude u vlasništvu samog vlasnika aplikacije, u potpunosti odvojen od sistema koji je trenutno implementiran u aplikaciji „eTeka“, gdje se vrši *tracking* fizičkih kopija knjiga u bibliotekama na nekom određenom nivou. Važno je istaći i to da je planirano uvesti i sistem plaćanja po posudbi u slučaju elektronskih knjiga.

Na kraju, jedan od prioriteta bi svakako bilo uvođenje razina članstva za korisnike, tj. pružanje dodatnih usluga, odnosno funkcionalnosti članovima sa većim „rangom“ članarine.

Naprimjer, „VIP“ članovi aplikacije bi imali dodatne usluge kao što su niža cijena rezervacije elektronskih knjiga, ili pravo na veću dužinu trajanja posudbe i sl.

Naravno, pored svih navedenih, uvijek postoje razne mogućnosti za poboljšanje aplikacije i sistema generalno, kako u vidu poboljšanja preformansi, tako i u vidu popravki na dizajnu, tj. izgledu aplikacije, ali i u vidu raznih sigurnosnih „zakrpi“ koje mogu biti implementirane.

## 5. ZAKLJUČAK

Ovaj projekat ima izuzetan poslovni potencijal iz više razloga. Glavni razlog jeste taj što, barem na našim prostorima, ne postoji projekat, tj. aplikacija koja nudi ovakvu vrstu usluge. Sprovedena tržišna analiza pokazala je da ovaj projekat ne bi imao prijetnju koja je skoro pa i integrisana na novim projektima u današnje vrijeme – postojanje konkurentnog proizvoda. Međutim, uvijek postoji rizik od pojave konkurencije, koja će svoj proizvod bazirati na snagama već postojećeg proizvoda, te ga dekorisati na način koji bi bio prikladniji običnim korisnicima proizvoda.

Stoga, postupci koji treba da slijede u fazi razvoja ovog projekta jesu proširenja projekta sa već postojećim planovima, te vođenje računa o odzivu (*feedback-u*) korisnika proizvoda, kako bi se istim pružilo što je moguće bolje iskustvo pri korištenju aplikacije. Na kraju dana, najbitnije od svega jeste da je sam korisnik proizvoda zadovoljan, a samo bonusom se može smatrati činjenica da trenutno ne postoji jaka konkurencija. Važno je u nečemu biti prvi, međutim, nakon dobrog starta, cilj je i završiti utrku na prvom mjestu.

## 6. LITERATURA

- [1] [https://sco.wikipedia.org/wiki/Oracle\\_Corporation](https://sco.wikipedia.org/wiki/Oracle_Corporation) (dostupno: 17.10.2021.)
- [2] [https://en.wikipedia.org/wiki/Java\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Java_(programming_language)) (dostupno: 06.10.2021.)
- [3] <https://www.tiobe.com/tiobe-index/> (dostupno: 19.10.2021.)
- [4] <https://www.jetbrains.com/lp/devecosystem-2021/java/> (dostupno: 20.10.2021.)
- [5] <https://www.javatpoint.com/java-swing> (dostupno: 19.10.2021.)
- [6] <https://www.edureka.co/blog/java-jframe/> (dostupno: 18.10.2021.)
- [7] <https://en.wikipedia.org/wiki/NetBeans> (dostupno: 19.10.2021.)
- [8] <https://netbeans.apache.org/about/history.html> (dostupno: 19.10.2021.)
- [9] <https://netbeans.org/downloads/8.2/rc/start.html?platform=windows&lang=en&option=all>  
(dostupno: 13.10. 2020.)
- [10] <https://vertabelo.com/blog/cardinality-in-data-modeling/> (dostupno: 04.10.2021.)
- [11] <https://sourceforge.net/projects/wampserver/> (dostupno: 07.12.2020.)
- [12] <https://dev.mysql.com/downloads/workbench/> (dostupno: 07.12.2020.)