UDK 004.9
stručni rad

Harun Smriko
Nevzudin Buzađija

IDEJNO RJEŠENJE ZA RAZVOJ APLIKACIJA U JAVA OKRUŽENJU PRIMJENOM JFRAME

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*

Uvod

Pandemija virusa COVID-19 mnogo 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.

Informacioni sistem je kreiran u formi desktop 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 upoznavanje svih funkcionalnosti aplikacije i načina na koje se realizuju.

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).

Elementi java projekta

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

[1] https://www.javatpoint.com/java-swing

JFrame

JFrame je klasa unutar javax.swing paketa. Korištenje ove klase obezbjeđuje prikaz potpuno funkcionalnog prozora 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)[2].

NetBeans

NetBeans je razvojno okruženje (tzv. IDE) za programski jezik Java. On omogućava da se aplikacije razvijaju iz seta modularnih softverskih komponenti zvanih moduli. U okviru praktičnog projekta ovog rada korištena je verzija 8.2, preuzeta sa zvanične stranice[3]. 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.

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 „klikom miša“. Također, vrijedi istaći da tzv. event listenere možemo postavljati na bilo koji od elemenata, pa tako naprimjer, u implementacijskom dijelu rada, bit će korišten elent 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.

Implementacija

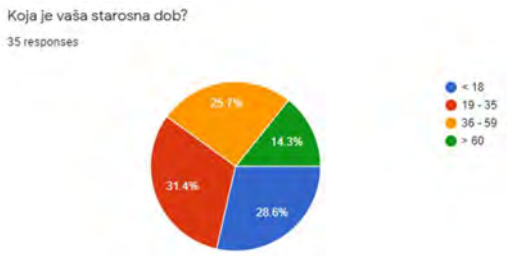
Kao zadatak ovog rada odabrano je kreiranje informacionog sistema za biblioteke na nekom određenom nivou (gradskom, kantonalnom, državnom i sl.). Naziv kreirane aplikacije je „eTeka“.

Analiza zahtjeva

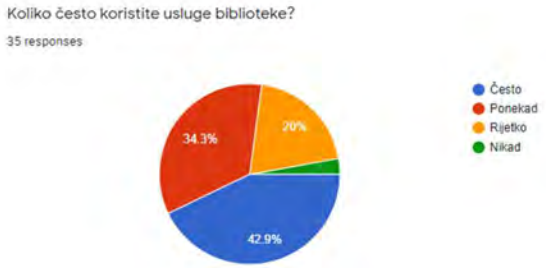
Prva faza razvoja sistema je analiza korisničkih zahtjeva. Ova faza podrazumijeva razne aktivnosti kao što su ispitivanje, te sprovođenje anketa nad potencijalnim korisnicima, o tome koje funkcionalnosti bi oni željeli vidjeti u aplikaciji, a koju bi potencijalno kupili. U sklopu ove faze razvoja projekta, izvršeno je anketiranje potencijalnih korisnika putem platforme Google Forms, te anketi su imali pristup 50 osoba, koju su dobili putem platforme Facebook.

[2] https://www.edureka.co/blog/java-jframe/
[3] https://netbeans.org/downloads/8.2/rc/start.html?platform=windows&lang=en&option=all

Dostavljeno je 35 rezultata ankete (odgovora), a analiza sprovedenog upitnika se nalazi na slikama ispod.



Grafikon 1. Analiza odgovora na prvo pitanje u sprovedenoj anketi



Grafikon 2. Analiza odgovora na drugo pitanje u sprovedenoj anketi

Grafikon 3. Analiza odgovora na treće 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 poslovnicama (što uključuje CRUD operacije);
- Upravljanje uposlenicima (što uključuje CRUD operacije);
- Pregled aktivnih i neaktivnih rezervacija, svih korisnika u svim poslovnicama;
- 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.

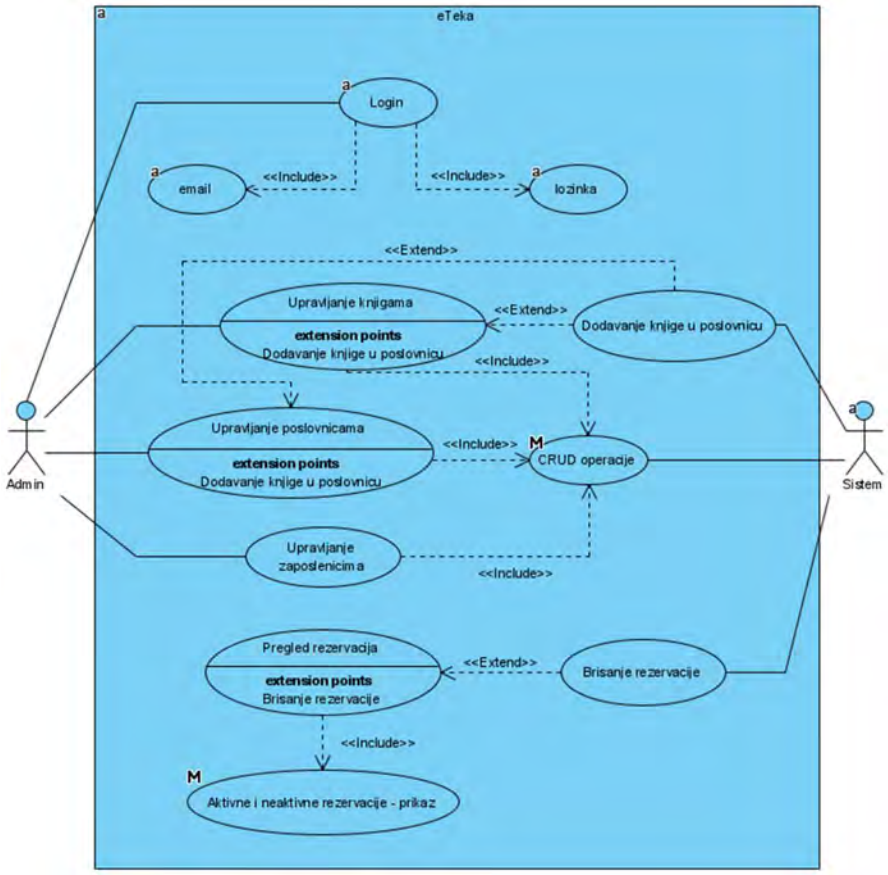
Za korisnika aplikacije:

- 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.

Analiza i dizajn sistema

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).

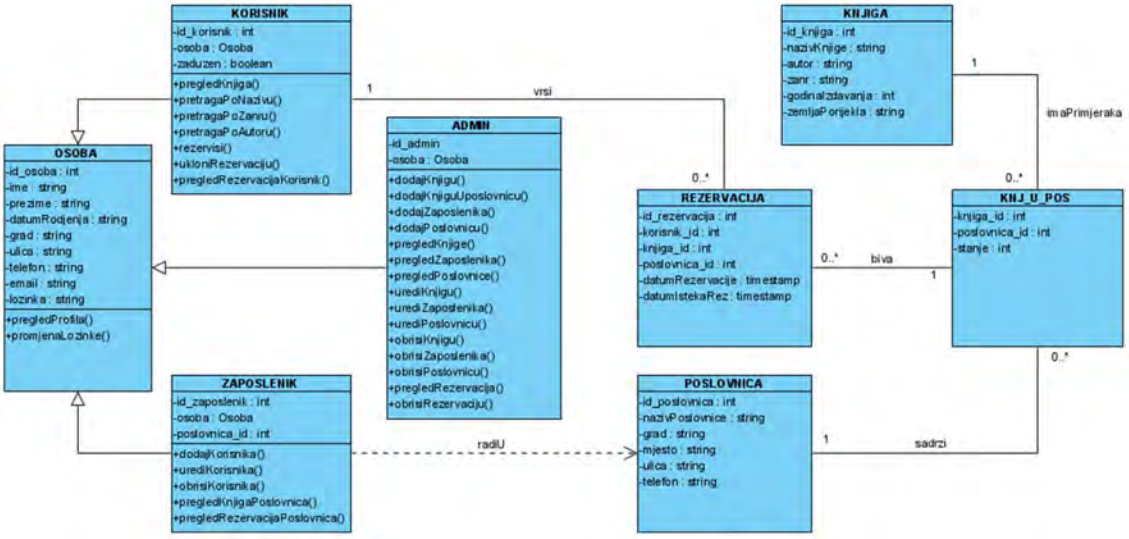


Slika 1. Use Case dijagram za admin korisnika

Ono što možemo ustanoviti na use case dijagramima jeste da, pored pobrojanih funkcionalnosti, svaki akter ima i zajedničku funkcionalnost logina (prijave) 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 sa slučajevima upravljanja, te sa akterom „Sistem“, zbog vršenja promjena unutar same aplikacije.

Class dijagram

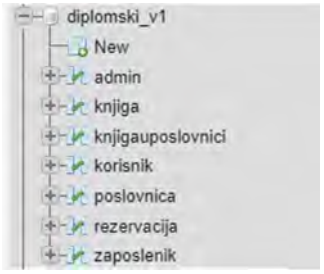
Class (klasni) dijagram je dijagram koji dosta detaljnije opisuje strukturu samog sistema. Elementi ovog dijagrama su klase, što se i osnovni elementi ovog dijagrama, a predstavljaju zasebne cjeline u sistemu. Okarakterisane su atributima i operacijama. Atributi su elementi koji pobliže opisuju klasu, 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 2. Class dijagram

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. Kreiranje i struktuiranje baze podataka znatno je olakšano alatom koji dolazi u sklopu WampServera, koji se zove phpMyAdmin. Ovaj alat uveliko olakšava kreiranje tabela i definisanje atributa, njihovih ograničenja, ali i definisanje primarnih ključeva.



Slika 3. 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. Dakle, baza je uspješno kreirana, i sljedeći korak jeste početak rada u NetBeans okruženju.

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

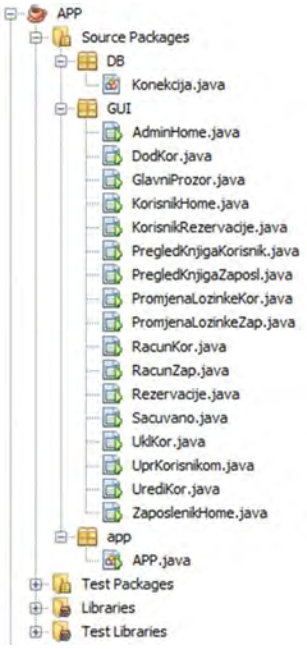
```
public class Konekcija {
   private static final String korisnik = "root";
   private static final String sifra = "";
  private static final String kon = "jdbc:mysql://localhost:3306/dip?serverTimezone=UTC";
public static Connection veza = null;
public Konekcija() throws SQLException, ClassNotFoundException {
      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 4. Prikaz svih kreiranih klasa u Java projektu

Na slici 4. 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);
   } }
```

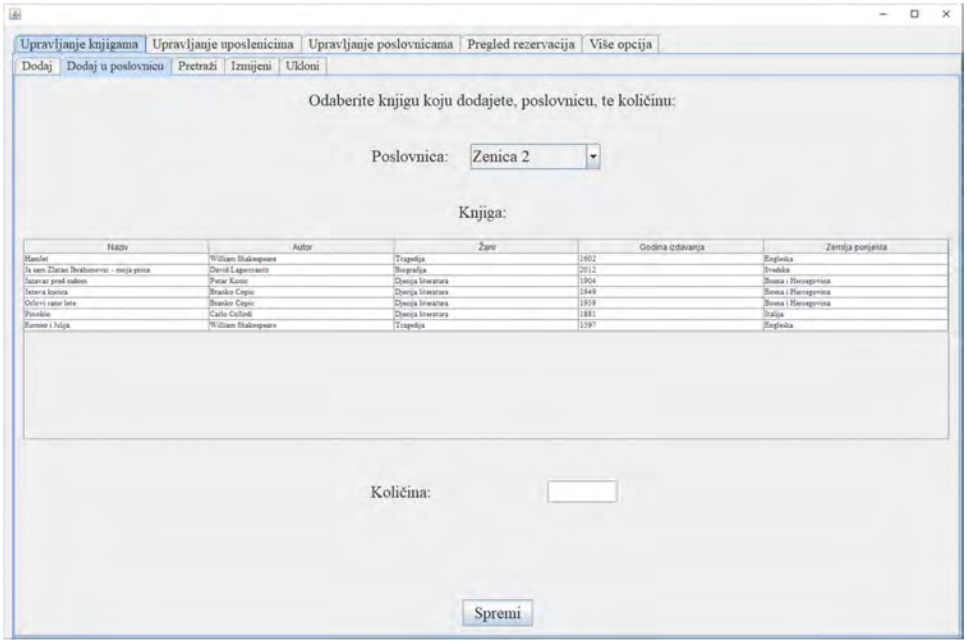
U glavnoj klasi programa se ne nalazi ništa osim inicijalizacije objekta klase početnog prozora aplikacije, nazvanog GlavniProzor.

Slika 5. Prikaz glavnog prozora, te pokušaj prijave sa nevalidnim podacima

Na slici iznad prikazan je glavni prozor, tj. login page aplikacije. Unošenjem validnih podataka, nakon prijave se otvara jedan od 3 moguća prozora: prozor za admin korisnika, prozor za uposlenika, ili prozor za korisnika aplikacije.

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:



Slika 6. 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 btnSumbitKnjiPosActionPerformed(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 iznad prikazana je logika unosa određenog broja knjiga u odabranu poslovnicu. U konstruktoru klase AdminHome (tj. pri otvaranju prozora admin stranice) dešava se dohvatanje 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 odabrane 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.

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 u 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);
    }   }
```

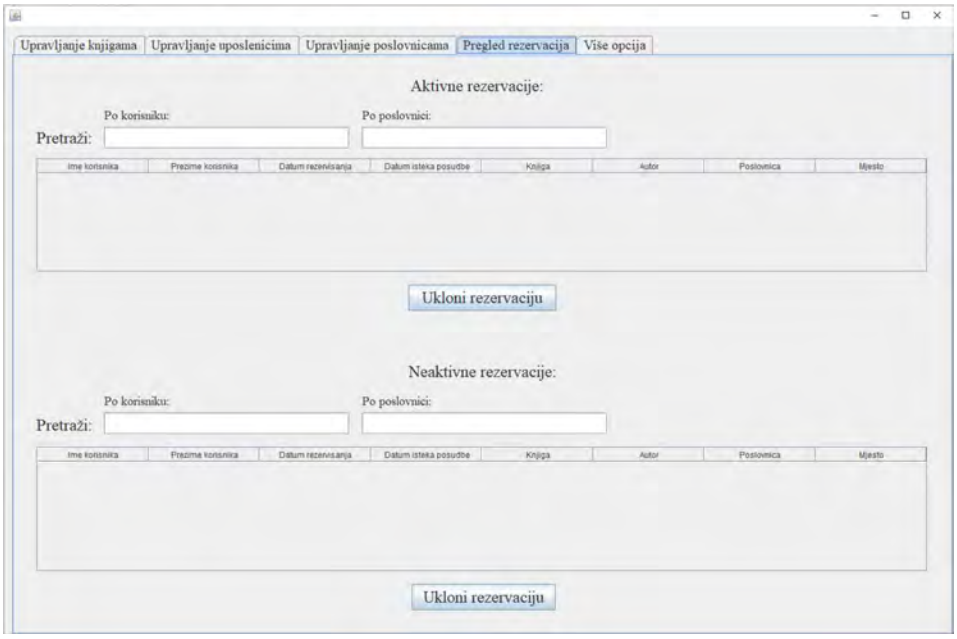
Preostala dva elementa, tj. atributa, koji su integrisani u svaku tabelu sa stajališta poboljšanja sigurnosti jesu datum kreiranja i datum modifikovanja (posljednjeg modifikovanja) svakog od zapisa. Po default-u, i jedan i drugi atribut imaju vrijednost CURRENT_TIMESTAMP (tip podatka je timestamp), što 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 poslovnicama, 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). Admin korisnik ima i opciju brisanja rezervacije, bilo da je aktivna, ili neaktivna. Ipak, detaljnije o rezervacijama biće govora u nastavku rada.

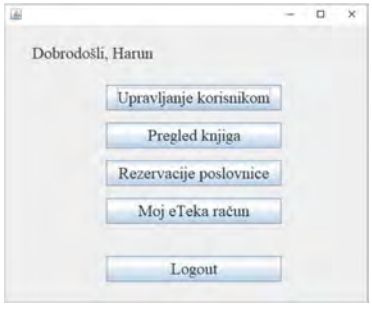
Posljednja funkcionalnost admin korisnika je logout, gdje dolazi do zatvaranja admin page-a, te ponovnom otvaranju login page-a.



Slika 7. Prikaz rezervacija za admin korisnika

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 8. 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.

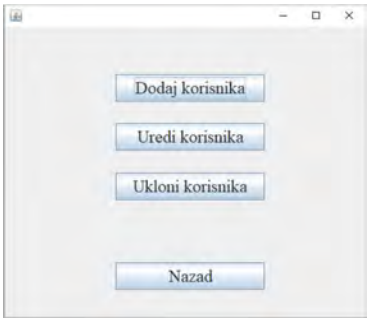
Izvršavanje tranzicije sa prozora na prozor je jako jednostavno implementirati, potrebno je samo napraviti konstruktor koji će prihvatati string argument koji bi predstavljao email adresu korisnika, te izvršiti inicijalizaciju atributa u koji će podatak biti pohranjen. Pri zatvaranju postojećeg i otvaranju novog prozora, on 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. 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 9. Prikaz prozora za upravljanje korisnicima

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.

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 dvaju 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);
        }    }
```

Planovi za budućnost

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.

Zaključak

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.

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.

Literatura

1. J. McGovern, S. Tyagi, S. Mathew, M. Stevens, Java Web Services Architecture. Morgan Kaufmann, 2010.

2. Netbeans, Securing a Web Application in NetBeans IDE, dostupno na URL: https://netbeans.apache.org/kb/docs/web/security-webapps.html
3. Nourie D. (2006), Java Technologies for Web Applications , dostupno na URL: https://www.oracle.com/technical-resources/articles/java/webapps.html
4. https://www.tiobe.com/tiobe-index/
5. https://www.jetbrains.com/lp/devecosystem-2021/java/
6. https://www.javatpoint.com/java-swing
7. https://www.edureka.co/blog/java-jframe/
8. https://netbeans.apache.org/about/history.html
9. https://vertabelo.com/blog/cardinality-in-data-modeling/
10. https://sourceforge.net/projects/wampserver/
11. https://dev.mysql.com/downloads/workbench/

CONCEPTUAL SOLUTION FOR DEVELOPMENT OF APPLICATIONS IN JAVA ENVIRONMENT USING JFRAM

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*