GUI aplikacija za vođenje knjižnice

Ljubić, Robert

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Stručni Studij

GUI APLIKACIJA ZA VOĐENJE KNJIŽNICE

Završni rad

Robert Ljubić

Osijek, 2019.

SADRŽAJ

1. UVOD

Cilj ovog završnog rada je stvaranje GUI (engl. *Graphical User Interface*) aplikacije za vođenje knjižnice. Zaposlenici knjižnice moraju imati jednostavan i uredan pristup bazi podataka kako bi mogli ne smetano raditi svoj posao. To uključuje unos novih knjiga u bazu, njihovo brisanje, dodavanje novih članova u knjižnicu, posudba knjiga tim članovima, te njihovo brisanje. Isto tako, potrebno je da aplikacija omogući zaprimanje posuđenih knjiga i produživanje posudbi ukoliko je potrebno. Kako bi se poboljšao rad knjižnice potrebno je omogućiti zaposlenicima kompletni uvid u statistiku posudbi svih knjiga i statistiku odabranih članova kako bi lakše mogli pratiti popularnost određenih knjiga. Jedan od zaposlenika će biti administrator i on pri prijavi mora dobiti sučelje koje uz ove prijašnje opcije ima i opciju da dodaje i briše zaposlenike iz sustava.

Pri razvoju aplikacije korišteno je *Eclipse* razvojno okruženje. Programski jezici korišteni pri razvoju su Java i FXML. Za pristup bazi podataka, korišten je *SQLite Studio*. Sve ove tehnologije i jezici su detaljnije opisani u drugom poglavlju. U trećem poglavlju je prikazan proces stvaranja bitnijih dijelova aplikacije, dok je u četvrtom poglavlju objašnjeno kako je aplikacija zamišljena da se koristi.

1.1. Zadatak završnog rada

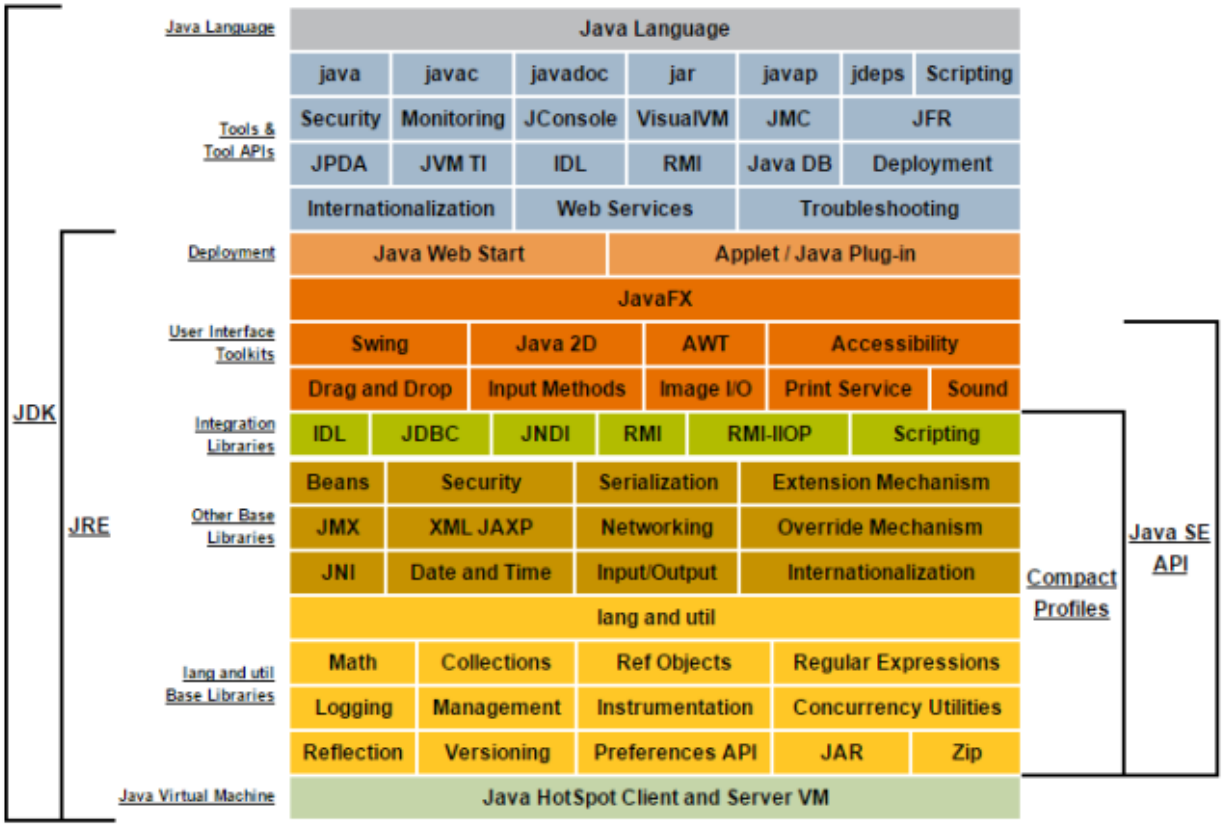
Zadatak ovog završnog rada je izrada desktop aplikacije za vođenje knjižnice koja omogućuje zaposlenicima jednostavno rukovanje posudbom knjiga, dodavanjem i brisanjem knjiga i članova knjižnice, te uvid u statistike članova i knjiga.

2. KORIŠTENE TEHNOLOGIJE I PROGRAMSKI JEZICI

Ovo poglavlje služi za opis svih korištenih tehnologija pri razvoju aplikacije.

2.1. Java

Javu možemo definirati kao programski jezik ili kao platformu. Razvio ju je James Gosling iz tvrtke Sun Microsystems, te je objavljena 1995 godine. 2009. godine tvrtku je otkupio Oracle Corporation, te oni od tada nastavljaju razvoj Jave. Java kao programski jezik je objektno-orijentirani jezik koji je nastao kao odgovor na stvaranje sve više i više platformi i programskih okruženja što je otežavalo stvaranje aplikacija za svako od njih. Ideja je bila da se stvori programski jezik koji se može napisati jednom, te zatim pokrenuti na bilo kojem računalu neovisno o njegovom operacijskom sustavu. Iz te ideje proizlazi temeljni slogan jave „*write once, run anywhere*“ (WORA), odnosno „napiši jednom, pokreni bilo gdje“. Takav način rada doveo je do izrazite popularnosti Jave, te je kasnije prouzrokovalo razvijanje i Microsoftovog .NET Framework-a koji je inkorporirao najuspješnije aspekte Jave. Java je zapravo vrlo slična C-u i C++-u, ali izostavlja neka svojstva kao što su pokazivači i ručno oslobađanje memorijskog prostora (uvođenjem automatskog „*garbage collectora*“). Aplikacije pisane u Javi se tipično kompiliraju u „*bytecode*“-u pomoću kojega se mogu pokrenuti na bilo kojem JVM-u (engl. *Java Virtual Machine*). JVM je virtualni stroj koji računalu omogućuje pokretanje programa pisanih u Java „*bytecode“*-u. Java kao platforma je zapravo skup *software*-a koji služe kako bi se pokrenuo Java kôd. To uključuje već spomenuti JVM, kompajler, te skup raznih biblioteka koje je moguće pozivati u kôdu. Kompajler služi kako bi izvorni kôd preveo u „*bytecode*“, te je dio JDK-a (engl. *Java Development Kit*), koji je zapravo skup svih bitnih programskih alata za pokretanje Java kôda. Još jedna komponenta JDK-a je JRE (engl. *Java Runtime Environment*). JRE sadrži alate za korisnička sučelja kao npr. *JavaFX* koji se koristio za razvijanje aplikacije za ovaj završni rad, integracijske biblioteke, te mnoge druge biblioteke. Spomenuto se može pročitati s Java konceptualnog dijagrama na slici 2.1. [1]



Slika 2.1. *Java Konceptualni Dizajn. [2]*

2.2. JavaFX

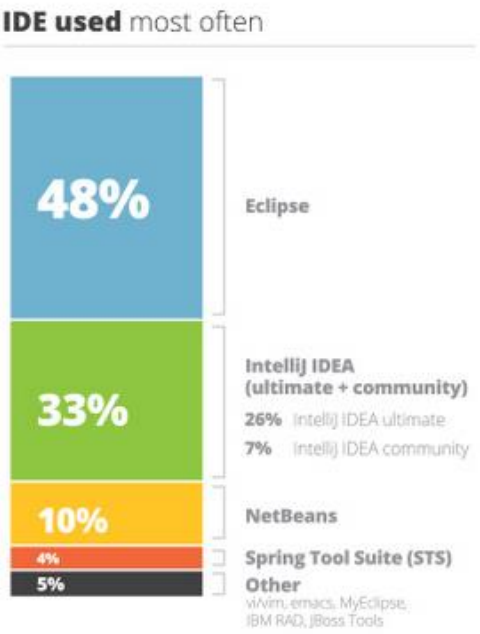
JavaFX je jedan od alata koji služi za kreiranje aplikacija s grafičkim korisničkim sučeljem koje se mogu pokretati na mnogobrojnim uređajima. Namijenjen je kao zamjena za *Swing* i to je alat koji je korišten pri realizaciji ovoga završnoga rada. Jedna od novih funkcionalnosti koje *JavaFX* donosi je FXML – novi *markup* jezik za definiranje korisničkih sučelja baziran na XML-u (engl. *eXtensible Markup Language*) . Primjer FXML kôda se može vidjeti u programskom kôdu 2.1.. Za uređivanje FXML dokumenata korišten je program *SceneBuilder* koji koristi jednostavan „*Drag & Drop*“ sustav kako bi se stvorilo korisničko sučelje. [3]

```xml
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.TableColumn?>
<?import javafx.scene.control.TableView?>
<?import javafx.scene.layout.AnchorPane?>


<AnchorPane xmlns:fx="http://javafx.com/fxml/1" xmlns="http://javafx.com/javafx/8.0.171">
   <children>
      <TableView fx:id="member_table" layoutX="-9.0" prefHeight="368.0" prefWidth="815.0">
        <columns>
          <TableColumn fx:id="member_id" prefWidth="77.0" text="ID_Člana" />
          <TableColumn fx:id="name" prefWidth="150.0" text="Ime" />
            <TableColumn fx:id="last_name" prefWidth="155.0" text="Prezime" />
            <TableColumn fx:id="address" prefWidth="285.0" text="Adresa" />
            <TableColumn fx:id="OIB" prefWidth="147.0" text="OIB" />
        </columns>
      </TableView>
   </children>
</AnchorPane>

```

Programski kôd 2.1. *Primjer FXML* kôd*a.*

2.3. Eclipse

Eclipse je platforma koja programerima pruža razvoj aplikacija u integriranom razvojnom okruženju. Integrirano razvojno okruženje je ono okruženje koje programeru pruža pristup lako shvatljivim alatima koji mu mogu pripomoći u razvoju aplikacije. [4] Prema podacima iz ankete provedene 2014. godine (Slika 2.2.), gdje su ispitanici bili preko 2000 Java programera, *Eclipse* je najpopularnije integrirano razvojno okruženje za razvoj Java aplikacija sa čak 48% korištenosti.



Slika 2.2. *Najkorištenija integrirana razvojna okruženja za razvoj Java aplikacija. [5]*

Drugi popularni alati su *IntelliJ IDEA* s 33%, i *NetBeans* s 10%. Spomenutu anketu je moguće posjetiti na poveznici [5] u literaturi.

2.4. SQL

SQL (engl. *Structured Query Language*) je strukturirani jezik za upite. SQL omogućuje lako upravljanje strukturiranim podacima, specifično onima koji imaju entitete međusobno povezane različitim vezama. Nastao je kao odgovor na sve veće i veće nezadovoljstvo programera koji su htjeli brzi i jednostavni način pristupa bazi podataka. Iako je prije bilo moguće pristupiti bazi podataka, vrlo često je to zahtijevalo pisanje specijalnog programa za različite situacije što u većini slučajeva nije bilo ekonomski isplativo. Način na koji SQL zapravo funkcionira je da dopušta korisnicima da pošalju jedan zahtjev bazi koji nazivamo upit (engl. *Query*), te se taj upit onda obrađuje i izvodi. Pomoću tih upita lako je moguće dodavati, brisati, odabrati, te ažurirati podatke unutar neke tablice podataka. [6] Jedan od najjednostavnijih primjera upita bi bio *SELECT* upit. U sljedećem primjeru (Programski kôd 2.1.) upit odabire sve redove iz tablice „*Customers*“ gdje stupac „*City*“ sadrži *String* „Berlin“ ili „München“.

```
SELECT * FROM Customers
WHERE City='Berlin' OR City='München';
```

Programski kôd 2.1. *Primjer SELECT naredbe.[7]*

3. RAZVOJ APLIKACIJE

Ovo poglavlje opisuje strukturu baze podataka aplikacije, te prikazuje određene dijelove kôda i detaljnije objašnjava njihovu funkciju.

3.1. Baza podataka

Za izradu i korištenje baze podataka u ovom projektu korišten je program *SQLite Studio*. Ova aplikacija sadrži 4 tablice: „*Clanovi*“, „*Posudba*“, „*Knjige*“ i „*Zaposlenici*“. Tablica „*Clanovi*“ sadrži ime, prezime, adresu, OIB, te automatski dodijeljeni identifikacijski broj člana pri dodavanju u sustav. SQL upit pomoću kojega je tablica napravljena prikazan je u programskom kôdu 3.1..

```
CREATE TABLE Clanovi (
    id      INTEGER       PRIMARY KEY AUTOINCREMENT,
    ime     VARCHAR (100),
    prezime VARCHAR (100),
    adresa  VARCHAR (300),
    OIB     BIGINT (11)   UNIQUE
);
```

Programski kôd 3.1. *Stvaranje tablice „Clanovi“.*

Tablica „*Knjige*“ sadrži ime knjige, ime autora, godinu izdanja knjige, identifikacijski broj knjige, te vrijednost koja predstavlja zastavicu čija je uloga praćenje posudbi knjige. Upit pomoću kojega je tablica napravljena prikazan je programskim kôdom 3.2..

```
CREATE TABLE Knjige (
    ID_knjige      INTEGER       PRIMARY KEY AUTOINCREMENT
                                 NOT NULL,
    ime_knjige     VARCHAR (100) NOT NULL,
    ime_autora     VARCHAR (100) NOT NULL,
    godina_izdanja INTEGER (4),
    posudena       INTEGER       DEFAULT (0)
);
```

Programski kôd 3.2. *Stvaranje tablice „Knjige“.*

Tablica „*Zaposlenici*“ sadrži korisničko ime i lozinku zaposlenika potrebnih za prijavu u sustav, ime i prezime zaposlenika, OIB zaposlenika, njegovu adresu, njegov identifikator, te vrijednost koja govori je li zaposlenik administrator kako bi mu sustav mogao pružiti dodane opcije pri prijavi. Upit pomoću kojega je tablica napravljena prikazan je programskim kôdom 3.3..

```
CREATE TABLE Zaposlenici (
    ID_zaposlenika INTEGER       PRIMARY KEY AUTOINCREMENT,
    ime            VARCHAR (100),
    prezime        VARCHAR (100),
    username       VARCHAR (100),
    password       VARCHAR (100),
    adresa         VARCHAR (300),
    OIB            BIGINT (11)   UNIQUE,
    administrator  INTEGER (1)   DEFAULT (0)
);
```

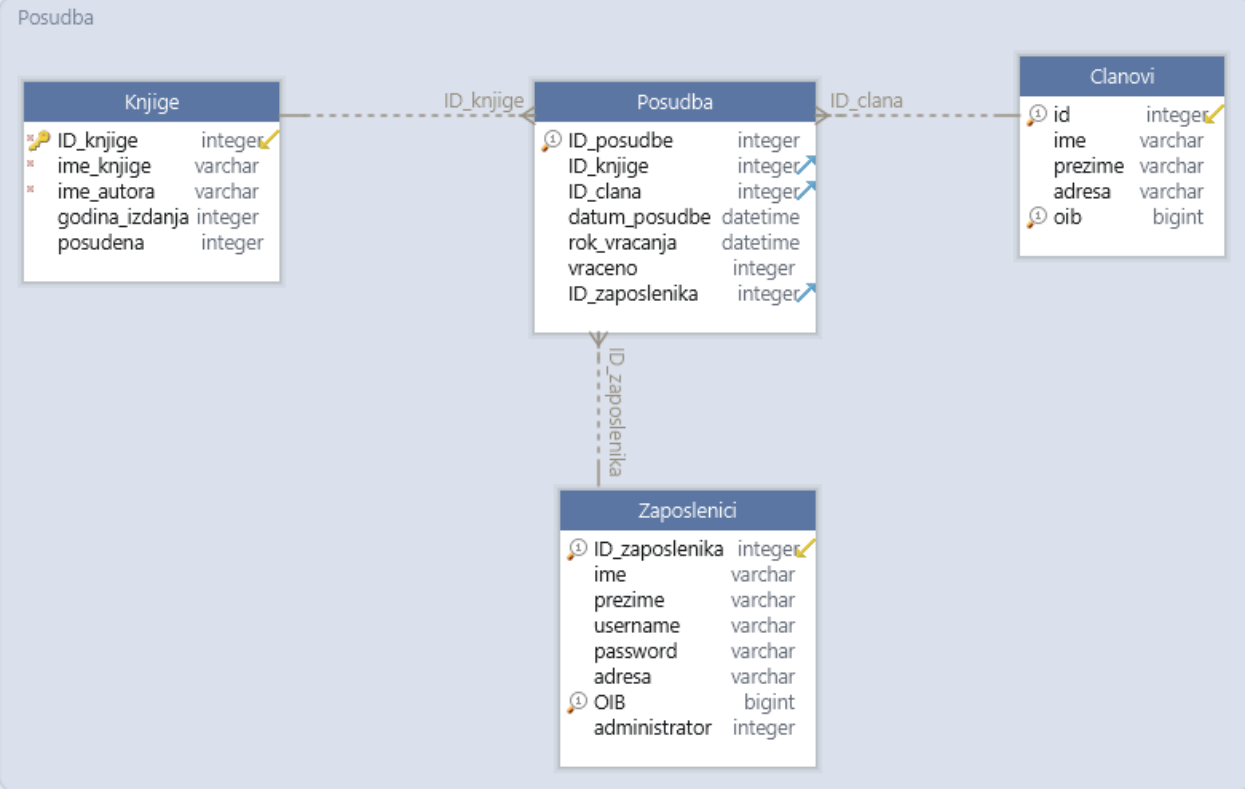
Programski kôd 3.3. *Stvaranje tablice „Zaposlenici“.*

Zadnja tablica je tablica „*Posudba*“ preko koje su sve druge tablice povezane. Primarni ključevi iz svih prijašnjih tablica se nalaze u ovoj tablici kao strani ključevi. Osim toga, ova tablica sadrži još jedan primarni ključ koji služi kao identifikator posudbe. Pomoću tih identifikatora, lako se može pratiti tko je sve sudjelovao u procesu određene posudbe kako bi se moglo lakše riješili određene probleme kao mogući nestanak knjige. Pored identifikatora, tablica još sadrži i datum na koji se vrši posudba, datum do kojeg se knjiga mora vratiti, te varijablu „*vraceno*“ koja nam govori je li knjiga vraćena ili ne. Upit pomoću kojega je stvorena tablica „*Posudba*“ moguće je vidjeti u programskom kôdu 3.4..

```
CREATE TABLE Posudba (
    ID_posudbe     INTEGER     PRIMARY KEY AUTOINCREMENT,
    ID_knjige      INTEGER     REFERENCES Knjige (ID_knjige),
    ID_clana       INTEGER     REFERENCES Clanovi (id),
    ID_zaposlenika INTEGER     REFERENCES Zaposlenici (ID_zaposlenika),
    datum_posudbe  DATETIME,
    rok_vracanja   DATETIME,
    vraceno        INTEGER (1) DEFAULT (0)
);
```

Programski kôd 3.4. *Stvaranje tablice Posudba.*

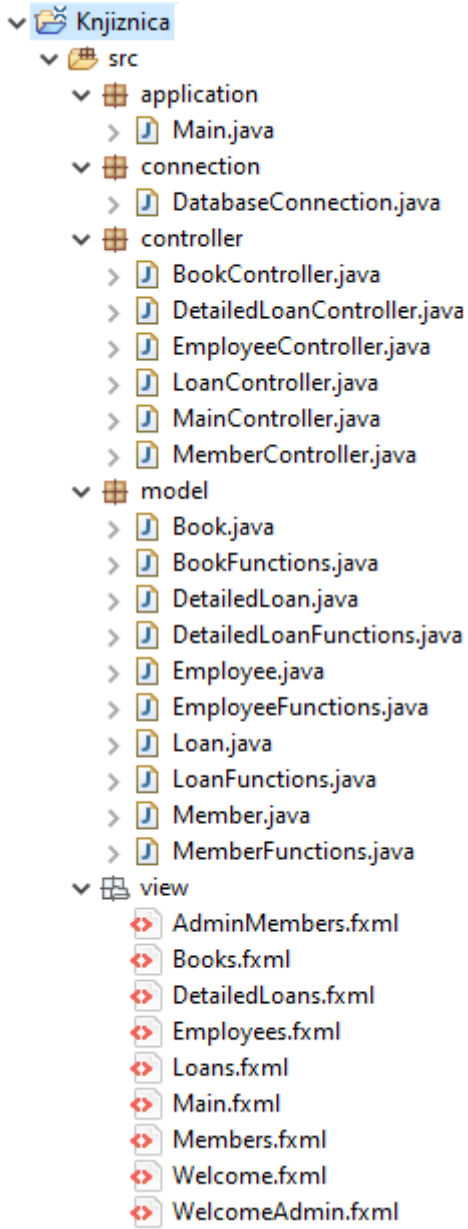
Iz programskih kôdova 3.1. – 3.4. vidljivo je da svaki atribut koji predstavlja identifikator ima svojstvo „*AUTOINCREMENT*“. To svojstvo govori da će se svakom novom redu tablice automatski dodati identifikator prijašnjeg reda + 1. Zahvaljujući takvom sustavu zaposlenici ne moraju unositi identifikatore sami, već ih sustav sam unosi što štedi vrijeme i smanjuje mogućnost ljudske greške. Na slici 3.1. može se vidjeti pregledniji pogled na veze između spomenutih tablica.



Slika 3.1. *Pregled veza između tablica koje tvore bazu podataka aplikacije.*

3.2. Programska rješenja

Projekt „*Knjižnica*“ sastoji se od 5 paketa. To su paketi „*application“*, *„connection“*, *„controller“*, *„model“* i „*view“* (Slika 3.2.). U „*application“* paketu nalazi se „*Main.java“* klasa (Programski kôd 3.5.) programa preko koje se pokreće program. U klasi „*Main.java“* može se vidjeti „*main()“* metoda čija je svrha pokrenuti *JavaFX* aplikaciju ukoliko se aplikacija nije pokrenula bez nje kao što bi trebala. Osim „*main()“* metode, postoji još i preopterećena „*start()“* metoda. U toj metodi dodjeljuje se fxml dokument koji se treba učitati pomoću „*FXMLLoader.load()“* metode, te se sprema u referencu klase „*Parent“* kao objekt. Referenca klase „*Parent“* se tada koristi kako bi se postavila scena, te na kraju i sama „pozornica“ programa.



Slika 3.2. *Strukturni pregled projekta.*

„Connection“ paket sadrži klasu *„DatabaseConnection.java“* (Programski kôd 3.6.) koja služi kako bi se otvarala i zatvarala veza sa *SQLite* bazom podataka *„Library.db“* u kojoj se nalaze sve tablice iz prijašnjeg potpoglavlja. Za otvaranje veze s bazom podataka služi nam metoda *„getConnection()“* koja uspostavlja vezu s JDBC (engl. *Java DataBase Connectivity)* driverom preko *„forName()"* metode. Nakon toga, pomoću *„DriverManager.getConnection()“* metode, šalje se relativni put do korištene baze podataka. U slučaju ove aplikacije, baza podataka se nalazi direktno u korijenskom (engl. *root*) direktoriju projekta. Metoda *„closeConnection()“* služi kako bi se konekcija s bazom podataka prekinula, pod uvjetom da je uopće uspostavljena.

```
package application;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class Main extends Application{

	public static void main(String[] args) {
		launch(args);
	}

	@Override
	public void start(Stage primaryStage) throws Exception {

		Parent root =
FXMLLoader.load(getClass().getResource("/view/Main.fxml"));

		Scene scene = new Scene(root);

		primaryStage.setTitle("Knjiznica");
		primaryStage.setScene(scene);
		primaryStage.sizeToScene();
		primaryStage.centerOnScreen();
		primaryStage.show();
	}

}
```

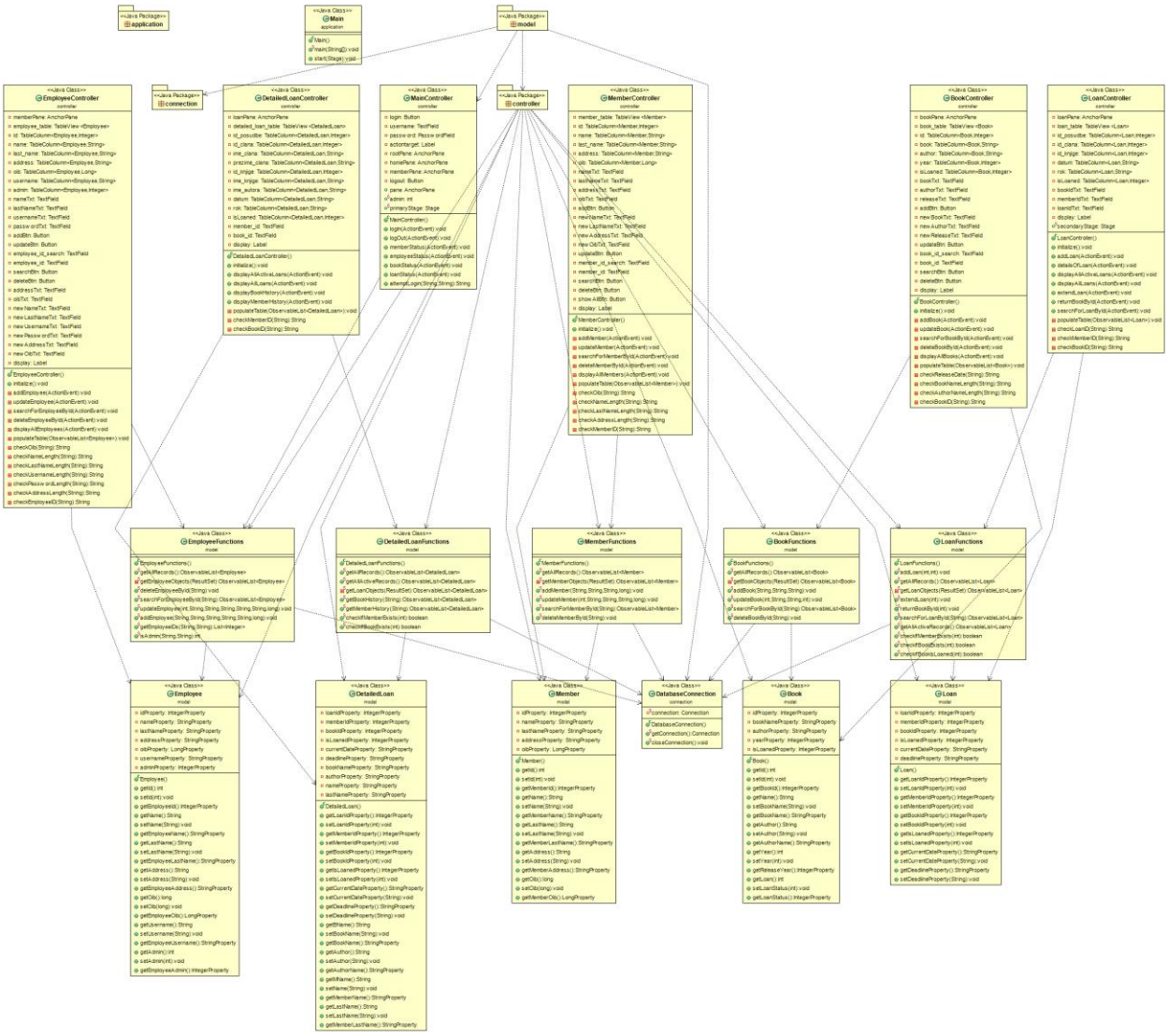
Programski kôd 3.5. *Izgled main.java klase.*

Sljedeća tri paketa su „*model*“, „*controller*“ i „*view*“. Ti paketi sadrže najveći dio kôda projekta. U paketu „*model*“ nalazi se deset klasa što sadrže sve funkcije koje rade direktno s bazom podataka. Paket „*view*“ sadrži devet fxml dokumenta koji služe za direktnu komunikaciju s korisnikom, odnosno predstavlja takozvano korisničko sučelje. Paket „*controller*“ sadrži šest klasa čija je svrha validiranje korisnikovih unosa, te slanje validiranih podataka odgovarajućoj metodi u „*model*“ klasi. „*Controller*“ paket služi i za prikazivanje pravilnog pogleda korisniku ovisno o pritisnutoj tipci. Uzimajući u obzir da aplikacija sadrži puno metoda koji imaju sličnu funkciju, komentirane i objašnjene su samo neke od njih. Kao primjer dodavanja, čitanja, ažuriranja, te brisanja redaka iz baze podataka korišten je „*Books.fxml*“ pogled.

```
package connection;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;


	public class DatabaseConnection {

		private static Connection connection = null;

		public static Connection getConnection() throws SQLException,
	ClassNotFoundException {

			Class.forName("org.sqlite.JDBC");
			String url = "jdbc:sqlite:Library.db";
			return DriverManager.getConnection(url);
		}

		public static void closeConnection() throws SQLException{
			try {
				if(connection != null && !connection.isClosed()) {
					connection.close();
				}
			}catch(Exception e) {
				throw e;
			}
		}




	}
```

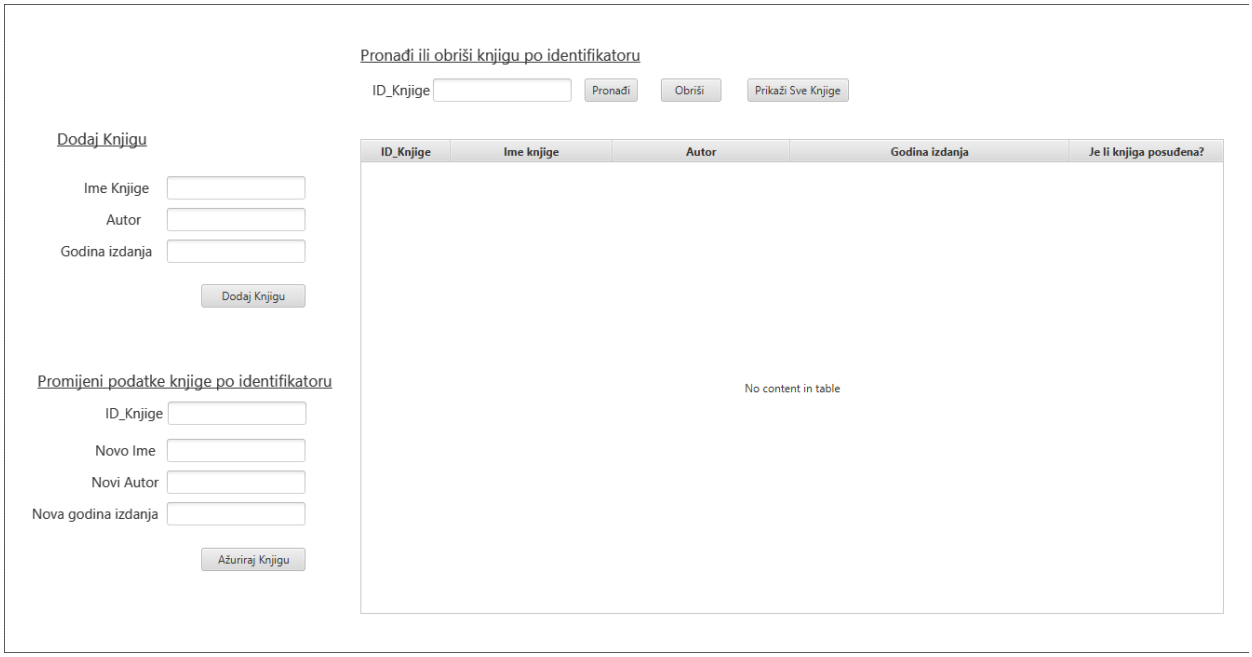
Programski kôd 3.6. *Izgled DatabaseConnection.java klase*.

Veze između određenih klasa programa mogu se detaljnije vidjeti na UML (engl. *Unified Modeling Language*) dijagramu (Slika 3.3.) koji je dobiven korištenjem *Eclipse* „*ObjectAid UML Diagram" plugin*-a. Osim veza između klasa, UML dijagram još sadrži i listu svih atributa i metoda koje klasa sadrži. Atributi klasa su označeni kvadratićem sa crvenim obrubom. Zeleni kružići pored metoda naznačavaju da su javne, dok crveni kvadratići naznačavaju da su privatne.



Slika 3.3. *UML dijagram aplikacije*

Na slici 3.4. prikazano je korisničko sučelje za „*Books.fxml"* pogled, koje je napravljeno koristeći *SceneBuilder* aplikaciju. U ovom dijelu poglavlja objašnjeno je kako se od korisničkog unosa podaci spremaju u bazu podataka, te se prikazuju na tablici.



Slika 3.4. *Izgled „Books.fxml“ u SceneBuilder-u.*

Za početak, ukoliko je pritisnuta tipka „*Dodaj Knjigu*“ poziva se metoda „*addBook()*“ iz klase „*BookController.java*“ (Programski kôd 3.7.).

```
@FXML
	private void addBook(ActionEvent event) throws Exception {

		String godina_izdanja = checkReleaseDate(releaseTxt.getText());
		String ime_autora = checkAuthorNameLength(authorTxt.getText());
		String ime_knjige = checkBookNameLength(bookTxt.getText());

		if (ime_knjige == null || ime_autora == null || godina_izdanja ==
		null) {
			return;
		}
		BookFunctions.addBook(ime_knjige, ime_autora, godina_izdanja);
		ObservableList<Book> bookList = BookFunctions.getAllRecords();
		populateTable(bookList);

		display.setTextFill(Color.GREEN);
		display.setText("Knjiga dodana!");

	}
```

Programski kôd 3.7. *Kôd metode „BookController.addBook()“.*

Metoda „*addBook()*“ prvo vrši validaciju nad unesenim parametrima ime knjige, ime autora, te godina izdanja. Kako bi se te validacije izvršile, koriste se tri pomoćne metode: „*checkReleaseDate()*“, „*checkAuthorNameLength()*“ i „*checkBookNameLength()*“. Na primjer,

metoda „*checkBookNameLength()*“ (Programski kôd 3.8.) provjerava je li unos za ime knjige prazan ili *null*, te ukoliko nije provjerava je li broj znakova između dva i dvjesto.

```
private String checkBookNameLength(String text) {
          if (text.equals(null) || text.isEmpty()) {
                display.setTextFill(Color.FIREBRICK);
                display.setText("Ime knjige ne smije biti prazno!");
                return null;
          } else if (text.length() < 2 || text.length() > 200) {
                display.setTextFill(Color.FIREBRICK);
                display.setText("Ime knjige mora biti u intervalu [2,200]");
                return null;
          } else {
                return text;
          }

    }
```

Programski kôd 3.8. *Kôd metode „checkBookNameLength()“.*

Ukoliko unos prođe obje provjere šalje se dalje u metodu „*BookFunctions.addBook()*“ (Programski kôd 3.9.) gdje se odrađuje dodavanje knjige u bazu.

```
public static void addBook(String book, String author, String releaseYear)
                throws ClassNotFoundException, SQLException {

          Connection conn = null;
          try {
                conn = DatabaseConnection.getConnection();
                String sql = "INSERT INTO Knjige(ime_knjige, ime_autora,
   godina_izdanja) VALUES(?, ?, ?)";
                PreparedStatement myStat = conn.prepareStatement(sql);
                myStat.setString(1, book);
                myStat.setString(2, author);
                myStat.setString(3, releaseYear);
                myStat.executeUpdate();
          } catch (SQLException e) {
                throw e;
          } finally {
                try {
                      DatabaseConnection.closeConnection();
                } catch (SQLException e) {
                      throw e;
                }
          }

    }
```

Programski kôd 3.9. *Kôd metode „BookFunctions.addBook()“.*

Ukoliko bilo koji uneseni podatak ne prođe validaciju, niti jedan od njih se ne šalje u klasu u „*model*“ paketu, već se na „*Books.fxml*“ šalje poruka greške kako bi ju korisnik mogao ispraviti. Metoda „*BookFunctions.addBook()*“ prvo pokušava uspostaviti konekciju s bazom podataka pozivanjem metode „*DatabaseConnection.getConnection()*“ koja je opisana ranije. Ukoliko je konekcija s bazom podataka uspješna, izvršava se SQL upit koji unesene parametre sprema u odgovarajuća mjesta u tablici „*Knjige*“. U ovom slučaju, za izvršavanje upita korišten je „*PreparedStatement*“. „*PreparedStatement*“ predstavlja objekt koji može spremiti i kompilirati SQL upit i prije nego dobije parametre. Takvim načinom komunikacije s bazom podataka osigurano je što brže izvršavanje upita. Osim brzine, „*PreparedStatement*“ omogućava i zaštitu od „SQL *Injection*“ napada tako što ne dopušta unos cijelih SQL naredbi, već samo unos parametara. Nakon što metoda dobije parametre, unosi ih u SQL upit, te ga izvrši. Nakon izvršavanja SQL upita, metoda prestaje s radom, te se izvršavanje nastavlja u „*BookController.addBook()*“ metodi. U sljedećem koraku se novo unesenu knjigu, zajedno s ostalim knjigama, prikazuju na tablici koja se nalazi na slici 3.4. Također se poziva metoda „*BookFunctions.getAllRecords()*“ (Programski kôd 3.10.).

```
public static ObservableList<Book> getAllRecords() throws
ClassNotFoundException, SQLException {
            Connection conn = null;

            try {
                  conn = DatabaseConnection.getConnection();
                  String sql = "SELECT * FROM Knjige";
                  PreparedStatement myStat = conn.prepareStatement(sql);
                  ResultSet myRs = myStat.executeQuery();

                  ObservableList<Book> oblist = getBookObjects(myRs);
                  return oblist;
            } catch (SQLException e) {
                  throw e;
            } finally {
                  try {
                        DatabaseConnection.closeConnection();
                  } catch (SQLException e) {
                        throw e;
                  }
            }
      }
```

Programski kôd 3.10. *Kôd metode „BookFunctions.getAllRecords()“.*

Metoda „*getAllRecords()*“ služi kako bi prikazala sve podatke o svim knjigama na tablicu. Prvo se uspostavlja veza s bazom podataka, te se zatim SQL upitom dohvaćaju svi podaci iz tablice „*Knjige*“. Nakon toga se ti podaci spremaju u varijablu „*myRs*“, te se prosljeđuju metodi „*getBookObjects()*“ (Programski kôd 3.11.).

```
	private static ObservableList<Book> getBookObjects(ResultSet myRs) throws
SQLException {
		try {
			ObservableList<Book> bookList =
			FXCollections.observableArrayList();

			while (myRs.next()) {
				Book book = new Book();
				book.setId(myRs.getInt("ID_knjige"));
				book.setBookName(myRs.getString("ime_knjige"));
				book.setAuthor(myRs.getString("ime_autora"));
				book.setYear(myRs.getInt("godina_izdanja"));
				book.setLoanStatus(myRs.getInt("posudena"));
				bookList.add(book);

			}
			return bookList;
		} catch (SQLException e) {
			throw e;
		} finally {
			try {
				DatabaseConnection.closeConnection();
			} catch (SQLException e) {
				throw e;
			}
		}
	}
```

Programski kôd 3.11. *Kôd metode „BookFunctions.getBookObjects()“.*

U metodi „*getBookObjects()*“ se stvaraju lista knjiga i objekt klase „*Book.java*“ (Programski kôd 3.12.). Zatim se pomoću „*gettera()*“ i „*settera()*“ iz varijable „*myRs*“ dodjeljuju podaci iz baze podataka na objekt tipa „*Book*“ , te se taj objekt dodaje u listu knjiga. Nakon što se svi novostvoreni objekti spreme u listu knjiga, ona se vraća nazad na „*BookController.addBook()*“, gdje se poziva metoda „*populateTable()*“ (Programski kôd 3.13.) koja prikazuje sve stupce tablice „*Knjige*“ iz baze podataka na tablicu „*book_table*“ u „*Books.fxml*“. Time je završen proces dodavanja knjige u knjižnicu. Na sličan način radi i ažuriranje podataka knjige, brisanje knjige, te ispis knjige.

```
package model;

import javafx.beans.property.IntegerProperty;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;

public class Book {

	private IntegerProperty idProperty;
	private StringProperty bookNameProperty;
	private StringProperty authorProperty;
	private IntegerProperty yearProperty;
	private IntegerProperty isLoanedProperty;

	public Book() {
		super();
		this.idProperty = new SimpleIntegerProperty();
		this.bookNameProperty = new SimpleStringProperty();
		this.authorProperty = new SimpleStringProperty();
		this.yearProperty = new SimpleIntegerProperty();
		this.isLoanedProperty = new SimpleIntegerProperty();
	}

	public int getId() {
		return idProperty.get();
	}

	public void setId(int id) {
		this.idProperty.set(id);
	}

	public IntegerProperty getBookId() {
		return idProperty;
	}
```

Programski kôd 3.12. *Dio kôda „Book.java“ klase.*

```
private void populateTable(ObservableList<Book> bookList) {

		book_table.setItems(bookList);

	}
```

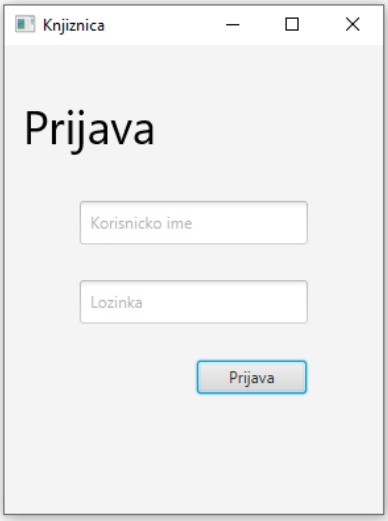
Programski kôd 3.13. *Metoda „populateTable()“.*

4. KORIŠTENJE APLIKACIJE

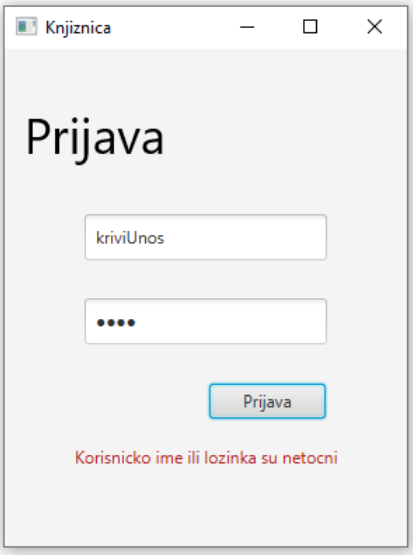
U ovom poglavlju detaljno je objašnjeno kako se aplikacija koristi.

4.1. Prijava

Pri pokretanju aplikacije, korisnici dobiju sljedeći prozor (Slika 4.1.). Na tom prozoru se vidi tekst u gornjem lijevom kutu koji korisniku naznačuje da taj prozor služi za prijavu u sustav. Zatim, u sredini prozora, postoje jedan „*TextField*“ i jedan „*PasswordField*“ koji služe za unos korisničkog imena i lozinke. „*TextField*“ je prostor u koji korisnik može unijeti tekst koji se onda može obrađivati unutar aplikacije. „*PasswordField*“ radi istu stvar, osim što je pri unosu tekst enkriptiran kao što možemo vidjeti na slici 4.2. . Kako bi korisnicima bilo jasno za što služi koja kućica, postoji “*placeholder text*“ koji naznačuje što ide u koju kućicu. Kada korisnik unese tražene podatke, pritiskom na tipku Prijava dogode se jedna od tri mogućnosti. Prva mogućnost je da su korisničko ime i lozinka ne točni, odnosno da ne postoji niti jedan red u bazi podataka koji sadrži to korisničko ime i tu lozinku. U tom slučaju, korisnik će na dnu aplikacije dobiti poruku prikazanu na slici 4.2.

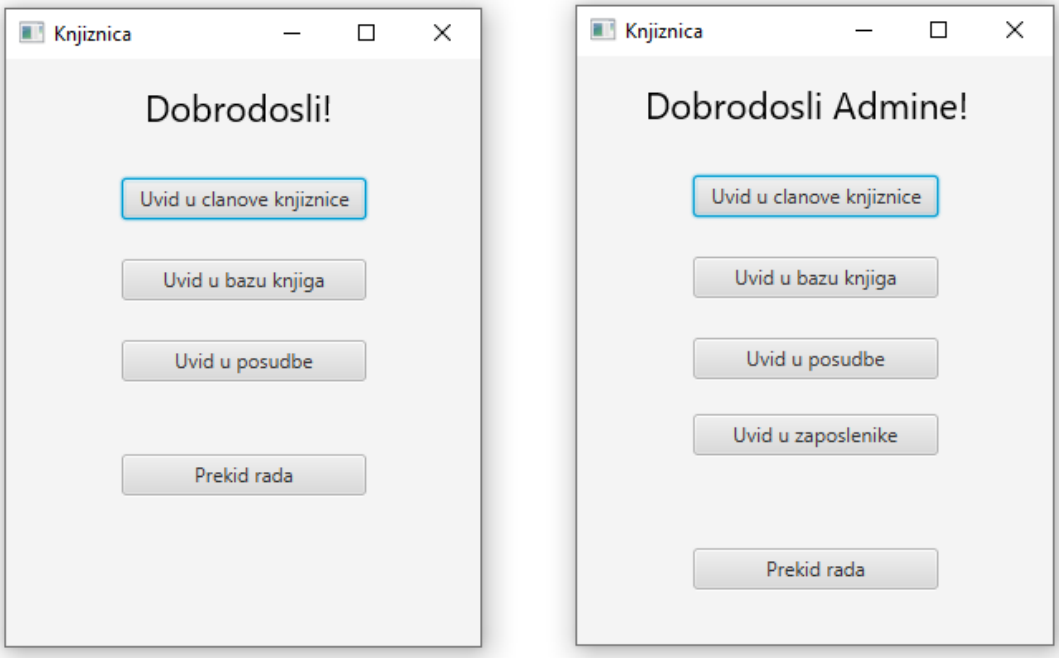


Slika 4.1. *Izgled sučelja za prijavu u sustav.*



Slika 4.2. *Izgled sučelja za prijavu u sustav pri krivom unosu.*

Druga dva slučaja su kada su korisničko ime i lozinka uneseni ispravno. Ovisno o tome da li je korisnik administrator ili ne, na pregled dobiju različita sučelja (Slika 4.3.).



Slika 4.3. *Izgled sučelja nakon prijave za običnog zaposlenika (lijevo) i administratora (desno).*

Na slici se može vidjeti da zaposlenik ima većinu opcija. Jedina opcija koja mu je nedostupna je „*Uvid u zaposlenike*“. Ta opcija služi kako bi administrator mogao brisati ili dodavati nove

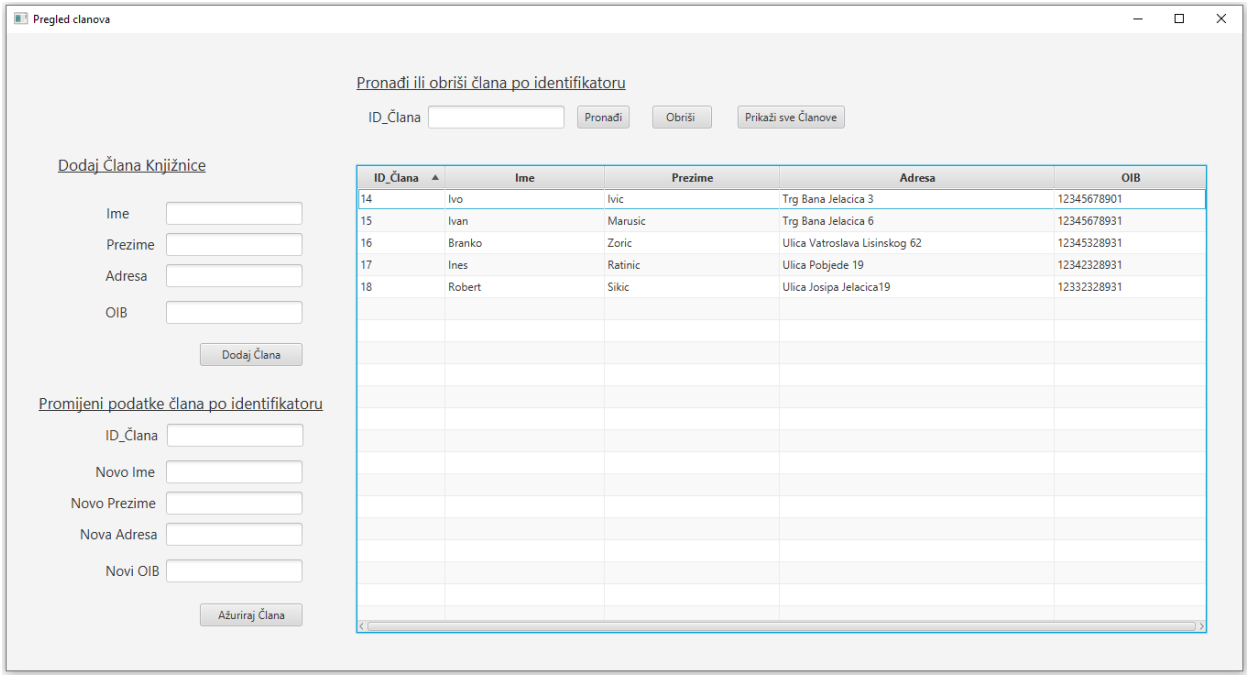
zaposlenike. Kako bi se korisnik odjavio iz sustava potrebno je kliknuti na tipku „*Prekid rada“* koja će ga zatim vratiti na prvi pogled koji vidimo na slici 4.1.

4.2. Uvid u članove knjižnice

Pritiskom na tipku „*Uvid u clanove knjiznice“* korisniku se otvori novi prozor pod nazivom „*Pregled clanova“* (Slika 4.4.). U tom prozoru zaposlenik može, ukoliko želi, dodati novog člana, ažurirati podatke već postojećeg člana, prikazati sve članove na tablici, pronaći podatke o određenom članu, te obrisati člana iz knjižnice. Kako bi se korisnik mogao što lakše koristiti aplikacijom prozor je podijeljen na više dijelova. Na lijevoj strani tablice postoji prostor za dodavanje novog člana knjižnice, te prostor za ažuriranje već postojećeg člana, dok se iznad tablice nalazi prostor za brisanje i traženje člana knjižnice preko njegovog identifikatora, te tipka za prikaz svih članova knjižnice.

Tablica članova sadrži identifikator člana, njegovo ime, prezime, adresu, te njegov OIB. Korisnik tablicu može sortirati uzlazno ili silazno po bilo kojem od tih parametara pritiskom na sam stupac.

Ukoliko se korisnik odluči za dodavanje novoga člana, potrebno je u prostor za dodavanje novoga člana unijeti odgovarajuće parametre (Ime, Prezime, Adresa i OIB) te pritisnuti tipku „*Dodaj Člana“*. Nakon što je tipka pritisnuta, program vrši sve verifikacije kako bi ustanovio jesu li svi parametri uneseni, i ukoliko jesu, provjerava jesu li uneseni pravilno. U slučaju da korisnik ne unese ništa, u gornjem lijevom kutu programa će iskočiti poruka napisana crvenim slovima koja naznačuje kako ime člana ne smije biti prazno. Ako uneseno ime nije između dva i dvjesto znakova, na isto mjesto će iskočiti crvena poruka koja govori kako ime člana mora biti u intervalu [2, 200]. Nakon što korisnikov unos za polje „*Ime“* zadovoljava prijašnje spomenutim uvjetima, ista ta verifikacija se vrši i za prezime i za adresu, te konačno i za OIB, gdje će program, ukoliko je polje ostavljeno prazno, izbaciti poruku „Morate unijeti važeći OIB – 11 znakova!“. U slučaju da korisnik unese bilo što osim 11 znamenki, program će izbaciti poruku „OIB mora imati 11 brojeva!“. Ukoliko su svi podaci pravilno uneseni, u gornjem lijevom kutu prozora će iskočiti zelena poruka „Član dodan!“. Nakon uspješnog dodavanja člana knjižnice, sustav će članu automatski dodijeliti identifikacijski broj, te ga upisati u tablicu na desnoj strani.



Slika 4.4. *Prozor koji se otvori nakon pritiska na tipku „Uvid u clanove knjiznice“.*

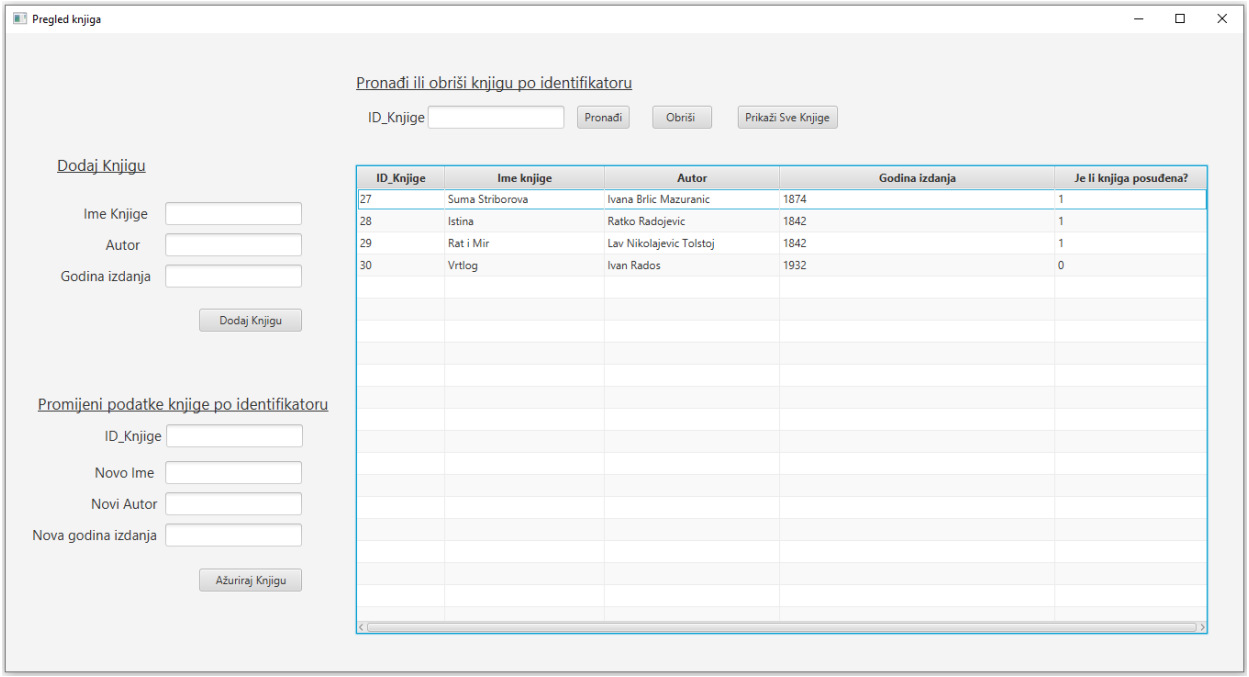
Promjena podataka o članu funkcionira na sličan način. Prva stvar koju korisnik mora unijeti je identifikator člana čije podatke želi mijenjati. Ukoliko korisnik stisne tipku „*Ažuriraj Člana“*, a da je pritom ostavio polje „*ID_Člana“* prazno, u gornjem lijevom kutu aplikacije će crvenim slovima iskočiti poruka „ID člana ne smije biti prazan!“. U slučaju da su u polje uneseni znakovi koji nisu brojevi, iskočiti će poruka „ID člana mora biti broj“. Verifikacija za novo ime, prezime, adresu i OIB vrši se na isti način kao i pri unosu novoga člana. Pritiskom na tipku „*Ažuriraj Člana“*, pod uvjetom da su sva polja pravilno ispunjena, u gornjem lijevom kutu pojavljuje se zelena poruka „Član ažuriran!“.

Pronalazak člana po identifikatoru vrši se tako što se u polje za unos upiše identifikator člana knjižnice čije podatke želimo vidjeti u tablici, nakon čega se pritisne tipka „*Pronađi“*. Zatim, tablica na ekranu prikaže sve podatke člana knjižnice s unesenim identifikatorom. Isto kao i kôd ažuriranja podataka člana, uneseni identifikator člana mora biti broj. Kako bi korisnik opet mogao vidjeti sve članove knjižnice u tablici, potrebno je kliknuti na tipku „*Prikaži se Članove“*.

Kako bi korisnik obrisao člana knjižnice, u polje za unos je potrebno unijeti broj koji predstavlja identifikator člana knjižnice kojeg želi obrisati, te zatim pritisnuti tipku „*Obriši“*. Nakon pritiska tipke, član s unesenim identifikatorom će biti izbrisan iz baze podataka, te će se tablica članova automatski ažurirati.

4.3. Uvid u bazu knjiga

Pritiskom na tipku „*Uvid u bazu knjiga*“ otvara se novi prozor „*Pregled knjiga*“ koji sadrži tablicu sa svim ključnim informacijama o knjizi (Slika 4.5.). To su ime knjige, ime autora knjige, godina izdanja knjige, njezin identifikator, te broj koji govori je li knjiga posuđena ili ne („1“ označava da je knjiga posuđena, dok „0“ označava da nije). Isto kao i kôd tablice članova, svi ti parametri se mogu sortirati ulazno i silazno pritiskom na željeni stupac. Osim tablice, prozor još sadrži dijelove koji su jako slični onima iz uvida o članovima knjižnice. Prostor za dodavanje knjige nalazi se na istom mjestu kao i prostor za dodavanje novih članova, prostor za ažuriranje podataka knjige nalazi se na istom mjestu kao i prostor za ažuriranje podataka člana knjižnice i sl.



Slika 4.5. *Prozor koji se otvori nakon pritiska na tipku „Uvid u bazu knjiga“.*

Dodavanje nove knjige zahtjeva unos imena i autora koji moraju biti između dva i dvjesto znakova kako bi prošli validaciju, dok godina izdanja mora biti u intervalu cijelih brojeva [1000, 2019]. Ukoliko su sva polja pravilno ispunjena, pritiskom na tipku „*Dodaj Knjigu*“ podaci se spremaju u bazu podataka, te se informacije o knjizi automatski prikazuju na tablici. Identifikator knjige se automatski dodjeljuje na sljedeći dostupni broj, dok je zadana vrijednost stupca „*Je li knjiga posuđena?*“ postavljena na „0“. Za ažuriranje knjige, potrebno je unijeti njen identifikator koji mora biti broj, novo ime autora i knjige koji moraju biti u intervalu između dva i dvjesto znakova, te nova godina izdanja koja mora biti u intervalu cijelih brojeva [1000, 2019]. Nakon što su svi

parametri pravilno uneseni, pritiskom na tipku „*Ažuriraj knjigu“* podaci se automatski ažuriraju u bazi podataka, te u tablici.

Za pronalazak ili brisanje knjige potrebno je unijeti identifikator knjige nakon čega se pritisne tipka „*Pronađi“* ili „*Obriši“*. Ovisno o pritisnutoj tipci, red koji sadrži uneseni identifikator će biti ispisan u tablici ili izbrisan iz nje. Kako bi se nakon pronalaska određene knjige opet mogle vidjeti informacije o svim knjigama potrebno je samo pritisnuti tipku „*Prikaži Sve Knjige“*.

4.4. Uvid u zaposlenike

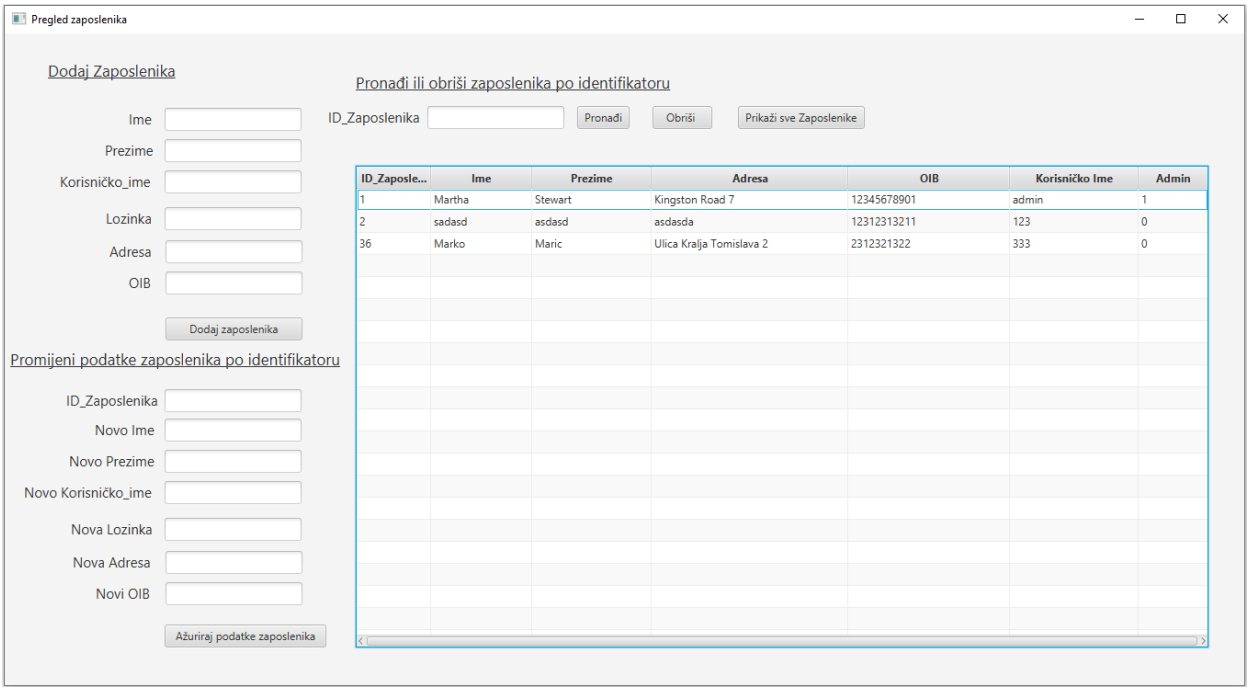
Tipka „*Uvid u zaposlenike“* otvara novi prozor „*Pregled zaposlenika“* (Slika 4.6.) koji je strukturno jako sličan prozorima za pregled članova i knjiga. Tablica zaposlenika sadrži njihov identifikator, ime, prezime, adresu, OIB, korisničko ime, te varijablu koja nam govori je li zaposlenik administrator ili ne („1“ označava da je zaposlenik administrator, dok „0“ da nije). Pristup pregledu zaposlenika imaju samo oni zaposlenici koji su administratori.

Kako bi administrator uspješno dodao novog zaposlenika potrebno je u polja za unos unijeti ime, prezime, korisničko ime, lozinku, adresu, OIB zaposlenika, te pritisnuti tipku „*Dodaj zaposlenika“*. Prvih pet polja moraju sadržavati tekst koji je između dva i dvjesto znakova, dok OIB mora biti 11 znamenki. Ukoliko ti uvjeti nisu ispunjeni u gornjem desnom kutu će se ispisati poruke slične onima u prošlim prozorima kako bi upozorile na krivi unos. Ukoliko su svi podaci uneseni kako treba, ispisati će se zelena poruka „*Zaposlenik dodan!“,* podaci će se spremiti u bazu podataka, te će se tablica automatski ažurirati. Pri unosu, kao i prije, identifikator se dodjeljuje automatski kao sljedeća slobodna vrijednost u tablici, te zadana vrijednost stupca „*admin“* iznosi „0“ što znači da zaposlenik nije administrator. Unesena korisnička imena i lozinke svih zaposlenika mogu se koristiti kako bi se prijavili u sustav.

Administrator može ažurirati podatke zaposlenika tako što u dijelu prozora predodređenom za ažuriranje unese identifikator zaposlenika čije podatke želi mijenjati, novo ime, prezime, korisničko ime, lozinku, adresu, te OIB. Verifikacija tih parametara je ista kao i u prošlim prozorima. Nakon što administrator pravilno unese sve parametre i pritisne tipku „*Ažuriraj podatke zaposlenika“* podaci se automatski ažuriraju u bazi i tablici, te se u gornjem desnom kutu prikaže zelena poruka „Zaposlenik ažuriran!“.

Kako bi administrator pronašao ili obrisao zaposlenika potrebno je u odgovarajuće polje unijeti identifikator zaposlenika koji mora biti broj. Nakon pravilnog unosa, pritiskom na tipku „*Pronađi*“ ili „*Obriši“* administrator može vidjeti sve podatke o zaposleniku ili obrisati zaposlenika s

unesenim identifikatorom, ovisno o pritisnutoj tipci. Ukoliko administrator unese identifikator zaposlenika koji je isto administrator, uključujući i samoga sebe, brisanje se neće izvršiti. Pritiskom na tipku „*Prikaži sve Zaposlenike*“ na tablici se prikažu svi zaposlenici u bazi podataka.



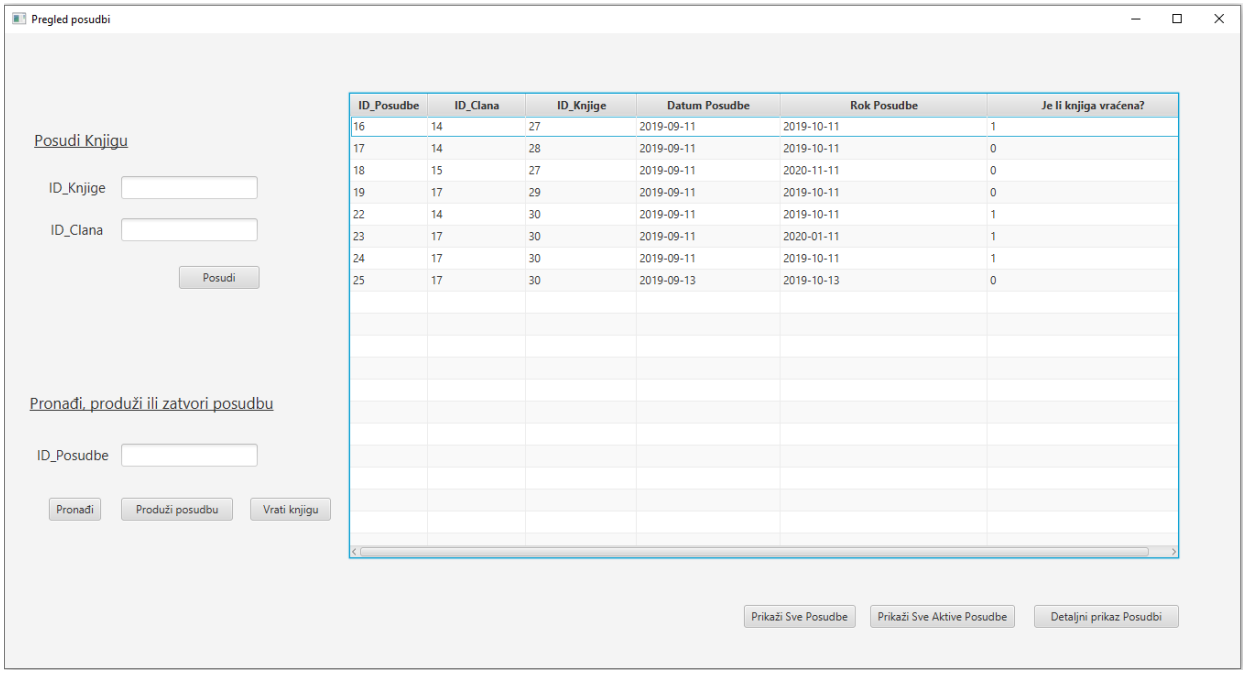
Slika 4.6. *Prozor koji se otvori nakon pritiska na tipku „Uvid u zaposlenike“.*

4.5. Uvid u posudbe

U slučaju da korisnik pritisne tipku „*Uvid u posudbe*“ otvara se novi prozor „*Pregled posudbi*“ (Slika 4.7.) koji sadrži tablicu posudbi, prostor za odrađivanje posudbe, prostor za pronalazak, produženje ili zatvaranje posudbe, te prostor koji sadrži nekoliko dodatnih opcija.

U prostoru za odrađivanje posudbe korisnika se traži unos identifikatora knjige koja se posuđuje, te identifikator člana kojem se knjiga posuđuje. Korisnik bi te podatke trebao moći pročitati sa članske iskaznice člana, te knjige. Nakon što korisnik unese identifikatore i pritisne tipku „*Posudi*“ provjerava se jesu li podaci brojevi, te ukoliko jesu, postoji li član i knjiga s takvim identifikatorima u knjižnici. Ukoliko ne postoje, u gornjem lijevom kutu aplikacije će se ispisati crvena poruka „Knjiga ili član ne postoje u knjižnici“. U slučaju da član i knjiga postoje u bazi podataka, provjerava se da li je knjiga s unesenim identifikatorom već posuđena, ukoliko je na ekran se ispisuje crvena poruka „Knjiga je već posuđena!“. U suprotnom, na ekran se ispiše zelena poruka „Knjiga posuđena“, te se u bazu podataka spremaju identifikatori člana i knjige, dodjeljuje im se jedinstveni identifikator posudbe, datum na koji je posudba izvedena, datum do kojega se

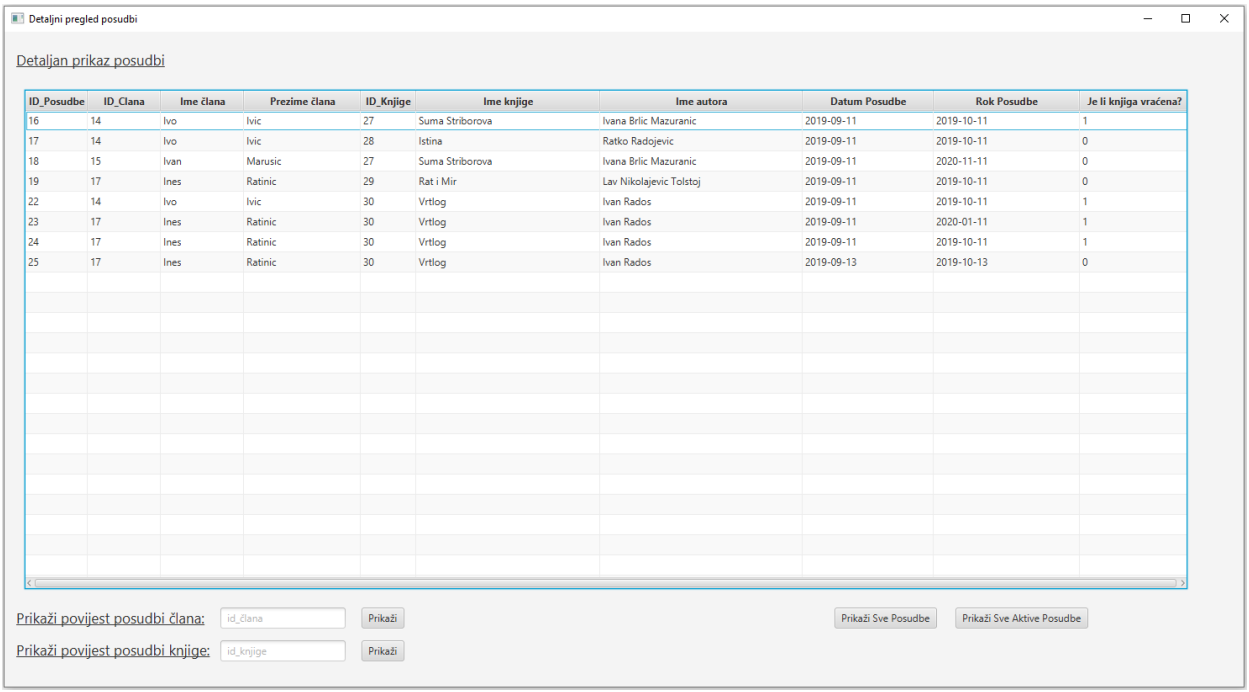
knjiga mora vratiti (mjesec dana od datuma posudbe), te varijabla koja nam označava da li je knjiga vraćena ili ne („0“ predstavlja da knjiga nije vraćena, dok „1“ predstavlja da je), koja je automatski zadana na „0“. Isto tako, u tablici „*Knjige*“ u bazi podataka se stupac koji govori je li knjiga posuđena promijeni na „1“ što označava da je knjiga posuđena.



Slika 4.7. *Prozor koji se otvori nakon pritiska na tipku „Uvid u posudbe“.*

Za pronalazak određene posudbe, njeno produživanje ili njen prekid, korisnik mora unijeti identifikator te posudbe. Taj identifikator se unosi u odgovarajuće polje u donjem lijevom kutu prozora, te se na njemu vrši ista verifikacija kao i na prijašnjim identifikatorima. Nakon unosa identifikatora i pritiska na tipku „*Pronađi*“, na tablicu posudbi se ispiše posudba s tim identifikatorom, ukoliko postoji. U slučaju da korisnik stisne tipku „*Produži posudbu*“, posudba s unesenim identifikatorom se produžuje za mjesec dana, te se promjena roka posudbe odmah ažurira na tablici. Ako korisnik pritisne tipku „*Vrati knjigu*“, red u tablici „*Posudba*“ koji sadrži uneseni identifikator postavlja vrijednost stupca koja govori je li knjiga vraćena ili ne na „1“, te se vrijednost stupca iz tablice „*Knjige*“ koji označava je li knjiga posuđena ili ne postavlja na „0“ što označava da je knjiga vraćena. Ukoliko korisnik želi vidjeti samo one posudbe kod kojih knjige nisu još vraćene, potrebno je samo kliknuti na tipku „*Prikaži Sve Aktivne Posudbe*“ u donjem desnom kutu nakon čega će se tablica posudbi automatski ažurirati i prikazivati samo one posudbe čije knjige nisu vraćene. Klikom na tipku „*Prikaži Sve Posudbe*“ korisnik može opet ažurirati tablicu tako da prikazuje sve posudbe.

Tablica posudbi sadrži samo identifikatore članova i knjiga, no ne i imena tih članova i knjiga. Kako bi korisnik imao detaljniji pogled na posudbe, on može stisnuti na tipku „*Detaljni prikaz Posudbi*“ nakon čega se otvara novi prozor „*Detaljni pregled posudbi*“ (Slika 4.8.) koji sadrži veću tablicu koja osim podataka koji se nalaze u prijašnjoj tablici još sadrži i ime knjige, ime autora knjige, te ime i prezime člana koji je posudio knjigu. U ovom prozoru korisnik može u donjem lijevom kutu unijeti identifikator određenog člana, te pritisnuti odgovarajuću tipku „*Prikaži*“ kako bi dobio prikaz svih prošlih i trenutnih posudbi toga člana. Isto tako, korisnik može unijeti i identifikator knjige kako bi dobio uvid u statistiku posudbe te knjige. Kao i na prošlom prozoru, moguće je prikazati sve posudbe kod kojih knjiga nije vraćena, te ponovno prikazati sve posudbe tako što se pritisnu tipke „*Prikaži Sve Aktivne Posudbe*“ i „*Prikaži Sve Posudbe*“.



Slika 4.8. *Prozor koji se otvori nakon pritiska na tipku „Detaljni pregled posudbi“.*

.

5. ZAKLJUČAK

Zadatak ovog završnog rada je bio napraviti GUI aplikaciju za vođenje knjižnice koja će olakšati rad zaposlenicima. Rješenje ovog zadatka pruža korisnicima jednostavno i jasno sučelje za korištenje sustava. Aplikacija im pruža razne opcije kao što su dodavanje i brisanje zaposlenika, članova i knjiga, te uvide u razne statistike posudbi i članova. Pregled tih statistika može pomoći pri planiranju daljnjih poslovnih strategija knjižnice. Pri odabiru tehnologija i jezika za razvoj aplikacije bilo je potrebno na umu imati njihovu raznovrsnost, mogućnost jednostavnog adaptiranja različitim operacijskim sustavima, te funkcionalnost koju pružaju aplikaciji. Iz tog razloga, aplikacija je napravljena u *Eclipse* razvojnom okruženju koristeći programski jezik Java. U dijelu rada gdje se opisuje razvoj aplikacije detaljno je opisano stvaranje baze podataka i njenih tablica. Isto tako, detaljno je opisan i proces dodavanja nove knjige u bazu.

Kao poboljšanje ove aplikacije moguće je dodatno automatizirati proces posudbe knjiga na način da zaposlenik ne mora unositi identifikatore knjiga i članova, već da se identifikatori iščitavaju skenerima.

LITERATURA

[1] Java *: "The Java Language Specification, 2nd Edition"* [20.9.2019.]

[2] Java konceptualni dizajn Oracle: https://www.oracle.com/technetwork/es/java/javase/tech/index.html [20.9.2019.]

[3] Greg Brown, „Introducing FXML“, 2011.

[4] Eclipse:

http://help.eclipse.org/kepler/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2Fint_eclipse.htm [20.9.2019.]

[5] Anketa iz 2014, JRebel: https://jrebel.com/resources/java-tools-and-technologies-landscape-2014/?utm_source=Java%2520Tools%2520&%2520Technologies%25202014&utm_medium=allreports&utm_campaign=rebellabs&utm_rebellabsid=88 [20.9.2019.]

[6] Structured-Query-Language, Microsoft, 2017: https://docs.microsoft.com/en-us/sql/odbc/reference/structured-query-language-sql?view=sql-server-2017 [20.9.2019.]

[7] SQL primjer SELECT naredbe, w3schools: https://www.w3schools.com/sql/sql_and_or.asp [20.9.2019.]

SAŽETAK

Title: GUI aplikacija za vođenje knjižnice

U ovom završnom radu razvijena je desktop aplikacija za olakšanje vođenja knjižnice zaposlenicima. Omogućeno im je dodavanje i brisanje članova knjižnice i knjiga. Osmišljen je i sistem posuđivanja knjiga pomoću kojega možemo jasno pratiti odstupanja ukoliko se dogodi nekakav incident. Zaposlenik može vidjeti uvid o svim posudbama, članovima knjižnice, ili, ukoliko je administrator, uvid o zaposlenicima, te njihovo dodavanje ili brisanje. Korištene tehnologije u radu su opisane u drugom poglavlju, razvoj aplikacije opisan je u trećem, dok način na koji se aplikacija koristi u četvrtom poglavlju.

Ključne riječi: *Eclipse*, *JavaFx*, knjižnica, knjiga, posudba

ABSTRACT

Title: GUI application for Library management

In this final paper, a desktop application has been developed to facilitate library management for employees. Employees can add and delete members and books of the library. A book borrowing system is also designed in a way that helps clear out deviations if some sort of incident occurs. The employee can view all borrowings, library members, or, if an employee is an administrator, employee information and can add or delete them if he wishes to do so. The technologies used in the paper are described in the second chapter. The way in which the application is used, and its development is described in the third chapter.

Key words: Book, Borrowing, *Eclipse, JavaFx,* Library

ŽIVOTOPIS

Robert Ljubić rođen je 23. studenog 1996. godine u Đakovu. U Đakovu završava Osnovnu školu Ivana Gorana Kovačića, nakon čega upisuje opću gimnaziju Antuna Gustava Matoša u Đakovu. 2016. godine upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, stručni smjer Informatika.

_____

Robert Ljubić

PRILOZI

Mapa projekta koja sadrži kôd aplikacije nalazi se na optičkom disku koji je priložen uz isprintanu verziju rada, zajedno s word i pdf dokumentima.