IZRADA WEB TRGOVINE SPECIJALIZIRANE ZA IZNAJMLJIVANJE KNJIGA

Vidović, Petar

Undergraduate thesis / Završni rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Split / Sveučilište u Splitu**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:228:559687

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-15**

Repository / Repozitorij:

Repository of University Department of Professional Studies

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Računarstvo

PETAR VIDOVIĆ

Z A V R Š N I R A D

IZRADA WEB TRGOVINE SPECIJALIZIRANE ZA IZNAJMLJIVANJE KNJIGA

Split, rujan 2023.

SVEUČILIŠTE U SPLITU
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Računarstvo

Predmet: Baze podataka

Z A V R Š N I R A D

Kandidat: Petar Vidović

Naslov rada: Izrada web trgovine specijalizirane za iznajmljivanje knjiga

Mentor: Mr. sc. Ivica Ružić, viši predavač

Split, rujan 2023.

Sadržaj

Sažetak **1**

1 Uvod **2**

2 Korištene tehnologije i alati **3**

2.1 Java . . . 3

2.2 Java Spring Boot . . . 3

2.3 Apache Maven . . . 4

2.4 Spring Security . . . 4

2.5 Thymeleaf . . . 5

2.6 Docker . . . 5

2.6.1 PostgreSQL . . . 7

2.6.2 MailHog . . . 8

2.7 Izgled korisničkog sučelja . . . 8

2.7.1 JavaScript . . . 8

2.7.2 Bootstrap . . . 9

3 Struktura baze podataka **10**

3.1 Dijagram relacijske baze podataka . . . 10

3.2 Struktura i relacije pojedinih entiteta . . . 11

4 Opis implementacije programskog rješenja **18**

4.1 Inicijalizacija Spring Boot projekta . . . 18

4.1.1 Spring Initializr . . . 18

4.1.2 Struktura Spring Boot aplikacije . . . 21

4.2 Interakcija s bazom podataka . . . 22

4.2.1 Flyway migracijske skripte . . . 22

4.2.2 Entiteti . . . 23

4.2.3 Pristup podatcima - JPA repozitoriji . . . 29

4.2.4 Hibernate ORM . . . 32

4.2.5 Punjenje podatcima - Java Faker . . . 33

4.3 Servisi i kontroleri . . . 35

4.4 Autentikacija 39
4.5 Autorizacija 41
4.6 Zakazani poslovi - Java Scheduler 43
4.7 Opis značajki specifičnih za projekt 46
4.7.1 Nadzorna ploča i statistika - Chart.js 46
4.7.2 Upravljanje korisnicima i članstvima 49
4.7.3 Administracija knjižnične građe 51
4.7.4 Rezervacija i posudba knjižnične građe 54
4.7.5 Vraćanje posuđenih knjiga i obrada zakasnine 57

5 Zaključak **59**

Literatura **60**

Dodatci **62**

Sažetak

Razvoj interneta i digitalnih tehnologija utječe na preobrazbu svih društvenih sfera, stoga je izrada web aplikacija ključna u svrhu povećanja produktivnosti poslovnih procesa i interakcije među njihovim korisnicima. U ovom završnom radu predstavljena je web aplikacija čiji je cilj omogućiti iznajmljivanje knjiga registriranim članovima. Izrađena je koristeći razvojni okvir Java Spring Boot, a korisničko sučelje je dizajnirano pomoću alata otvorenog kôda (engl. *open source*) Bootstrap. Korisnici aplikacije podijeljeni su u tri uloge: administrator, knjižničar i član koje određuju različita korisnička prava. Aplikacija pruža usluge iznajmljivanja te rezervacije knjižnične građe uz prikaz svih posudbi, knjižničkog fonda i statističkih podataka vezanih za cjelokupni proces. U nastavku završnog rada detaljno će biti opisane korištene tehnologije te način implementacije navedenih značajki.

Ključne riječi: *Baza podataka, Docker, Flyway, Java Spring Boot, Maven*

Summary

Developing a Web Shop Specialized in Book Lending

The development of the internet and digital technologies is influencing the transformation of all social spheres, thus making web application development crucial for enhancing the productivity of business processes and interaction among their users. This final paper presents a web application aimed at enabling book rentals for registered members. It was built using the Java Spring Boot framework, and the user interface was designed using the open-source Bootstrap toolkit. Application users are divided into three roles: administrator, librarian, and member, each with different user rights. The application offers services for book rental and reservation, displaying all borrowings, library inventory, and statistical data related to the entire process. The following sections of the final paper will provide a detailed description of the technologies used and the implementation process of the mentioned features.

Keywords: *Database, Docker, Flyway, Java Spring Boot, Maven*

1. Uvod

U današnjem modernom i užurbanom svijetu, vrijeme je dragocjeno. Većina populacije s naglaskom na mlade sve su više usmjereni kupovini na internetu (engl. *online shopping*) kako bi uz što manje uložene energije i vremena u bilo kojem trenutku mogli obaviti kupnju robe ili usluga. Također, veliki dio poslodavaca usmjerava se prema digitalnom tržištu s ciljem da, uz minimalna ulaganja, svoje usluge prezentiraju velikom broju korisnika interneta. Motivacija za izradu aplikacije opisane u ovom završnom radu je da se i knjižnice aktivno uključe u internetsko poslovanje te da korisnicima omoguće rezervaciju knjiga bez neophodnog dolaska. Svrha same aplikacije je da na zanimljiv način korisnicima svih dobnim skupinama omogući aktivno posuđivanje knjiga. Znajući koliko je čitanje važno i utječe na duhovni razvoj čovjeka, cilj je da razvoj tehnologije ne uskraćuje osobno kritičko razmišljanje i da služi u svrhu njegova promicanja i povećanja kreativnosti koje čitanje nudi.

Web aplikacija koja je izrađena kao praktični dio ovog završnog rada registriranim članovima omogućava rezerviranje knjižnične građe koja je trenutno dostupna u knjižničnom fondu. Korisnici imaju uvid u detalje svih svojih posudbi te na kontrolnoj ploči (engl. *dashboard*) mogu pratiti osobne i općenite statističke podatke. Korisnici s ulogom knjižničara potvrđuju rezervirane posudbe, kreiraju nove te kontroliraju vraćanje posuđenih knjiga uz moguću naplatu zakasnine ako primjerak nije vraćen u za to pretviđenom vremenu. Još jedna važna uloga knjižničara je kontinuirano praćenje statističkih podataka koji se nalaze na kontrolnoj ploči na temelju kojih će ažurirati veličinu knjižničkog fonda ili ga nadopuniti novim vrstama građe. Administratorsko sučelje omogućuje sve radnje unutar aplikacije uključujući pregled svih korisnika neovisno o ulozi. Obavještavanje članova o mogućoj zakasnini, isteku članstva i svim bitnim informacijama realizirano je putem elektroničke pošte.

Završni rad sastoji se od 5 poglavlja. Nakon uvoda u kojem je istaknuta motivacija za izradu web aplikacije i kratak opis njenih funkcionalnosti, u drugom poglavlju predstavljene su korištene tehnologije i alati. U trećem poglavlju je prikazana struktura baze podataka, opisani su odnosi među entitetima i njihovi atributi. Četvrto poglavlje detaljno prezentira implementaciju programskog rješenja same aplikacije. U zadnjem poglavlju iznesen je zaključak.

2. Korištene tehnologije i alati

2.1. Java

Java je robustan i siguran objektno-orijentirani programski jezik objavljen 1995. godine. Popularan je programski jezik, a koristi se za izradu mobilnih, web i *desktop* aplikacija. Jedna od glavnih karakteristika Jave je portabilnost. Cilj je da se jednom napisani Java kôd, nepromijenjen bez prilagodbe može pokrenuti na različitim operativnim sustavima koji imaju instaliran Java virtualni stroj (JVM - *Java virtual machine*). Kompatibilnost na različitim uređajima postignuta je tako da se izvorni kôd Java programa ne prevodi direktno u strojni, već u objektni kôd (engl. *bytecode*). Objektni kôd sastavljen je od jedne ili više datoteka s ekstenzijom `.class` koje se izvršavaju pomoću Java interpretera, sastavnog dijela Java virtualnog stroja [1].

2.2. Java Spring Boot

Java Spring Boot je razvojni okvir otvorenog kôda te je jedan od modula Spring *Frameworka*. Njegov izvorni kôd, kao i od ostalih modula Springa dostupan je te se slobodno može vidjeti na platformi GitHub [2]. Za popularnost Springa zaslužne su dvije najvažnije značajke: injektiranje ovisnosti (engl. *dependency injection*) i inverzija kontrole (engl. *inversion of control*). Injektiranje i upravljanje ovisnostima tehnika je pomoću koje se olakšava dodavanje i povezivanje objekata o kojima ovisi rad kôda te eliminira potrebu da klijent sam određuje uslugu koju će koristi. Injektiranjem ovisnosti postiže se veća modularnost aplikacije i inverzija kontrole gdje sami razvojni okvir preuzima kontrolu toka izvršavanja programskog kôda. Spring Boot modul nadograđuje razvojni okvir Spring s automatskom konfiguracijom projekta bez dodatnih ručnih podešavanja i ugrađenim poslužiteljem. Pogodan je za izgradnju mikroservisa, web, samostalnih (engl. *standalone*) i proizvodno spremnih (engl. *production-grade*) aplikacija. Osnovni cilj Spring Boota je da programerima uz što manje uloženog truda i vremena omogući konfiguriranje aplikacije spremne za daljnji razvoj. Spring Boot aplikacija ima slojevitu arhitekturu u kojoj svaki sloj neposredno komunicira sa slojem ispod ili iznad njega po hijerarhijskoj strukturi [3]. U poglavljima koja slijede detaljno je prikazan opis arhitekture aplikacije. Ulazna točka za pokretanje Spring Boot aplikacije je klasa koja sadrži anotaciju `@SpringBootApplication` i glavnu metodu prikazanu u ispisu 1.

```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableScheduling;

@SpringBootApplication
@EnableScheduling
public class LibraryManagementSystemApplication {
    public static void main(String[] args) {
        SpringApplication.run(LibraryManagementSystemApplication.class,
    args);
    }
}
```

Ispis 1: Prikaz metode za pokretanje Spring Boot aplikacije

2.3. Apache Maven

Apache Maven je razvojni alat otvorenog kôda za izgradnju i upravljanje aplikacijama primarno napisanih u Java programskom jeziku. Baziran je na primjeni objektnog modela projekta (POM - Project Object Model) u kojem su definirane sve ovisnosti potrebne za upravljanje projektom. Datoteka `pom.xml` predstavlja jezgru (engl. core) konfiguracije Maven projekta. Maven osigurava veliko skladište korištenih biblioteka, jednostavno postavljanje projekta, automatsko ažuriranje ovisnosti i kompatibilnost s prethodnim verzijama. Glavna svrha mu je upravljanje ovisnostima, a najpoznatija vrsta upravljanja paketima je JAR - *Java Archive* koja je ujedno korištena i u ovome radu [4].

2.4. Spring Security

Spring Security kao i već spomenuti Spring Boot, također je modul razvojnog okvira Spring. Prilagodljiv je i moćan te je standard za osiguranje aplikacija baziranih na Springu. Fokus Spring Securitya je omogućiti autentikaciju i autorizaciju Java aplikacijama. Osim toga pruža zaštitu od raznih napada poput fiksiranja sesije (engl. *session fixation attack*) i krivotvorenja zahtjeva na više web stranica (engl. *CSRF attack*). Lako je proširiv i odgovara na prilagođene korisničke zahtjeve [5].

Korištenje Spring Securitya za osiguranje aplikacije bit će prikazano u poglavljima 4.4 i 4.5. Prezentiran je detaljni opis konfiguracije i korištenja mnogih klasa dobivenih nakon uključivanja Spring Security ovisnosti u projekt.

2.5. Thymeleaf

Thymeleaf je moderni Java mehanizam na poslužiteljskoj strani za stvaranje i obradu HTML - *HyperText Markup Language* datoteka. Thymeleaf donosi mogućnost izrade elegantnih predložaka dodavajući atribute standardnim HTML oznakama. Obavlja transofrmacije XHTML/HTML5 datoteka, s ciljem prikazivanja podataka generiranih u web aplikaciji te je zaslužan za dinamički sadržaj prikazan na korisničkom sučelju. Omogućava izravan pristup Java i Spring objektima inicijaliziranim kroz Spring spremnik (engl. *Spring beans*). Pogodan je za integraciju s modulima Springa, a u aplikaciju se dodaje kao ovisnost [6]. Ako se koristi uz Spring Security, važna značajka Thymeleafa je pristup vjerodajnicama trenutno prijavljenog korisnika unutar predloška [7]. Također, ima mogućnost prikazivnaja različitog sadržaja ovisno o korisničkoj ulozi. Pruža uslugu generiranja teksta, linkova, uvjetnih naredbi i brojnih drugih funkcionalnosti.

2.6. Docker

Docker je najpopularnija platforma koja olakšava razvoj i pokretanje aplikacija koristeći spremnike (engl. *containers*). Nastao je 2014. godine, a temelji se na uporabi Linux spremnika. Spremnici služe za pakiranje aplikacija uz pripadne biblioteke i komponente o kojima aplikacija ovisi. Pohrana aplikacija u spremnike osigurava im prijenos i pokretanje neovisno o vrsti operativnog sustava i infrastrukture stroja na kojem se pokreću. Fleksibilnost, efikasnost korištenja resursa i smanjenje potrebnog prostora za pohranu su glavne prednosti uporabe spremnika [8].

Za korištenje navedenih mogućnosti, na lokalnom uređaju potrebno je imati instaliran Docker Engine pomoću kojeg se stvaraju spremnici i pokreću aplikacije pohranjene u njima. Spremnici se konstruiraju na temelju Docker slika (engl. *image*). Slika je datotečni sustav koji služi za čitanje uputa na temelju kojih se spremnik kreira. Korisnici mogu koristiti javne ili izrađivati vlastite Docker slike. `Dockerfile` je datoteka unutar aplikacije, koja sadrži skriptu s prethodno navedenim uputama iz kojih se generira instanca slike odnosno spremnik.

Aplikacija može imati više komponenti ili servisa koji se također pohranjuju u spremnike da bi se uspješno pokretali. Pomoću Docker Compose ugrađenog alata svi aplikacijski servisi se pakiraju u spremnike. Datoteka `compose.yaml` sadrži konfiguraciju pomoću koje se svi dodatni servisi pokreću zajedno s aplikacijom. Osim toga unutar navedene datoteke upravlja se stvaranjem Docker volumena (engl. *volumes*), s ulogom čuvanja podataka koje spremnici koriste. Terminalna naredba `docker-compose up` služi za stvaranje Docker spremnika i pokretanje aplikacije, dok se naredbom `docker-compose down` zaustavljaju svi servisi, a spremnici brišu. Ispis 2 sadrži prikaz aplikacijskih servisa koji se pokreću pomoću Docker spremnika [9].

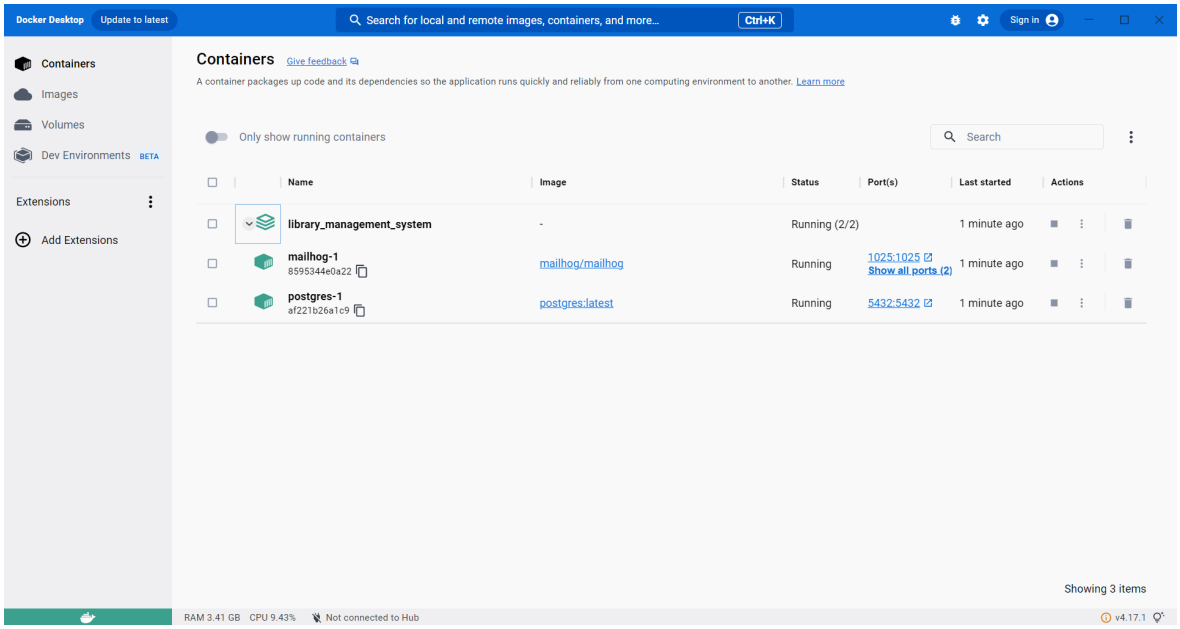
```
services:
  web-app:
    build: .
    image: 'library-management-system.jar:latest'
    ports:
      - '8080:8080'
    depends_on:
      postgres:
        condition: service_healthy
    environment:
      - SPRING_DATASOURCE_URL=jdbc:postgresql://postgres:5432/postgres
      - SPRING_DATASOURCE_DB=postgres
      - SPRING_DATASOURCE_USERNAME=postgres
      - SPRING_DATASOURCE_PASSWORD=postgres

  postgres:
    image: 'postgres:latest'
    environment:
      - 'POSTGRES_DB=postgres'
      - 'POSTGRES_PASSWORD=postgres'
      - 'POSTGRES_USER=postgres'
    ports:
      - '5432:5432'
    volumes:
      - ./postgres-data:/var/lib/postgresql/data
    healthcheck:
      test: [ "CMD-SHELL", "pg_isready -U postgres" ]
```

```
28        interval: 10s
29        timeout: 5s
30        retries: 5
31
32    mailhog:
33      image: mailhog/mailhog
34      ports:
35        - '1025:1025' # smtp server
36        - '8025:8025' # web ui
```

Ispis 2: Konfiguracija servisa u `compose.yaml` datoteci

Na slici 1 prikazani su pokrenuti Docker spremnici prilikom rada aplikacije.



Slika 1: Aktivni Docker spremnici

U slijedećim potpoglavljima kratko su predstavljene tehnologije konfigurirane kao servisi u datoteci `compose.yaml`, ključni za pravilno pokretanje i izvođenje aplikacije.

2.6.1. PostgreSQL

PostgreSQL je objektno-relacijski sustav otvorenog kôda za upravljanje bazama podataka (engl. RDBMS - *Relational Database Management System*). Koristi i proširuje SQL jezik (engl. *Structured Query Language*), najpopularniji jezik korišten za pohranu i obradu

informacija u relacijskoj bazi podataka. Značajke PostgreSQL-a sigurno pohranjuju i skaliraju najsloženija radna opterećenja podataka. Stekao je snažnu reputaciju svojom pouzdanošću, integritetom podataka, proširivošću i robusnim skupom funkcija. Nije u vlasništvu niti jedne kompanije, već njime upravlja aktivna globalna zajednica programera, entuzijasta i drugih volonteračitePostgreSQL.

Korištenje PostgreSQL baze podataka u Spring Boot aplikaciji omogućeno je uključivanjem dvije ovisnosti unutar `pom.xml` datoteke. Spring Boot Starter Data JPA ovisnost zaslužna je za uspješno povezivanje Spring Boot projekta i relacijske baze podataka, dok PostgreSQL JDBC Driver regulira komunikaciju PostgreSQL baze sa aplikacijom. Osnovna konfiguracija nalazi se u datoteci `application.properties` sa svrhom podešavanja različitih svojstava, kao što su port poslužitelja, lozinka, i drugi detalji veze s bazom podataka.

2.6.2. MailHog

MailHog je alat otvorenog kôda za testiranje usluga elektroničke pošte s lažnim SMTP poslužiteljem. Kratica SMTP (engl. *Simple Mail Transfer Protocol*) predstavlja uobičajeni način za siguran i pouzdan prijenos elektroničke pošte na internetu. MailHog svu poslanu elektroničku poštu pohranjuje i prikazuje na web korisničkom sučelju. Podržava rad u Docker okruženju bez ikakve potrebne instalacije i korisnicima uvelike olakšava razvoj aplikacija te reducira moguće probleme sa SMTP poslužiteljem. Predefinirano (engl. *default*) SMTP poslužitelj se pokreće na priključku 1025 (engl. *port*), web poslužitelj zaslužan za prikaz pretinca elektroničke pošte na priključku 8025, a podatci iz pretinca pohranjeni su u glavnoj memoriji (engl. *in-memory database*) te se gube prestankom rada aplikacije. Spring Boot mail pokretač nalazi se u skupu ovisnosti unutar `pom.xml` datoteke [10].

2.7. Izgled korisničkog sučelja

2.7.1. JavaScript

JavaScript je skriptni programski jezik koji omogućuje implementaciju složenih značajki na web stranicama. Koristi se za izradu dinamičnih stranica, upravljanje multimedijom i animacijama. jQuery jedna je od najkorištenijih JavaScript biblioteka. Bogata je značajkama, a koristi se za manipulaciju HTML dokumenata, rukovanje događajima i izradu

animacija. Uvelike je promijenila i olakšala način pisanja JavaScript kôda. Select2 je mala JavaScript biblioteka, osmišljena s namjerom pojednostavljanja HTML `select` elementa. Nudi mogućnost pretrage podataka dostupnih u izborniku vidljivom na korisničkom sučelju. Poziva se unutar jQuery funkcije selektiranjem elementa na kojeg se želi primijeniti ova značajka [11].

2.7.2. Bootstrap

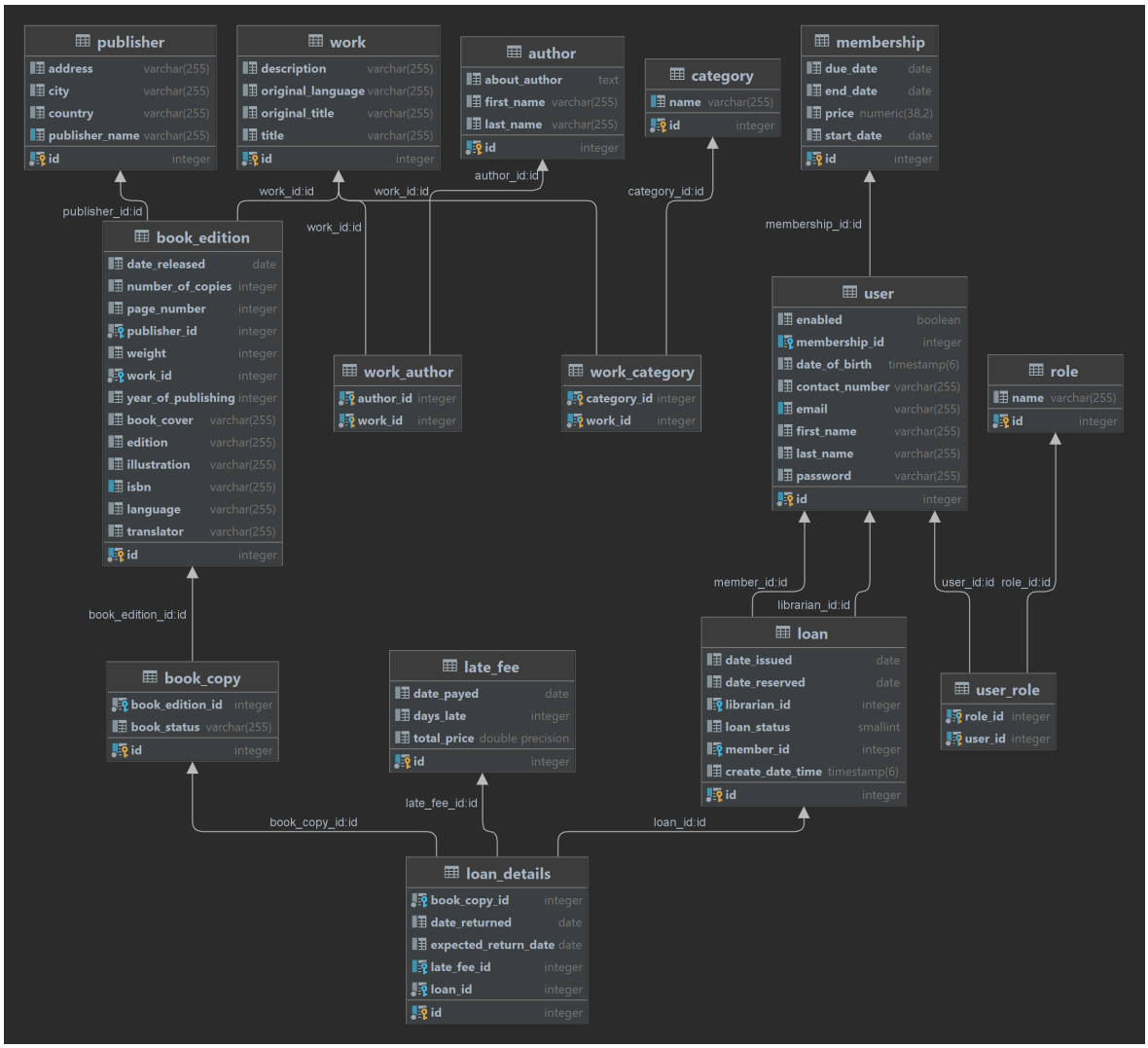
Bootstrap je najpopularniji CSS (engl. *Cascading Style Sheets*) razvojni okvir. Sastoji se od niza alata i komponenti za brz i jednostavan razvoj responzivnih web stranica. Sadrži predloške dizajna temeljene na CSS-u, a koristi se za izradu modernih, ponajprije mobilno prilagodljivih web stranica [12].

3. Struktura baze podataka

Svaka baza podataka vjerno predstavlja presliku stvarnih procesa iz okoline. Sastoji se od međusobno povezanih podataka čije vrijednosti opisuju dio realnog svijeta. Pravilan dizajn baze podataka izuzetno je važan te je uloženo vrijeme u njenu konstrukciju bitan dio u izradi aplikacije. Kvalitetno osmišljena baza podataka štiti integritet, osigurava ažuran pristup i manipulaciju točnim podatcima s jednostavnom mogućnosti prilagodbe na potencijalne promjene.

3.1. Dijagram relacijske baze podataka

Na slici 2 prikazana je shema relacijske baze podataka (engl. *database scheme*). Prikazuje stvarnu konstrukciju baze podataka. Dijagram omogućava grafički prikaz entiteta, atributa koji ih opisuju i međusobnih relacija među entitetima. Automatski je generirana u sklopu integriranog razvojnog okruženja IntelliJ IDEA pogodnog za izradu aplikacija napisanih u programskim jezicima utemeljenim na JVM-u. Radi preglednosti na dijagramu sa slike 2 izostavljena je tablica `flyway_schema_history` zaslužna za praćenje flyway migracija. Detaljni opis migracija prikazan je u poglavlju 4.2.1.



Slika 2: Shema baze podataka

3.2. Struktura i relacije pojedinih entiteta

U tablicama 1 - 12 prikazana je struktura pojedinih entiteta uz opis njihovih relacija. Unutar tablica prikazani su svi atributi koji opisuju određeni entitet. Pojedini atribut definiran je svojim nazivom, tipom, veličinom podatka, ključem i kardinalitetom. Primarni ključ je minimalni skup atributa koji jedinstveno identificira element entiteta, dok je strani ključ atribut koji se odnosi na primarni ključ druge tablice s ciljem uspostavljanja veze između te dvije tablice. Polje kardinalnosti sadrži uređeni par brojeva koji govori koliko vrijednosti pojedini atribut koristi za opis odabranog elementa entiteta. Prvi broj određuje je li atribut opcionalan (engl. *min card*), **0** označava neobavezan, a **1** obavezan unos. Drugi broj unutar para predstavlja maksimalni broj vrijednosti atributa (engl. *max card*) [13]. Međutablice `user_role`, `work_author` i `work_category` s kompozitnim primarnim ključem bez

dodatnih polja kod veza više-na-više, prikazane na dijagramu 2 nisu tablično opisane. Navedene tablice tj. relacije objašnjene su u sklopu opisa entiteta `user` i `work`.

Tablica 1: Entitet `user`

Naziv atributa	Tip podatka	Primarni ključ	Strani ključ	Kardinalnost
`id`	integer	DA		(1,1)
`first_name`	varchar(50)			(1,1)
`last_name`	varchar(50)			(1,1)
`date_of_birth`	timestamp(6)			(1,1)
`contact_number`	varchar(255)			(1,1)
`email`	varchar(255)			(1,1)
`password`	varchar(255)			(1,1)
`enabled`	boolean			(1,1)
`membership_id`	integer		DA	(0,1)

Entitet *user* opisuje sve aplikacijske korisnike. Svaki korisnik može imati više uloga tj. `rola`, a svaka uloga može biti dodjeljena većem broju korisnika, zato su entiteti `user` i `role` povezani vezom **više-na-više**. Kardinalitet za oba entiteta koji sudjeluju u relaciji više-na-više je (0,n). Međutablica `user_role` koristi se za pohranu viševrijednosnih atributa, jer unos više vrijednosti unutar jednog polja, stupca tablice nije moguć. Detaljni opis korisničkih ovlasti ovisno o vrsti uloge, predstavljen je u poglavlju 4.5. Strani ključ `membership_id` povezuje korisnika s članstvom tj. tablicom `membership`. Vezani su relacijom **jedan-na-jedan** koja omogućava da svaki korisnik ima jedno članstvo i obrnuto. Polje `membership_id` može poprimiti *null* vrijednost, ako korisnik nije član, odnosno nema ulogu `membera`. Kardinalitet entiteta povezanih relacijom jedan-na-jedan je (0,1).

Tablica 2: Entitet `role`

Naziv atributa	Tip podatka	Primarni ključ	Strani ključ	Kardinalnost
`id`	integer	DA		(1,1)
`name`	varchar(255)			(1,1)

Entitet `role` predstavlja jednostavnu tablicu sa svrhom upravljanja korisničkim ovlastima. Različiti korisnici mogu imati istu ulogu koja je odgovorna za pravo pristupa određenim putanjama i aplikacijskim resursima. Tri aplikacijske uloge su `Member`, `Librarian`

i Administrator s mogućnosti dodavanja novih uloga, ako proširenje poslovnih procesa to bude zahtijevalo.

Tablica 3: Entitet membership

Naziv atributa	Tip podatka	Primarni ključ	Strani ključ	Kardinalnost
id	integer	DA		(1,1)
start_date	date			(1,1)
due_date	date			(1,1)
end_date	date			(0,1)
price	numeric(38,2)			(1,1)

Tablicom 3 opisan je entitet membership. Jedan element entiteta zaslužan je za praćenje podataka o trajanju i cijeni članstva određenog člana.

Tablica 4: Entitet loan

Naziv atributa	Tip podatka	Primarni ključ	Strani ključ	Kardinalnost
id	integer	DA		(1,1)
member_id	integer		DA	(1,1)
librarian_id	integer		DA	(1,1)
date_reserved	date			(0,1)
date_issued	date			(0,1)
loan_status	smallint			(1,1)
create_date_time	timestamp(6)			(1,1)

Entitet loan opisan u tablici 4 prikazuje i skladišti podatke svih posudbi te predstavlja središte aplikacijske poslovne logike. Sadrži dva strana ključa, member_id i librarian_id. Oba pružaju vezu **više-na-jedan** prema tablici user omogućavajući jednom knjižničaru ili članu veći broj posudbi. Atribut loan_status je tipa **enum** te vrijednost polja poprima iz konačnog skupa unaprijed definiranih konstanti. Prema tome status posudbe može biti: IN_PROGRESS, RESERVED, IN_LOAN, CANCELLED, ili DONE. Ovisno o atributu loan_status razlikujemo rezervirane, aktivne, završene i otkazane posudbe te posudbe u stvaranju. Stvaranje posudbe omogućeno je dodavanjem knjiga u košaricu koja predstavlja posudbu statusa IN_PROGRESS. Polje create_date_time pohranjuje vremenski zapis kreiranja posudbe. Navedeni atribut korišten je prilikom otkazivanja koša-

rice otvorene dulje od 30 minuta s ciljem oslobađanja knjiga kako bi bile dostupne ostalim članovima. Otkazivanje je realizirano pomoću izvršavanja planiranih zadatka u određenom vremenskom intervalu (engl. *Scheduled Tasks*). Zakazani poslovi detaljno su predstavljeni u poglavlju 4.6

Tablica 5: Entitet `loan_details`

Naziv atributa	Tip podatka	Primarni ključ	Strani ključ	Kardinalnost
`id`	integer	DA		(1,1)
`loan_id`	integer		DA	(1,1)
`book_copy_id`	integer		DA	(1,1)
`late_fee_id`	integer		DA	(0,1)
`expected_date_returned`	date			(1,1)
`date_returned`	date			(0,1)

Struktura entiteta `loan_details` prikazana je u tablici 27. Pomoću veze više-na-jedan entitet je povezan s tablicama `loan` i `book_copy`. Navedene relacije omogućavaju posuđivanje više primjeraka knjižnične građe na jednoj posudbi te njihovo vraćanje u različitom vremenu. Kada se primjerak vrati automatski je dostupan drugim korisnicima. Ako su sve knjige s određene posudbe vraćene ona je završena i status joj je postavljen na `DONE`. U slučaju zakašnjelog vraćanja primjerka obračunava se zakasnina. Stranim ključem `late_fee_id` osigurana je veza jedan-na-jedan prema tablici `late_fee` koja nudi detalje o datumu plaćanja i cijeni naknade u slučaju zakašnjelog vraćanja knjige iz pripadajuće posudbe.

Tablica 6: Entitet `late_fee`

Naziv atributa	Tip podatka	Primarni ključ	Strani ključ	Kardinalnost
`id`	integer	DA		(1,1)
`days_late`	integer			(1,1)
`date_payed`	date			(0,1)
`total_price`	double precision			(1,1)

Tablica 7: Entitet `book_copy`

Naziv atributa	Tip podatka	Primarni ključ	Strani ključ	Kardinalnost
`id`	integer	DA		(1,1)
`book_edition_id`	integer		DA	(1,1)
`book_status`	varchar(255)			(1,1)

Entitet `book_copy` predstavlja primjerak knjižnične građe te sadrži dva atributa od kojih je jedan strani ključ `book_edition_id` koji pruža vezu više-na-jedan s tablicom `book_edition`. Na temelju navedene relacije omogućeno je da izdanje knjižnične građe može imati više primjeraka dok pojedini primjerak isključivo pripada jednom izdanju knjige. Polje `book_status` nudi podatke o stanju primjerka knjige, a moguće vrijednosti ovog enum atributa su `OK`, `LOST` ili `DAMAGED`. Samo primjerci s vrijednosću statusa `OK` su dostupni za iznajmljivanje.

Tablica 8: Entitet `book_edition`

Naziv atributa	Tip podatka	Primarni ključ	Strani ključ	Kardinalnost
`id`	integer	DA		(1,1)
`date_released`	date			(0,1)
`number_of_copies`	integer			(0,1)
`page_number`	integer			(1,1)
`publisher_id`	integer		DA	(1,1)
`weight`	integer			(0,1)
`work_id`	integer		DA	(1,1)
`year_of_publishing`	integer			(1,1)
`book_cover`	varchar(255)			(1,1)
`edition`	varchar(255)			(1,1)
`illustration`	varchar(255)			(0,1)
`isbn`	varchar(255)			(1,1)
`language`	varchar(255)			(1,1)
`translator`	varchar(255)			(0,1)

Entitet `book_edition` strukturno je prikazan pomoću tablice 8, uz detaljan prikaz svih potrebnih atributa za njegovo opisivanje. Među atributima ističu se dva strana ključa,

work_id i publisher_id pomoću kojih je vezama više-na-jedan entitet book_edition povezan s entitetima work i publisher. Na taj način je realizirana preslika stvarnog procesa iz poslovne okoline gdje određeno književno djelo ima više različitih izdanja, a pojedino izdanje samo jednog izdavača koji može tiskati više različitih edicija i književnih djela.

Tablica 9: Entitet publisher

Naziv atributa	Tip podatka	Primarni ključ	Strani ključ	Kardinalnost
id	integer	DA		(1,1)
publisher_name	varchar(255)			(1,1)
country	varchar(255)			(1,1)
city	varchar(255)			(1,1)
address	varchar(255)			(0,1)

Tablica 10: Entitet work

Naziv atributa	Tip podatka	Primarni ključ	Strani ključ	Kardinalnost
id	integer	DA		(1,1)
title	varchar(255)			(1,1)
original_title	varchar(255)			(1,1)
original_language	varchar(255)			(1,1)
description	varchar(255)			(0,1)

Tablicom 10 prikazana je shema entiteta work koji predstavlja književno djelo definirano svojim nazivom, opisom i jezikom na kojem je napisano. Entitet je povezan s dvije jednostavne tablice author i category relacijama više-na-više. Implementacijom spomenutih relacija omogućeno je da djelo može imati više autora te da se ubraja u više književnih žanrova. Nastavno tome pojedini autor može napisati više knjiga dok različita književna djela unutar knjižničnog fonda mogu pripadati istoj književnoj vrsti. Veze entiteta work realizirane su korištenjem međutablica work_author i work_category. Zbog svoje jednostavnosti entiteti work i category nisu detaljnije objašnjeni, već samo strukturno prikazani pomoću tablica 11 i 12.

Tablica 11: Entitet `author`

Naziv atributa	Tip podatka	Primarni ključ	Strani ključ	Kardinalnost
`id`	integer	DA		(1,1)
`first_name`	varchar(50)			(1,1)
`last_name`	varchar(50)			(0,1)
`about_author`	text			(0,1)

Tablica 12: Entitet `category`

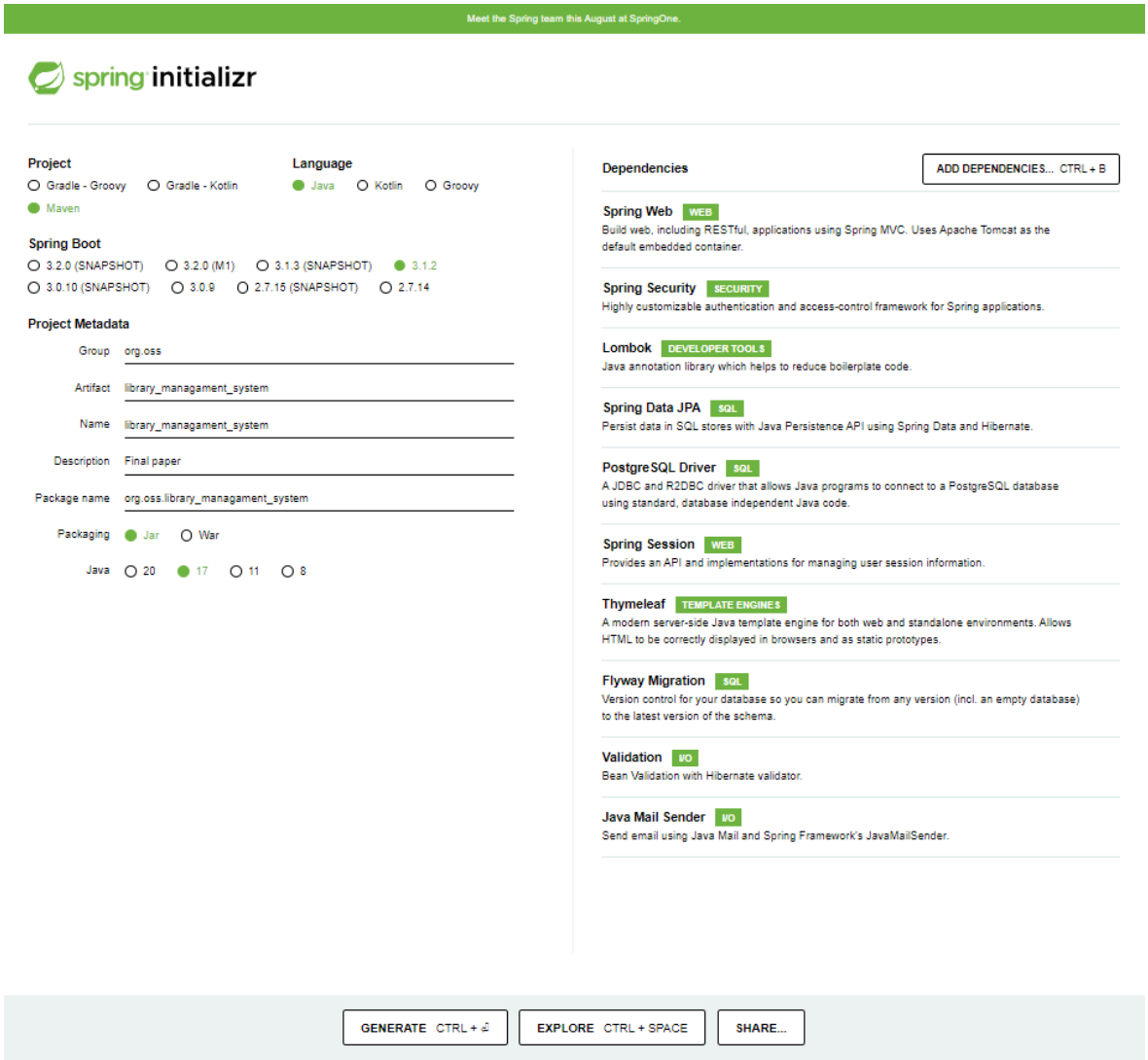
Naziv atributa	Tip podatka	Primarni ključ	Strani ključ	Kardinalnost
`id`	integer	DA		(1,1)
`name`	varchar(255)			(1,1)

4. Opis implementacije programskog rješenja

4.1. Inicijalizacija Spring Boot projekta

Kreiranje i postavljanje novog projekta predstavlja izazov programerima prilikom razvoja aplikacija. Glavna značajka Spring Boota je da veliki dio konfiguracije obavlja automatski te olakšava cijeli proces početnog oblikovanja projekta. **Spring Boot Initializr** predstavlja web aplikaciju koja služi za inicijalizaciju Spring Boot projekta bez generiranja dodatnog aplikacijskog koda. Omogućava dodavanje ovisnosti i paketa unutar aplikacije te pruža osnovnu strukturu **Maven** ili **Gradle** projekta.

4.1.1. Spring Initializr



Slika 3: Spring Initializr web sučelje

Za konfiguriranje web aplikacije opisane u ovom završnom radu, korišteno je Spring Initializr web korisničkog sučelje dostupno na poveznici `https://start.spring.io`. Prilikom podešavanja aplikacije, odabiru se vrsta projekta, programski jezik, verzija Spring Boota, metapodatci o projektu te se dodaju potrebne ovisnosti.

Na slici 3 prikazana je početna stranica alata Spring Initializr s odabranim vrijednostima i ovisnostima potrebnim za pravilno konfiguriranje aplikacije. Nakon postavljanja svih potrebnih podataka i dodavanja paketa koji će se koristiti u izradi aplikacije, projekt se generira i automatski preuzima klikom na botun *GENERATE* koji se nalazi u dnu slike 3. Odabrani programski jezik je Java, tip projekta *Maven*, a Spring Boot verzija je 3.1.2 koja je zadnja stabilna verzija u trenutku inicijalizacije. Dodatno slijedi unos metapodataka koji pružaju pojedinosti o svim podržanim svojstvima konfiguracije. Unosom grupe, artefakta i imena kreira se ime projekta tj. *package name*. Sukladno konvenciji nazivi paketa napisani su malim slovima kako bi se izbjegao konflikt s imenovanjem Java klasa i sučelja (engl. *interface*). U konačnici naziv paketa odgovara obrnutom nazivu internetske domene, a za potrebe završnog rada naziv paketa je `org.oss.library_management_system`. Pakiranje pomaže u upravljanju aplikacijom te njenim modulima, klasama i sučeljima.

Nakon unosa podataka bitnih za upravljanje paketima i cjelokupni rad projekta, dodaju se ovisnosti klikom na botun *ADD DEPENDENCIES*. Dodane ovisnosti pohranjene su u sekciji `<dependencies>` unutar `pom.xml` datoteke. Predstavljena web aplikacija koristi veći broj ovisnosti čije su uloge ukratko opisane.

- `spring-boot-starter-web` - Ovisnost koja automatski konfigurira Spring Boot projekt s ciljem izgradnje web aplikacije koristeći Spring MVC (engl. *Model-View-Controller*). Također osigurava ugrađeni Tomcat server.
- `spring-boot-starter-security` - Uključuje korištenje razvojnog okvira Spring Security. Osnovna svrha ove ovisnosti je osigurati sigurnost web aplikacije te pružiti autentikaciju i autorizaciju.
- `spring-boot-starter-data-jpa` - Ovisnost koja pruža usluge razvojnog okvira Spring Data JPA (engl. *Java Persistence API*). JPA je zadužen za pohranu podataka u relacijsku bazu podataka te je standardna specifikacija ORM-a (engl. *Object Relational Mapping*) u Javi. Detaljniji opis alata korištenih za interakciju s bazom

podataka predstavljen je u poglavlju 4.2.

- `spring-boot-starter-thymeleaf` - Osigurava aplikaciji korištenje usluga alata za obradu HTML datoteka Thymleaf.
- `postgresql` - Aplikaciji omogućava povezivanje i interakciju s PostgreSQL bazom podataka, uz osnovne funkcionalnosti utvrđivanja veze, slanja upita i dohvaćanja podataka iz baze podataka.
- `lombok` - Spring Boot ovisnost, koja pomaže pri smanjenju ponavljanja istih šablonskih djelova kôda (engl. *boilerplate code*), pomoću Spring Boot anotacija. Lombok se najčešće koristi pri generiranju *get* i *set* metoda te konstruktora pomoću kojih se stvaraju objekti tj. instance pojedinih entiteta.
- `flyway-core` - Ovisnost `flyway-core` omogućava korištenje alata otvorenog kôda Flyway. Razvojni okvir Flyway koristi se za ažuriranje verzije baze podataka pomoću migracijskih alata. Flayway migracijske skripte će biti posebno predstavljene u poglavlju 4.2.1.
- `javafaker` - Ovisnost koja pruža korištenje JavaFaker biblioteke, a olakšava popunjavanje baze podataka slučajno generiranim podatcima po određenim pravilima. Spomenuta biblioteka detaljnije je predstavljena u poglavlju 4.2.5.
- `spring-boot-docker-compose` - Pruža uslugu pokretanja servisa Spring Boot projekta unutar Docker spremnika.
- `spring-boot-starter-validation` - Omogućava validaciju podataka korištenjem Spring anotacija nad podatkovnim članovima modela. Osigurava i štiti integritet podataka. Kada Spring Boot pronađe argument označen s anotacijom `@Valid`, automatski pokreće zadanu implementaciju **Hibernate Validatora** i potvrđuje argument. Opisana ovisnost osigurava i štiti integritet podataka. Korisničke zahtjeve koji narušavaju suvislost podataka, sustav za upravljanje bazom podataka prepoznaje i odbacuje.
- `spring-boot-starter-mail` - Omogućava slanje elektroničke pošte unutar razvojnog okvira Spring Boot korištenjem različitih klasa i sučelja.

Nakon generiranja projekta, ovisnostima je moguće ručno manipulirati unutar `pom.xml`

datoteke. Ukoliko razvoj aplikacije iziskuje potrebu za dodavanjem novih ovisnosti, one se mogu naknadno nadodati unutar datoteke `pom.xml`. Dodavanje se vrši upisivanjem grupe i artefakta pojedine ovisnost unutar skupa svih ovisnosti. Web stranica *Maven repository* sadrži pohranjene ovisnosti te se podatci potrebni za upis unutar `pom.xml` datoteke mogu pronaći *online* na poveznici `https://mvnrepository.com`.

4.1.2. Struktura Spring Boot aplikacije

Nakon inicijalizacije Spring Boot projekta, aplikacija je spremna za daljnji razvoj. Osnovna struktura neposredno generiranog *Spring Boot Maven* projekta pomoću alata Spring Initializr jest sljedeća:

```
Library_Managament_System
├── src
│   ├── main
│   │   ├── java
│   │   └── resources
│   └── test
│       ├── java
│       └── resources
├── target
├── pom.xml
├── Dockerfile
└── compose.yaml
```

Direktorij `src/main` je najvažniji direktorij unutar Maven projekta te je u njemu sadržan sav izvorni kôd. Unutar `main` direktorija nalaze se Java klase, sučelja, pripadajući metapodatci i resursi koji se pakiraju unutar JAR datoteka. Sastoji se od dva poddirektorija `src/main/java` i `src/main/resources`. Unutar mape `java` nalaze se klase, Spring objekti i sav programski kôd korišten za izradu logike web aplikacije, dok direktorij `resources` sadrži statičke datoteke, slike, HTML predloške, migracijske skripe i konfiguracijske datoteke (`application.properties`). U trenutku kada se projekt *builda* sve klase, paketi i JAR datoteke smještaju se u `target` direktorij.

4.2. Interakcija s bazom podataka

4.2.1. Flyway migracijske skripte

Flyway je razvojni okvir za verzioniranje baze podataka i primjenjivanje željenih promjena na aplikacijsku bazu korištenjem migracija.

Potrebno je dodati Flyway kao ovisnost u `pom.xml` kao što je prikazano u ispisu 3.

```
<dependency>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-core</artifactId>
    <version>9.10.2</version>
</dependency>
```

Ispis 3: Dodavanje Flyway ovisnosti u `pom.xml`

Kako bi se na najlakši način Flyway integrirao s PostgreSQL bazom (također prethodno dodanoj kao ovisnost) i jednostavno konfigurirao instaliran je i korišten Flyway Maven Plugin kao što je prikazano u ispisu 4.

```
<plugin>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-maven-plugin</artifactId>
    <version>9.10.2</version>
    <configuration>
        <url>jdbc:postgresql://localhost:5432/postgres</url>
        <user>postgres</user>
        <password>postgres</password>
    </configuration>
    <dependencies>
        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <version>42.5.1</version>
            <scope>runtime</scope>
        </dependency>
    </dependencies>
```

```
18 </plugin>
```

Ispis 4: Dodavanje i konfiguracija Flyway Maven Plugina u `pom.xml`

Migracijske skripte nalaze se u direktoriju `resources/db/migration`. Flyway prati sljedeću konvenciju za nazivanje verzioniranih migracijskih skripti: `<Prefix><Version><Separator><Description><Suffix>`, pri čemu su:

- `<Prefix>` - Zadani prefiks za verzionirane migracije jest `V`, ali može se postaviti drugačije mijenjanjem Flyway postavki.
- `<Version>` - Broj verzije migracije. Brojevi *minor*, *major* ili *patch* verzija mogu se odvojiti donjim crticama (_) ili točkama (.).
- `<Separator>` - Zadano su separator dvije donje crtice tj. __.
- `<Description>` - Kratki tekstualni opis migracije pri čemu se riječi odvajaju donjim crticama (_) ili razmacima.
- `<Suffix>` - Zadano postavljen kao `.sql`.

U ispisu 5 prikazan je dio kôda migracije `V1__Create_author_table.sql`.

```
1 CREATE TABLE IF NOT EXISTS author (
2     id SERIAL NOT NULL PRIMARY KEY,
3     first_name VARCHAR NOT NULL,
4     last_name VARCHAR,
5     about_author TEXT
6 );
```

Ispis 5: Dio kôda migracije `V1__Create_author_table.sql`

Flyway pokreće migracije koje će se primijeniti na shemu baze naredbom `.\mvnw flyway:migrate`.

4.2.2. Entiteti

Entitet u JPA predstavlja **POJO** (engl. *Plain Old Java Object*) sa podatcima koje je moguće spremiti u bazu podataka. POJO je obični Java objekt bez specifičnih ograničenja

te nema poveznice ni s jednim razvojnim okvirom. Koristi se za opisivanje entiteta koji predstavljaju tablicu pohranjenu u bazi podataka, a njegovi podatkovni članovi jednaki su stupcima tablice unutar baze podataka. Instanca određenog entiteta odgovara retku tablice u bazi podataka.

Entiteti predstavljene aplikacije kreirani su pomoću Java klasa i razvojnog okvira Spring Boot. Klase koje predstavljaju entitete definiraju se korištenjem Spring Boot anotacije `@Entity` iznad imena klase. Pozadinski mikroservisi Spring Boota pregledavaju sve klase te one s `@Entity` anotacijom definiraju kao klase entiteta. Anotacija `@Table` se također postavlja nad navedenu klasu s ciljem povezivanja tablice u bazi podataka i Java klase uz mogućnost postavljanja imena tablice. Ako se ime eksplicitno ne postavi, tablica poprima ime klase entiteta. Korištenjem lombok anotacija automatski se definiraju konstruktori i metode za dohvaćanje i postavljanje vrijednosti varijabli klase odnosno vrijednosti stupaca tablice. Isto tako u klasi entiteta navode se svi stupci tablice te se definiraju veze između tablica. Uz već spomenute anotacije koje se koriste nad klasom entiteta, opisane su i sljedeće anotacije korištene nad njenim podatkovnim članovima:

- `@Id` - Koristi se za definiranje primarnog ključa entiteta. Primarni ključ je primitivnog tipa, a najčešće *Integer* ili *Long*.
- `@GeneratedValue` - Definira strategiju za generiranje vrijednosti primarnog ključa entiteta. Strategija korištena u aplikaciji je `GenerationType.IDENTITY` koja osigurava jedinstvenu vrijednost primarnog ključa odnosno da se za primarni ključ koristi *identity* stupac.
- `@Column` - Anotacija korištena za definiranje imena stupca unutar tablice baze podataka. Ako se naziv ne postavi, ime stupca će odgovarati imenu podatkovnog člana klase entiteta.
- `@Transient` - Spring Boot anotacija nad članom klase entiteta, pomoću koje ga razvojni okvir ignorira prilikom mapiranja stupaca baze podataka. Koristi se kada vrijednosti polja ne trebaju biti pohranjene unutar baze podataka.
- `@JoinColumn` - Anotacija za imenovanje stupca tablice koji je ujedno definiran kao strani ključ te pruža vezu prema drugoj tablici.

- `@JoinTable` - Koristi se za kreiranje međutablice s kompozitnim primarnim ključem prilikom realiziranja veze više-na-više. Omogućava definiranje imena međutablice te koristi primarne ključeve entiteta koji sudjeluju u relaciji na temelju kojih kreira kompozitni primarni ključ.
- `@OneToOne` - Pomoću anotacije `@JoinColumn` povezuje strani ključ matične tablice s primarnim ključem drugog entiteta i osigurava relaciju jedan-na-jedan. Anotacija `@OneToOne` je korištena u aplikaciji kako bi se povezale tablice `user` i `membership` te `loan_details` i `late_fee`.
- `@ManyToMany` - Anotacija kojom se između dva entiteta realizira relacija više-na-više. Spomenuta anotacija korištena je unutar entiteta `work` te određenom književnom djelu pruža veći broj autora i žanrova dok pojedini autor ili kategorija mogu imati više različitih književnih djela.
- `@ManyToOne` - Anotacija za definiranje veze više-na-jedan.

U poglavlju 3.2 detaljno je objašnjena struktura entiteta korištenih u aplikaciji uz pripadajuće relacije. Zbog toga su u nastavku prikazani primjeri izvornog kôda zaslužni za generiranje entiteta i njihovih relacija bez dodatnog opisa. Ispisi 6 i 7 predstavljaju dvije krucijalne Java klase u kojima je prikazano korištenje svih prethodno navedenih anotacija s ciljem pravilnog kreiranja entiteta i njihovih međusobnih relacija.

```
1  @Entity
2  @Getter
3  @Setter
4  @AllArgsConstructor
5  @NoArgsConstructor
6  @Table(name = "loan")
7  public class Loan {
8
9      @Id
10     @GeneratedValue(strategy = GenerationType.IDENTITY)
11     @Column(name = "id", nullable = false)
12     private Integer id;
13
```

```
14      @ManyToOne
15      @JoinColumn(name = "member_id", referencedColumnName = "id")
16      private User member;
17
18      @ManyToOne
19      @JoinColumn(name = "librarian_id", referencedColumnName = "id")
20      private User librarian;
21
22      @Column(name = "date_reserved")
23      private LocalDate dateReserved;
24
25      @Column(name = "date_issued")
26      private LocalDate dateIssued;
27
28      @Column(name = "loan_status")
29      private LoanStatus loanStatus;
30
31      @CreationTimestamp
32      private Timestamp createDateTime;
33
34      @Transient
35      public String getLoanStatusClass() {
36          return switch (this.loanStatus) {
37              case IN_PROGRESS -> "warning";
38              case RESERVED -> "info";
39              case IN_LOAN -> "primary";
40              case DONE -> "success";
41              case CANCELLED -> "danger";
42          };
43      }
44  }
```

Ispis 6: Prikaz klase entiteta `loan`

```java
@Entity
@Getter
@Setter
@NoArgsConstructor
@Table(name = "\"user\"", uniqueConstraints = {@UniqueConstraint(
    columnNames = {"email"})})
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    private Integer id;

    @Column(name = "first_name", length = 50)
    @NotNull(message = "first_name shouldn't be null")
    private String firstName;

    @Column(name = "last_name", length = 50)
    @NotNull(message = "last_name shouldn't be null")
    private String lastName;

    @Column(name = "password")
    @NotNull(message = "password shouldn't be null")
    private String password;

    @Column(name = "email")
    @NotEmpty(message = "Email cannot be empty")
    @NotNull(message = "email shouldn't be null")
    private String email;

    @Column(name = "date_of_birth")
    @NotNull(message = "date_of_birth shouldn't be null")
    private Timestamp dateOfBirth;

    @Column(name = "contact_number")
    @NotNull(message = "contact_number shouldn't be null")
    private String contactNumber;

```

```
38      @Column(name = "enabled")
39      private boolean enabled;
40
41      @OneToOne
42      @JoinColumn(name = "membership_id", referencedColumnName = "id")
43      private Membership membership;
44
45      @ManyToMany(fetch = FetchType.EAGER)
46      @JoinTable(
47              name = "user_role",
48              joinColumns = @JoinColumn(name = "user_id"),
49              inverseJoinColumns = @JoinColumn(name = "role_id")
50      )
51      private Set<Role> roles = new HashSet<>();
52
53      @Transient
54      public boolean hasMemberRole() {
55          return this.roles.stream()
56                  .filter(i -> i.getName().equals("MEMBER"))
57                  .findAny()
58                  .orElse(null) != null;
59      }
60  }
```

Ispis 7: Prikaz klase entiteta `user`

Nakon stvaranja entiteta potrebno je kreirati **DTO** (engl. *Data transfer object*) klase koje će predstavljati podatke primljene odnosno poslane na korisničko sučelje. Navedene klase nalaze se u paketu `org.oss.library_management_system.dto` unutar `src/main/java` direktorija. DTO klase su identične klasama entiteta te je njihova uloga slanje vrijednosti generiranih u aplikaciji na korisničko sučelje i obrnuto. Koriste se kada je potrebno prikazati ili dobiti objedinjene podatke različitih entiteta. Nad atributima DTO klasa postavljene su anotacije koje validiraju unesene podatke s korisničkog sučelja kako bi se zaštitio njihov integritet te kako se nesuvislo uneseni podatci ne bi mapirali u element entiteta. DTO klasa `book_edition` s anotacijama korištenim za validaciju podataka prikazana je u ispisu 8.

```java
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Data
public class BookEditionPayload {
    private Integer id;
    private Integer work;
    private Integer publisher;
    private Integer yearOfPublishing;
    @Pattern(regexp = "^(?:ISBN(?:-13)?:? )?(?=[0-9]{13}$|(?=(?:[0-9]+[-
    ]){4})[- 0-9]{17}$)97[89][- ]?[0-9]{1,5}[- ]?[0-9]+[- ]?[0-9]+[-
    ]?[0-9]$", message = "Isbn must contain 13 digits!")
    private String isbn;
    private String bookCover;
    private Integer weight;
    @Pattern(regexp="^[A-Za-z_ ]*$",message = "Invalid Input for the book
     edition language")
    @NotBlank(message = "language must not be empty")
    private String language;
    private Integer pageNumber;
    @NotBlank(message = "edition must not be empty")
    private String edition;

    private LocalDate dateReleased;

    @NotBlank(message = "illustration must not be empty")
    private String illustration;
}
```

Ispis 8: Prikaz DTO klase `book_edition`

4.2.3. Pristup podatcima - JPA repozitoriji

U današnjem vremenu stvarni poslovni procesi generiraju veliku količinu podataka koji se zbog informatizacije neophodno pohranjuju unutar baza i skladišta podataka. Pravilna pohrana i brzo dohvaćanje podataka predstavljaju bitan segment svake aplikacije i

informacijskog sustava. Glavni cilj je iskoristiti procesorsku snagu računala te na temelju dohvaćenih podataka pomoću raznih programskih algoritama doći do složenih spoznaja u svrhu unaprijeđenja poslovnog sustava.

Razvojni okvir Spring pomoću modula Spring Data rad s bazama podataka čini znatno jednostavnijim. Spring Data olakšava cijeli proces manipulacije podatcima i reducira količinu kôda potrebnog za interakciju s bazom podataka. Dohvaćanje podataka u opisanoj aplikaciji realizirano je pomoću JPA repozitorija. Za svaki entitet u aplikaciji definirano je sučelje tj. repozitorij pomoću kojeg se pristupa elementima tog entiteta s ciljem dohvaćanja potrebnih podataka. Kreirano sučelje definirano je kao repozitorij Spring Boot anotacijom `@Repository` te nasljeđuje sučelje `JpaRepository<T,ID>`. `T` predstavlja entitet čijim podatcima će se manipulirati, a `ID` tip primarnog ključa tog entiteta. Metode unutar JPA repozitorija omogućavaju filtriranje, sortiranje, paginaciju i CRUD (*engl. Create Read Update Delete*) operacije nad podatcima. Metode se pišu intuitivno te se podatcima manipulira na temelju samog naziva. IntelliJ IDEA pomaže u kreiranju upita prikazivanjem mogućih opcija u padajućem izborniku. Konvencija imenovanja metoda je *lowerCamelCase* tj. ime metode započinje malim slovom, a svaka slijedeća riječ se odvaja velikim. Upiti prema bazi podataka se također mogu pisati pomoću anotacije `@Query` koja se postavlja nad metodu, a za parametar prima posebno napisani SQL upit. Pisanje upita s `@Query` anotacijom nije preporučljivo jer u slučaju promjene baze podataka i programskog jezika za interakciju s njom, pristup i upravljanje podatcima bit će onemogućeno. Komunikaciju s bazom podataka treba prepustiti razvojnom okviru koji na jednostavan način osigurava pristup podatcima neovisno o vrsti baze podataka i programskog jezika za izradu upita [14]. Ispis 9 sadrži primjer JPA repozitorija koji omogućava dohvaćanje i obradu potrebnih podataka o posudbama knjižnične građe, dok se na slici 4 prikazuje proces kreiranja upita unutar repozitorija `LateFeeRepository`.

```
1 @Repository
2 public interface LoanRepository extends JpaRepository<Loan, Integer> {
3     Optional<Loan> findLoanByLoanStatusAndMemberId(LoanStatus status,
    Integer memberId) ;
4
5     Optional<Loan> findLoanByLoanStatusAndLibrarianId(LoanStatus status,
    Integer librarianId) ;
```

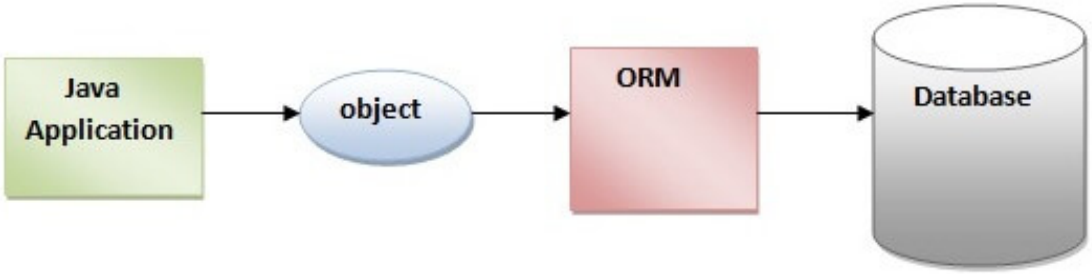
```
6
7       Integer countLoansByLoanStatusAndMemberId(LoanStatus status, Integer
        memberId);
8
9       Page<Loan> findAllByLoanStatusNot(LoanStatus status, Pageable
        pageable);
10
11      Page<Loan> findAllByMember_IdAndLoanStatusNot(Integer memberId,
        LoanStatus status, Pageable pageable);
12
13      List<Loan> findAllByMember_Id(Integer memberId);
14
15      Page<Loan> findAllByMember_Id(Integer memberId, Pageable pageable);
16
17      List<Loan> findAllByLoanStatus(LoanStatus status);
18
19      Page<Loan> findAllByLoanStatus(LoanStatus status, Pageable pageable);
20
21      Page<Loan> findAllByLoanStatusAndMember_Id(LoanStatus status, Integer
         memberId, Pageable pageable);
22      long countAllByDateIssuedBetween(LocalDate from, LocalDate to);
23
24      long countAllByDateIssuedBetweenAndMember_Id(LocalDate from,
        LocalDate to, Integer memberId);
25  }
```

Ispis 9: Prikaz repozitorija `LoanRepository`

Slika 4: Prikaz kreiranja upita pomoću JPA repozitorija

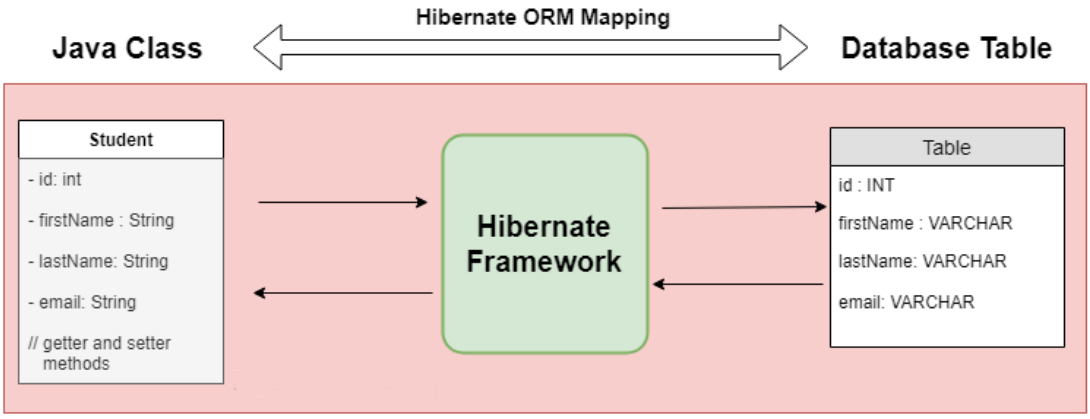
4.2.4. Hibernate ORM

Hibernate je Java razvojni okvir otvorenog kôda koji Java aplikaciji olakšava interakciju s bazom podataka te pruža objektno relacijsko mapiranje (ORM - *Object Relational Mapping*). Razvojni okvir Hibernate obavlja mapiranje Java objekata u tablice baze podataka i obratno te mapiranje Java tipova podataka u SQL tipove koje koristi baza podataka. Koristeći JPA pohranjuje se, ažurira i dohvaća podatke iz relacijskih baza podataka, a Hibernate implementira i omogućuje sve JPA specifikacije. Hibernate pruža interakciju s relacijskim bazama na visokoj razini te samostalno generira SQL upite, služi kao posrednik između klase entiteta i određene tablice relacijske baze podataka. Prednosti Hibernatea su neovisnost o bazi podataka jer podržava HQL (engl. *Hibernate Query Language*), automatsko kreiranje tablica u bazi podataka i brzina postignuta korištenjem privremene memorije (franc. *cache*) [15]. Na slici 5 prikazan je dijagram korištenja ORM-a povezivajući aplikaciju s relacijskom bazom podataka. Postupak mapiranja jednostavne Java klase entiteta u tablicu baze podataka i obratno opisan je slikom 6.



Slika 5: Proces spremanja objetka u bazu podataka korištenjem ORM-a

Preuzeto s poveznice `https://www.javatpoint.com/hibernate-tutorial`



Slika 6: Prijem mapiranja Java objekta u tablicu baze podataka i obratno

Preuzeto s poveznice `https://www.javaguides.net/2018/11/hibernate-framework-overview-architecture-bascis.html`

4.2.5. Punjenje podatcima - Java Faker

Nakon kreiranja Spring Boot projekta i izrade klasa entiteta obavlja se migracija baze podataka. Pomoću Hibernatea kreiraju se tablice, a sljedeći korak u razvoju aplikacije je populacija baze podataka testnim podatcima. Nad pohranjenim podatcima će se vršiti razne operacije, a bit će prikazani na korisničkom sučelju. Punjenje podataka odvija se pomoću metode `run` unutar *Spring Beana* koji implementira `CommandLineRunner` sučelje, a definiran je anotacijom `@Component`. Metoda `run` izvršava se uvijek pri pokretanju aplikacije. U slučaju postojanja više *Spring Beanova* koji implementiraju `CommandLineRunner` i metodu `run`, pomoću anotacije `@Order` određuje se prioritet izvršavanja. Ispis 10 prikazuje kreiranje Spring komponente implementirajući `CommandLineRunner` sučelje.

```
1 @Component
2 public class DataSeedLoader implements CommandLineRunner{
3     @Override
4     public void run(String... args) throws Exception {
5   loadRoleData();
6         loadUserData();
7         loadAuthorData();
8         loadCategoryData();
9         loadWorkData();
```

```
10          loadPublisherData();
11          loadBookEditionData();
12          loadBookCopyData();
13          loadLoanData();
14          loadLoanDetailsData();
15      }
16  }
```

Ispis 10: *Spring Bean* za populiranje baze podataka

Punjenje podatcima realizirano je korištenjem pomoćnih metoda pozvanih unutar glavne `run` metode. Podatci se generiraju pomoću **JavaFaker** biblioteke uključene kao ovisnost u `pom.xml` datoteci. JavaFaker korišten je za generiranje širokog spektra lažnih podataka. Raznim klasama i metodama navedena biblioteka generira podatake stvarnog izgleda koji mogu biti nazivi ljudi, država, jezika, adresa, knjiga i mnogi drugi. Generiranje podataka pomoću JavaFaker biblioteke je korisno prilikom testiranja aplikacije jer se postiže provjera valjanosti bez upotrebe stvarnih vrijednosti. U praktičnom radu korištene su razne klase za simuliranje naziva knjiga, ISBN identifikatora te imena autora i izdavača [16]. Ispis 11 opisuje korištenje JavaFaker klasa u metodi za generiranje književnih djela.

```
1  private void loadWorkData() {
2      if (workRepository.count() <= 10) {
3          for (int i = 0; i < 20; i++) {
4              var faker = new Faker();
5              var work = new Work();
6              work.setTitle(faker.book().title());
7              work.setDescription(faker.lorem().sentence());
8              work.setOriginalTitle(work.getTitle());
9              work.setOriginalLanguage(faker.nation().language());
10             var category = categoryRepository.getOneRandom();
11             Set<Category> categories = new HashSet<>();
12             categories.add(category);
13             work.setCategories(categories);
14             var author = authorRepository.getOneRandom();
15             Set<Author> authors = new HashSet<>();
16             authors.add(author);
```

```
17              work.setAuthors(authors);
18              workRepository.save(work);
19          }
20          System.out.println("Seeded work table successfully");
21      }
22  }
```

Ispis 11: Metoda za punjenje podataka tablice `work`

4.3. Servisi i kontroleri

Spring Boot aplikacija ima slojevitu arhitekturu. Repozitoriji predstavljaju sloj odgovoran za komunikaciju s bazom, servisi sloj zaslužan za obradu podataka i kontroleri sloj upravljanja putanjama za prikaz podataka na korisničkom sučelju. Servisi su klase tj. *Spring Beanovi* označeni anotacijom `@Service`, a sadržavaju svu poslovnu logiku aplikacije. Sloj poslovne logike zaslužan je za povezivanje preostala dva sloja. Servisi obrađuju podatke primljene iz repozitorija, mapiraju ih u DTO objekte te ih prosljeđuju *kontrolerima* tj. upravljačima. Unutar servisa također se hvataju i obrađuju iznimke s ciljem spriječavanja grešaka koje prekidaju daljnje izvođenje aplikacije [17]. U ispisu 12 prikazan je primjer servisa `CategoryServiceImpl` i metoda za dohvaćanje svih žanrova.

```
1  @Service
2  public class CategoryServiceImpl implements CategoryService {
3
4      private final CategoryRepository categoryRepository;
5
6      private final ModelMapper modelMapper;
7
8      public CategoryServiceImpl(CategoryRepository categoryRepository,
       ModelMapper modelMapper) {
9          this.categoryRepository = categoryRepository;
10         this.modelMapper = modelMapper;
11     }
12
13     @Override
14     public Page<CategoryPayload> getAllCategories(String keyword, int
```

```
    page, int size, String[] sort) {
15          var sortField = sort[0];
16          var sortDirection = sort[1];
17          var direction = sortDirection.equals("desc") ? Sort.Direction.
    DESC : Sort.Direction.ASC;
18          var order = new Sort.Order(direction, sortField);
19
20          Pageable paging = PageRequest.of(page - 1, size, Sort.by(order));
21
22          Page<Category> categoryPage;
23
24          if (keyword == null) {
25              categoryPage = categoryRepository.findAll(paging);
26          } else {
27              categoryPage = categoryRepository.
    findByNameContainingIgnoreCase(keyword, paging);
28          }
29          var categoryPayloadList = categoryPage.stream().map(category ->
    modelMapper.map(category, CategoryPayload.class)).toList();
30
31          Page<CategoryPayload> categoryPayloadPage  = new PageImpl<>(
    categoryPayloadList, paging, categoryPage.getTotalElements());
32          return categoryPayloadPage;
33      }
34  }
```

Ispis 12: Prikaz dijela servisa `CategoryServiceImpl`

Upravljači tj. kontroleri predstavljaju Java klase nad kojima je postavljena anotacija `@Controller`. Uloga kontrolera je obrada dolaznih korisničkih zahtjeva te priprema modela i vraćanje podataka na određene putanje koji će se prikazati na korisničkom sučelju. Također nad klasom kontrolera nalazi se anotacija `@RequestMapping` koja definira prefiks putanje svih metoda unutar kontrolera, metode predstavljaju krajnje točke aplikacije. Svaka metoda ima anotaciju ovisno o HTTP (engl. *Hypertext Transfer Protocol*) zahtjevu kojeg obrađuje, sukladno tome korištene anotacije su `@GetMapping`, `@PostMapping`, `@PutMapping` i `@DeleteMapping`. Svaka od navedenih anotacija prima tekstualni parametar, definirajući ostatak putanje ako je to potrebno. Primjer kontrolera s metodom koja

obrađuje *GET* zahtjev te dohvaća detalje pojedine posudbe i prikazuje ih na ruti `http://localhost:8080/loans/id` opisan je u ispisu 13.

```
@Controller
@RequestMapping(value = "/loans")
public class LoanController {
    @GetMapping("/{id}")
     public String loanDetails(Model model, @PathVariable Integer id)
    throws NotFoundException, ForbiddenResourcesException {
         var authentication = SecurityContextHolder.getContext().
    getAuthentication();
         var currentUser = userRepository.findByEmail(authentication.
    getName());
         var currentUserRoles = new ArrayList<String>();
         for (var role : currentUser.getRoles()) {
             currentUserRoles.add(role.getName());
         }

         if(Objects.equals(currentUserRoles.get(0), "MEMBER")){
             var currentUserLoansId = loanRepository.findAllByMember_Id(
    currentUser.getId()).stream().map(Loan::getId).toList();
             if(!currentUserLoansId.contains(id)) throw new
    ForbiddenResourcesException("you do not have access rights! ");
         }
         var loan = loanService.getLoan(id);
         if (loan != null) {
             if(loan.getLoanStatus() == LoanStatus.IN_PROGRESS) throw new
    ForbiddenResourcesException("you do not have access rights! ");
             model.addAttribute("loan", loan);
             model.addAttribute("loanDetailsList", loanDetailsRepository.
    findAllByLoanId(id));
         }
         return "loan/loanDetails";
     }
}
```

Ispis 13: Primjer korištenja `@GetMapping` anotacije u kontroleru `LoanController`

U ispisu 13 također je prikazano korištenje anotacije `@PathVariable` koja unutar URL-a prosljeđuje varijablu potrebnu za definiranje točne rute tj. krajnje točke. U navedenom primjeru se pretražuju sve posudbe, a prikazuje se samo ona s točnim identifikatorom u putanji, ako element entiteta `loan` nije pronađen obrađuje se iznimka. Preostale anotacije korištene za dohvat podatka s kljentske strane su `@RequestParam` i `@ModelAttribute`. Anotacija `@RequestParam` služi za primanje parametara iz URL-a, dok anotacija `@ModelAttribute` dohvaća podatke unesene na korisničkom sučelju. Ispis 14 prikazuje primjer korištenja `@ModelAttribute` i `@PostMaping` anotacija prilikom kreiranja i spremanja novog književnog djela u bazu podataka na putanji `http://localhost:8080/works/saveWork`. Anotacijom `@Valid` validiraju se podatci uneseni u formi na korisničkom sučelju te u slučaju nesuvislog unosa korisniku se prikazuje poruka o greški [18].

```
1     @PostMapping("/saveWork")
2     public RedirectView saveNewWork(@Valid
3             @ModelAttribute("workPayload") WorkPayload workPayload,
4             BindingResult result, RedirectAttributesredirectAttributes){
5           if (result.hasErrors() && result.getFieldError() != null){
6              redirectAttributes.addFlashAttribute("error", "The form is
    not valid! " +  result.getFieldError().getDefaultMessage());
7           }
8           else {
9              try {
10                 var work = workService.createWork(workPayload);
11                 redirectAttributes.addFlashAttribute("success",
12                 "The work: " + work.getTitle() + " successfully
           created!");
13             }catch (Exception e) {
14                 redirectAttributes.addFlashAttribute("error",
15                 e.getMessage());
16             }
17          }
18          return new RedirectView("/works");
19    }
```

Ispis 14: Primjer korištenja `@PostMapping` i `@ModelAttribute` anotacija

4.4. Autentikacija

Autentikacija u aplikaciji realizirana je zahvaljujući razvojnom okviru Spring Security koji automatski, uz minimalno dodatnog konfiguriranja, pruža autentikacijske usluge uključujući i enkripciju lozinki. Konfiguracija Spring Security značajki omogućuje se pomoću Java klase s anotacijama `@Configuration` i `@EnableWebSecurity`. Da bi se služio aplikacijom korisnik mora biti prijavljen u istu. Samostalna registracija nije moguća već nove članove knjižnice u aplikaciju registriraju knjižničari i administratori. Prilikom prijave korisnika potrebno je pohraniti podatke o njegovom korisničkom računu sadržane u klasi `CustomUserDetails` koja implementira Spring Security sučelje `UserDetails`. Sučelje `UserDetails` pruža dohvaćanje i postavljanje korisničkih podataka potrebnih za prijavu te je prikazano u ispisu 15. Glavna uloga autentikacije pružena je implementiranjem sučelja `AuthenticationManager` koje obrađuje `AuthenticationException` iznimku ako korisnički podatci nisu ispravni i onemogućuje pristup resursima aplikacije neautenticiranim korisnicima. Klasa za konfiguraciju Spring Securityja prikazana je u u ispisu 16.

```
1  public interface UserDetails extends Serializable {
2      Collection<? extends GrantedAuthority> getAuthorities();
3
4      String getPassword();
5
6      String getUsername();
7
8      boolean isAccountNonExpired();
9
10     boolean isAccountNonLocked();
11
12     boolean isCredentialsNonExpired();
13
14     boolean isEnabled();
15 }
```

Ispis 15: Prikaz Spring Security sučelja `UserDetails`

```java
@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class SecurityConfiguration{

    @Bean
    public UserDetailsService userDetailsService() {
        return new CustomUserDetailsService();
    }

    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public DaoAuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new
    DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService());
        authProvider.setPasswordEncoder(passwordEncoder());

        return authProvider;
    }

    @Bean
    public AuthenticationManager authenticationManager(
    AuthenticationConfiguration authConfig) throws Exception {
        return authConfig.getAuthenticationManager();
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http)
    throws Exception {
        http
                .authorizeHttpRequests((requests) -> requests
                        .requestMatchers("/").permitAll()
                        .anyRequest().authenticated()
```

```
36                  )
37                  .formLogin(withDefaults())
38                  .logout(LogoutConfigurer::permitAll);
39
40          return http.build();
41      }
42  }
```

Ispis 16: Spring Security konfiguracijska klasa

4.5. Autorizacija

Nakon konfiguracije značajki Spring Securitya za autentikaciju korisnika, postavljaju se autorizacijska pravila također omogućena pomoću Spring Securitya. Pravila pristupa aplikacijskim resursima ovise o vrsti korisničke uloge. Uloge unutar predstavljene aplikacije su `Administrator`, `Librarian` i `Member`. Korisnici s ulogama `Administrator` i `Librarian` zaduženi su za upravljanje ostalim korisnicima, izdavačima, autorima, kategorijama, posudbama, knjižničnom građom te njenim primjercima. Administrator ima najveće ovlasti, može pristupati svim krajnjim točkama aplikacije i izvršavati sve operacije nad podatcima. Knjižničarima je onemogućen pristup korisničkim računima drugih administratora i knjižničara te nemaju pravo brisanja aplikacijskih resursa. Članovima je dostupan prikaz vlastitih korisničkih i statističkih podataka. Svaki aktivni član može rezervirati i posuđivati knjige prateći osobnu evidenciju posudbi i mogućih zakasnina. Autorizacija je realizirana implementiranjem sučelja `SecurityFilterChain` u istoimenoj metodi unutar Spring Security konfiguracijske datoteke (pogledati ispis 16). `SecurityFilterChain` metoda autenticiranim korisnicima dopušta pristup svim putanjama i omogućava korištenje formi za prijavu i odjavu dobivene sa Spring Securityjem. Ograničenja korisničkih prava za određene putanje i korisničke uloge definiraju se naknadno unutar kontrolera pomoću anotacije `@PreAuthorize`. Anotacija `@PreAuthorize` koristi parametre napisane pomoću Spring Boot izraza definirajući prava pristupa određenim korisničkim ulogama za putanju metode nad kojom je anotacija postavljena. Izraz se provjerava prije izvršavanja metode te se na taj način omogućava pristup putanji kontrolera samo korisnicima s ulogama unesenim u izraz. Korištenje anotacije `@PreAuthorize` kako bi se članovima zabranio pristup formi za dodavanje novih autora u sustav prikazano je u ispisu 17. Spring Security autori-

zacijskim značajkama moguće je pristupiti i unutar HTML predložaka pomoću Thymeleaf atributa, time je omogućeno prikazivanje ili skrivanje odabranih HTML elemenata pojedinim korisničkim ulogama. U ispisu 18 prikazano je korištenje autorizacije pomoću Thymeleafa kako bi se botun s poveznicom na putanju za brisanje autora prikazao samo korisnicima s administratorskim ovlastima [19] [20].

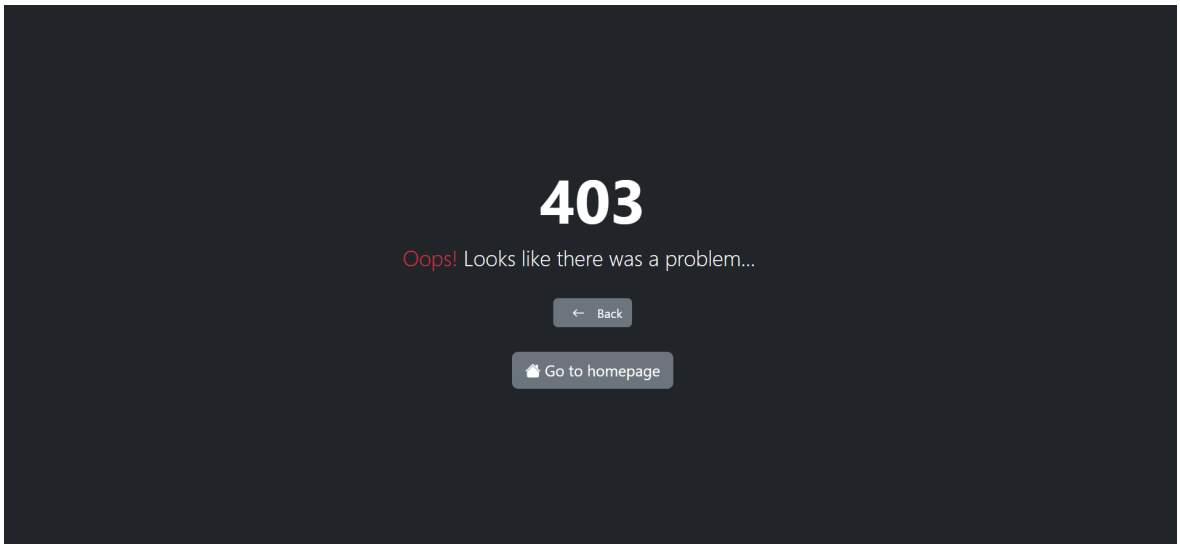
```
1     @PreAuthorize("hasAnyAuthority('ADMIN', 'LIBRARIAN')")
2     @GetMapping("/add")
3     public String addNewAuthor(Model model,
4                     AuthorPayload authorPayload) {
5         model.addAttribute("authorPayload", authorPayload);
6         return "author/addNewAuthor";
7     }
```

Ispis 17: Primjer korištenja @PreAuthorize anotacije

```
1 <a class="btn btn-danger ms-2"
2                 role="button"
3     th:if="${#authorization.expression('hasAuthority(''ADMIN'')')}"
4                 data-bs-toggle="modal" data-bs-target="#deleteModal"
5                 title="Delete author"
6   >
7                  <i class="bi bi-trash3-fill"></i>
8                  Delete
9 </a>
```

Ispis 18: Prikaz korištenja autorizacije pomoću Thymeleafa

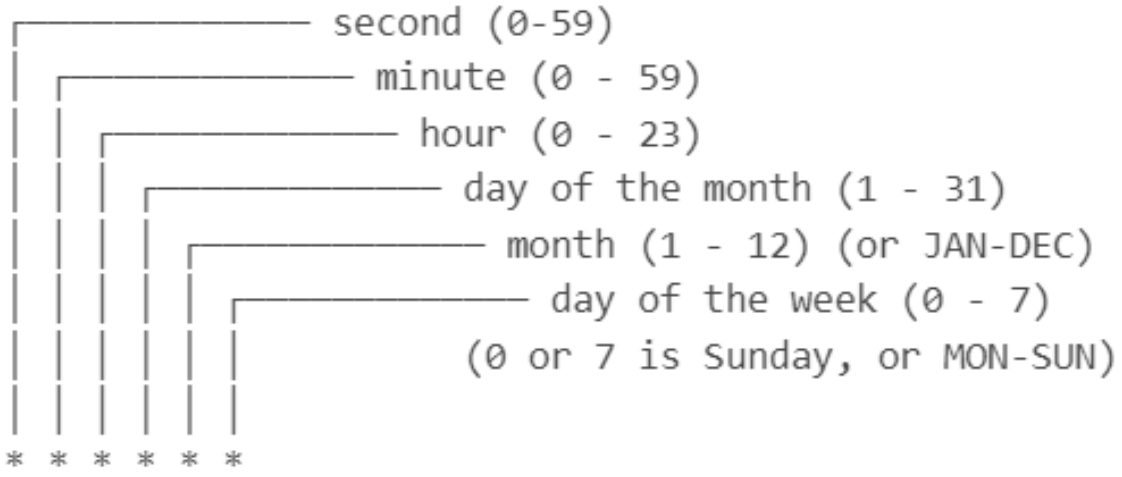
U slučaju korisničkog unosa putanje unutar URL-a koju nema ovlasti posjetiti, prikazuje se *custom* web stranica s prikazom HTTP kôda greške. Kôd greške za neautoriziran pristup traženom resursu je 403 - *forbidden*. Web stranica s kôdom greške prikazana je na slici 7.



Slika 7: Odbijen pristup na rutu za koju korisnik nema potrebne ovlasti

4.6. Zakazani poslovi - Java Scheduler

Zakazani poslovi (engl. *Scheduled jobs*) su dijelovi kôda poslovne logike koji se izvršavaju periodično u određenom vremenu. Spring Boot omogućava jednostavno korištenje planiranih zadataka, a njihova uporaba u aplikaciji omogućava se anotacijom `@EnableScheduling` nad klasom koja sadržava glavnu *main* metodu za pokretanje Spring Boot aplikacije (pogledati ispis 1). Metode koje implementiraju zakazane poslove definirane su anotacijom `@Scheduled` te ne primaju parametre i nemaju povratnu vrijednost [21]. Postoji više načina za određivanje vremena pokretanja zadatka, ali najkorišteniji su cron izrazi. Cron izraz predstavlja *String* tj. niz od šest znakova odvojenih razmakom koji opisuju pojedinačne djelove rasporeda izvršavanja zadatka, a prosljeđuje se unutar anotacije `@Scheduled` kao parametar. Cron izrazi omogućuju periodičko pokretanje zadataka na određeni datum i vrijeme, te predstavljaju alat za automatizaciju procesa bez korisničke interakcije. Na slici 8 prikazan je format cron izraza s opisom uloge i rasponom vrijednosti svakog znaka, asteriks tj. zvjezdica predstavlja sve moguće vrijednosti pojedinog polja. Primjer cron izraza
`cron = "* * * * * *"` izvršava zakazani zadatak svaku sekundu [22].



Slika 8: Prikaz formata cron izraza

Zakazani poslovi korišteni su u aplikaciji kako bi programski kôd implementiran unutar poslovne logike pokrenut pomoću Cron izraza obavljao važne zadatke umjesto administratora i knjižničara. Unutar aplikacije korišteno je pet primjera zakazanih poslova opisanih u nastavku.

- Posudba je omogućena dodavanjem više knjiga u košaricu koja se na kraju treba potvrditi kako bi se posudba aktivirala, status nepotvrđene posudbe je `IN_PROGRESS`. Zakazani zadatak koji se pokreće svakih pola sata te provjerava sve otvorene košarice implementiran je s ciljem otkazivanja svih posudbi statusa `IN_PROGRESS` tj. košarica otvorenih dulje od trideset minuta. Sve knjige iz košarice se uklanjaju kako bi ih drugi korisnici mogli nesmetano posuditi. Realizacija opisanog zadatka prikazana je u ispisu 19.
- Zakazani zadatak provjerava sve aktivne posudbe te ako utvrdi istek predviđenog roka za vraćanje pojedine knjige s posudbe kreira zakasninu i šalje obavijest o naknadi korisniku putem elektroničke pošte. Zadatak se izvršava svaki dan te se naknada povećava dok se knjižnična građa ne vrati.
- Planirani zadatak sa svrhom otkazivanja rezerviranih posudbi korisnika, ako knjižničar ne potvrdi posudbu u roku od četiri dana. Također sve knjige prilikom otkazivanja postaju dostupne za iznajmljivanje drugim članovima.

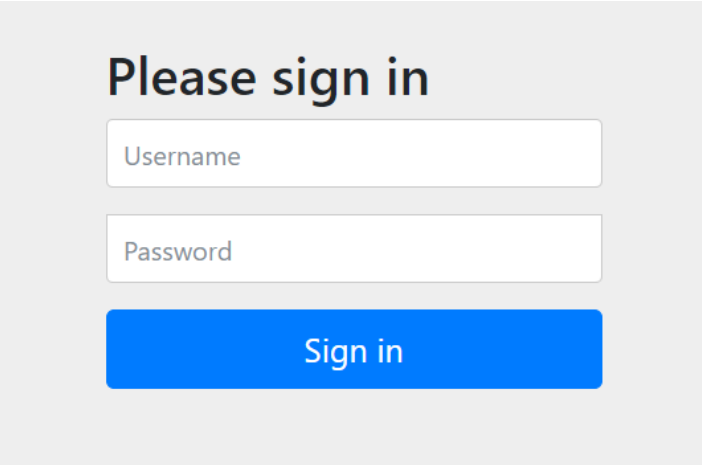
- Praćanje svih aktivnih članstava te slanje obavijesti elektroničkom poštom o isteku članstva pet dana prije krajnjeg roka produženja. Zakazani zadatak se također pokreće svakog dana te deaktivira pojedino članstvo ako nije produženo na vrijeme.
- Zakazani posao za reguliranje cijena članarina ovisno o životnoj dobi člana, a izvršava se prvog dana svakog mjeseca.

```
1  @Scheduled(cron = "0 */30 * * * *")
2  public void checkInProgressLoans(){
3  var loansInProgress = loanRepository.findAllByLoanStatus(LoanStatus.
   IN_PROGRESS);
4      Timestamp currentTimeStamp = new Timestamp(System.
   currentTimeMillis());
5      for(var loanInProgress : loansInProgress){
6          var diff = currentTimeStamp.getTime() - loanInProgress.
   getCreateDateTime().getTime();
7          long diffMinutes = diff / (60 * 1000) % 60;
8          if(diffMinutes > 30){
9              var loanDetailsList = loanDetailsRepository.
   findAllByLoanId(loanInProgress.getId());
10             try {
11                 for (var loanDetails : loanDetailsList ){
12                     loanDetailsService.
13                     removeLoanDetailsFromCart(loanDetails.getId());
14                 }
15                 deleteLoanInProgress(loanInProgress.getId());
16             }catch (Exception e) {
17                 e.printStackTrace();
18             }
19         }
20     }
21 }
```

Ispis 19: Primjer korištenja anotacije `@Scheduled` i cron izraza

4.7. Opis značajki specifičnih za projekt

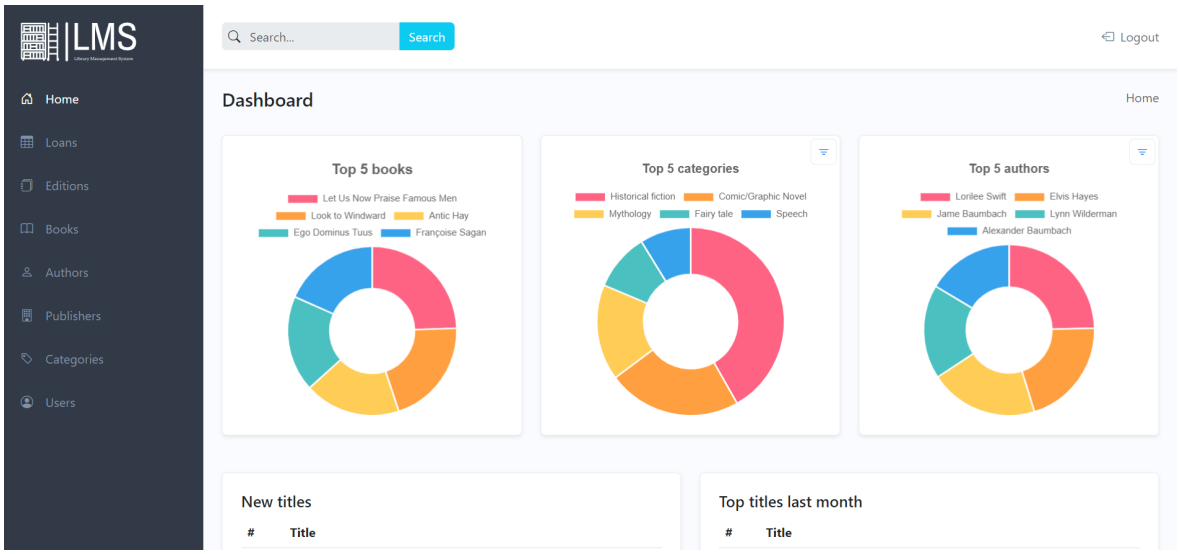
U ovom poglavlju detaljno su opisane funkcionalnosti implementirane u praktičnom dijelu završnog rada uz prikaz korisničkog sučelja pri korištenju aplikacije. Na slici 9 prikazana je jednostavna forma za prijavu korisnika. Dobivena je u sklopu razvojnog okvira Spring Security te je korištena kako bi se neautenticiranim korisnicima zabranio pristup aplikacijskim resursima.



Slika 9: Forma za prijavu

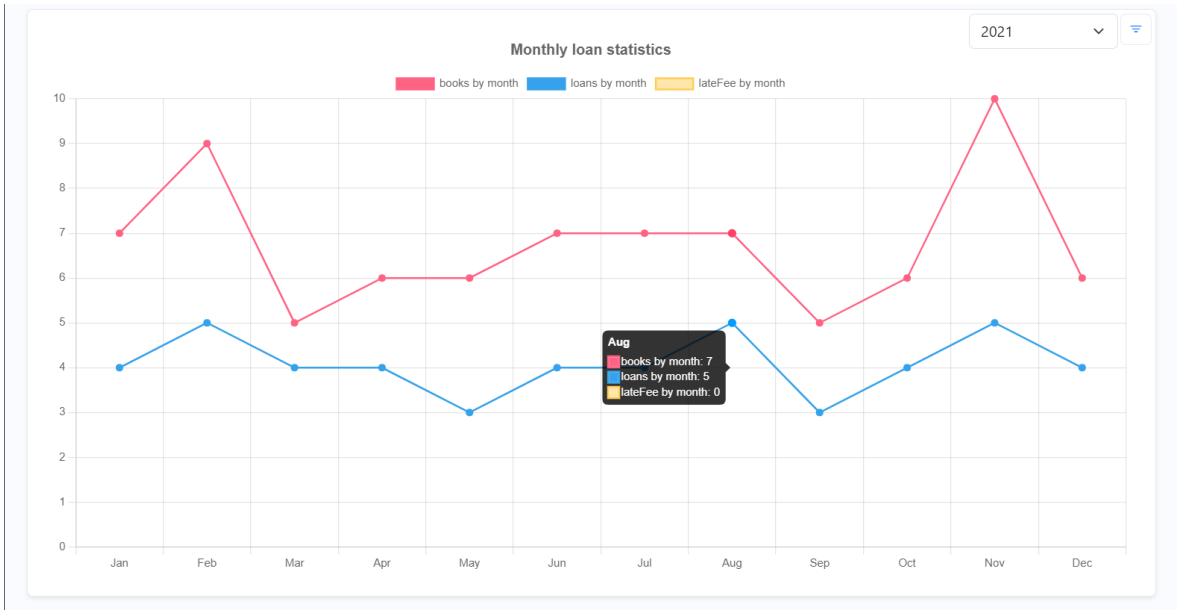
4.7.1. Nadzorna ploča i statistika - Chart.js

Uspješnom prijavom korisnik započinje s radom unutar aplikacije. Autenticirani korisnik nakon prijave preusmjerava se na početnu stranicu koja sadrži nadzornu ploču s osobnim statističkim podatcima te općenitim korisnim informacijama koje su generirane obradom podataka poslovnih procesa koje pruža aplikacija. Bočna navigacijska traka olakšava korištenje aplikacije, pristup rutama i poboljšava korisničko iskustvo (engl. UX - *User Experience*). Prikaz elemenata navigacijske trake ovisi o korisničkoj ulozi, stoga administratori i knjižničari imaju veću razinu pristupa resursima od članova knjižnice. Na slici 10 prikazano je korisničko sučelje glavnog izbornika, neposredno nakon prijave korisnika.

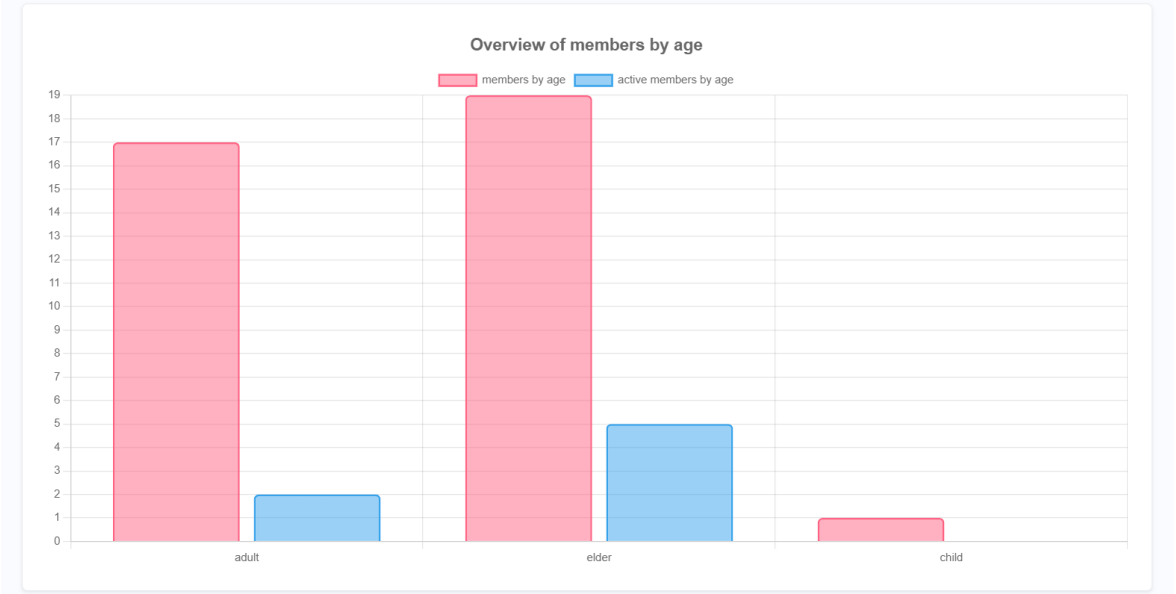


Slika 10: Nadzorna ploča s bočnom navigacijskom trakom

Na slici 11 grafički je prikazana godišnja statistika posudbi, pojedinih primjeraka te iznosa zakasnine za svaki mjesec. Članovima su prikazani osobni podatci dok knjižničari i administratori imaju uvid u cjelokupnu statistiku svih članova knjižnice. Slika 12 prikazuje grafikon na kojem se knjižničarima i administratorima omogućava prikaz aktivnih te neaktivnih članova po životnoj dobi. Pomoću dobivenih podataka administratori mogu prilagoditi svoje poslovanje kako bi zainteresirali i privukli ciljanu grupu ljudi.



Slika 11: Godišnji statistički prikaz posudbi i zakasnina



Slika 12: Statistički prikaz broja članova po životnoj dobi

Grafički prikaz podataka generiranih unutar poslovne logike aplikacije omogućava se pomoću JavaScript biblioteke Chart.js. Pomoću ove besplatne biblioteke znatno je olakšano dizajniranje grafova raznih oblika koji na zanimljiv način prikazuju podatke na korisničkom sučelju. Interakcija između Spring Boota i chart.js biblioteke omogućena je pomoću Thymeleafa. Podatci koji se žele prikazati pomoću grafova mapiraju se u model te se unutar JavaScript kôda zaduženog za kreiranje grafa pozivaju pomoću Thymeleaf sintakse. Grafikon je smješten unutar `canvas` HTML elementa s kojim JavaScript kôd komunicira pomću identifikatora [23]. U ispisu 20 prikazan je kôd korišten za generiranje kružnog grafikona koji prikazuje podatke o najposuđenijim književnim djelima. U prikazu grafikona omogućeno je filtriranje podataka za određeni vremenski interval.

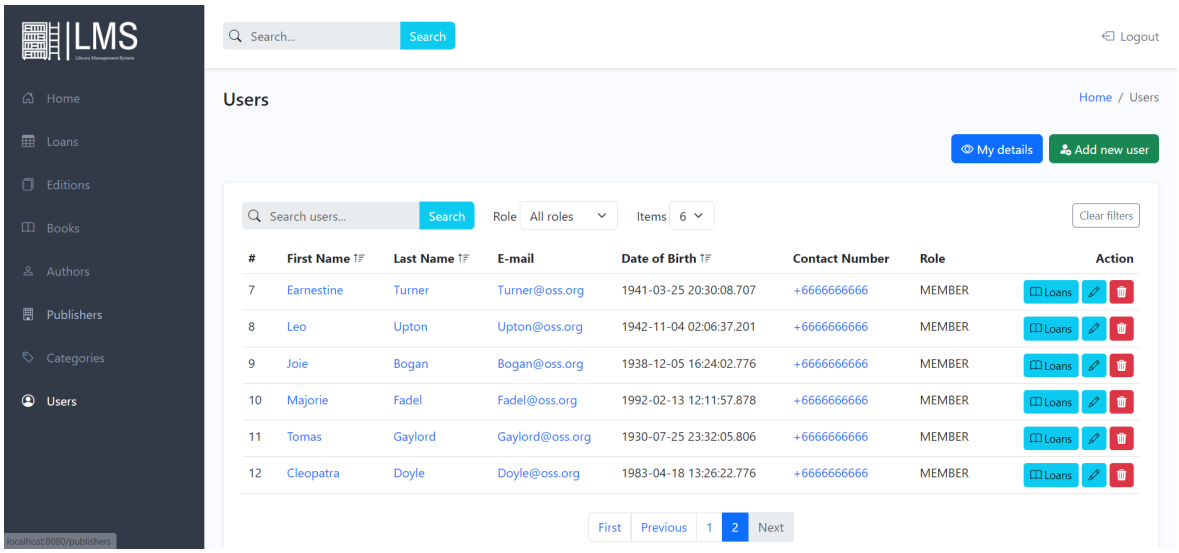
```
1 var topFiveWorks = new Chart(topFiveWorksChrt.getContext("2d"), {
2     type: 'doughnut',
3     data: {
4         labels: [[${workMap.keySet()}]],
5         datasets: [{
6             label: "percentage of borrowed books by work",
7             data: [[${workMap.values()}]],
8             backgroundColor: Object.values(CHART_COLORS),
9             hoverOffset: 5,
```

```
        }],
    },
    options: {
        responsive: true,
        plugins: {
            legend: {
                position: 'top',
            },
            title: {
                display: true,
                text: 'Top 5 books',
                font: {
                    size: 18,
                },
            },
        },
    },
});
```

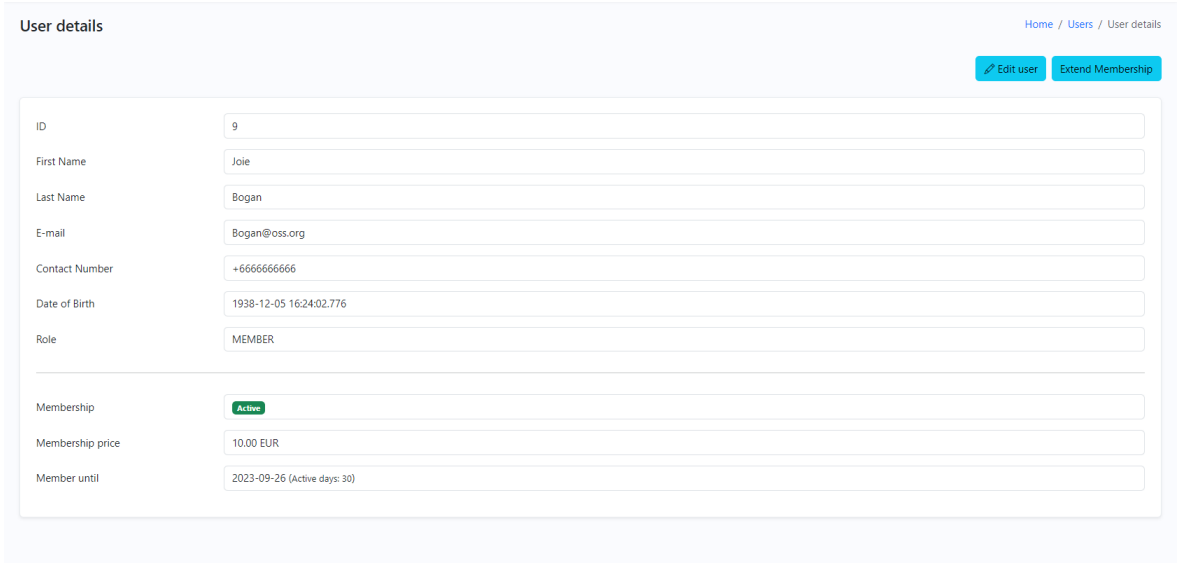
Ispis 20: Kreiranje kružnog grafikona

4.7.2. Upravljanje korisnicima i članstvima

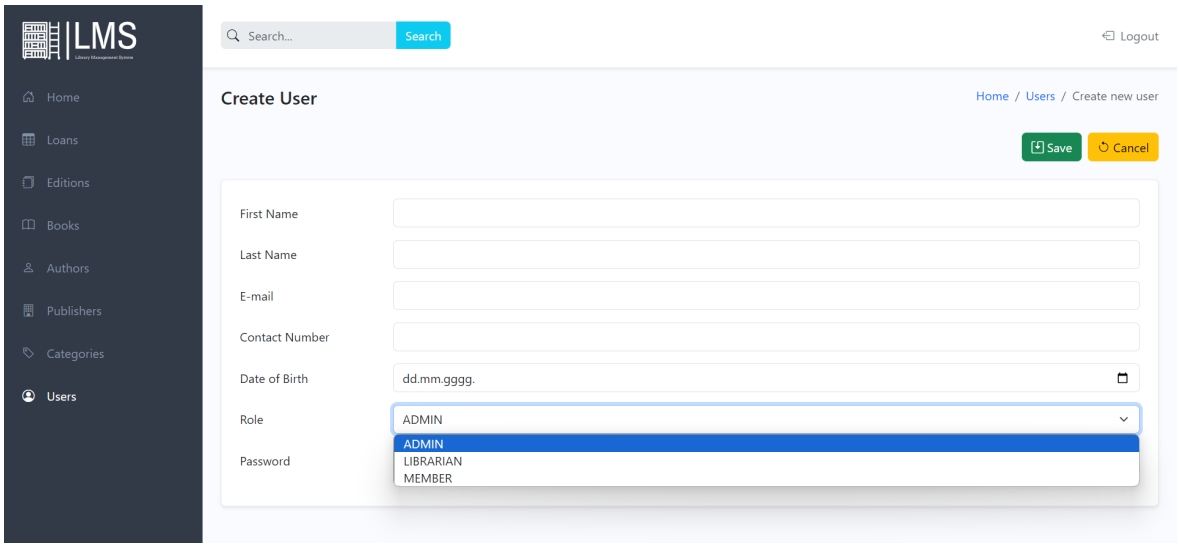
Članovi knjižnice se ne mogu samostalno registrirati te za njih to obavljaju administratori ili knjižničari. Administrator ima uvid u sve korisnike neovisno o korisničkoj ulozi te ih može dodavati, ažurirati i brisati. Knjižničari nemaju mogućnost brisanja, a omogućen im je prikaz, dodavanje i ažuriranje korisnika s ulogom člana. Pojedini član knjižnice ima uvid samo u svoje osobne podatke, uključujući cijenu članarine te status i datum isteka članstva. Na slikama 13, 14 i 15 prikazano je administratorsko korisničko sučelje za upravljanje korisnicima.



Slika 13: Prikaz svih korisnika za administratore

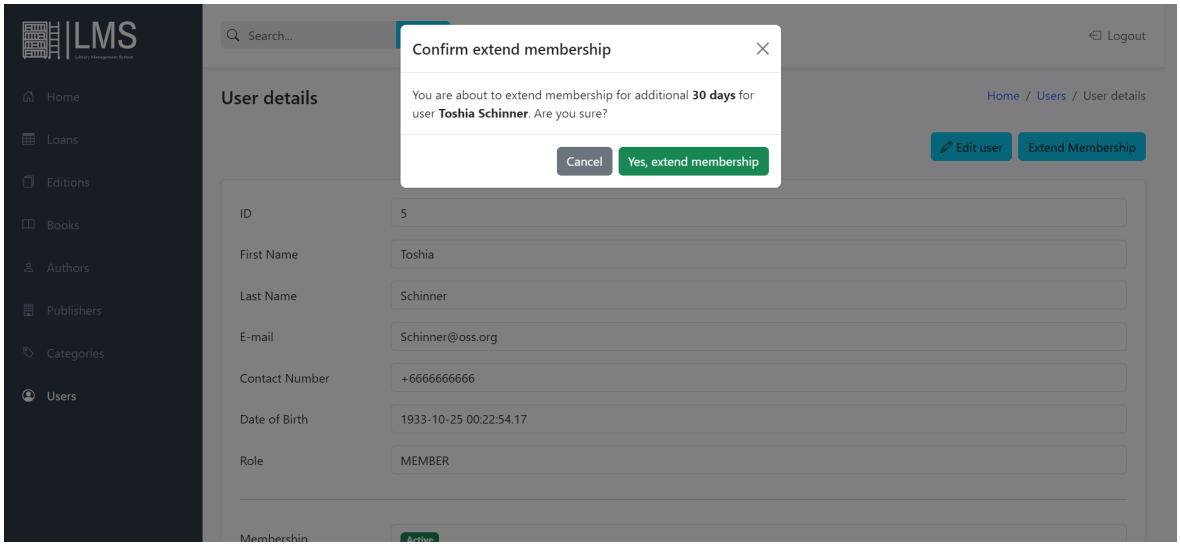


Slika 14: Detalji korisnika



Slika 15: Registracija novih korisnika

Prilikom registracije člana automatski se kreira članstvo te se određuje cijena članarine ovisno o dobi u trajanju od trideset dana. U slučaju neplaćanja mjesečne naknade, članstvo se može otkazati, ali i ponovo obnoviti. Sve navedene operacije nad članstvom korisnika obavljaju administratori i knjižničari. Korisnik čije je članstvo isteklo nema pravo rezerviranja i posuđivanja knjižnične građe. Na slici 16 prikazano je produljenje članstva.

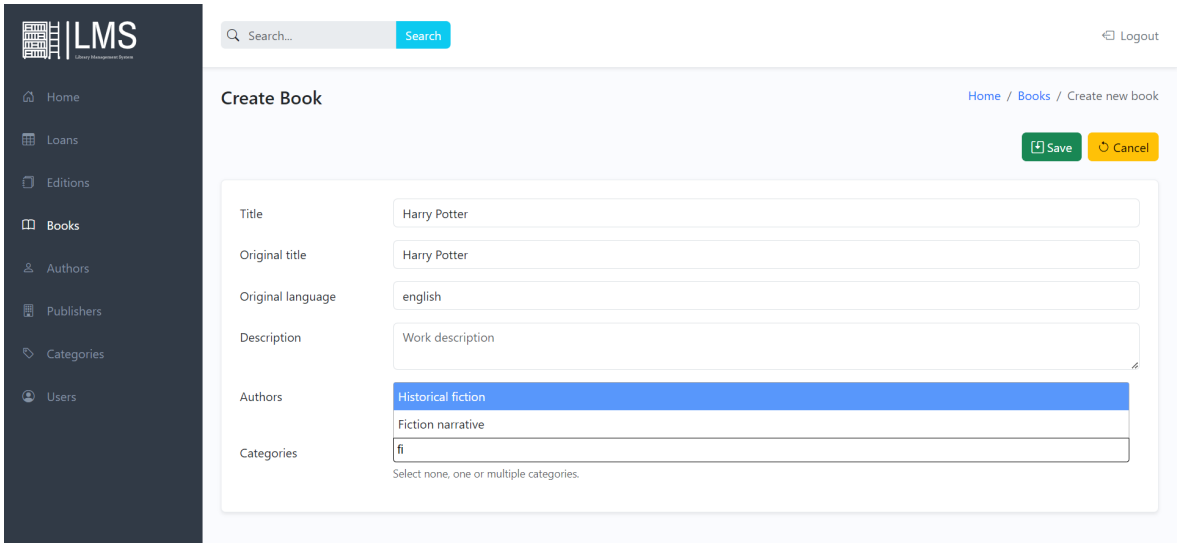


Slika 16: Prikaz skočnog prozora pri produljenju članstva

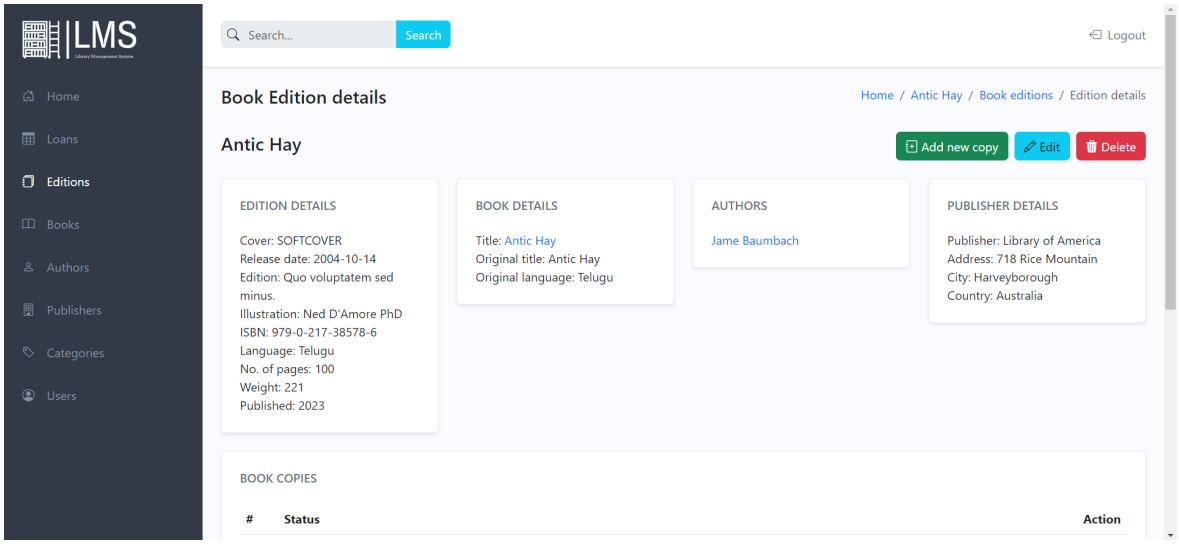
4.7.3. Administracija knjižnične građe

Administratori i knjižničari zaslužni su za upravljanje cjelokupnim sustavom s tim da je ovlast brisanja podataka svih resursa prepuštena samo administratorskim korisnicima

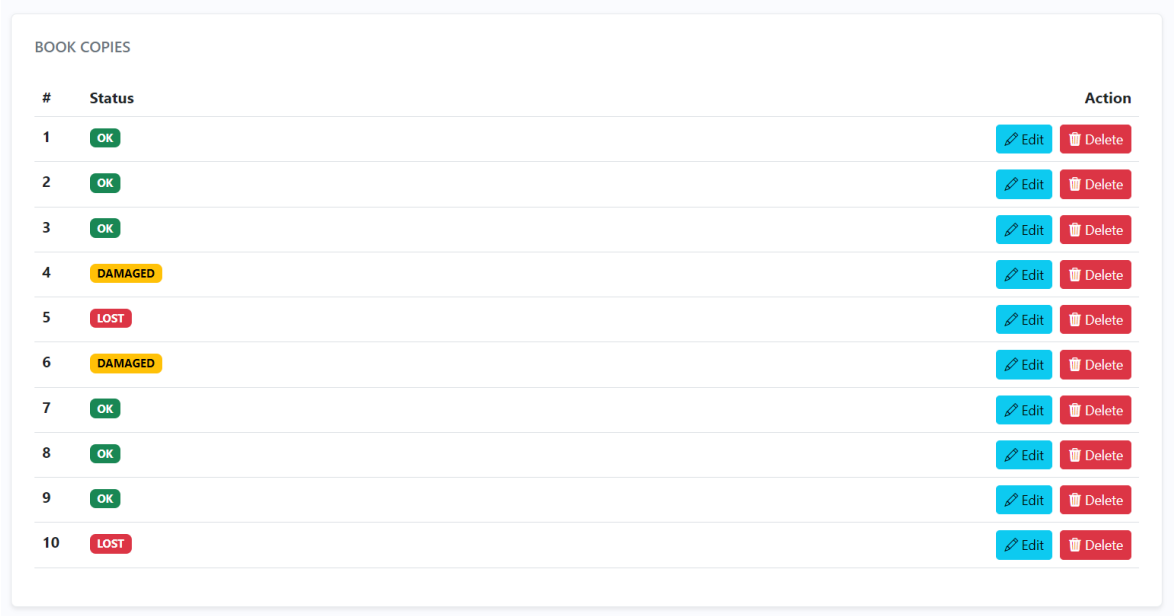
u svrhu povećane sigurnosti. Korisnici s navedenim ulogama upravljaju te imaju pregled podataka o autorima, žanrovima, književnim djelima, izdavačima, određenim izdanjima te njihovim fizičkim primjercima. Omogućeno im je pretraživanje, sortiranje, filtriranje i paginacija svih aplikacijskih resursa te polje za globalnu pretragu knjižnične građe prikazano u zaglavlju (engl. *header*) svake web stranice. Uloga administratora i knjižničara je da na temelju svih podataka o književnim djelima i stanju njihovih primjeraka kontinuirano održavaju knjižnični fond. Svaki primjerak nekog izdanja književnog djela ima svoj status kojeg je moguće ažurirati. Posuđivati se mogu samo primjerci knjižnične građe čiji je status `OK`. Administracija knjižnične građe prikazana je na slikama 17, 18, 19, 20 i 21.



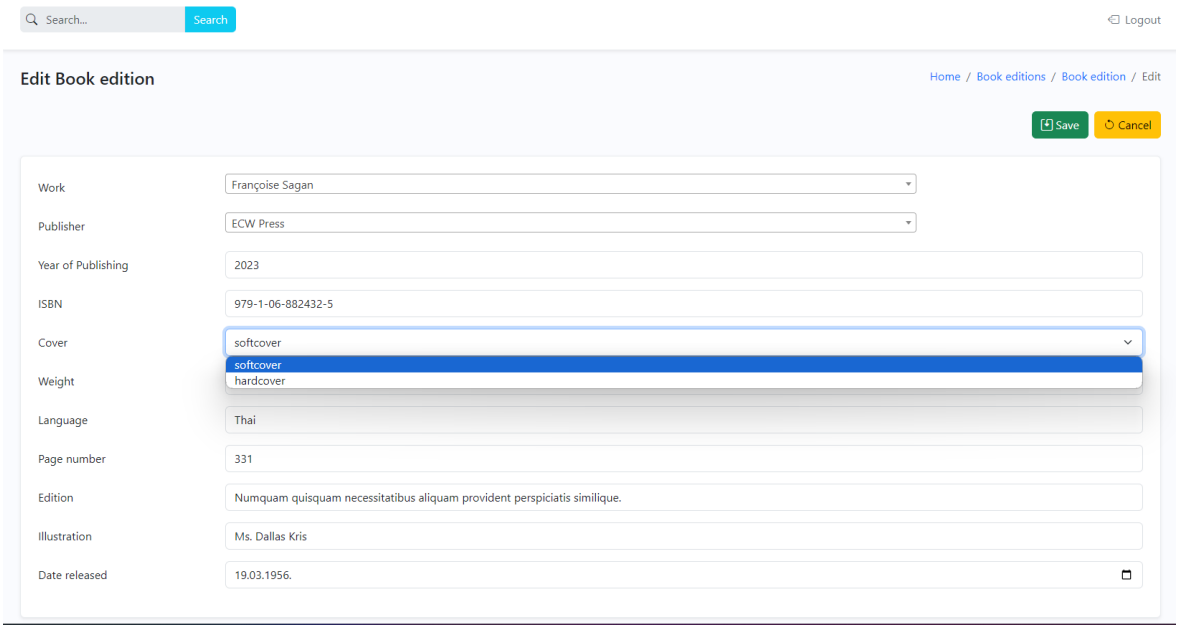
Slika 17: Dodavanje novog književnog djela



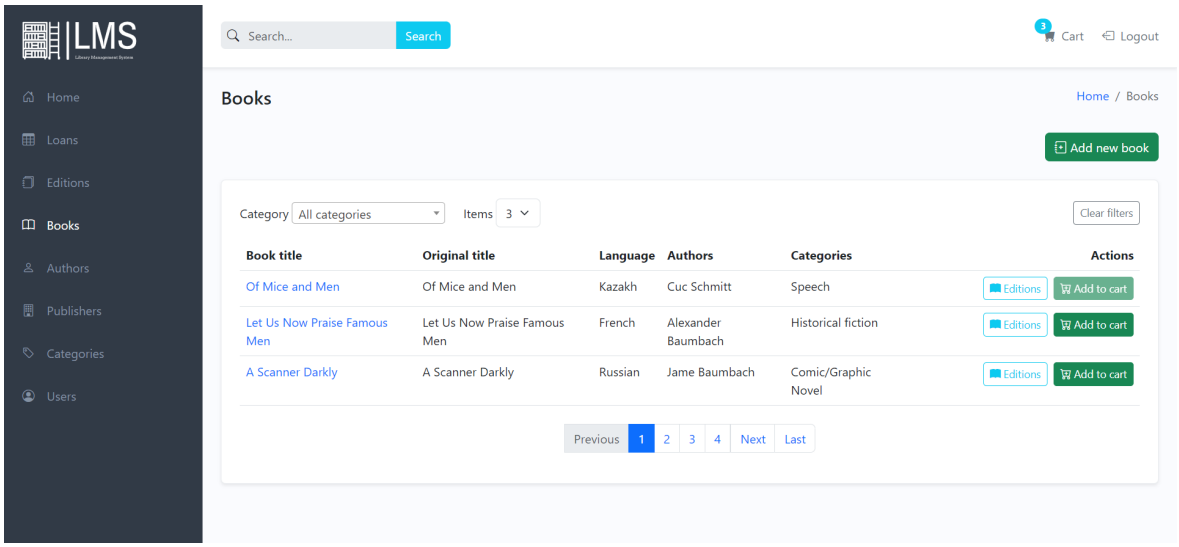
Slika 18: Prikaz izdanja knjige



Slika 19: Primjerci određenog knjižnog izdanja



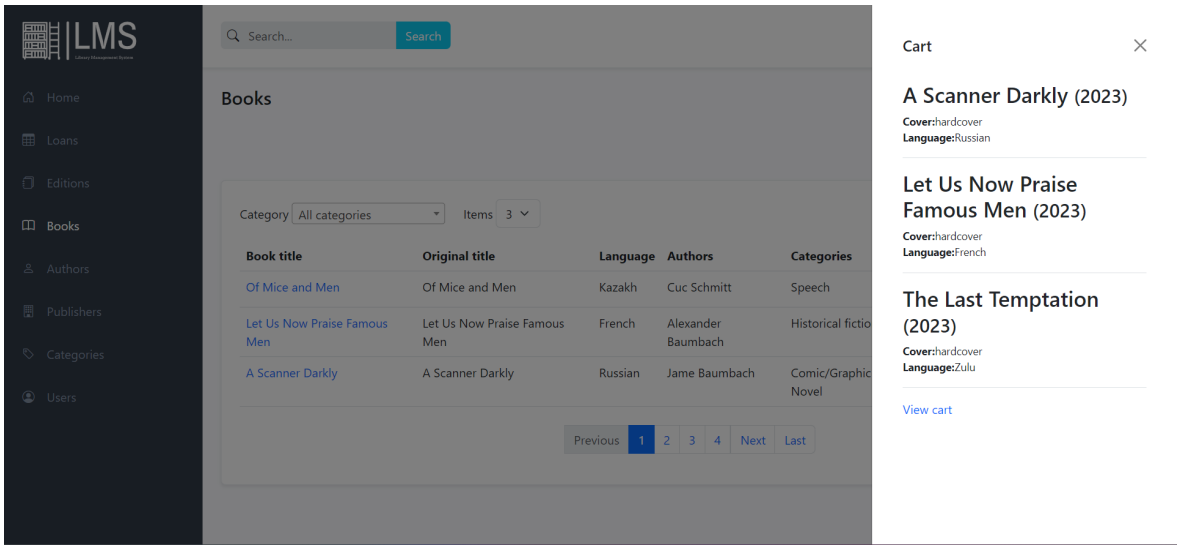
Slika 20: Ažuriranje podataka izdanja



Slika 21: Prikaz knjiga

4.7.4. Rezervacija i posudba knjižnične građe

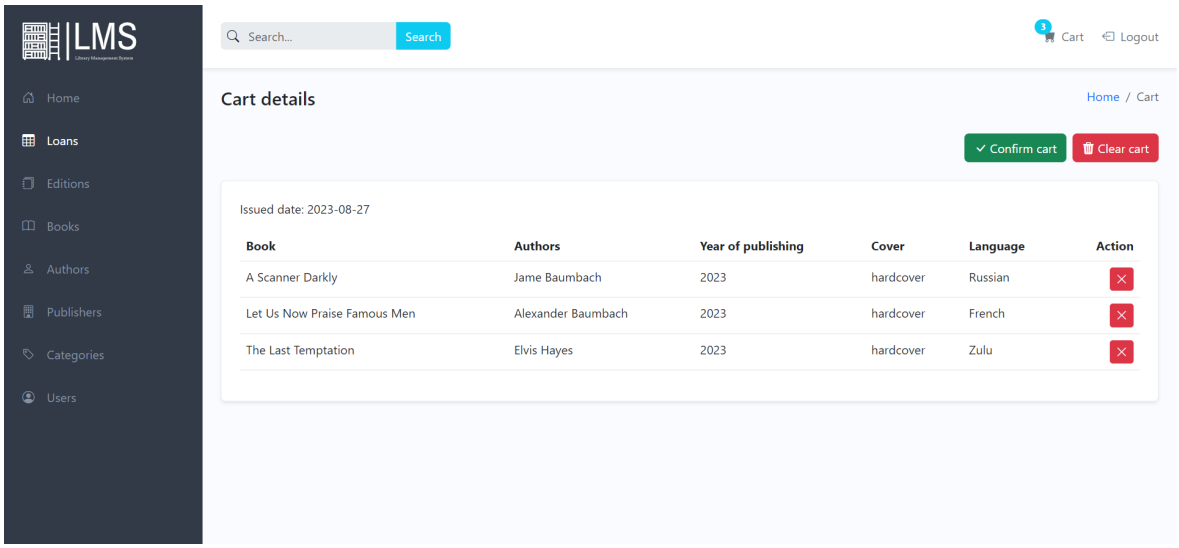
Izanjmljivanje knjižnične građe predstavlja jezgru aplikacije, a realizirano je dodavanjem stavki u košaricu preko botuna na književnom djelu ili izdanju. Pojedina košarica vidljiva je samo korisniku koji ju je započeo. Administratori i knjižničari kreiraju posudbe za određene članove, dok sami članovi imaju mogućnost rezervacije knjiga. U košaricu se može dodati knjižnična građa koja ima bar jedan dostupan primjerak statusa `OK`. Košarica nakon dodavanja odabranih stavki prikazana je na slici 22.



Slika 22: Prikaz košarice s knjigama za posudbu

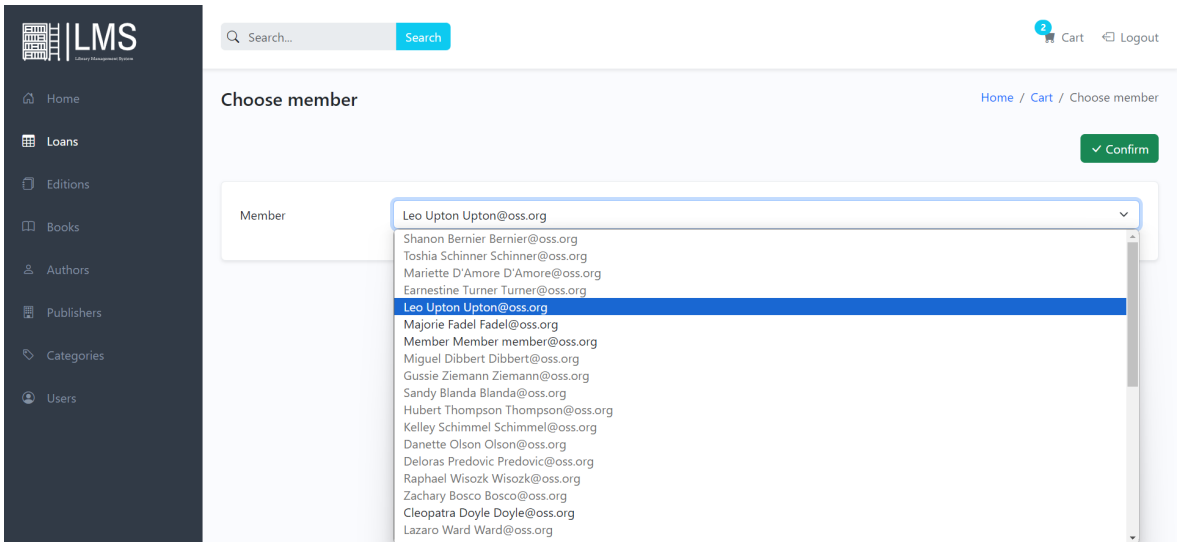
Pojedini član istvoremeno može imati tri posuđena primjerka knjižnične građe te je

također u košaricu moguće maksimalno dodati tri stavke. Botun za dodavanje knjige u košaricu je onemogućen za odabir ako član premaši svoj osobni limit ili odabrana knjiga nema dostupnih primjeraka (pogledati sliku 21). Knjiga je zauzeta ako je posuđena, rezervirana ili se trenutno nalazi u košarici drugog aplikacijskog korisnika. Detalji košarice prikazani su na slici 23.

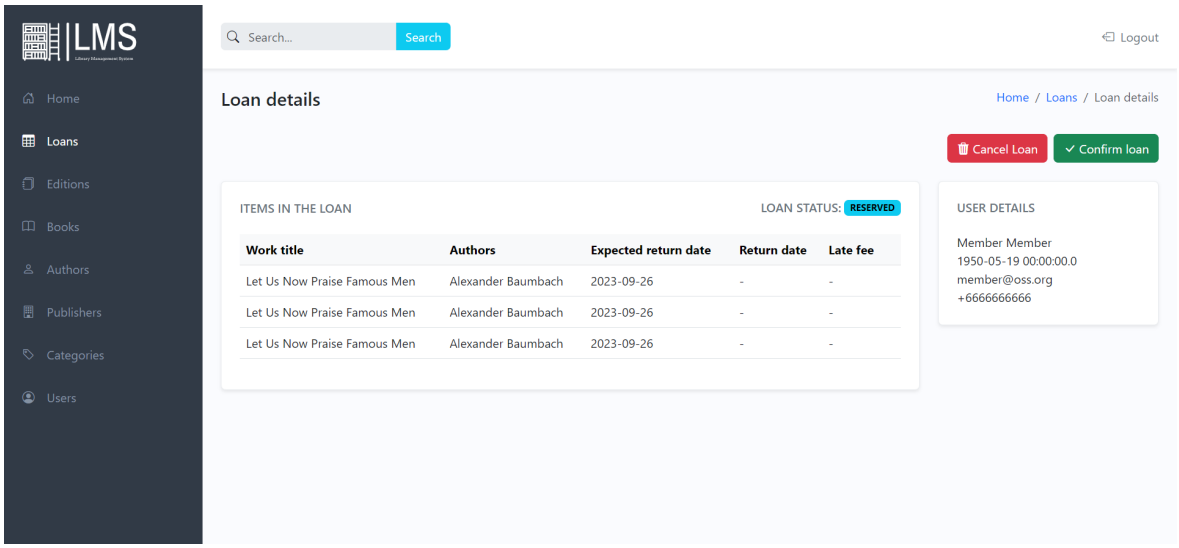


Slika 23: Detalji košarice

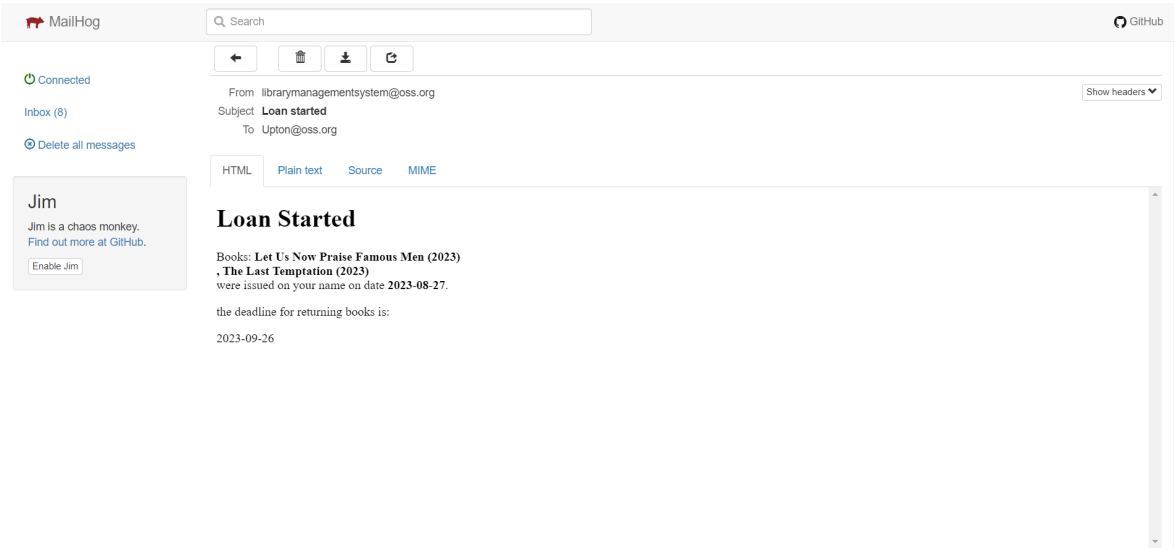
Potvrđivanjem detalja košarice sa slike 23 član rezervira posudbu, a knjižničar i administrator u međukoraku odabiru člana koji posuđuje knjižničnu građu iz košarice. Knjižničari odabiru članove u padajućem izborniku uz mogućnost pretrage. Za odabir su onemogućeni članovi koji imaju tri nevraćene knjige, neobnovljeno članstvo ili aktivnu zakasninu. Odabir članova pri kreaciji posudbe knjižničara ili administratora prikazan je na slici 24. Rezervirana posudba se treba preuzeti u trodnevnom roku ili se rezervacija otkazuje. Otkazivanje rezervacije omogućeno je članu koji je rezervirao posudbu u bilo kojem trenutku. Administratosko sučelje za potvrdu rezervacije prikazano je na slici 25. Obavijesti o članarinama, rezervacijama, posudbama i mogućim zakasninama šalju se elektroničkom poštom. Pretinac elektroničke pošte i primjer poruke prikazan je na slici 26.



Slika 24: Odabir člana za kreiranje posudbe (za knjižničare i administratore)



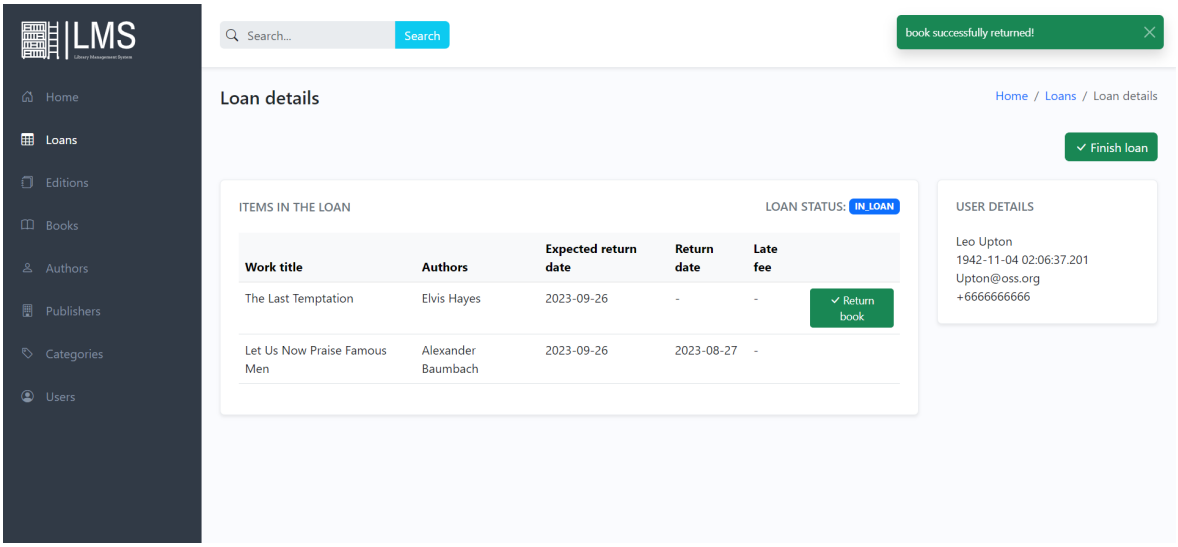
Slika 25: Potvrda rezervirane posudbe (za knjižničare i administratore)



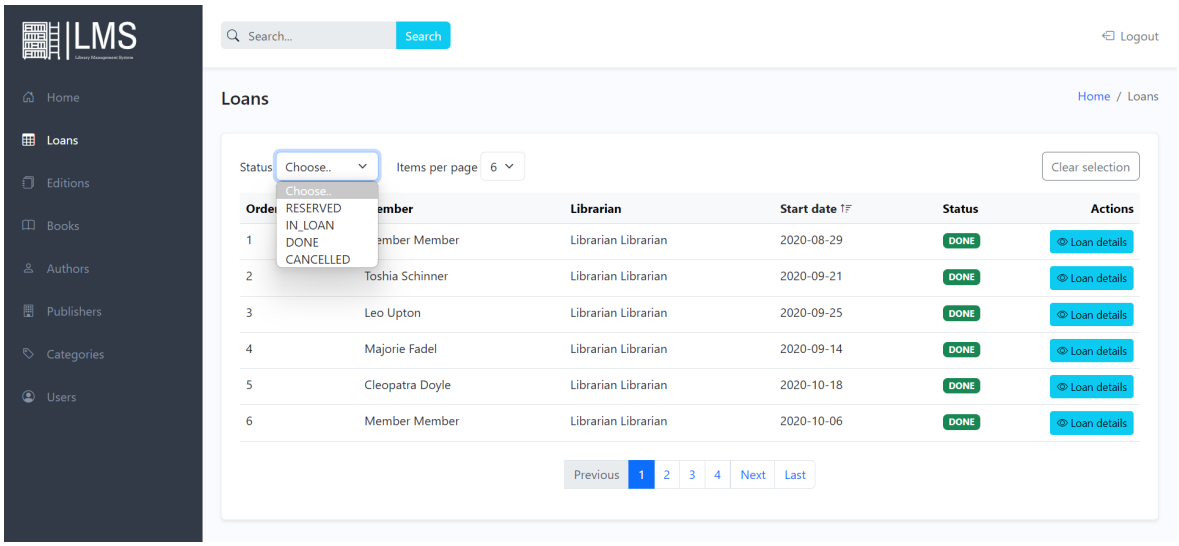
Slika 26: Pretinac elektroničke pošte s primjerom poruke

4.7.5. Vraćanje posuđenih knjiga i obrada zakasnine

U aplikaciji je implementirano vraćanje knjiga s iste posudbe u različito vrijeme. Administratorima i knjižničarima omogućeno je potvrđivanje vraćanja knjižnične građe i naplata zakasnine pojedinačno za svaku stavku s posudbe. Ako korisnik kasni s vraćanjem knjige obračunava se dnevna naknada, a knjiga se može vratiti kada se ukupan iznos plati. Posudba je završena kada se sve knjige vrate te joj je status `DONE` (pogledati sliku 27). Administratorima i knjižničarima omogućen je pregled povijesti posudbi. Članovima je dopušten pristup vlastitim posudbama uz mogućnost filtriranja po statusu i sortiranja prema datumu (pogledati sliku 28).



Slika 27: Detalji posudbe i vraćanje knjiga



Slika 28: Prikaz svih posudbi

5. Zaključak

U ovom završnom radu predstavljena je moderna web aplikacija s ciljem olakšavanja procesa iznajmljivanja knjižnične građe. Objašnjene su korištene tehnologije pri realizaciji, struktura baze podataka te detaljni opis implementacije aplikacijskih značajki popraćen ispisima kôda, slikama i tablicama.

Izrađena Spring Boot aplikacija knjižničarima omogućava upravljanje knjižničnom građom i svim elementima knjižničnog sustava. Grafički prikazani statistički podatci znatno olakšavaju održavanje i povećavanje knjižničnog fonda ovisno o zahtjevima poslovne okoline. Aplikacijski je realiziran proces istodobnog iznajmljivanja većeg broja primjeraka knjižnične građe. Članovima je pružena mogućnost rezervacije knjiga uz prikaz povijesti vlastitih posudbi i statističkih podataka s ciljem njihove integracije kako bi aktivno sudjelovali u poslovnim procesima knjižnice.

Aplikacija implementira sve unaprijed postavljene zahtjeve, a po potrebi omogućava brzo i lako proširenje dodatnim značajkama. U aplikaciji je realizirana osnova za plaćanje članarina i potencijalnih zakasnina ukoliko se knjižnična građa ne vrati u predviđenom vremenu. Shodno tome, aplikaciju je moguće proširiti realizacijom stvarnog procesa plaćanja uz evidenciju računa.

Koristeći razvojni okvir Spring Boot olakšano je postavljanje projekta uz daljnju implementaciju poslovne logike. Spring Security gotovim funkcionalnostima osigurava sigurnost, autentikacijski sustav i korisničku forme za prijavu. Povezivanje pozadinski generiranih podataka i HTML predložaka realizirano je pomoću alata Thymeleaf kompatibilnog s razvojnim okvirom Spring Boot. Korištenjem Bootstrapa i JavaScripta omogućena je izrada dinamičkog, modernog korisničkog sučelja, a Chart.js biblioteka pruža atraktivan prikaz statističkih podataka pomoću grafikona i dijagrama.

Literatura

[1] International Business Machines Corporation, "IBM - Java," https://www.ibm.com/topics/java, posjećeno 6.8.2023.

[2] https://github.com/spring-projects/spring-boot, posjećeno 5.8.2023.

[3] International Business Machines Corporation, "IBM - Java Spring Boot," https://www.ibm.com/topics/java-spring-boot, posjećeno 6.8.2023.

[4] The Apache Software Foundation, "Apache Maven," https://maven.apache.org/, posjećeno 8.8.2023.

[5] Tarnum Java SR, "Baeldung - Spring Security," https://www.baeldung.com/security-spring, posjećeno 14.8.2023.

[6] The Thymeleaf Team, "Thymeleaf - Spring Boot," https://www.thymeleaf.org/doc/tutorials/3.0/thymeleafspring.html, posjećeno 14.8.2023.

[7] ——, "Thymeleaf - Spring Security," https://www.thymeleaf.org/doc/articles/springsecurity.html, posjećeno 14.8.2023.

[8] Docker Inc, "Docker - containers," https://www.docker.com/resources/what-container/, posjećeno 12.8.2023.

[9] ——, "Docker overview," https://docs.docker.com/get-started/overview/, posjećeno 12.8.2023.

[10] Tarnum Java SR, "Baeldung - Flyway," https://www.baeldung.com/database-migrations-with-flyway, posjećeno 11.8.2023.

[11] Select2 Team, "Select2," https://select2.org/, posjećeno 6.8.2023.

[12] Bootstrap Team, "Bootstrap," https://getbootstrap.com/docs/5.3/getting-started/introduction/, posjećeno 6.8.2023.

[13] Z. Torba, *Baze podataka*. Veleučilište u Splitu, 2001.

[14] VMware Tanzu, "Spring - repositories," https://docs.spring.io/pring-data/jpa/docs/current/reference/html/#repositories, posjećeno 18.8.2023.

[15] Tarnum Java SR, "Baeldung - Jpa and Hibernate ," https://www.baeldung.com/learn-

jpa-hibernate, posjećeno 6.8.2023.

[16] ——, “Baeldung - Java Faker,” https://www.baeldung.com/java-faker, posjećeno 6.8.2023.

[17] ——, “Baeldung - Spring Boot services,” https://www.baeldung.com/spring-component-repository-service, posjećeno 16.8.2023.

[18] ——, “Baeldung - Spring Boot controllers,” https://www.baeldung.com/spring-controllers, posjećeno 16.8.2023.

[19] VMware Tanzu, “Spring - authorization,” https://docs.spring.io/spring-security/reference/servlet/authorization/index.html, posjećeno 20.8.2023.

[20] ——, “Spring - SecurityfilterChain,” https://docs.spring.io/spring-security/site/docs/3.0.x/reference/security-filter-chain.html, posjećeno 20.8.2023.

[21] Tarnum Java SR, “Baeldung - Scheduled Tasks,” https://www.baeldung.com/spring-scheduled-tasks, posjećeno 21.8.2023.

[22] ——, “Baeldung - Cron expressions,” https://www.baeldung.com/cron-expressions, posjećeno 21.8.2023.

[23] Chart.js Team, “Chart.js,” https://www.chartjs.org/docs/latest/, posjećeno 22.8.2023.

Dodatci

Popis slika

1 Aktivni Docker spremnici 7
2 Shema baze podataka 11
3 Spring Initializr web sučelje 18
4 Prikaz kreiranja upita pomoću JPA repozitorija 31
5 Proces spremanja objetka u bazu podataka korištenjem ORM-a 32
6 Prijem mapiranja Java objekta u tablicu baze podataka i obratno 33
7 Odbijen pristup na rutu za koju korisnik nema potrebne ovlasti 43
8 Prikaz formata cron izraza 44
9 Forma za prijavu 46
10 Nadzorna ploča s bočnom navigacijskom trakom 47
11 Godišnji statistički prikaz posudbi i zakasnina 47
12 Statistički prikaz broja članova po životnoj dobi 48
13 Prikaz svih korisnika za administratore 50
14 Detalji korisnika 50
15 Registracija novih korisnika 51
16 Prikaz skočnog prozora pri produljenju članstva 51
17 Dodavanje novog književnog djela 52
18 Prikaz izdanja knjige 52
19 Primjerci određenog knjižnog izdanja 53
20 Ažuriranje podataka izdanja 53
21 Prikaz knjiga 54
22 Prikaz košarice s knjigama za posudbu 54
23 Detalji košarice 55
24 Odabir člana za kreiranje posudbe (za knjižničare i administratore) 56
25 Potvrda rezervirane posudbe (za knjižničare i administratore) 56
26 Pretinac elektroničke pošte s primjerom poruke 57
27 Detalji posudbe i vraćanje knjiga 57
28 Prikaz svih posudbi 58

Popis tablica

1 Entitet `user` . . . 12
2 Entitet `role` . . . 12
3 Entitet `membership` . . . 13
4 Entitet `loan` . . . 13
5 Entitet `loan_details` . . . 14
6 Entitet `late_fee` . . . 14
7 Entitet `book_copy` . . . 15
8 Entitet `book_edition` . . . 15
9 Entitet `publisher` . . . 16
10 Entitet `work` . . . 16
11 Entitet `author` . . . 17
12 Entitet `category` . . . 17

Popis ispisa kôda

1 Prikaz metode za pokretanje Spring Boot aplikacije . . . 4
2 Konfiguracija servisa u `compose.yaml` datoteci . . . 6
3 Dodavanje Flyway ovisnosti u `pom.xml` . . . 22
4 Dodavanje i konfiguracija Flyway Maven Plugina u `pom.xml` . . . 22
5 Dio kôda migracije `V1__Create_author_table.sql` . . . 23
6 Prikaz klase entiteta `loan` . . . 25
7 Prikaz klase entiteta `user` . . . 27
8 Prikaz DTO klase `book_edition` . . . 29
9 Prikaz repozitorija `LoanRepository` . . . 30
10 *Spring Bean* za populiranje baze podataka . . . 33
11 Metoda za punjenje podataka tablice `work` . . . 34
12 Prikaz dijela servisa `CategoryServiceImpl` . . . 35
13 Primjer korištenja `@GetMapping` anotacije u kontroleru `LoanController` 37
14 Primjer korištenja `@PostMapping` i `@ModelAttribute` anotacija . . . 38
15 Prikaz Spring Security sučelja `UserDetails` . . . 39
16 Spring Security konfiguracijska klasa . . . 40

17 Primjer korištenja @PreAuthorize anotacije 42
18 Prikaz korištenja autorizacije pomoću Thymeleafa 42
19 Primjer korištenja anotacije `@Scheduled` i cron izraza 45
20 Kreiranje kružnog grafikona 48