ВИСОКА ТЕХНИЧКА ШКОЛА СТРУКОВНИХ СТУДИЈА У НОВОМ САДУ
СТУДИЈСКИ ПРОГРАМ: МАСТЕР ИНФОРМАЦИОНЕ ТЕХНОЛОГИЈЕ

Коначно поена:

Софтверско инжењерство

Предспитни пројекат

“Private Library”

Студенти:

Катарина Грковић МИТ 19/23
Лука Гаћа МІТ 8/23
Ненад Тубић МИТ 5/23

Ментори:

професор: др Тања Крунић
асистент: мр Нинослава Тихи

Телетубићи

Нови сад, 2023/24

Питања://*оставите празну страницу за одговоре*

Пројекат

Задатак: Израда групног пројекта кпји се брани путем power point презентације или докуемнтације у виду доцџ фајла којемсе претходно шаљу асистенту на емаил. На пројекту се може освојити до 50 поена, а за предиспитне обавезе је минимално неопходно 30. Пројектни задатак подразумева израду сајта или апликације (по договору) користећи агилне методе развоја софтвера (scrum) кпји се води у радном окружењу Jira, а сарадња међу девелоперима преко Git-a. Софтвер се развија у технологијама по жељи при чему је потребно на самом почетку дефинисати корисничке захтеве (финкционалне и нефункционалне) и моделирати случајеве коришћења. Такође је потребно урадити дијаграм класа, активности и ЕР дијаграм базе. На крају је потребно тестирати апликацију или сајт помоћу Селениум алата или било ког другог алата.

До 10 поена: Дати су кориснички захтеви - функционални и нефункционални

До 10 поена: Урађени су дијаграми коришћења (барем 2-3), дијаграм класа, дијаграм активности, ЕР дијаграм

До 10 поена: приказан је код и база података који су урађени у технологијама по жељи

До 10 поена: приказани су скриншотови тестова

До 10 поена: приказани су скриншотови спринтова и докази сарадње на пројекту из Git-a.

Укупно се на пројекту може освојити 50 поена.

За све горе наведен обезбедити screenshot.

Коментари у коду нису дозвољени. Код пројекта се брани усмено.

САДРЖАЈ

1. УВОД

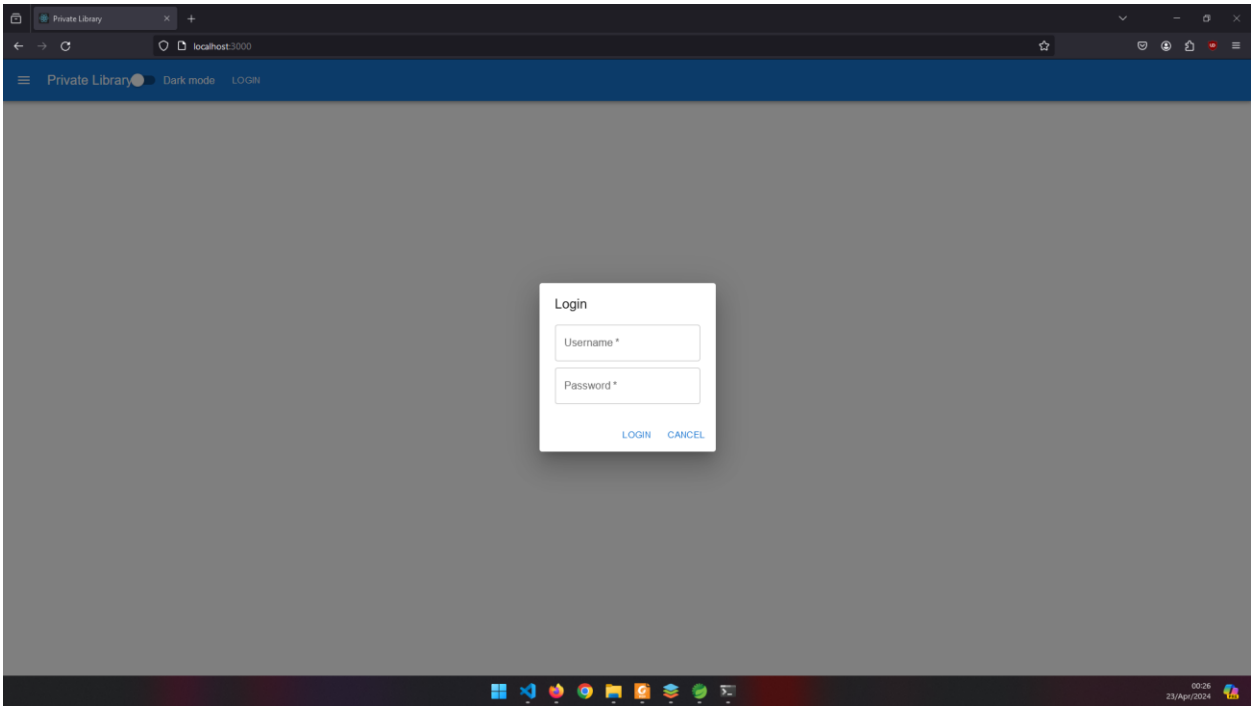
1.1. Резиме

Циљ и сврха апликације је имплементација веб апликације за приватну библиотеку, те се и сама апликација зове “Приватна библиотека”. Будући да је у питању приватна библиотека није неопходна функционалност издавања књига или сложенији систем категорије корисника, као и сигурносних мера, те постоје две роле корисника администратор и корисник, али само са стране фронтенда.

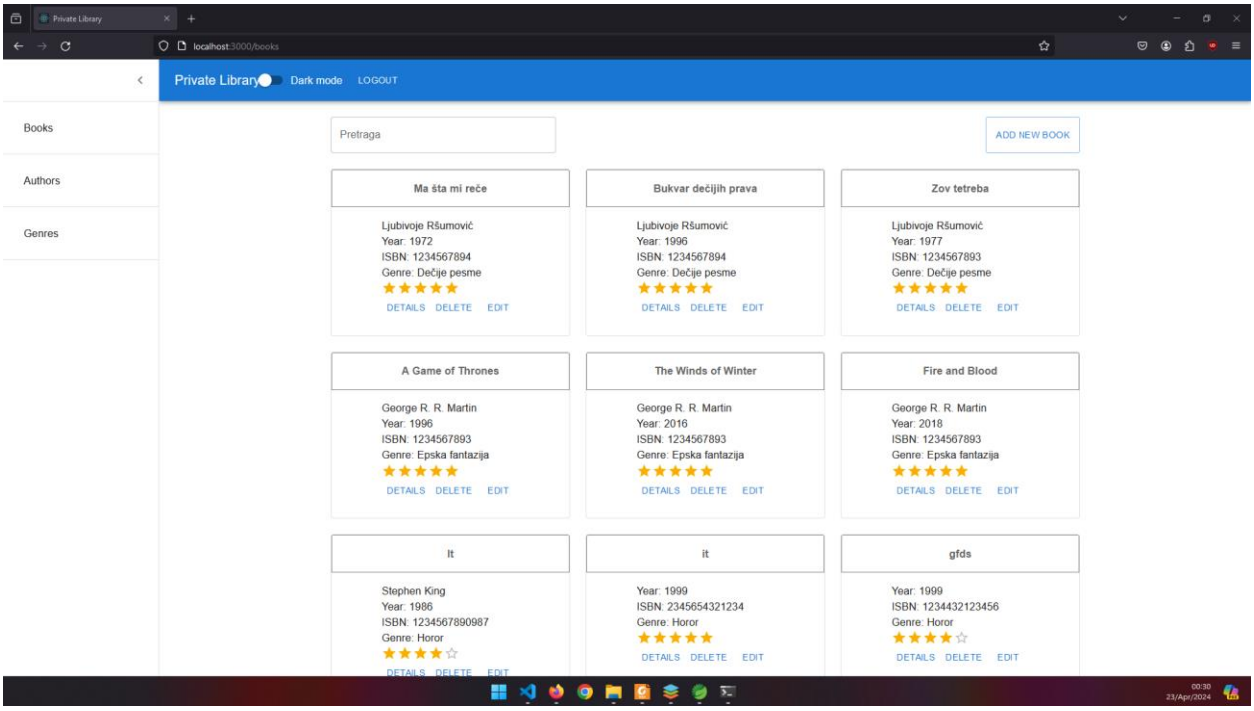
Основни задаци које апликација треба да врши је:

- Уписивање књига
- Преглед књига
- Уређивање књига
- Брисање књига

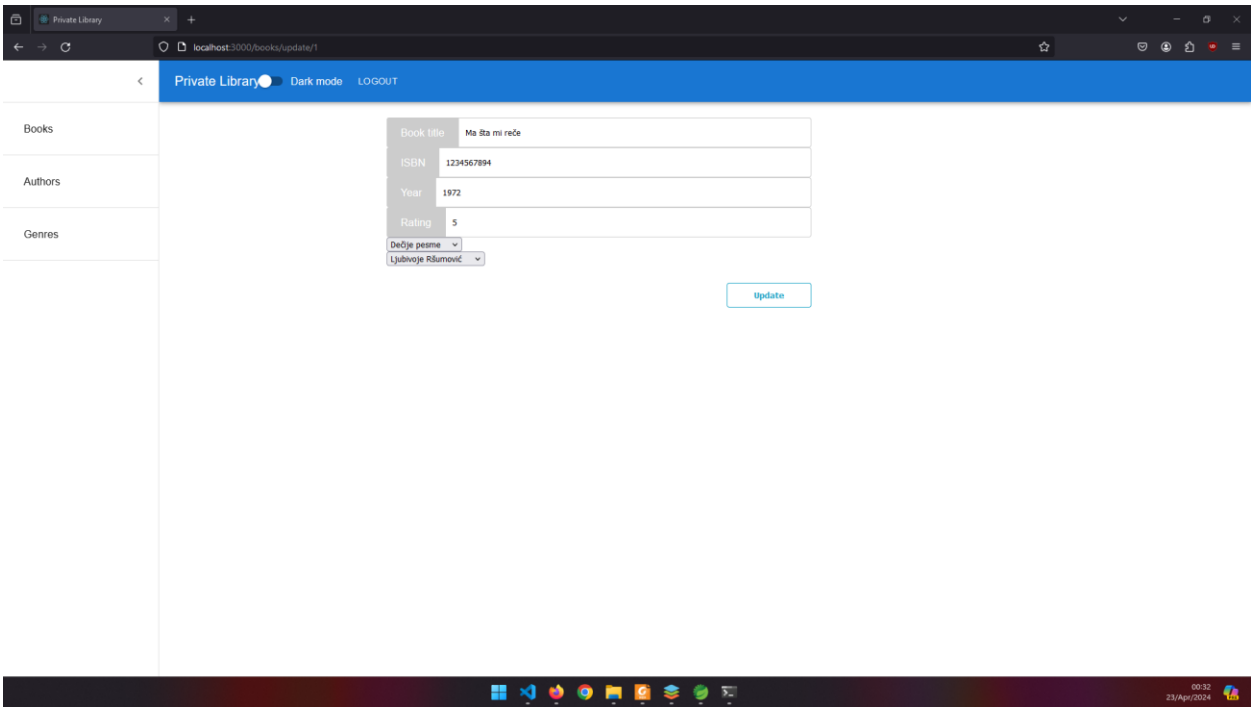
Имплементиране су и додатне функционалности како на ентитету Књига, тако и над ентитетима Аутор и Жанр.



Слика 1. Почетна страница апликације са отвореним модалом за логовање



Слика 2. Страница са приказом књига и опцијама за уписивање, преглед, брисање и уређивање



Слика 3. Након клика на дугме EDIT отвара се страница за уређивање књига

1.2. Намена апликације и циљна група

Апликација је намењена оним корисницима који поседују већу колекцију књига, односно приватну библиотеку, како би могли да имају приступ евиденцији књига, као и други корисници исте, такви корисници који поседују велике колекције књига су и циљана група за коришћење ове апликације. Апликација омогућава брз и једноставан преглед књига које су евидентиране у библиотеци, претраживање, уписивање књига, преглед књига, уређивање књига, као и брисање књига. Такође, исто је омогућено и над ентитетима Аутор и Жанр.

1.3. Категорије корисника

Због специфичности намене апликације, сасвим је довољно да апликација има два корисника са две различите роле, а то су администратор и корисник. Конкретно у овом случају то је решено на нивоу фронтенда са логиком која то решава, те уколико је улогован корисник са администраторском улогом, моћи ће да има све функционалности апликације, разуме се тај налог би користио сам власник библиотеке. Уколико је обичан корисник улогован у апликацију, он ће бити лимитиран функционалностима исте, конкретно моћи ће да прегледа ентитете и претражује, али неће моћи да додаје, уређује или их брише. Приступ овом налогу ће бити омогућен неком другом лицу од стране власника библиотеке, те су из тог разлога ограничене функционалности на том налогу.

2. ЗАХТЕВИ

2.1 Функционални захтеви

Обичан корисник има ограничен приступ апликацији. Конкретно, приликом одабира неког од ентитета, рецимо Књиге (Books), тада може само да прегледа књиге, да врши претрагу и преглед појединачне изабране књиге, али не може да врши додавање/уписивање књиге, брисање или уређивање/ажурирање књиге. У случају ентитета Аутори (Authors), такође наведени корисник може само да прегледа који су аутори у систему, све то али без функционалности додавања, ажурирања или брисања. Наравно што се тиче ентитета Жанрови (Genres), ситуација је слична. У овом случају корисник има само функционалност прегледа жанрова, наравно без функционалности додавања, ажурирања и брисања.

Администратор има на располагању све функционалности над свим ентитетима (Књиге/Books, Аутори/Authors, Жанрови/Genres). Те функционалности подразумевају уписивање, преглед, ажурирање и брисање. Наравно, присутне су и додатне функционалности за претрагу, преглед појединих елемената ентитета, конкретно преглед одређене књиге што се тиче ентитета Књиге.

2.2 Нефункционални захтеви

Будући да је веб апликација намењена корисницима који имају приватну колекцију књига, самим тиме се намеће да ће бити имплементирана на интернету, што је у пракси локална рачунарска мрежа у просторијама власника приватне библиотеке.

Самим тиме акценат није на сигурности, будући да је систем на тај начин изолован од спољних корисника, а приступ апликацији омогућен искључиво физичким присуством корисника на некој од радних станица – терминала.

Власник библиотеке и апликације омогућава приступ спољним корисницима тако што ће им дати потребне креденцијале за приступ истој.

Пошто је аплликација намењена између осталог и за кориснике који ће бити посетиоци библиотеке, кориснички интерфејс треба да буде једноставан и интуитиван.

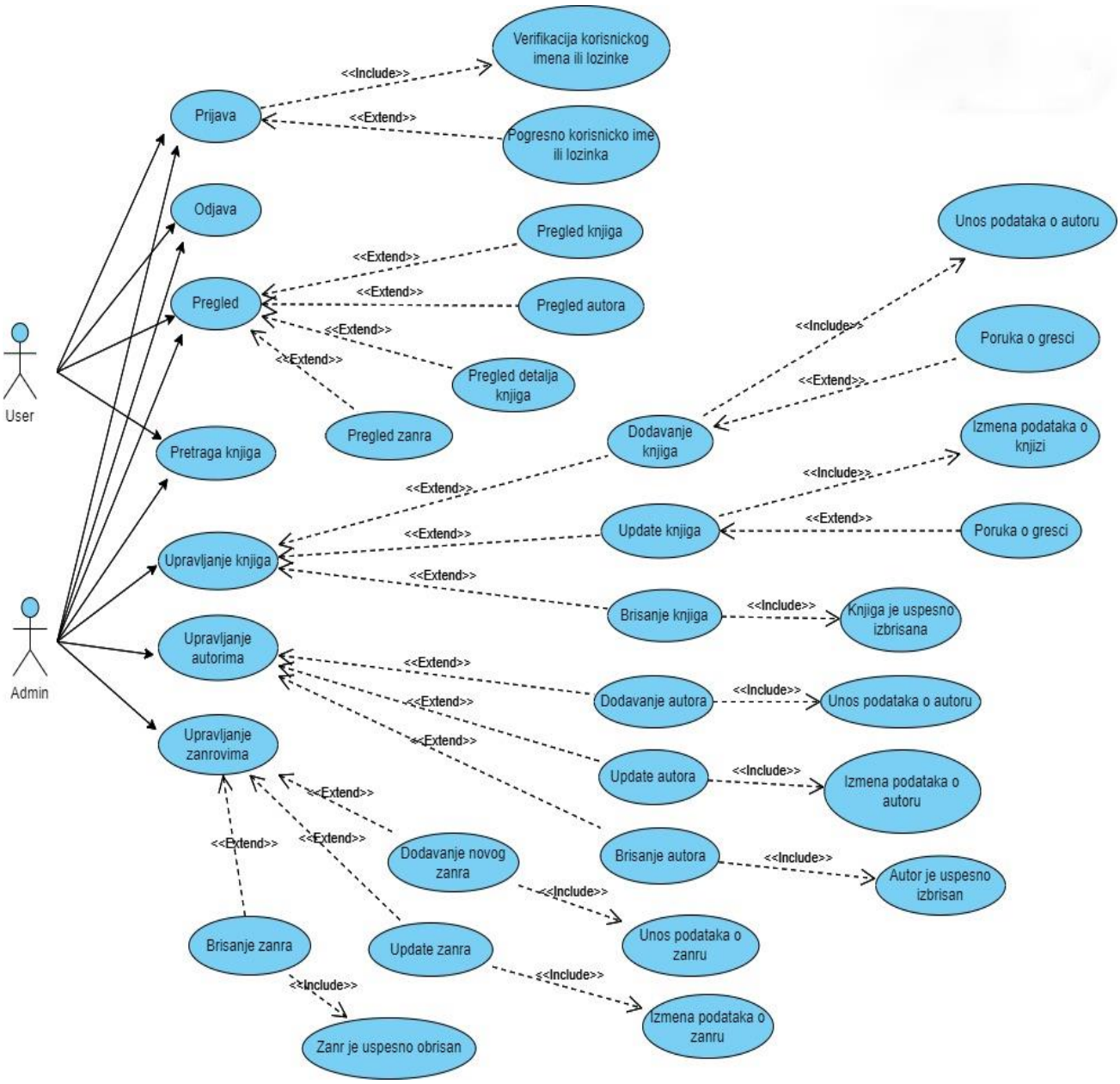
Што се тиче одржавања апликације, требало би да буде лако одржавана и надограђивана, у смислу перформанси, функционалности и сигурности, уколико се појаве потребе за истим.

3. UML МОДЕЛ АПЛИКАЦИЈЕ

3.1. Дијаграм случаја коришћења

Табела 1. Пример три случаја коришћења аутентификације корисника

Случај коришћења	Опис	Учесник	Услов	Резултат
СК1	Логовање	Корисник	Корисник мора унети исправно корисничко име и лозинку	Корисник има приступ
СК2	Управљање књигама	Администратор	Верификована улога администратора	Администратор може додавати, уређивати или брисати књиге
СК3	Управљање жанровима	Администратор	Верификована улога администратора	Администратор може додати, уређивати или брисати жанрове књига из система



Слика 4. Пример Use case дијаграма за систем управљања библиотеке

3.2. Дијаграм класа

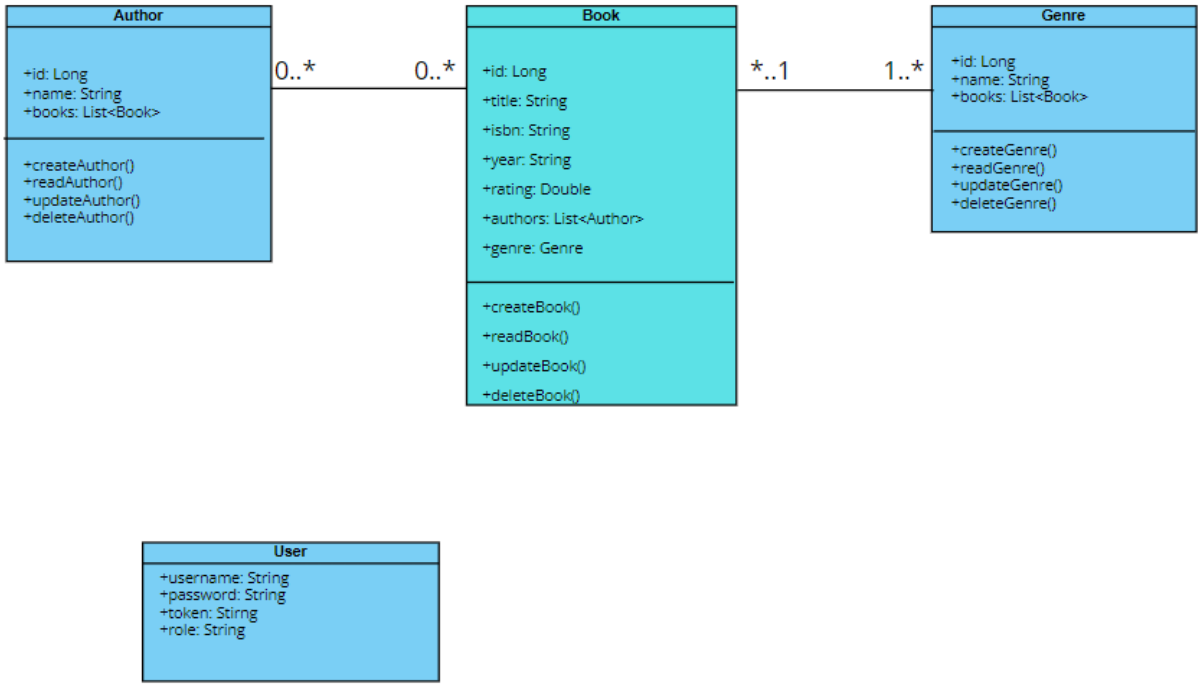
Апликација има четири класе:

- Класу Author
- Класу Book
- Класу Genre
- Класу User

Класа User је независна од класа Author, Book и Genre, што значи да се она користи на frontendu за интеракцију с корисником.

Author – Book: веза типа „0..*-то -0..*“ означава да један аутор може бити повезан с ниједном, једном или више књига, а једна књига може имати ниједног, једног или више аутора, то је (many-to-many relationship).

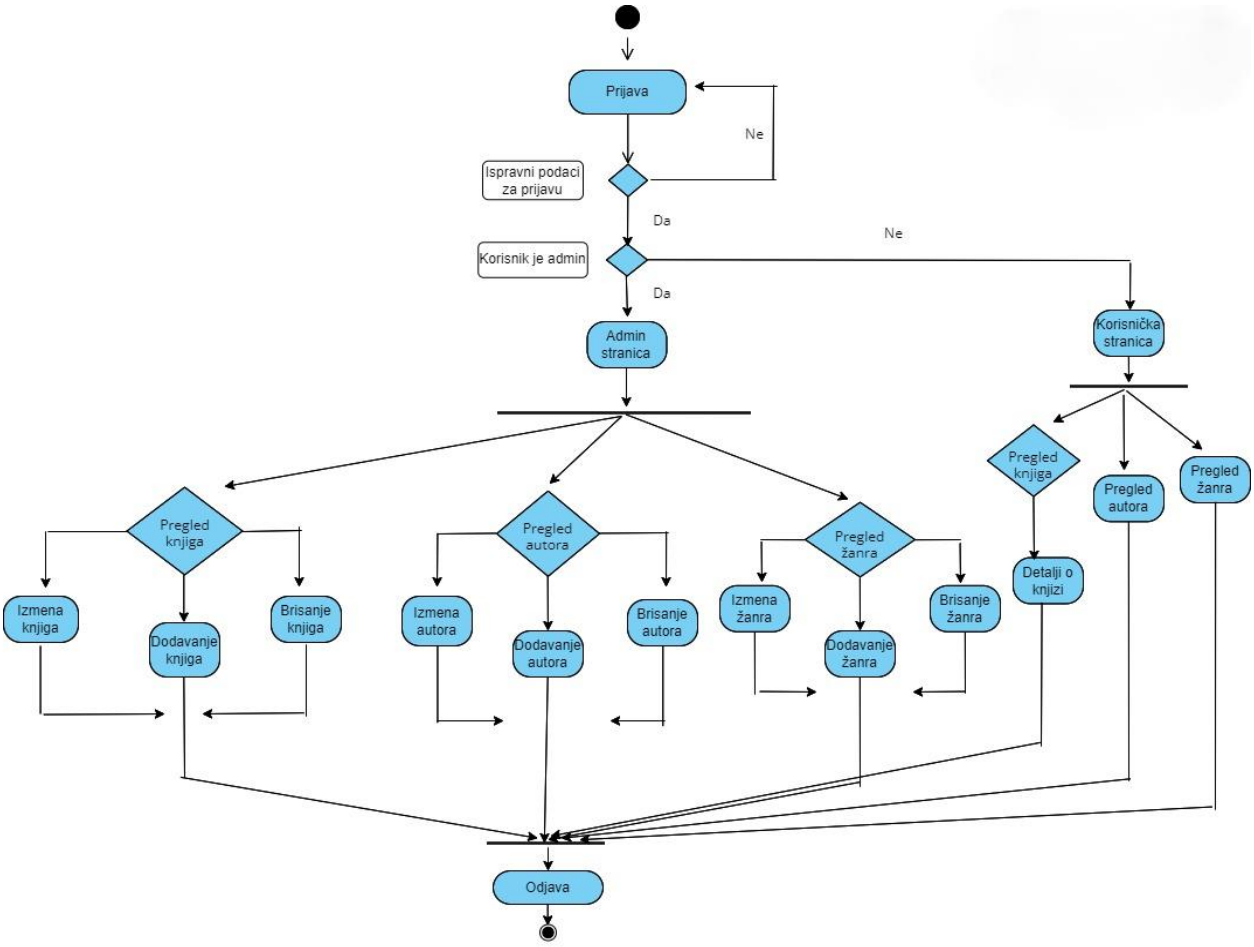
Book - Genre: веза типа „*..1-то-1..*“ што значи да књига мора имати тачно један жанр (обавезна веза типа one-to-one с жанром), док један жанр може обухватити једну или више књига. Ово је веза one-to-many из перспективе жанра према књигама.



Слика 5. Дијаграм класа

3.3. Дијаграм активности

Дијаграм активности моделира кораке који се одвијају у интеракцији корисника са системом управљања библиотеком након иницијалне пријаве. Процес почиње када корисник иницира акцију пријаве. Постоји провера да ли су унети подаци за пријаву исправни. Ако је корисник препознат као админ, приступа се админ страници. На админ страници, администратор има одређене могућности. Након обављања административних задатака, админ се може одјавити. Ако корисник није админ, прелази се на корисничку страницу. На корисничкој страници, корисник има одређене опције. Ако подаци за пријаву нису исправни, корисник остаје на страници за пријаву како би покушао поново. Без обзира на пут, и админ и корисник имају опцију за одјаву. На дијаграму су јасно означене гране активности које се одвијају након одређених услова.



Слика 6. Дијаграм активности који моделира кораке који се одвијају у интеракцији корисника са системом управљања библиотеком након иницијалне пријаве

3.4. ЕР дијаграм

Дати спецификацију базе података преко ЕР дијаграма који треба да буде почетни модел базе. Дати у кратким цртама опис ентитета и њихових атрибута, примарних и страних кључева и релација између тих ентитета. Описати која врста базе података ће се користити и зашто.

ЕР модел релацијске базе података са ентитетима book, author и genre, са спојеном табелом book_author. Ово је релациона база података која користи структуиран упитни језик (SQL) због њихове способности да ефикасно управљају сложеним упитима и релацијама између подацима.

Ентитет Book има атрибуте:

1. id – примарни кључ
2. genre_id – страни кључ (повезује са genre)
3. isbn – међународни стандардни број књиге
4. rating – просечна оцена књиге
5. title- наслов књиге
6. year – година издавања

Ентитет Author има атрибуте:

1. id – примарни кључ (јединствени идентификатор аутора)
2. name – име аутора

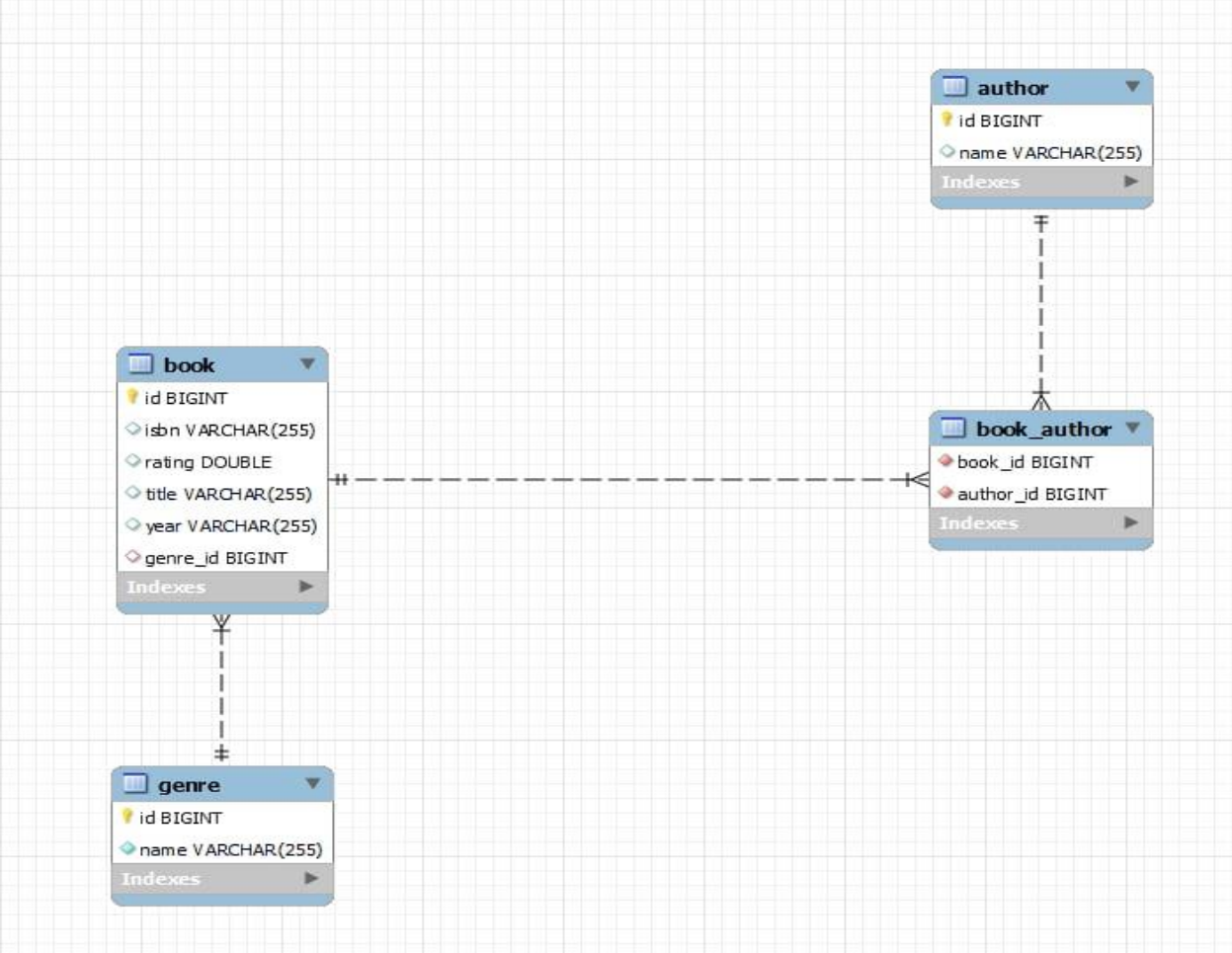
Ентитет Genre има атрибуте:

1. id – примарни кључ (јединствени идентификатор жанра)
2. name – име жанра

Ентитет Book_author има атрибуте: (спојна табла са релацију H-M)

1. book_id – страни кључ (повезује са book)
2. author_id - страни кључ (повезује са author)

Ентитети Book и Genre релација (many-to-one) где више књига може бити повезано са једним жанром. Ентитети Book и Author релација (many-to-many) где више књига може бити повезано са више аутора и обрнуто. Коришћење релацијске базе података је оптимано за овакав тип система јер омогућава строгу структуру и јасне релације међу подацима, што олакшава интегритет и консистентност података, као и сложене упите који су неопходни за претрагу и манипулацију подацима књигама, жанровима и ауторима.



Слика 7. ЕР дијаграм

id	isbn	rating	title	year	genre_id
1	1234567894	5	Ma šta mi reče	1972	1
2	1234567894	5	Bukvar dečijih prava	1996	1
3	1234567893	5	Zov tetreba	1977	1
4	1234567893	5	A Game of Thrones	1996	2
5	1234567893	5	The Winds of Winter	2016	2
6	1234567893	5	Fire and Blood	2018	2
7	1234567899	5	It	1986	3

Слика 8. Приказ тебеле Књига из базе података

4. КОРИШЋЕНЕ ТЕХНОЛОГИЈЕ

4.1 MySQL Community

MySQL Community је бесплатна, отворена база података која се широко користи у свету софтверског развоја. Овај систем за управљање базама података (DBMS) има богат историјат и пружа низ техничких карактеристика које га чине популарним избором за различите врсте апликација.

MySQL Community нуди широк спектар функционалности које задовољавају потребе различитих апликација. Његове кључне техничке карактеристике укључују:

- Релациони модел података са подршком за SQL језик.
- Могућност управљања огромним количинама података.
- Високу поузданост и скалирање.
- Подршку за различите оперативне системе, укључујући Windows, Linux и macOS.
- Свестране опције за репликацију података и сигурносне механизме.
- Ефикасан рад са великим оптерећењем и високом доступношћу.

Међу предностима MySQL Community-ја су:

- Бесплатна доступност и open-source модел, што га чини доступним за развојне тимове свих величина.
- Велика заједница корисника и развијалаца (програмера) која пружа подршку, ресурсе и додатне алате.
- Доказана поузданост и перформансе у широком спектру апликација, од веб сајтова до пословних система.

Међутим, постоје и одређене мане:

- Неке напредне функције могу бити ограничене у бесплатној верзији MySQL Community-ја, што може захтевати плаћене додатке или прелазак на комерцијалне верзије.
- Захтева вештину и знање за оптимално подешавање и управљање перформансама, посебно у окружењима са високим оптерећењем.
- Конкуренција у виду других DBMS-ова, као што су PostgreSQL или MongoDB, који могу имати специфичне предности у одређеним сценаријима употребе.

MySQL Community је моћан, флексибилан и популаран систем за управљање базама података који пружа широк спектар функционалности и подржава различите врсте апликација, уз подршку велике заједнице корисника и развијалаца (програмера).

4.2 Spring Boot

Spring Boot је популаран фрејмворк за развој Јава апликација који олакшава и убрзава процес изградње робустних, скалабилних и ефикасних софтверских решења.

Spring Boot нуди многе техничке карактеристике које га чине атрактивним за развој апликација:

- Аутоматско конфигурисање: Spring Boot аутоматски конфигурише многе делове апликације на основу конвенција и стандардних поставки, што олакшава почетак рада.
- Уграђени сервер: Долази са уграђеним сервером, као што су Tomcat, Jetty или Undertow, што елиминише потребу за екстерним сервером за покретање апликације.
- Велики екосистем: Има богат екосистем проширења (стартера) који олакшава интеграцију са различитим технологијама и алатима, као што су базе података, безбедност, RESTful сервиси, итд.
- Микросервиси: Подржава развој микросервисних архитектура, омогућавајући модуларност и скалабилност апликација.

Једна занимљивост везана за Spring Boot је да је настао као реакција на све већу потребу за поједностављивањем развоја Spring апликација, пружајући програмерима алате и окружење које убрзава процес развоја и олакшава одржавање апликација.

Укратко, Spring Boot је моћан и популаран фрејмворк за развој Јава апликација који нуди многе предности у брзом и ефикасном развоју софтвера, уз подршку јаког екосистема и заједнице корисника.

4.2.1 Микросервисна архитектура

Микросервисна архитектура је приступ развоју софтвера који се фокусира на изградњу апликација као скупа малих, независних сервиса, при чему сваки сервис обавља специфичан посао. Уместо да се апликација састоји од једне монолитне целине, микросервиси омогућавају да се апликација разбије на више мањих и флексибилних компоненти, које се могу развијати, тестирати, деплојовати и скалирати независно једна од друге. Сваки микросервис може имати своју базу података, свој програмски језик, технологију или чак тим који га одржава. Ова архитектура омогућава агилнији развој, лакше одржавање и скалирање апликација.

Кључна предност микросервисне архитектуре је њен модуларни приступ који омогућава лакшу разградњу сложених система на мање компоненте, што олакшава развој, тестирање и управљање апликацијама. Такође, омогућава већу флексибилност и скалираност, јер сваки сервис може бити деплојован, скалиран или замењен независно од других. Међутим, имплементација микросервисне архитектуре може бити изазовна у погледу управљања комуникацијом између сервиса, конзистентности података и комплексности конфигурације. Ипак, уз правилно планирање и алате, микросервиси могу значајно унапредити агилност и перформансе апликација.

4.2.2 Развој веб апликације у Spring Boot-u

Развој веб апликације Приватна библиотека је рађен по узору на миркосервисну архитектуру, с тим да је пословна логика смештена у саме контролере, ради лакшег прегледа, будући да је у питању мањи пројекат. Контролери су RESTFUL, што говори у прилог томе да је пројекат рађен по узору на микросервисну архитектуру. У оквиру овог одељка биће приказане кључне класе везане за развој апликације, такође су присутне и DTO класе, као и репозиторијум интерфејси који омогућавају рад са посебним функцијама из CrudRepository klase.

4.2.3 Контролери

AuthorController.java

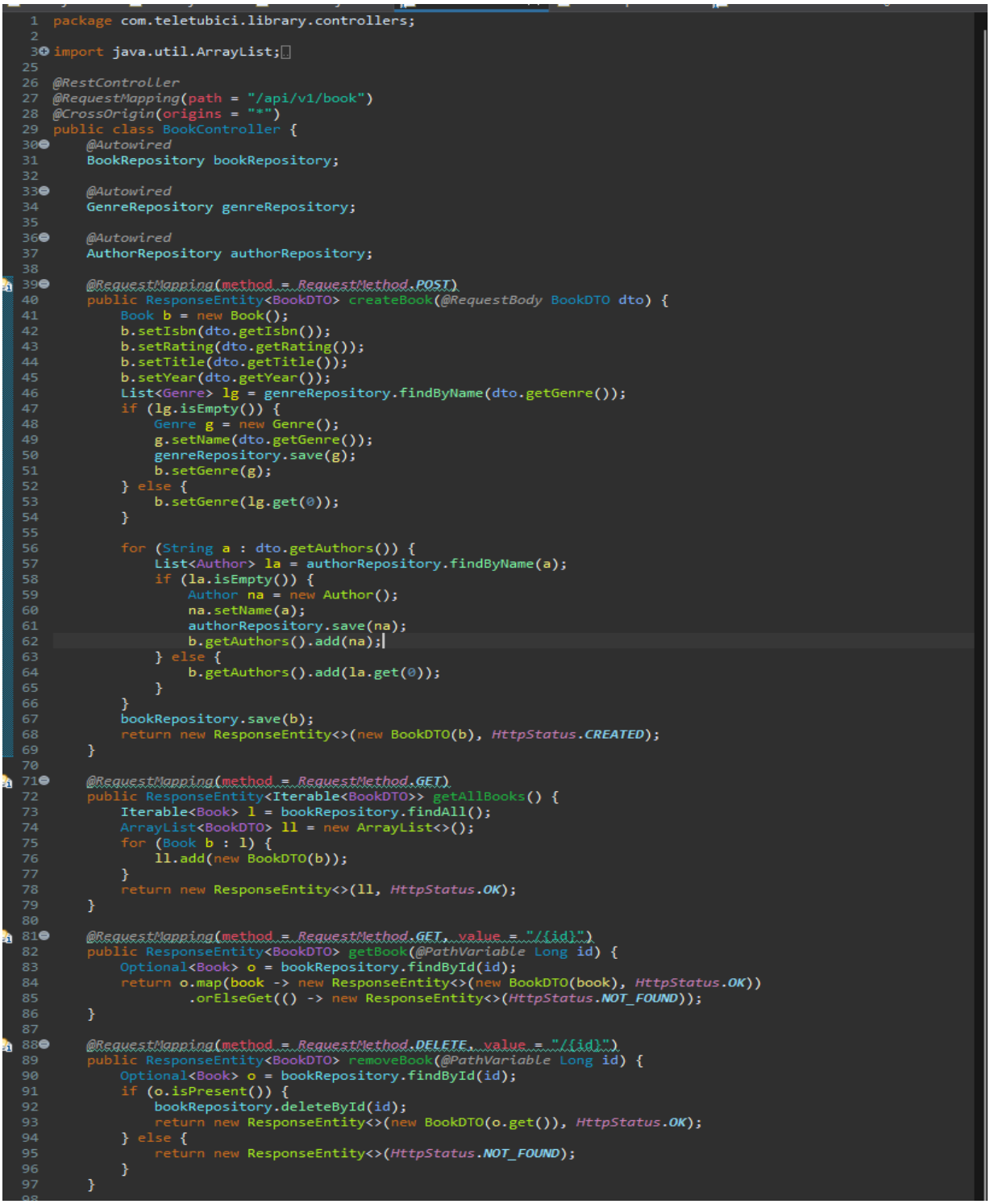
Има ендпоинте који омогућавају CRUD функционалности, конкретно упис, читање, брисање и ажурирање, функционалност враћања свих аутора, претраге по имену и претаге по садржају карактера у имену аутора.

```java
package com.teletubici.library.controllers;

import java.util.ArrayList;

@RestController
@RequestMapping(path = "/api/v1/author")
@CrossOrigin(origins = "*")
public class AuthorController {
    @Autowired
    AuthorRepository AuthorRepository;

    @RequestMapping(method=RequestMethod.POST)
    public AuthorDTO createAuthor(@RequestBody Author author) {
        Author a = new Author();
        a.setName(author.getName());
        AuthorRepository.save(a);
        return new AuthorDTO(a);
    }

    @RequestMapping(method=RequestMethod.GET)
    public Iterable<AuthorDTO> getAllAuthors() {
        Iterable<Author> l = AuthorRepository.findAll();
        ArrayList<AuthorDTO> ll = new ArrayList<>();
        for(Author a : l) {
            ll.add(new AuthorDTO(a));
        }
        return ll;
    }

    @RequestMapping(method=RequestMethod.GET, value="/{id}")
    public ResponseEntity<AuthorDTO> getAuthor(@PathVariable Long id) {
        Optional<Author> o = AuthorRepository.findById(id);
        if(o.isPresent()){
            return new ResponseEntity<AuthorDTO>(new AuthorDTO(o.get()), HttpStatus.OK);
        }else{
            return new ResponseEntity<AuthorDTO>(HttpStatus.NOT_FOUND);
        }
    }

    @RequestMapping(method=RequestMethod.DELETE, value="/{id}")
    public ResponseEntity<AuthorDTO> removeAuthor(@PathVariable Long id) {
        Optional<Author> o = AuthorRepository.findById(id);
        if(o.isPresent()){
            AuthorRepository.deleteById(id);
            return new ResponseEntity<AuthorDTO>(new AuthorDTO(o.get()), HttpStatus.OK);
        }else{
            return new ResponseEntity<AuthorDTO>(HttpStatus.NOT_FOUND);
        }
    }

    @RequestMapping(method=RequestMethod.PUT, value="/{id}")
    public ResponseEntity<AuthorDTO> updateAuthor(@PathVariable Long id, @RequestBody Author author) {
        Optional<Author> o = AuthorRepository.findById(id);
        if(o.isPresent()){
            Author a = o.get();
            a.setName(author.getName());
            AuthorRepository.save(a);
            return new ResponseEntity<AuthorDTO>(new AuthorDTO(a), HttpStatus.OK);
        }else{
            return new ResponseEntity<AuthorDTO>(HttpStatus.NOT_FOUND);
        }
    }

    @RequestMapping(method=RequestMethod.GET, value="/byname")
    public Iterable<AuthorDTO> getAuthorByName(@RequestParam String name) {
        Iterable<Author> l = AuthorRepository.findByName(name);
        ArrayList<AuthorDTO> ll = new ArrayList<>();
        for(Author a : l) {
            ll.add(new AuthorDTO(a));
        }
        return ll;
    }

    @RequestMapping(method=RequestMethod.GET, value="/search")
    public Iterable<AuthorDTO> searchAuthorByName(@RequestParam String name) {
        Iterable<Author> l = AuthorRepository.findByNameContainingIgnoreCase(name);
        ArrayList<AuthorDTO> ll = new ArrayList<>();
        for(Author g : l) {
            ll.add(new AuthorDTO(g));
        }
        return ll;
    }
}

```

Слика 9. AuthorController.java

BookController.java

Има ендпоинте који омогућавају CRUD функционалности, функционалност враћања свих књига и претаге по садржају карактера у имену књиге.



```java
package com.teletubici.library.controllers;

import java.util.ArrayList;

@RestController
@RequestMapping(path = "/api/v1/book")
@CrossOrigin(origins = "*")
public class BookController {
    @Autowired
    BookRepository bookRepository;

    @Autowired
    GenreRepository genreRepository;

    @Autowired
    AuthorRepository authorRepository;

    @RequestMapping(method = RequestMethod.POST)
    public ResponseEntity<BookDTO> createBook(@RequestBody BookDTO dto) {
        Book b = new Book();
        b.setIsbn(dto.getIsbn());
        b.setRating(dto.getRating());
        b.setTitle(dto.getTitle());
        b.setYear(dto.getYear());
        List<Genre> lg = genreRepository.findByName(dto.getGenre());
        if (lg.isEmpty()) {
            Genre g = new Genre();
            g.setName(dto.getGenre());
            genreRepository.save(g);
            b.setGenre(g);
        } else {
            b.setGenre(lg.get(0));
        }

        for (String a : dto.getAuthors()) {
            List<Author> la = authorRepository.findByName(a);
            if (la.isEmpty()) {
                Author na = new Author();
                na.setName(a);
                authorRepository.save(na);
                b.getAuthors().add(na);
            } else {
                b.getAuthors().add(la.get(0));
            }
        }
        bookRepository.save(b);
        return new ResponseEntity<>(new BookDTO(b), HttpStatus.CREATED);
    }

    @RequestMapping(method = RequestMethod.GET)
    public ResponseEntity<Iterable<BookDTO>> getAllBooks() {
        Iterable<Book> l = bookRepository.findAll();
        ArrayList<BookDTO> ll = new ArrayList<>();
        for (Book b : l) {
            ll.add(new BookDTO(b));
        }
        return new ResponseEntity<>(ll, HttpStatus.OK);
    }

    @RequestMapping(method = RequestMethod.GET, value = "/{id}")
    public ResponseEntity<BookDTO> getBook(@PathVariable Long id) {
        Optional<Book> o = bookRepository.findById(id);
        return o.map(book -> new ResponseEntity<>(new BookDTO(book), HttpStatus.OK))
                .orElseGet(() -> new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }

    @RequestMapping(method = RequestMethod.DELETE, value = "/{id}")
    public ResponseEntity<BookDTO> removeBook(@PathVariable Long id) {
        Optional<Book> o = bookRepository.findById(id);
        if (o.isPresent()) {
            bookRepository.deleteById(id);
            return new ResponseEntity<>(new BookDTO(o.get()), HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }
```

```java

    @RequestMapping(method = RequestMethod.PUT, value = "/{id}")
    public ResponseEntity<BookDTO> updateBook(@PathVariable Long id, @RequestBody BookDTO dto) {
        Optional<Book> o = bookRepository.findById(id);
        if (o.isPresent()) {
            Book b = o.get();
            b.setIsbn(dto.getIsbn());
            b.setRating(dto.getRating());
            b.setTitle(dto.getTitle());
            b.setYear(dto.getYear());
            List<Genre> lg = genreRepository.findByName(dto.getGenre());
            if (lg.isEmpty()) {
                Genre g = new Genre();
                g.setName(dto.getGenre());
                genreRepository.save(g);
                b.setGenre(g);
            } else {
                b.setGenre(lg.get(0));
            }
            b.getAuthors().clear();
            for (String a : dto.getAuthors()) {
                List<Author> la = authorRepository.findByName(a);
                if (la.isEmpty()) {
                    Author na = new Author();
                    na.setName(a);
                    authorRepository.save(na);
                    b.getAuthors().add(na);
                } else {
                    b.getAuthors().add(la.get(0));
                }
            }
            bookRepository.save(b);
            return new ResponseEntity<>(new BookDTO(b), HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }

    @RequestMapping(method = RequestMethod.GET, value = "/search")
    public ResponseEntity<Iterable<BookDTO>> searchBookByName(@RequestParam String title) {
        Iterable<Book> l = bookRepository.findByTitleContainingIgnoreCase(title);
        ArrayList<BookDTO> ll = new ArrayList<>();
        for (Book g : l) {
            ll.add(new BookDTO(g));
        }
        return new ResponseEntity<>(ll, HttpStatus.OK);
    }
}
```

Слика 10. BookController.java

GenreController.java

Садржи ендпоинте који омогућавају CRUD функционалности, функционалност враћања свих жанрова, претраге по имену и претаге по садржају карактера у имену жанра.

```java
package com.teletubici.library.controllers;

import java.util.ArrayList;

@RestController
@RequestMapping(path = "/api/v1/genre")
@CrossOrigin(origins = "*")
public class GenreController {
    @Autowired
    GenreRepository genreRepository;

    @RequestMapping(method=RequestMethod.POST)
    public ResponseEntity<GenreDTO> createGenre(@RequestBody Genre genre) {
        List<Genre> existingGenre = genreRepository.findByName(genre.getName());

        if(!existingGenre.isEmpty()){
            return new ResponseEntity<GenreDTO>(HttpStatus.CONFLICT);
        }else{
            Genre g = new Genre();
            g.setName(genre.getName());
            genreRepository.save(g);
            return new ResponseEntity<GenreDTO>(new GenreDTO(g), HttpStatus.CREATED);
        }

    }

    @RequestMapping(method=RequestMethod.GET)
    public Iterable<GenreDTO> getAllGenres() {
        Iterable<Genre> l = genreRepository.findAll();
        ArrayList<GenreDTO> ll = new ArrayList<>();
        for(Genre g : l) {
            ll.add(new GenreDTO(g));
        }
        return ll;
    }

    @RequestMapping(method=RequestMethod.GET, value="/{id}")
    public ResponseEntity<GenreDTO> getGenre(@PathVariable Long id) {
        Optional<Genre> o = genreRepository.findById(id);
        if(o.isPresent()){
            return new ResponseEntity<GenreDTO>(new GenreDTO(o.get()), HttpStatus.OK);
        }else{
            return new ResponseEntity<GenreDTO>(HttpStatus.NOT_FOUND);
        }
    }

    @RequestMapping(method=RequestMethod.DELETE, value="/{id}")
    public ResponseEntity<GenreDTO> removeGenre(@PathVariable Long id) {
        Optional<Genre> o = genreRepository.findById(id);
        if(o.isPresent()){
            genreRepository.deleteById(id);
            return new ResponseEntity<GenreDTO>(new GenreDTO(o.get()), HttpStatus.OK);
        }else{
            return new ResponseEntity<GenreDTO>(HttpStatus.NOT_FOUND);
        }
    }

    @RequestMapping(method=RequestMethod.PUT, value="/{id}")
    public ResponseEntity<GenreDTO> updateGenre(@PathVariable Long id, @RequestBody Genre genre) {
        Optional<Genre> o = genreRepository.findById(id);
        if(o.isPresent()){
            Genre g = o.get();
            g.setName(genre.getName());
            genreRepository.save(g);
            return new ResponseEntity<GenreDTO>(new GenreDTO(g), HttpStatus.OK);
        }else{
            return new ResponseEntity<GenreDTO>(HttpStatus.NOT_FOUND);
        }
    }

    @RequestMapping(method=RequestMethod.GET, value="/byname")
    public Iterable<GenreDTO> getGenreByName(@RequestParam String name) {
        Iterable<Genre> l = genreRepository.findByName(name);
        ArrayList<GenreDTO> ll = new ArrayList<>();
        for(Genre g : l) {
            ll.add(new GenreDTO(g));
        }
        return ll;
    }

    @RequestMapping(method=RequestMethod.GET, value="/search")
    public Iterable<GenreDTO> searchGenreByName(@RequestParam String name) {
        Iterable<Genre> l = genreRepository.findByNameContainingIgnoreCase(name);
        ArrayList<GenreDTO> ll = new ArrayList<>();
        for(Genre g : l) {
            ll.add(new GenreDTO(g));
        }
        return ll;
    }
}

```

Слика 11. GenreController.java

4.2.4 Класе Ентитета

Author.java

Класа у којој су дефинисани атрибути ентитета Аутор и у којој су анотацијама између осталог дефинисане релације са осталим класама.

```java
package com.teletubici.library.entities;

import java.util.ArrayList;

@Entity
public class Author {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    protected Long id;

    protected String name;

    @ManyToMany(fetch = FetchType.LAZY, cascade = CascadeType.REFRESH)

    @JoinTable(name = "Book_Author", joinColumns = @JoinColumn(name = "author_id",
referencedColumnName = "id", nullable = false, updatable = false, insertable = false),
inverseJoinColumns = @JoinColumn(name = "book_id", referencedColumnName = "id", nullable = false,
updatable = false, insertable = false))
    protected List<Book> books;

    public Long getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public List<Book> getBooks() {
        return books;
    }

    public void setBooks(List<Book> books) {
        this.books = books;
    }

    public Author() {
        super();
        this.books = new ArrayList<>();
    }

}
```

Слика 12. Author.java

Book.java

Класа у којој су дефинисани атрибути ентитета књига (Book) и у којој су анотацијама између осталог дефинисане релације са осталим класама. Наравно, гет и сет методе са конструктором се подразумевају.

```java
package com.teletubici.library.entities;

import java.util.ArrayList;

@Entity
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    protected Long id;
    protected String title;
    protected String isbn;
    protected String year;
    protected Double rating;

    @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.REFRESH)

    @JoinTable(name = "Book_Author", joinColumns = @JoinColumn(name = "book_id",
referencedColumnName = "id", nullable = false, updatable = false, insertable = false),
inverseJoinColumns = @JoinColumn(name = "author_id", referencedColumnName = "id", nullable = false,
updatable = false, insertable = false))
    protected List<Author> authors;

    @ManyToOne(cascade = CascadeType.REFRESH, fetch = FetchType.EAGER)
    protected Genre genre;

    public Long getId() {
        return id;
    }

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

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getIsbn() {
        return isbn;
    }

    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }

    public String getYear() {
        return year;
    }

    public void setYear(String year) {
        this.year = year;
    }

    public Double getRating() {
        return rating;
    }

    public void setRating(Double rating) {
        this.rating = rating;
    }

    public List<Author> getAuthors() {
        return authors;
    }

    public void setAuthors(List<Author> authors) {
        this.authors = authors;
    }

    public Genre getGenre() {
        return genre;
    }

    public void setGenre(Genre genre) {
        this.genre = genre;
    }

    public Book() {
        super();
        this.authors = new ArrayList<>();
    }

}

```

Слика 13. Book.java

Genre.java

Класа у којој су дефинисани атрибути ентитета жанр (Genre) и у којој су анотацијама између осталог дефинисане релације са осталим класама.

```java
package com.teletubici.library.entities;

import java.util.ArrayList;

@JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class, property = "name")
@Entity
public class Genre {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    protected Long id;

    @Column(nullable = false, unique = true)
    protected String name;

    @OneToMany(mappedBy = "genre", fetch = FetchType.EAGER, cascade = CascadeType.REFRESH)
    protected List<Book> books;

    public Long getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Genre() {
        super();
        this.books = new ArrayList<>();
    }

    public List<Book> getBooks() {
        return books;
    }

    public void setBooks(List<Book> books) {
        this.books = books;
    }

}

```

Слика 14. Genre.java

pom.xml

Фајл у којем су уписани подаци приликом иницијализације самог пројекта, као и подаци о библиотекама које се регулишу уз помоћ MAVENa, јер је сам пројекат тако иницијализован ‘постављен да користи MAVEN.

```
https://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation)
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.4</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.teletubici</groupId>
    <artifactId>PrivateLibrary</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>PrivateLibrary</name>
    <description>Privatna biblioteka, projekat za predispitnu obavezu</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <version>3.2.4</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
            <version>3.2.4</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
            <version>3.2.4</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.33</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <version>3.2.4</version>
            <optional>true</optional>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

Слика 15. pom.xml

4.3 React

Будући да колега који је био задужен за израду фронтенд дела апликације, није био у могућности да заврши пројекат на време из приватних разлога. Ванредно је одрађена фронтенд солуција како би пројекат био комплетиран као једна целина. Главне функционалности које су замишљене и дефинисане су покривене овом солуцијом, те ће исте бити описане у овој секцији документације.

React је јаваскрипт библиотека отвореног кода која се користи за израду корисничких интерфејса, посебно за веб апликације. Основна идеја је креирање компоненти које представљају делове корисничког интерфејса, а затим компоновање ових компоненти како би се креирао комплексан интерфејс. React користи JSX, синтаксу која омогућава писање ХТМЛ-а унутар Јаваскрипт-а, што олакшава рад са компонентама. Једна од главних карактеристика React-а је виртуелни ДОМ (Документ Објект Модел), који омогућава ефикасније управљање променама у интерфејсу и брже ажурирање корисничког искуства.

4.3.1 Развој фронтенд дела веб апликације у React-u

За израду фронтенда у React библиотеци за ову апликацију, фокус је био на стварању корисничког интерфејса који ће омогућити интуитивно коришћење апликације. Користећи RESTful API који пружа бекенд развијен у Spring Boot-u, можемо ефикасно комуницирати са сервером и приказивати релевантне податке корисницима.

index.js

У овом фајлу је постављен рутер који омогућава навигацију ка страницама, односно компонентама.

```
const router = createBrowserRouter([
  {
    path: '/',
    element: <App/>,
    children: [
      {
        path: 'books',
        element: <ShowBooks/>,
        loader: async() => {
          const user = check_login();
          return fetch('http://localhost:8080/api/v1/book');
        },
        errorElement: <ErrorDisplay entity="knjige"/>
      },
      {
        path: 'authors',
        element: <ShowAuthors/>,
        loader: async() => {
          const user = check_login();
          return fetch('http://localhost:8080/api/v1/author');
        },
        errorElement: <ErrorDisplay entity="autori"/>
      },
      {
        path: 'genres',
        element: <ShowGenres/>,
        loader: async() => {
          const user = check_login();
          return fetch('http://localhost:8080/api/v1/genre');
        },
        errorElement: <ErrorDisplay entity="zanrovi"/>
      },
      {
        path:'genres/add_new',
        element: <NewGenre/>
      },
      {
        path:'authors/add_new',
        element: <NewAuthor/>
      },
```

```
      },
      {
        path:'genres/add_new',
        element: <NewGenre/>
      },
      {
        path:'authors/add_new',
        element: <NewAuthor/>
      },
      {
        path:'books/add_new',
        element: <NewBook/>
      },
      {
        path:'books/update/:id',
        element: <EditBook/>,
        loader: async({params}) => {
          return fetch(`http://localhost:8080/api/v1/book/${params.id}`);
        }
      },
      {
        path:'/books/book/:id',
        element: <Book/>,
        loader: async({params}) => {
          return fetch(`http://localhost:8080/api/v1/book/${params.id}`);
        }
      },
      {
        path:'genres/update/:id',
        element: <EditGenre/>,
        errorElement: <ErrorDisplay entity="zanr"/>,
        loader: async({params}) => {
          const user = check_login(['admin']);
          return fetch(`http://localhost:8080/api/v1/genre/${params.id}`);
        },
        action: async({params,request}) =>{
          const data = Object.fromEntries(await request.formData());
          return fetch(`http://localhost:8080/api/v1/genre/${params.id}`, {
            method: "PUT",
            headers: {
                "Content-Type": "application/json",
            },
            body: JSON.stringify(data)
          });
        }
      }
      ]
    }
])

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <RouterProvider router={router}/>
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

Слика 16. index.js

login_logic.js

У овом фајлу је постављена логика логовања, у њој су дефинисани корисници са улогама и омогућава условно рендеровање компоненти што ће бити приказано у једној од компонената.

```
import { useState } from "react";

export const get_login = () => {
    const user = JSON.parse(localStorage.getItem('user'));
    console.log(user);
    return user;
}

export const check_login = (roles) => {
    const user = get_login();
    if(user === null){
        const err = {
            cause:"login",
            message: "Korisnik nije ulogovan"
        };
        throw err;
    }else if(roles){
        if(!roles.includes(user.role)){
            const err = {
                cause:"security",
                message: "Korisnik nema pravo pristupa"
            };
            throw err;
        }
    }

    return user;

}


export const valid_login = (roles) => {
    const user = get_login();
    if(user === null){
        return false;
    }else if(roles){
        if(!roles.includes(user.role)){
            return false;
        }
    }
    console.log("23456789");
    return true;

}
```

```
export const useLogin = () => {
    const [user, setUser] = useState(JSON.parse(localStorage.getItem('user')));
    return [
        user,

        (username, password)=>{
            if(username === 'test' && password === 'test'){
                const nuser = {
                  username: 'test',
                  name: 'Test Test',
                  token: 'abcd',
                  role: 'admin'
                }
                setUser(nuser);
                localStorage.setItem("user", JSON.stringify(nuser));
                return nuser;
            }else if(username === 'user' && password === 'user') {
                const nuser = {
                  username: 'user',
                  name: 'User User',
                  token: '1234',
                  role: 'user'
                }
                setUser(nuser);
                localStorage.setItem("user", JSON.stringify(nuser));
                return nuser;
          }else{
            return null;
          }
        },

        ()=>{
            setUser(null);
            localStorage.removeItem('user');
        }
    ]
}
```

Слика 17. login_logic.js

LoginModal.js

Компонента која уствари представља форму за логовање са пољима корисничко име и шифра, као и дугмима Login и Cancel.

```
const LoginModal = ({onCloseModal}) => {
    const [username, setUsername] = useState();
    const [password, setPassword] = useState();

    const {user, login, logout} = useContext(UserContext);

    const userLogin = () => {
        const user = login(username, password);
        onCloseModal();
    }
    return (
        <div>
          <Dialog
            open={true}
            aria-labelledby="alert-dialog-title"
            aria-describedby="alert-dialog-description"
          >
            <DialogTitle id="alert-dialog-title">
              {"Login"}
            </DialogTitle>
            <DialogContent sx={{display:"flex", flexDirection:"column"}}>
              <TextField label="Username" variant="outlined" required sx={{marginBottom:"10px"}} onChange=
              {(e)=>{setUsername(e.target.value)}}/>
              <TextField label="Password" variant="outlined" required onChange={(e)=>{setPassword(e.target.
              value)}}/>
            </DialogContent>
            <DialogActions>
              <Button onClick={userLogin}>Login</Button>
              <Button onClick={()=>{onCloseModal()}}>Cancel</Button>
            </DialogActions>
          </Dialog>
        </div>
      );
}

export default LoginModal;
```

Слика 18. LoginModal.js

ShowBooks.js

Ова компонента омогућава приказ компоненте ShowBook.js која представља заправо једну од књига из базе података, односно биће их рендеровано онолико колико их има у бази. Такође у овој компоненти имамо и условно рендеровање компоненте, односно елемента.

```
const ShowBooks = () => {
    const books = useLoaderData();
    const navigate = useNavigate();
    const [user, setUser] = useState(null)
    const [isLogin, setIsLogin] = useState(false);

    const [genres, setGenres] = useState([]);
    const [filteredBooks, setFilteredBooks] = useState(books);
    const [searchedBooks, setSearchedBooks] = useState(books);


const search = (value) => {
    if (value === "") {
        setSearchedBooks(filteredBooks);
    } else {
        const filtered = filteredBooks.filter((book) =>
            book.title.toLowerCase().includes(value.toLowerCase())
        );
        setSearchedBooks(filtered);
    }
};


    useEffect(() => { //A
        let ignore = false;
        const ff = async () => { //B
            let r = await fetch("http://localhost:8080/api/v1/genre");
            let rr = await r.json();
            if (!ignore) {
                setGenres(rr);
            }
        };
        ff();
        return () => { //C
            ignore = true;
        };
    }, []);


    const handleDelete = (bookId) => {
        const updatedFilteredBooks = filteredBooks.filter((b) => b.id !== bookId);
        setFilteredBooks(updatedFilteredBooks);

        const updatedSearchedBooks = searchedBooks.filter((b) => b.id !== bookId);
        setSearchedBooks(updatedSearchedBooks);
    };

```

```
    useEffect(() => {
        const u = localStorage.getItem("user");
        console.log(u);
        if (u) {
            setUser(JSON.parse(u));
            setIsLogin(true);
        }
    }, [isLogin])


    return <Container>
        <Box sx={{ display: "flex", justifyContent: "space-between", marginBottom: 3 }}>
            <FormControl sx={{ width: "30%" }}>
                <TextField
                    placeholder="Pretraga..."
                    label="Pretraga"
                    onChange={(e) => search(e.target.value)}
                />
            </FormControl>

            {isLogin && user.role === 'user' ? <></> : <Button variant="outlined" onClick={() => { navigate
            ('add_new') }}>Add New Book</Button>}


        </Box>
        <Grid container spacing={3}>
                {searchedBooks.map((b) => (
                    <ShowBook onDelete={handleDelete} book={b} />
                ))}
            </Grid>
    </Container>
}

export default ShowBooks;
```

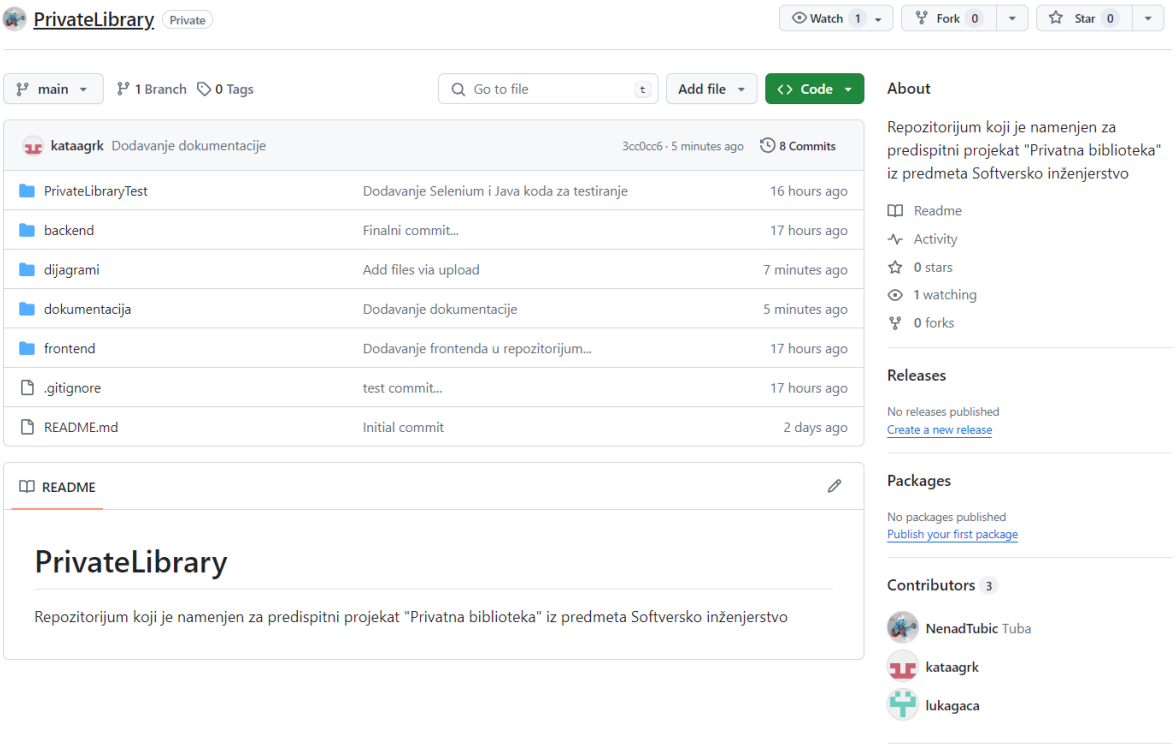
Слика 19. ShowBooks.js

4.4 Git/GitHub

За потребе апликације Приватна библиотека креиран је нови репозиторијум Private-library на GitHub профилу колеге Лазара Велимировића у којем се налазе сви потребни пројекти, документација и све што је везано за целокупну апликацију.

Линк до GitHub профила:

https://github.com/NenadTubic/PrivateLibrary



Слика 20. GitHub репозиторијум

5. ТЕСТИРАЊЕ

Коришћена технологија је Селениум, специфично Селениум Web Driver. Верзија Селениум Java Client-а је 4.19.1, извачена марта 29.ог, 2024. године. Верзија TestNG је најмодернија - не пишу детаљи у вези ње. Селениум WebDriver функционише тако што користи тзв. WebDriver интерфејс са предефинисаним командама и методама. Прво што је неопходно учинити јесте дефинисати који претраживач ћемо користити (да ли Chrome, Safari, Firefox..). Омогућава безброј интеракција са веб страницом и свим елементима апликације. На пример, интерфејс омогућава манипулацију елемената на основу атрибута, xpath-а итд (ово је тзв. WebElement елемент), затим скакање са алерт прозора на други алерт прозор, интеракције везане за чекање да се садржај учита итд. Такође, корисник има опцију да угаси прозор, саму web-страницу, да је освежи, увећа, смањи..

Наравно, вреди и напоменути да се неће баш и сваки елемент странице одмах учитати. Ово је од круцијалне важности када радимо са, на пример, поп-уп прозорима, алерт-овима итд. У сврху чекања да се садржај учита, постоји класа WebDriverWait. Ова класа такорећи "онемогућује" некомпатибилност елемената због неке разлике у времену учитавања, тј. синхронизације. Она је изузетно корисна / потентна када знамо или можемо претпоставити да ће се одређени елемент можда касније појавити или учитати. WebDriverWait се, дакле, имплементира када постоји разлика у синхронизацији унутар једног или више @Test-а. Када смо већ на теми чекања (wаит-ова), вреди споменути две (главне, јер има их три) врсте чекања - имплицитно и експлицитно. Имплицитни wait се дефинише када желимо да одредимо време чекања учитавања сваког елемента странице или апликације. Такође вреди споменути да он важи на глобалном нивоу. Његова главна мана је у томе што жури, тј. не чека доследно баш алерт-ове, JS позиве и слично. Експлицитни wаит служи сличну функцију и као имплицитни, с тим да с експлицитним специфирамо време чекања учитавања једног, специфичног елемента. Много је кориснији од имплицитног у смислу прецизности, иако логично захтева више кода.

Вреди споменути још неколико занимљивости / елемената тестирања са Селениумом, а то је тзв. "testng.xml" фајл. У питању је дакле XML / конфигурациони фајл који суштински помаже са организацијом тестоца. У овом фајлу се сви тестови смештају такорећи у један фајл, по потреби се могу и заједно покретати (ово се назива паралелно тестирање). Тестеру омогућава и груписање свих његових или њених тестова (суитес). Пружа детаљније извештаје о имплементацији тестова и омогућава да један тест зависи од другог, што може бити корисно када не желимо да се један тест изврши (на пример 6 од 10) уколико су тестови 4 и 5 пропали. Једно од најкориснијих ствари у вези testng.xml фајла јесте да када се он покрене, покрену се сви тестови / суите-ови написани, тј. референцирани у њему.

Последња ствар коју треба споменути у вези писања кода јесте POM, тј. Page Object Model. У питању није нити објекат нити елемент, већ једноставно принцип посматрања и писања сваке странице web-сајта, као засебног објекта (или касе). Другим речима: свака страница апликације или сајта је, или се третира као Page Object. Оно је очигледно базирано на основним принципима објектно оријентисаног програмирања. Сама PageFactory класа се дефинише унутар конструктора, али тек након што је WebDriver иницијализован. Оно је суштински класа која симплификује иницијализацију веб елемената. Дефинисање елемента БЕЗ PageFactory изгледа овако: WebElement inputF = driver.findElement(By.cssSelector("#inputF"));. Дефинисање СА PageFactory изгледа овако: @FindBy(css = "#inputF"); WebElement inputF;. Уколико, на пример, покушамо да искористимо ту анотацију без да смо иницијализовали елемент, компајлер ће вратити "no such element exception"! Селениум WebDriver је, дакле, изузетно ефикасан и флексибилан алат за аутоматизацију претраживача који је, поред можда JEST-а, један од најпопуларнијих алата у данашњици који се користи за тестирање.

5.1 Test case 1

Опис: Верификовати да је могуће улоговати се у налог као админ, затим покушати да додамо нову књигу.

Корак 1 - покренути апликацију Private Library

Корак 2 - кликнути на LOGIN дугме

Корак 3 - у новоотвореном прозору унети "test" као и корисничко име, и као лозинку

Корак 4 - након успешног логовања, отићи на Books страницу и затим кликнути "ADD NEW BOOK" дугме

Корак 5 - испунити сва поља, пазећи да унешени подаци испуњавају критеријуме њихових кореспондирајућих поља

Корак 6 - кликнути дугме за финализацију

Корак 7 - отићи назад на Books страницу и проверити да ли се новоунешена књига налази на страници

Очекивани резултат 1: логин прозор је отворен / интерактиван

Очекивани резултат 2: допуштено је логовање са претходно дефинисаним креденцијалима

Очекивани резултат 3: могућ је унос нове књиге

Очекивани резултат 4: нова књига је успешно унешена у базу података

Прави резултат: логин прозор је отворен / интерактиван, допуштено је логовање са претходно дефинисаним креденцијалима, могућ унос нове књиге, нова књига је присутна не само на Books страници, већ се јавља и у бази података

5.1.1 Код и спровођење теста

За први тест је коришћена релативно симплистична методологија рада.

Наиме, прво што сам урадио јесте да сам креирао засебан, нов јава пројекат. Затим, креирао сам ново паковање у овом пројекту и класу - у ствари тест који ћемо касније покренути. Име сваког теста је везано за његову сврху.

У првом случају је нужно првенствено дефинисати који драјвер ћемо користити - у мом случају, то је ChromeDriver. Затим, направили смо инстанцу WebDriverWait интерфејса како би касније могли да искористимо тзв. експлицитни wаит, тј. методу(е) уз помоћу које ћемо форсирати да тест сачека да се одређени (специфирани) елемент апликације учита. На самом крају, Селениум ће навигирати (отворити) локацију апликације, у ствари порт 3000 пошто је апликација покренута ЛОКАЛНО, не преко хостинга.

Следи низ мање-више идентичних радњи, што само указује на софистицирану симплистичност Селениума. Овај интерфејс функционише тако што тестер прво мора да специфира елемент којим ће се манипулисати, те тек затим ће моћи Селениум (и Јава) метода да спроводи над њима. У одређеним случајевима је нужно сместити га / их као WebElement објекат (нешто налик смештању вредности "променљивој"), но зарад одржавања релативне концизности и прегледности кода на сцреенсхот-овима, ја ово (углавном) нисам радио. Инстанце где се појављују су углавном биле форсиране, пошто сам добијао ерроре приликом имплементације, тј. иницијализације тестова.

Првенствено сам дефинисао тип и назив првог елемента потребног за тест - логин дугмета. Затим сам, позивајући Селениум WebDriver методу .findElement, уз помоћ xpath путање специфирао тачну локацију елемента са којим желим да располажем. Како бих био сигуран да је елемент учитан (како не би дошло до некомпатибилности изазваних разликама у синхронизацији), одлучио сам да позовем инстанцу WebDriverWait објекта, звану "wаit", и помоћу његових метода да експлицитно нагласим да сачека бар (претходно наглашених) десет секунди да се дати елемент учита.

У новоотвореном прозору за логовање тражим сва три елемента. Над корисничким именом и лозинка елементима вршим методу .sendKeys, која омогућава да се нешто напише и пошаље специфираном елементу, и затим лоцирам, чекам кликабилност и онда тек кликам на дугме којим прослеђујем унешене податке. Као што рекох, имплементирам ову методу рада зарад одржавања концизности кода.

Следеће, кликам на навигациони мени, како бих добио опције интеракција са расположивим страницама. Кликам на Books страницу и чекам да се учита. Затим, лоцирам и кликам на ADD BOOK дугме, које се у прозору налази горе десно. Иако део следеће етапе, објаснићу укратко овде: експлицитним wait-ом форсирам обуставу тестирања све док елемент / прозор у ком додајемо нову књигу не постане видљив.

У секцији где додајемо књигу лоцирам и испуњавам свако поље са кореспондирајућим информацијама. Међутим, овде имамо два тзв. селект елемент - један где бирамо жанр књиге, други са именима аутора. Селениум поседује одличне и флексибилне методе рада и са - листама! У оба случаја креирам инстанцу класе Select, затим као улазни параметар конструктора дефинишем тачне локације оба елемената (очекивано користећи њихове тачне путање). Користећи уграђену .selectByIndex(Index) методу, лоцирам желељене чланове обе листе. Финални корак док смо још у овом прозору је кликање на дугме да се подаци региструју.

Након успешног уноса свих података, враћамо се на Books страницу. Затим, користим assert assertEquals, уз помоћу којег валидирам да се неке, две вредности подударају. У овом нашем случају, проверавам да ли се на страници актуелној налази било који HTML span елемент који садржи текст који кореспондира наслову новоунете књиге (”Na Drini Cuprija"), наравно написано у латиници.

NewBook.java | BookDetails.java | BookDeletion.java | BookEdit.java

```java
package com.teletubici;

import java.time.Duration;

public class NewBook {

    public static void main(String[] args) {
        WebDriver driver = new ChromeDriver();
        WebDriverWait wait;
        wait = new WebDriverWait(driver, Duration.ofSeconds(10));
        driver.navigate().to("http://localhost:3000/books");

        WebElement loginBtn = driver.findElement(By.xpath("/html/body/div/div/header/div/button[2]"));
        wait.until(ExpectedConditions.elementToBeClickable(loginBtn)).click();
        WebElement userName = driver.findElement(By.xpath("/html/body/div[2]/div[3]/div/div[1]/div[1]/div/input"));
        wait.until(ExpectedConditions.visibilityOf(userName)).sendKeys("test");
        WebElement passWord = driver.findElement(By.xpath("/html/body/div[2]/div[3]/div/div[1]/div[2]/div/input"));
        wait.until(ExpectedConditions.visibilityOf(passWord)).sendKeys("test");
        wait.until(ExpectedConditions.elementToBeClickable(driver.findElement(By.xpath("/html/body/div[2]/div
[3]/div/div[2]/button[1]")))).click();

        WebElement navBtn = driver.findElement(By.xpath("/html/body/div/div/header/div/button[1]"));
        wait.until(ExpectedConditions.elementToBeClickable(navBtn)).click();

        WebElement booksBtn = driver.findElement(By.xpath("/html/body/div/div/div/div/ul[1]/li/a"));
        wait.until(ExpectedConditions.elementToBeClickable(booksBtn)).click();

        WebElement addNewBookBtn = driver.findElement(By.xpath("/html/body/div/div/main/div[2]/div[1]/button"));
        wait.until(ExpectedConditions.elementToBeClickable(addNewBookBtn)).click();

        wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.xpath("/html/body/div/div/main/div
[2]/div/div[1]/input")))).sendKeys("Na Drini Cuprija");
        wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.xpath("/html/body/div/div/main/div
[2]/div/div[2]/input")))).sendKeys("1231231231");
        wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.xpath("/html/body/div/div/main/div
[2]/div/div[3]/input")))).sendKeys("1995");
        wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.xpath("/html/body/div/div/main/div
[2]/div/div[4]/input")))).sendKeys("5");
        Select bookGenre = new Select(driver.findElement(By.xpath("/html/body/div/div/main/div[2]/div/div
[5]/select")));
        bookGenre.selectByIndex(2);
        Select authorNames = new Select(driver.findElement(By.xpath("/html/body/div/div/main/div[2]/div/div
[6]/select")));
        authorNames.selectByIndex(2);
        wait.until(ExpectedConditions.elementToBeClickable(driver.findElement(By.xpath("/html/body/div/div/main/div
[2]/div/button")))).click();

        wait.until(ExpectedConditions.elementToBeClickable(booksBtn)).click();
        assertEquals(driver.findElement(By.xpath("//span[text()='Na Drini Cuprija']")).getText(), "Na Drini
Cuprija");
        driver.quit();
    }
}
```

Слика 21. Код првог теста, од почетка до краја

Излазимо из апликације / странице употребом .quit() методе, која гаси прозор у ком се налазимо

Коментар: иако у коду нисам користио у појединим случајевима експлицитна имена нити дескрипција елемената, он функционише. На крају је видљив метод који сам користио за саму валидацију успешности тестирања.

id	isbn	rating	title	year	genre_id
1	1234567894	5	Ma šta mi reče	1972	1
2	1234567894	5	Bukvar dečijih prava	1996	1
3	1234567893	5	Zov tetreba	1977	1
4	1234567893	5	A Game of Thrones	1996	2
5	1234567893	5	The Winds of Winter	2016	2
6	1234567893	5	Fire and Blood	2018	2
41	1231231231	5	Na Drini Cuprija	1995	3
NULL	NULL	NULL	NULL	NULL	NULL

Слика 22. Приказ базе података након извршења нашег теста

Коментар: на слици је приказана секција софтвера MySQL Workbench, који сам користио за приступ бази података. Приказана је листа свих претходно-присутних елемената, као и наше новододане књиге. Сви подаци унешени у базу и видљиви на слици се могу видети у слици кода одозго.

5.2 Test Case 2

Опис: Верификовати да је могуће видети све податке о новокреираној књизи

Корак 1 - покренути апликацију Private Library

Корак 2 - кликнути на LOGIN дугме

Корак 3 - у новоотвореном прозору унети "test" као и корисничко име, и као лозинку

Корак 4 - отићи на Books страницу и лоцирати "details" линк / дугме на нашој новој књизи

Корак 5 - валидирати да ли су присутни СВИ подаци: од наслова до аутора

Очекивани резултат 1: логин прозор је отворен / интерактиван

Очекивани резултат 2:допуштено је логовање са претходно дефинисаним креденцијалима

Очекивани резултат 3: могуће је имати интеракцију са књигом

Очекивани резултат 4: присутни су сви претходно-унешени подаци

Прави резултат: логин прозор је отворен / интерактиван, допуштено је логовање са претходно дефинисаним креденцијалима, могуће је имати интеракцију са књигом, сви подаци су присутни и валидирани од стране теста

5.2.1 Код и спровођење теста

Образац на самом почетку је фактички идентичан оном од пре, тако да нећу много тога понављати: палимо апликацију тако што је покрећемо са порта 3000.

Затим, улазимо у LOGIN страницу кликом на кореспондирајуће дугме. Чекамо да се учитају оба поља за унос података, и затим уносимо у оба случајева податке ”test" који омогућују административне привилегије у апликацији. Кликамо дугме за валидирање унешених података и, ако успешно, поново одлазимо на страницу Books.

Када се налазимо на претходно споменутој страници, онда уз помоћ xpath путање тражимо "details" дугме наше новододане књиге. Очекивано чекамо да се прво учита како би избегли потенцијалне некомпатибилности посредством разлике у синхронизацији, затим кликамо на дугме и одлазимо на секцију са детаљима.

Овде идентификујем све присутне елементе, такође помоћу њихових xpath путања - једну за другом, сваку индивидуално ”провлачим” кроз посебан assertTrue код, који валидира да ли је оно поднешено као улазни параметар истинито, такорећи. У нашим случајевима, принцип валидације је базиран на томе да ли се текстуалне вредности обележених елемената поклапају са експлицитно дефинисаним текстуалним вредностима (игноришемо разлике у великим и малим словима коришћењем уграђене методе .equalsIgnoreCase(text)). Уколико су сви assert-ови прошли, апликација се гаси и тест сматрамо успешно имплементираним.

NewBook.java | BookDetails.java | BookDeletion.java | BookEdit.java

```java
package com.teletubici;

import static org.testng.Assert.assertTrue;

public class BookDetails {

    public static void main(String[] args) {

        WebDriver driver = new ChromeDriver();
        WebDriverWait wait;
        wait = new WebDriverWait(driver, Duration.ofSeconds(10));
        driver.navigate().to("http://localhost:3000/books");

        WebElement loginBtn = driver.findElement(By.xpath("/html/body/div/div/header/div/button[2]"));
        wait.until(ExpectedConditions.elementToBeClickable(loginBtn)).click();
        WebElement userName = driver.findElement(By.xpath("/html/body/div[2]/div[3]/div/div[1]/div[1]/div/input"));
        wait.until(ExpectedConditions.visibilityOf(userName)).sendKeys("test");
        WebElement passWord = driver.findElement(By.xpath("/html/body/div[2]/div[3]/div/div[1]/div[2]/div/input"));
        wait.until(ExpectedConditions.visibilityOf(passWord)).sendKeys("test");
        wait.until(ExpectedConditions.elementToBeClickable(driver.findElement(By.xpath("/html/body/div[2]/div[3]/div/div[2]/button[1]")))).click();

        WebElement navBtn = driver.findElement(By.xpath("/html/body/div/div/header/div/button[1]"));
        wait.until(ExpectedConditions.elementToBeClickable(navBtn)).click();

        WebElement booksBtn = driver.findElement(By.xpath("/html/body/div/div/div/div/ul[1]/li/a"));
        wait.until(ExpectedConditions.elementToBeClickable(booksBtn)).click();

        driver.findElement(By.xpath("/html/body/div/div/main/div[2]/div[2]/div[7]/div/div[2]/div/button[1]")).click();

        assertTrue(driver.findElement(By.xpath("/html/body/div/div/main/div[2]/div[1]/p")).getText().equalsIgnoreCase("Na Drini Cuprija"));
        assertTrue(driver.findElement(By.xpath("/html/body/div/div/main/div[2]/div[2]/div[2]")).getText().equalsIgnoreCase("Year: 1995"));
        assertTrue(driver.findElement(By.xpath("/html/body/div/div/main/div[2]/div[2]/div[3]")).getText().equalsIgnoreCase("ISBN: 1231231231231"));
        assertTrue(driver.findElement(By.xpath("/html/body/div/div/main/div[2]/div[2]/div[4]")).getText().equalsIgnoreCase("Genre: Horor"));
        assertTrue(driver.findElement(By.xpath("/html/body/div/div/main/div[2]/div[2]/div[5]")).getText().equalsIgnoreCase("Rating: 5"));
        driver.quit();
    }

}
```

Слика 23. Код другог теста, од почетка до краја

Коментар: у овом случају нећу приказивати измене у бази података, пошто их нема.

5.3 Test Case 3

Опис: Верификовати да је могуће изменити све податке о књизи

Корак 1 - покренути апликацију Private Library

Корак 2 - кликнути на LOGIN дугме

Корак 3 - у новоотвореном прозору унети "test" као и корисничко име, и као лозинку

Корак 4 - отићи на Books страницу и лоцирати "edit" линк / дугме на нашој новој књизи

Корак 5 - обрисати садржај свих поља

Корак 6 - испунити сва поља, пазећи да унешени подаци испуњавају критеријуме њихових кореспондирајућих поља

Корак 7 - отићи назад на Books страницу и проверити да ли су подаци књиге измењени

Очекивани резултат 1: логин прозор је отворен / интерактиван

Очекивани резултат 2: допуштено је логовање са претходно дефинисаним креденцијалима

Очекивани резултат 3: могуће је изменити све податке о постојећој књизи

Очекивани резултат 4: присутни су сви претходно-унешени подаци

Прави резултат: логин прозор је отворен / интерактиван, допуштено је логовање са претходно дефинисаним креденцијалима, могуће је изменити све податке о постојећој књизи, сви подаци су присутни и валидирани од стране теста

5.3.1 Код и спровођење теста

Образац на самом почетку је фактички идентичан оном од пре, тако да нећу много тога понављати: палимо апликацију тако што је покрећемо са порта 3000.

Затим, улазимо у LOGIN страницу кликом на кореспондирајуће дугме. Чекамо да се учитају оба поља за унос података, и затим уносимо у оба случајева податке ”test" који омогућују административне привилегије у апликацији. Кликамо дугме за валидирање унешених података и, ако успешно, поново одлазимо на страницу Books.

Када се налазимо на претходно споменутој страници, уз помоћ xpath путање тражимо "edit" дугме наше новододане књиге. Очекивано чекамо да се прво учита како би избегли потенцијалне некомпатибилности посредством разлике у синхронизацији, затим кликамо на дугме и одлазимо на секцију са детаљима.

Отворена је страница или секција апликације која нам омогућава да мењамо или модификујемо податке о нашој одабраној књизи. Иначе бих користио методу .clear() у намери да обришем садржај свих инпут поља, међутим Eclipse из неког разлога није желео да сарађује са мном. Као алтернативу сам осмислио друго, занимљивије решење: симулирање селектовања целог и затим брисања текста. Ово сам урадио тако што сам првенствено лоцирао сва инпут поља уз помоћ њихових путања. Затим, клонирао сам сваку линију кода, с тим да је само последњих неколико линија кода измењено. Наиме, свако инпут поље се прво селектује, затим се искористи CTRL + а пречица (уз помоћ које се селектује све присутно у том, неком елементу, прозору, апликацији итд). Затим, тест ”притиска” DELETE тастер, којим брише све податке унутар инпут поља. Након што је цео текстуални садржај обрисан, следи низ линија кода уз помоћу којих опонашамо понашање кода у првом тестном случају, у смислу да селектујемо редом сва инпут поља и убацујемо податке које желимо.

Када смо завршили, враћамо се на Books страницу и тражимо да ли постоји HTML span елемент који садржи текстуални податак који кореспондира недавно-измењеном наслову наше књиге, ”Na Dunavu Morava”. Користимо assert assertEquals како би проверили и валидирали успешност теста.

Уколико је assert прошао, апликација се гаси и тест сматрамо успешно имплементираним.

NewBook.java | BookDetails.java | BookDeletion.java | BookEdit.java

```
1  package com.teletubici;
2
3  import static org.testng.Assert.assertEquals;
15
16 public class BookEdit {
17
18     public static void main(String[] args) {
19         WebDriver driver = new ChromeDriver();
20         WebDriverWait wait;
21         wait = new WebDriverWait(driver, Duration.ofSeconds(10));
22         driver.navigate().to("http://localhost:3000/books");
23
24         WebElement loginBtn = driver.findElement(By.xpath("/html/body/div/div/header/div/button[2]"));
25         wait.until(ExpectedConditions.elementToBeClickable(loginBtn)).click();
26         WebElement userName = driver.findElement(By.xpath("/html/body/div[2]/div[3]/div/div[1]/div[1]/div/input"));
27         wait.until(ExpectedConditions.visibilityOf(userName)).sendKeys("test");
28         WebElement passWord = driver.findElement(By.xpath("/html/body/div[2]/div[3]/div/div[1]/div[2]/div/input"));
29         wait.until(ExpectedConditions.visibilityOf(passWord)).sendKeys("test");
30         wait.until(ExpectedConditions.elementToBeClickable(driver.findElement(By.xpath("/html/body/div[2]/div
   [3]/div/div[2]/button[1]")))).click();
31
32         WebElement navBtn = driver.findElement(By.xpath("/html/body/div/div/header/div/button[1]"));
33         wait.until(ExpectedConditions.elementToBeClickable(navBtn)).click();
34
35         WebElement booksBtn = driver.findElement(By.xpath("/html/body/div/div/div/div/ul[1]/li/a"));
36         wait.until(ExpectedConditions.elementToBeClickable(booksBtn)).click();
37
38         wait.until(ExpectedConditions.elementToBeClickable(driver.findElement(By.xpath("/html/body/div/div/main/div
   [2]/div[2]/div[7]/div/div[2]/div/button[3]")))).click();
```

Слика 24. Први део кода трећег теста

```
        wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.xpath("/html/body/div/div/main/div
[2]/div/div[1]/input")))).sendKeys(Keys.CONTROL + "a");
        wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.xpath("/html/body/div/div/main/div
[2]/div/div[1]/input")))).sendKeys(Keys.DELETE);
        wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.xpath("/html/body/div/div/main/div
[2]/div/div[2]/input")))).sendKeys(Keys.CONTROL + "a");
        wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.xpath("/html/body/div/div/main/div
[2]/div/div[2]/input")))).sendKeys(Keys.DELETE);
        wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.xpath("/html/body/div/div/main/div
[2]/div/div[3]/input")))).sendKeys(Keys.CONTROL + "a");
        wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.xpath("/html/body/div/div/main/div
[2]/div/div[3]/input")))).sendKeys(Keys.DELETE);
        wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.xpath("/html/body/div/div/main/div
[2]/div/div[4]/input")))).sendKeys(Keys.CONTROL + "a");
        wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.xpath("/html/body/div/div/main/div
[2]/div/div[4]/input")))).sendKeys(Keys.DELETE);
        wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.xpath("/html/body/div/div/main/div
[2]/div/div[1]/input")))).sendKeys("Na Dunavu Morava");
        wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.xpath("/html/body/div/div/main/div
[2]/div/div[2]/input")))).sendKeys("5431253405");
        wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.xpath("/html/body/div/div/main/div
[2]/div/div[3]/input")))).sendKeys("1112");
        wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.xpath("/html/body/div/div/main/div
[2]/div/div[4]/input")))).sendKeys("0");
        Select bookGenre = new Select(driver.findElement(By.xpath("/html/body/div/div/main/div[2]/div/div
[5]/select")));
        bookGenre.selectByIndex(0);
        Select authorNames = new Select(driver.findElement(By.xpath("/html/body/div/div/main/div[2]/div/div
[6]/select")));
        authorNames.selectByIndex(1);
        wait.until(ExpectedConditions.elementToBeClickable(driver.findElement(By.xpath("/html/body/div/div/main/div
[2]/div/button")))).click();

        wait.until(ExpectedConditions.elementToBeClickable(booksBtn)).click();
        assertEquals(driver.findElement(By.xpath("//span[text()='Na Dunavu Morava']")).getText(), "Na Dunavu
Morava");
        driver.quit();

    }

}
```

Слика 25. Други део кода трећег теста

Коментар: код је био сувише дугачак како би (лепо) стао на једну слику, па сам га морао убацити као две слике. Баш као што сам и споменуо у тексту одозго, алтернатива крајње примитивној CTRL, 'a' + DELETE комбинацији карактера би била просто метода .clear(), међутим Selenium има добро документован баг са том методом па сам био форсиран да нађем алтернативно решење.

id	isbn	rating	title	year	genre_id
1	1234567894	5	Ma šta mi reče	1972	1
2	1234567894	5	Bukvar dečijih prava	1996	1
3	1234567893	5	Zov tetreba	1977	1
4	1234567893	5	A Game of Thrones	1996	2
5	1234567893	5	The Winds of Winter	2016	2
6	1234567893	5	Fire and Blood	2018	2
41	5431253405	0	Na Dunavu Morava	1112	1
NULL	NULL	NULL	NULL	NULL	NULL

Слика 26. Приказ базе података након извршења трећег теста

Коментар: скроз доњи елемент базе је након покретања упита видљиво другачији од оног првобитно доданог. Сваки податак и по дужини и по типу одговара инпут пољима у апликацији.

5.4 Test Case 4

Опис: Верификовати да је могуће уколинити књигу

Корак 1 - покренути апликацију Private Library

Корак 2 - кликнути на LOGIN дугме

Корак 3 - у новоотвореном прозору унети "test" као и корисничко име, и као лозинку

Корак 4 - отићи на Books страницу и лоцирати "delete" линк / дугме на нашој новој књизи

Корак 5 - обрисати књигу

Очекивани резултат 1: логин прозор је отворен / интерактиван

Очекивани резултат 2: допуштено је логовање са претходно дефинисаним креденцијалима

Очекивани резултат 3: могуће је елиминисати књигу

Прави резултат: логин прозор је отворен / интерактиван, допуштено је логовање са претходно дефинисаним креденцијалима, књига је уклоњена и из апликације естетски, и из базе података.

5.4.1 Код и спровођење теста

Образац на самом почетку је фактички идентичан оном од пре, тако да нећу много тога понављати: палимо апликацију тако што је покрећемо са порта 3000.

Код овог теста је изузетно симплистичан, пошто ни у Селенијуму, ни у Јави не постоји неки, такорећи ”експлицитни” начин или метод валидације ОДСУСТВА неког елемента. Пало ми је на памет неколико метода, на пример неки try catch који проверава да ли елемент постоји уз помоћ .isDisplayed() методе, међутим ништа из неког разлога није хтело да функционише.

Улазимо у LOGIN страницу кликом на кореспондирајуће дугме. Чекамо да се учитају оба поља за унос података, и затим уносимо у оба случајева податке ”test" који омогућују административне привилегије у апликацији. Кликамо дугме за валидирање унешених података и, ако успешно, поново одлазимо на страницу Books.

Када се налазимо на претходно споменутој страници, уз помоћ xpath путање тражимо "delete" дугме наше новододане књиге. Очекивано чекамо да се прво учита како би избегли потенцијалне некомпатибилности посредством разлике у синхронизацији, затим кликамо на дугме. Као што сам напоменуо на почетку: не постоји ниједна уграђена валидација нити функција уз помоћу које би проверили да ли неки елемент **не** постоји, тако да у овом случају морамо да се задовољимо исходом теста без assert-а било које врсте.

NewBook.java | BookDetails.java | BookDeletion.java | BookEdit.java

```
package com.teletubici;

import java.time.Duration;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

public class BookDeletion {

    public static void main(String[] args) {
        WebDriver driver = new ChromeDriver();
        WebDriverWait wait;
        wait = new WebDriverWait(driver, Duration.ofSeconds(10));
        driver.navigate().to("http://localhost:3000/books");

        WebElement loginBtn = driver.findElement(By.xpath("/html/body/div/div/header/div/button[2]"));
        wait.until(ExpectedConditions.elementToBeClickable(loginBtn)).click();
        WebElement userName = driver.findElement(By.xpath("/html/body/div[2]/div[3]/div/div[1]/div
[1]/div/input"));
        wait.until(ExpectedConditions.visibilityOf(userName)).sendKeys("test");
        WebElement passWord = driver.findElement(By.xpath("/html/body/div[2]/div[3]/div/div[1]/div
[2]/div/input"));
        wait.until(ExpectedConditions.visibilityOf(passWord)).sendKeys("test");
        wait.until(ExpectedConditions.elementToBeClickable(driver.findElement(By.xpath("/html/body/div[2]/div
[3]/div/div[2]/button[1]")))).click();

        WebElement navBtn = driver.findElement(By.xpath("/html/body/div/div/header/div/button[1]"));
        wait.until(ExpectedConditions.elementToBeClickable(navBtn)).click();

        WebElement booksBtn = driver.findElement(By.xpath("/html/body/div/div/div/div/ul[1]/li/a"));
        wait.until(ExpectedConditions.elementToBeClickable(booksBtn)).click();

        WebElement delete = driver.findElement(By.xpath("/html/body/div/div/main/div[2]/div[2]/div[7]/div/div
[2]/div/button[2]"));
        wait.until(ExpectedConditions.elementToBeClickable(delete)).click();

        driver.quit();
    }
}
```

Слика 27. Код четвртог теста, од почетка до краја

Коментар: код је кратак и концизан, фактички идентичан коду другог теста. У валидност теста се уверавамо сопственим очима: видимо да се на страници више не налази књига, и видимо да елемент више не постоји у бази података.

Result Grid | Filter Rows: | Edit:

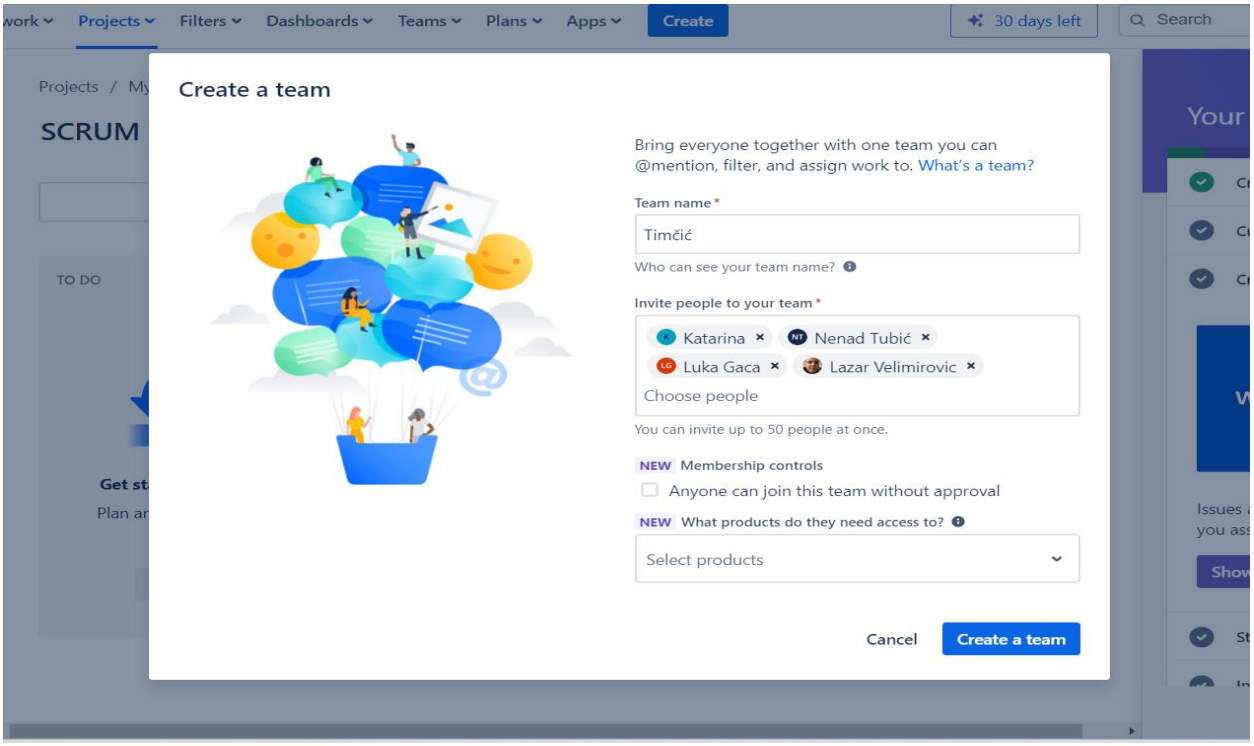
id	isbn	rating	title	year	genre_id
1	1234567894	5	Ma šta mi reče	1972	1
2	1234567894	5	Bukvar dečijih prava	1996	1
3	1234567893	5	Zov tetreba	1977	1
4	1234567893	5	A Game of Thrones	1996	2
5	1234567893	5	The Winds of Winter	2016	2
6	1234567893	5	Fire and Blood	2018	2
NULL	NULL	NULL	NULL	NULL	NULL

Слика 28. База података је лишена наше књиге.

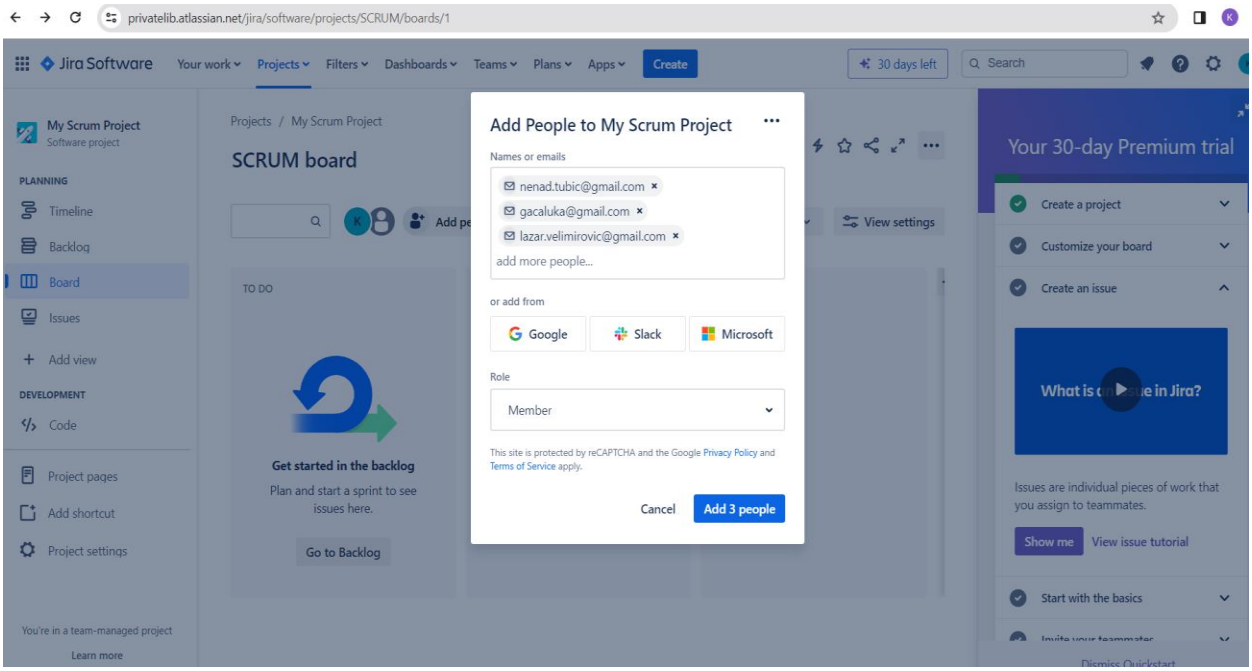
6. ПРИКАЗ РАЗВОЈА СОФТВЕРА УПОТРЕБОМ АГИЛНЕ МЕТОДОЛОГИЈЕ

Tabela 2 – Podela zadataka unutar tima za izradu praktičnog projekta

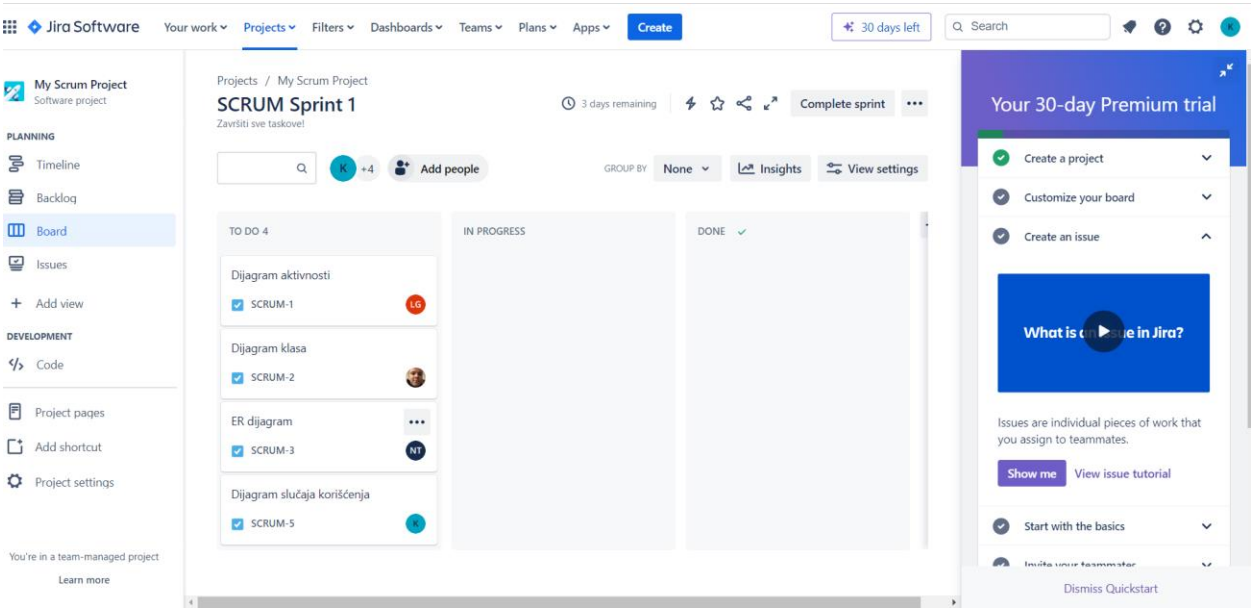
Aktivnost	Product owner	Srum master	Članovi Scrum tim-a
Organizacija sastanka za planiranje sprinta		+	
Sastavlja Epic/User Story-je u Backlog-u	+		
Postavlja prioritete u Backlog-u	+		
Planiranje sprintova i njihovih ciljeva (sprint goals)			+
Razvoj softvera			+
Testiranje softvera			+
Organizacija događaja *Sprint Review Meeting, Sprint Retrospective Meeting*		+	
Vođenje zapisnika o sastancima/događajima i njihova objava na boardu		+	
Daje povratnu informaciju (feedback)	+		



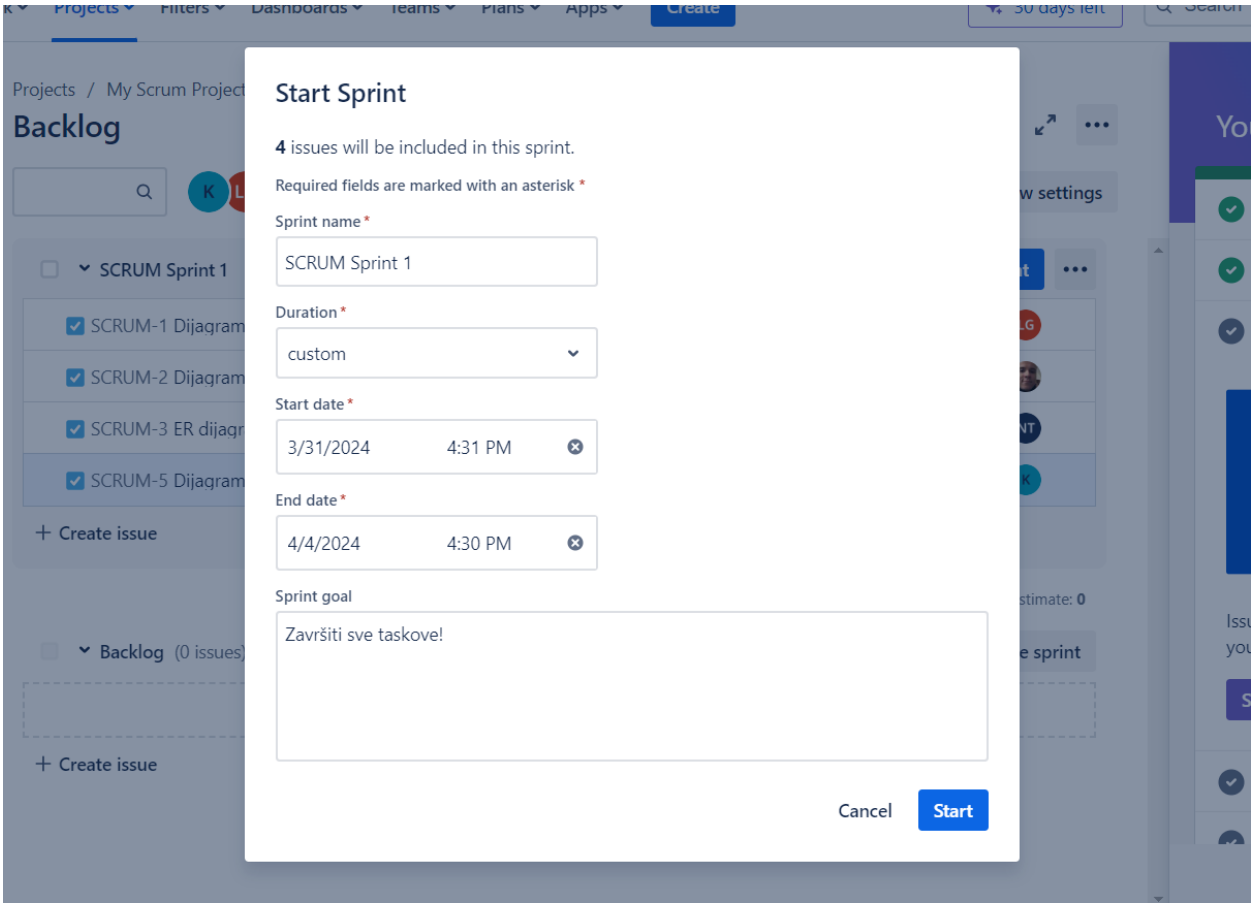
Слика . Креирање тима



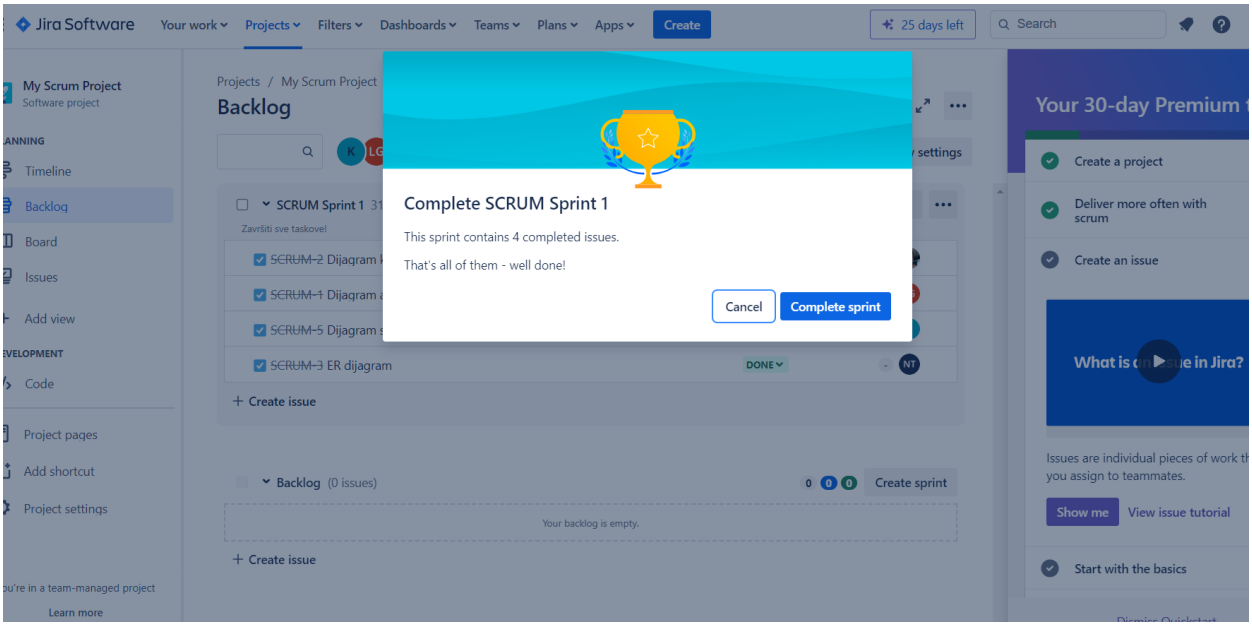
Слика . Додавање чланова тима на пројекту



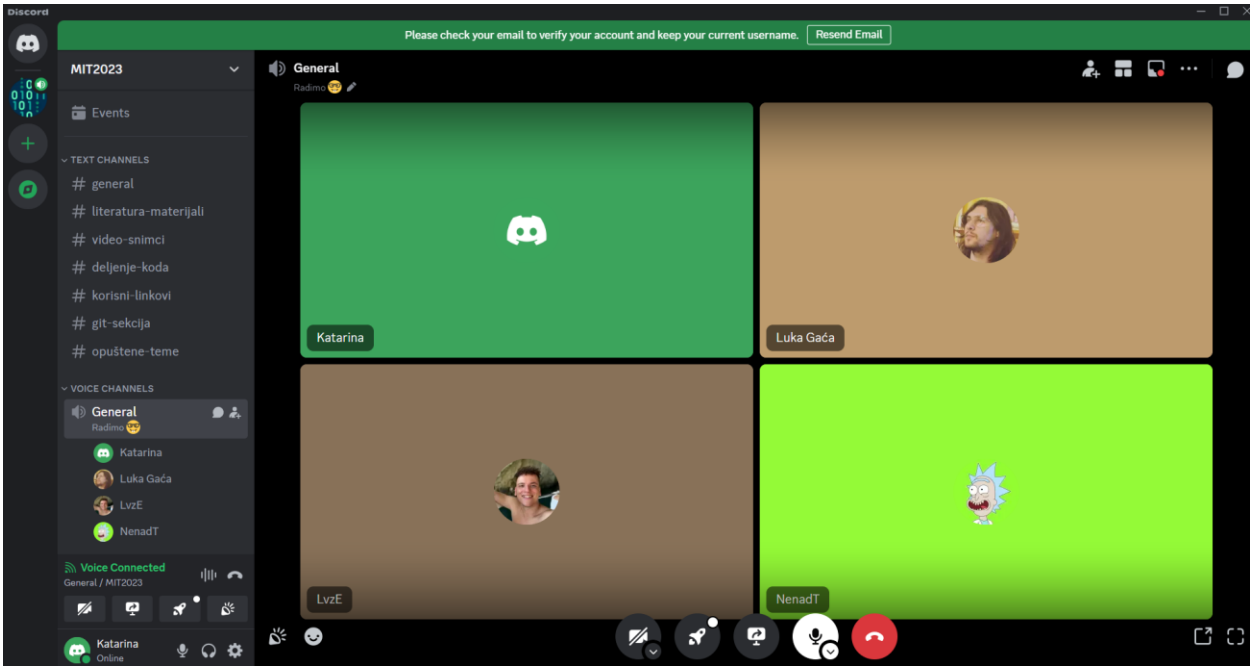
Слика . Додељени таскови унутар спринта



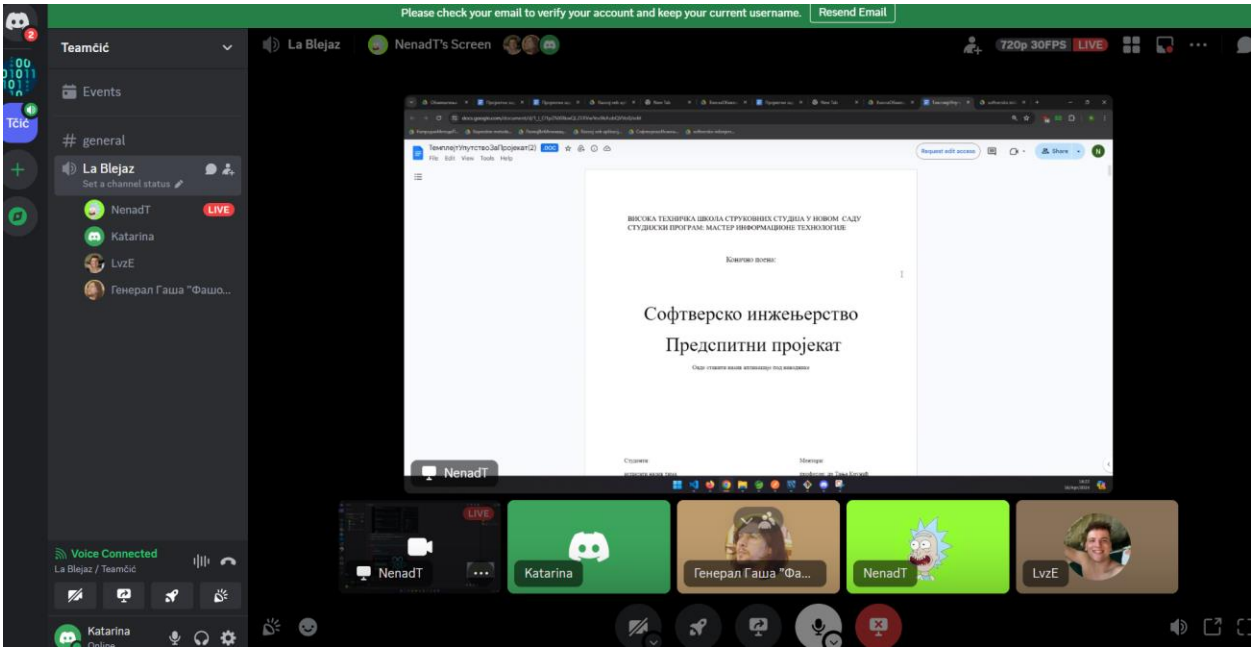
Слика . Стартовање сприта



Слика . Успешно одрађен спринт



Слика . Онлајн састанак преко Discord-а



Слика . Онлајн састанак преко Discord-а, пролажење кроз упутсто за пројекат

7. ЗАКЉУЧАК

Након завршетка пројекта, можемо закључити да је апликација релативно успешно креирана и главне функционалности функционишу без проблема. Све главне функционалности које смо желели да апликација има су доступне и раде онако како су и замишљене. Наравно, као и свака апликација, и ова је подложна променама, проширењима и дорадама.

Иако је један члан тима био спречен да испуни свој део задатка, што нас је приморало да приступимо изради резервне варијанте фронтенда, то није значајно умањило успешност израде пројекта. Тим је успео да се ефикасно прилагоди ситуацији и да оствари све задате циљеве. Ово искуство нам је било изузетно корисно, јер смо морали да савладамо нове технологије како бисмо испунили све што се од нас тражило у пројекту. Такође, пројекат нам је омогућио да се приближимо реалној слици израде једног пројекта у индустрији, показујући нам важност функционалног рада у тимовима и неопходност сваког члана да да свој максимум. Све у свему, иако смо се суочили са неочекиваном ситуацијом, тимски рад и посвећеност омогућили су нам да успешно завршимо пројекат.