Tehničko veleučilište u Zagrebu
Vrbik 8, Zagreb

Bibliotheca – web aplikacija za upravljanje knjižnicom

Napredne tehnike projektiranja web servisa

Neven Jakopčić

Zagreb, 30.11.2021.

Sadržaj

Popis slika

1. Sažetak

Ovaj seminar namijenjen je demonstraciji autorove sposobnosti da razvije full-stack aplikaciju, odnosno aplikaciju koja se sastoji od web aplikacije koja se prikazuje na pregledniku, te backend-a, odnosno programa koji prima i obrađuje zahtjeve koje prima od web aplikacije, te mu servira podatke potrebne za rad. U tu svrhu, korištene su Spring Boot biblioteka, s kojom imam dovoljno iskustva, koje sam stekao kroz obrazovanje i rad kao Junior Backend Developer. Za frontend, s kojim imam vrlo malo iskustva, koristio sam Vue.js biblioteku, koja je poznata po tome da je pristupačna početnicima u frontend razvoju, ali je jednako moćna kao i ostali, kompliciraniji frameworci, poput React-a i Angular-a.

Bibliotheca aplikacija je zamišljena kao sustav koji bi neka knjižnica mogla koristiti za katalog knjiga koje posjeduje, te kao sustav za upravljanje članstvima korisnika knjižnice. Osim toga, pruža običnim korisnicima uvid u katalog, te mogućnost rezervacije knjiga.

2. Uvod

Na tržištu postoje razne biblioteke, frameworci i rješenja koja omogućuju brz i lagan razvoj web aplikacija. Neki od frameworkova za razvoj frontend aplikacije su React i Angular, koji dominiraju trenutnim tržištem, te se često navode kao broj jedan i broj dva najpopularniji izbori. Na trećem mjestu bi se nalazio Vue.js, koji obećava pristupačnost početnicima u razvoju web aplikacija, te ubrzan tijek prilagođavanja i učenja kako se koristi. Zbog tog razloga sam odabrao upravo Vue.js, pošto imam jako malo iskustva s razvojom web aplikacija.

Postoje također i razni backend frameworkovi, no rijetko koji se može dičiti jednostavnošću i ubrzanim procesom učenja. Iz tog razloga sam odlučio koristiti Spring Boot, framework s kojim imam najviše iskustva, koje je stečeno kroz moje iskustvo na radu i studiranju.

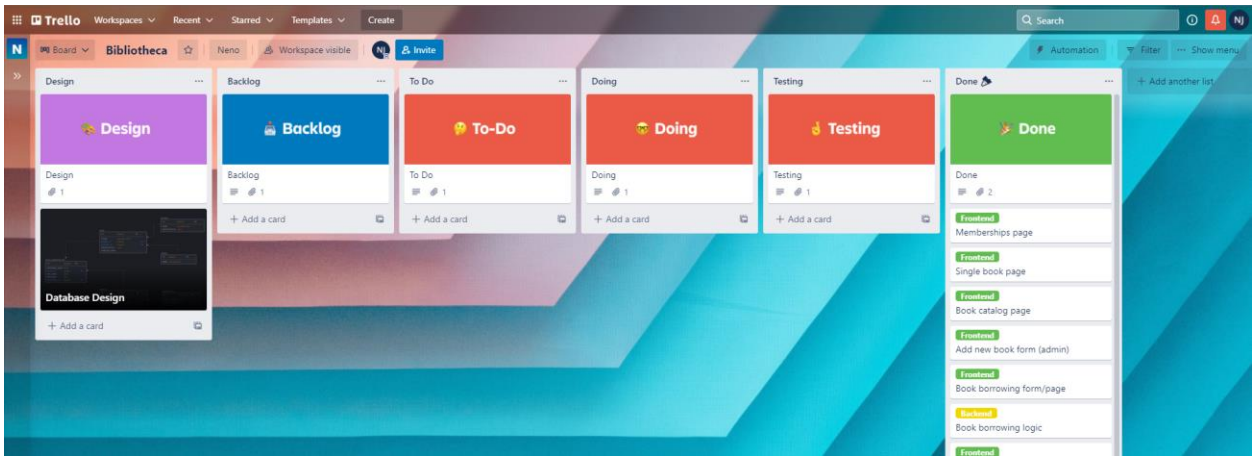
Osim Vue.js-a, odlučio sam koristiti još jednu tehnologiju s kojom nemam gotovo ništa iskustva, a to je Docker. Docker je iznimno moćan alat, te je gotovo sveprisutan u svijetu web development-a, ali i ostalim sferama programiranja.

Za projekt sam odabrao razviti relativno jednostavnu aplikaciju – aplikaciju za knjižnicu. Ovaj seminar opisuje elemente cijelog „stacka“ koji su bili potrebni za razvoj takve aplikacije.

3. Korištene tehnologije

3.1. Trello

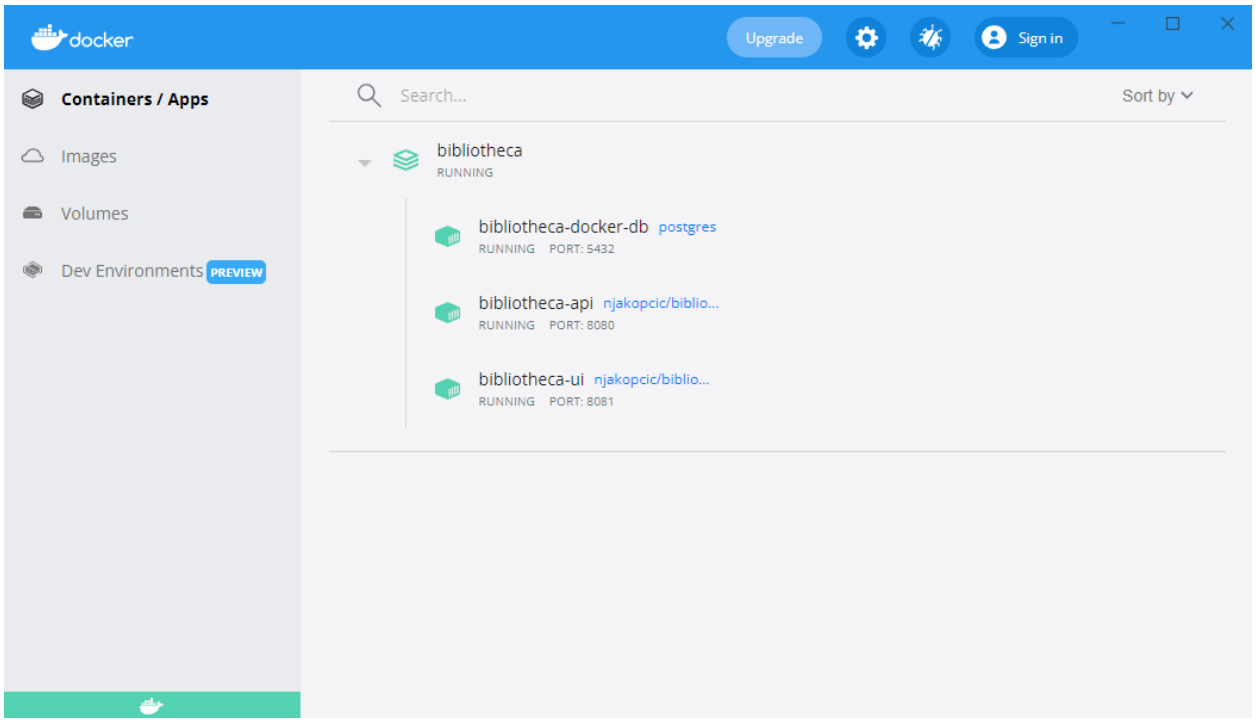
Trello je vizualni alat koji olakšava organiziranje projekata i kolaboraciju između ljudi koji rade na projektu. Omogućava razlaganje projekta na individualne zadatke koji se organiziraju na Kanban ploči. Na ovom projektu sam koristio Trello kako bih imao lakši pregled zadataka koje moram obaviti, te kako bih jasno definirao koje značajke projekta planiram implementirati. [1]



Slika 1 Trello ploča korištena za Bibliotheca projekt

3.2. Docker

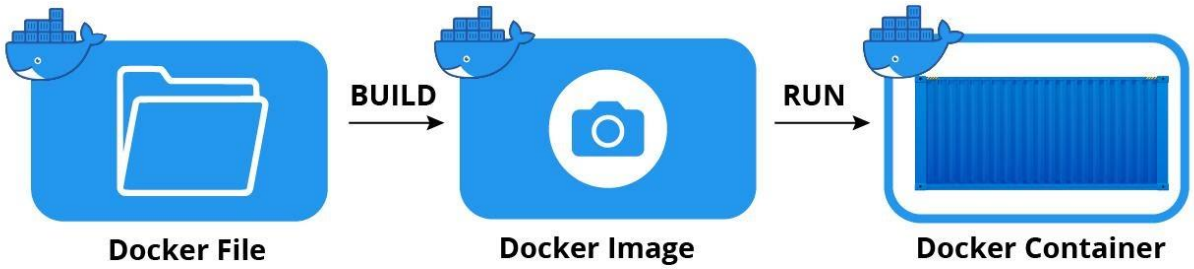
Docker je platforma otvorenog izvora za buildanje, deployment i upravljanje „kontejneriziranih“ aplikacija. Omogućuje programerima pakiranje aplikacija u „kontejnere“. Kontejneri (containers) su standardizirane komponente koje povezuju programski kod, ili buildane aplikacije, s bibliotekama operativnog sustava i bibliotekama o kojima aplikacija zavisi. Na taj način kontejneri omogućuju pokretanje i upravljanje aplikacija na bilo kojem radnom okruženju i operativnom sustavu, uz gotovo nepostojeći nivo truda potreban za pokretanje tih kontejnera.



Slika 2 Docker Desktop sučelje

Za razliku od virtual machine-ova, Docker kontejneri ne prave cijelu virtualnu instancu operativnog sustava, već uključuju samo procese operativnog sustava i potrebne elemente za pokretanje i rad aplikacije. Ova činjenica čini kontejnere mnogo efikasnijim, jer ne moraju glumiti cijeli operativni sustav, već koriste dijelove operativnog sustava na kojem se vrte. Također zauzimaju znatno manje memorije. Ove značajke dovode do povećanja produktivnosti programera koji razvijaju softver kad koriste Docker. [2]

Kod korištenja Dockera, bitno je razlikovati tri glavna pojma. Prvi pojam je Dockerfile, datoteka koja opisuje značajke image-a, te proces koji se treba provesti kod izgradnje istoga. Dockerfile se može podijeliti na više stadija, što omogućuje paralelizaciju kod izgradnje image-a, te cache-iranje već stadija koji se nisu promijenili između iteracija image-a.



Slika 3 Docker pipeline

Image je datoteka koja funkcionira kao predložak kontejnera, opisan u Dockerfile-u. Omogućuje pokretanje kontejnera na bilo kojem hardveru. Sadrže izvršiv kod aplikacije koju želimo pokrenuti. Dockerfile nije potreban za pokretanje kontejnera; dovoljno je da korisnik posjeduje kopiju image-a, te ju može pokrenuti kao kontejner.

Docker kontejner je pokrenuta instanca Docker image-a. Korisnici i administratori mogu vršiti interakciju s njima, te podešavati njihove postavke koristeći naredbe. Također je značajno spomenuti volumene (volumes), koji omogućavaju spremanje podataka kako ih ne bi izgubili prilikom gašenja i ponovnog paljenja kontejnera. Najjednostavniji primjer korištenja volumena je za održavanje trajnosti podataka spremljenih u bazama podataka. [3]

3.3. Docker-compose

Docker-compose, odnosno compose, je jednostavan alat koji omogućava definiranje i pokretanje Docker aplikacija koje se protežu kroz više kontejnera. Za definiranje servisa o kojima naša aplikacija ovisi koristimo YAML file, u kojem popisujemo servise koji tvore našu aplikaciju. [4]

```yaml
version: '3.3'

services:
   db:
     image: mysql:5.7
     volumes:
       - db_data:C:\Users\User\Desktop\dcompose
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: rootwordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress

   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "8000:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
volumes:
    db_data:
```

Slika 4 Primjer docker-compose.yml datoteke

3.4. PostgreSQL

PostgreSQL je napredna i moderna relacijska baza podataka otvorenog izvora. Jedna je od najmoćnijih baza podataka, te četvrta po popularnosti, ispod Oracle-a, MySQL-a i MS SQL Servera. [5] Neke od glavnih značajki PostgreSQL-a koje dovode do njene rasprostranjenosti su činjenice da je besplatna baza s visokom mogućnošću nadograđivanja, te to što je otvorenog izvora. [6]

3.5. Spring Boot

Spring Boot je framework otvorenog izvora koji pruža Java programerima platformu za razvoj serverske logike web aplikacija. Centralni dio Spring Boota je Spring framework, koji pruža mogućnosti automatskog ubrizgavanja ovisnih komponenti na mjesta gdje su potrebne (dependency injection), čime olakšava razvoj i mogućnost testiranja cijelokupne aplikacije, ali i njenih individualnih elemenata kroz unit testove.

Postoje mnogi dependency-ji koje možemo dodati u Spring Boot aplikaciju koji nam omogućuju dodavanje stvari poput povezivanja sa raznoraznim bazama podataka, LDAP sustavima, RabbitMQ-om, Kafkom, i tako dalje, ili ako želimo dodati neke značajke poput sigurnosnih mjera, validacije i sličnog. [7]

3.6. Liquibase

Liquibase je projekt otvorenog izvora koji omogućava brz razvoj i upravljanje shemama baza podataka. Kroz XML, YAML, JSON i SQL datoteke možemo definirati sheme, izmjenjivati ih, te dodavati podatke. Liquibase skripte se sastoje od „changesetova“, što su jedinstveni komadići promjena baze. Liquibase koristi „changelog“ koji prati koji su changesetovi već implementirani na bazi. [8]

3.7. Vue.js

Vue.js je progresivni JavaScript framework za razvoj korisničkih sučelja na webu, odnosno frontend-a web aplikacija. Razlikuje se od sličnih frameworkova po tome što je iznimno pristupačan svakome tko ima osnovno znanje HTML-a, CSS-a i JavaScript. Sintaksa kojom se definira izgled stranice je slična običnom HTML-u. Usprkos prividnoj jednostavnosti i brzini kojom ga osoba može naučiti, ne kompromitira na mogućnostima koje pruža, te je kao takav odlična alternativa frameworkovima poput Angular-a i React-a. [9]

3.8. Vuetify

Vuetify je biblioteka gotovih komponenti otvorenog izvora namijenjena za olakšani razvoj vizualno privlačnih Vue.js aplikacija. Prati Material Design specifikaciju, te je zbog toga koristan kod razvoja aplikacija koje se trebaju prikazivati na raznim sučeljima. Pruža širok spektar gotovih komponenti, uz veliku mogućnost prilagođavanja istih specifičnim potrebama web aplikacije u kojima se koriste. [10]

4. Pregled koda

4.1. docker-compose.yml

Polazna točka za buildanje i pokretanje ovog projekta je docker-compose.yml datoteka. Ona definira mjesta gdje se nalaze Dockerfileovi i prateći programski kod image-a koje je potrebno sastaviti. Kroz ovu datoteku definiramo neke potrebne postavke, kao što su korisnički podaci za bazu podataka te ime same baze koju ćemo uspostaviti. Navode se veze između kontejnera, odnosno između kontejnera s frontendom i kontejnera s backendom, te kontejnera baze podataka i kontejnera s backendom. Osim toga, ovdje određujemo koji kontejner ovisi o kojemu, te koje portove od kontejnera želimo izložiti (expose) kako bi s ugostujućeg operativnog sustava mogli imati pristup aplikacijama unutar kontejnera.

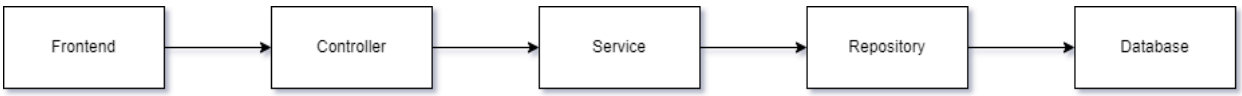
Slijedi dio docker-compose.yml datoteke vezan za backend kontejner, koji opisuje navedene stavke:

```
bibliotheca-api:
  build:
    context: ./bibliotheca-api
  image: njakopcic/bibliotheca
  container_name: bibliotheca-api
  ports:
    - "8080:8080"
  depends_on:
    - bibliotheca-db
  networks:
    - ui-api
    - api-db
  expose:
    - 8080
```

Docker-compose je u ovom projektu prvotno korišten kako bih stekao znanje korištenja Docker servisa, ali i radi omogućavanja čim lakše distribucije i pokretanja projekta. Kroz dvije naredbe (docker-compose build && docker-compose up) moguće je podići cijeli projekt, uključujući i bazu podataka.

4.2. Backend

Spring Boot backend Bibliotheca je, kao i što je tipično za ostale Spring Boot projekte, podijeljen na tri sloja. To su repository sloj, koji služi kao sučelje za komunikaciju između baze podataka i backend-a, controller sloj, koji „presreće“ HTTP zahtjeve koji dolaze s frontend-a, te ih potom šalje na service sloj, na kojem se odrađuje poslovna logika aplikacije, te služi kao posrednik između controller i repository sloja.



Slika 5 Slojevi backend-a

4.2.1. Dockerfile

Dockerfile koji opisuje proces skidanja nužnih komponenti s Interneta, te proces izgradnje i pokretanja gotove aplikacije je podijeljen u dva stadija. Prvi stadij (target) skida image koji sadrži Maven i JDK 11, koji su potrebni za izgradnju backenda. Kopira se relevantna pom.xml datoteka, koja opisuje sve komponente koje Maven treba skinuti sa online repozitorija, o kojima backend ovisi. Nakon toga se kopira izvorni kod aplikacije, te se poziva mvn package naredba, koja gradi samu aplikaciju.

```
FROM maven:3.6.3-jdk-11-slim AS target
WORKDIR /build
COPY pom.xml .
RUN mvn dependency:go-offline

COPY src/ /build/src/
RUN mvn clean package
```

Drugi stadij Dockerfilea (run) sadrži naredbe koje na Docker kontejneru prvo skinu JDK 11 s Interneta, potreban za rad aplikacije, te dodaju novog Linux korisnika „spring“. Ovo se tipično radi iz sigurnosnih razloga. Nakon toga se kopira sastavljeni backend iz target stadija, te se pokreće kroz Javu.

```
FROM openjdk:11-jre-slim AS run

RUN groupadd -r spring && useradd --no-log-init -r -g spring spring
USER spring:spring

COPY --from=target /build/target/bibliotheca-api-0.0.1-SNAPSHOT.jar
/usr/local/lib/bibliotheca.jar
ENTRYPOINT ["java","-jar","/usr/local/lib/bibliotheca.jar"]
```

4.2.2. Maven

Skidanje dependency-ija i build-anje backend aplikacije se provodi kroz Maven. U tu svrhu, pom.xml datoteka popisuje potrebne elemente. Slijedi primjer zapisa koji specificira spring-boot-starter-data-jpa dependency, koji se koristi za lakše programiranje poziva na bazu podataka:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
```

4.2.3. application.yml

Ova datoteka opisuje neke postavke Spring Boot aplikacije. Ovaj projekt ne zahtjeva dugačak application.yml zbog relativne jednostavnosti problematike koju ovaj backend rješava, niti korištenje profila koji se koriste u enterprise aplikacijama koje se pokreću na raznim okruženjima, kao što su lokalno, testno i produkcijsko okruženje. Međutim, u ovoj datoteci su opisani podaci potrebni za povezivanje na bazu podataka, te neke postavke koje određuju način na koji se ORM (Object-relational Mapper) treba ponašati. Također definira „secret“, što je „seed“ koji se koristi kod kreiranja JWT Tokena koje ova aplikacija koristi za autentifikaciju korisnika.

```
spring:
  datasource:
    driver-class-name: org.postgresql.Driver
    url: jdbc:postgresql://bibliotheca-db:5432/bibliotheca
    username: bibliotheca
    password: bibliotheca
  jpa:
```

```
    hibernate:
      ddl-auto: none
    show-sql: true
    open-in-view: true
  mvc:
    log-request-details: true
hr.njakopcic.bibliotheca:
  secret: bibliotheca-secret
debug: true
```

4.2.4. Liquibase skripte

Liquibase se u ovom projektu koristi kako bi se olakšalo i ubrzalo uspostavljanje baze podataka. U ovom projektu sam se odlučio na korištenje XML formata za skripte zbog toga što sam koristio JPA Buddy, plugin za IntelliJ IDEA razvojno okruženje, koji trenutno ne pruža mogućnost generiranja YML skripti, iako je taj format tipično čitljiviji. Liquibase je u ovom projektu korišten ne samo za generiranje sheme baze podataka, već i za ubacivanje nekih početnih podataka.

Slijedi primjer Liquibase changeset-a koji opisuje tablicu „Book“, u koju se spremaju podaci o knjigama:

```
<changeSet id="1635437569320-2-book" author="Neven Jakopčić">
  <createTable tableName="book">
    <column autoIncrement="true" name="id" type="BIGINT">
      <constraints nullable="false" primaryKey="true"
primaryKeyName="pk_book"/>
    </column>
    <column name="title" type="VARCHAR(500)">
      <constraints nullable="false"/>
    </column>
    <column name="genre_id" type="BIGINT"/>
    <column name="author_id" type="BIGINT"/>
    <column name="description" type="TEXT"/>
  </createTable>
</changeSet>
```

4.2.5. Modeli

Model je naziv za bilo koju klasu koja funkcionira kao jednostavni spremnik podataka kojim upravlja Spring Boot aplikacija. Najčešće je model samo popis atributa, međutim

ponekad može sadržavati i neke jednostavne funkcije, kao što su getter i setter funkcije. U ovom projektu modeli ne sadrže funkcije u izvornom kodu, već su navedene funkcije i ostali „boilerplate“ kod zamijenjeni Lombok anotacijama. Također, atributi modelne klase na sebi imaju anotacije koje opisuju na koji način Hibernate treba mapirati svojstva klase na stupce tablice u bazi podataka. Druga riječ za model, i riječ kojom se Spring Boot koristi je „Entity“, odnosno entitet. Shodno tome, modeli na sebi imaju @Entity() anotaciju.

Slijedi primjer klase koja opisuje model knjige:

```
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "BOOK")
@Entity(name = "BOOK")
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID")
    private Long id;

    @Column(name = "TITLE", nullable = false, length = 500)
    private String title;

    @ManyToOne
    @JoinColumn(name = "GENRE_ID")
    private Genre genre;

    @ManyToOne
    @JoinColumn(name = "AUTHOR_ID")
    private Author author;

    @Column(name = "DESCRIPTION")
    private String description;
```

4.2.6. DTO (Data Transfer Object)

Kako bi se ubrzalo slanje podataka sa backend na frontend, te kako bi se izbjegli problemi koji mogu nastupiti kad bi se slali modelni objekti (kao što je beskonačno ugnježđivanje kod modela koji imaju referencu jedno na drugoga), koriste se DTO klase.

DTO, odnosno Data Transfer Object-i su objekti koji imaju minimalnu količinu podataka koju je potrebno slati na frontend.

Na sljedećem primjeru DTO objekta koji predstavlja knjigu, bitno je istaknuti razliku. Modelna klasa (Book) je u sebi sadržavala referencu na objekt „Membership“ klase (koja opisuje članstvo u knjižnici), dok BookDto u sebi sadrži samo broj (Long) koji naznačuje identifikator članstva:

```
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class BookDto implements Serializable {

    private static final long serialVersionUID = -6616574079094780345L;

    private Long id;
    private String title;
    private GenreDto genre;
    private AuthorDto author;
    private String description;
    private Long reservationId;
```

Osim za slanje podataka s backend-a na web aplikaciju, tj. frontend, DTO objekti se koriste i za komunikaciju u suprotnom smjeru. Takve klase su sadržane u „request“ paketu. DTO-ovi koji predstavljaju odgovor sadržani su u shodno tome u „response“ paketu. Request DTO-ovi sadrže minimalnu količinu podataka koja je potrebna za opisivanje zahtjeva na server. U ovom projektu se request DTO-ovi koriste isključivo za kreiranje novih podataka.

Slijedi primjer request-a objekta za kreiranje nove knjige:

```
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class CreateBookRequest {

    @NotBlank
```

```
    private String title;

    @NotNull
    private Long genreId;

    @NotNull
    private Long authorId;

    @NotBlank
    private String description;
}
```

4.2.7. Mapperi

Kako bi se modelni podaci konzistentno transformirali u DTO objekte, koriste se Mapper klase. Spring Boot sadrži anotacije namijenjene specifično za Mappere, ali nisu korištene u ovom projektu zbog relativne jednostavnosti problematike koji rješavaju, odnosno kako bi se izbjegla nepotrebna komplikacija backend-a. Stoga, u Bibliotheca projektu, mapperi su jednostavne klase s privatnim konstruktorom (ne mogu se instancirati) koje sadrže u sebi statičku metodu koja mapira modelni objekt u DTO objekt.

Slijedi primjer klase koja mapira Book modelni objekt u BookDto objekt:

```
public class BookDtoMapper {

    public static BookDto map(Book source, Long reservationId) {
        return BookDto.builder()
            .id(source.getId())
            .title(source.getTitle())
            .genre(GenreDtoMapper.map(source.getGenre()))
            .author(AuthorDtoMapper.map(source.getAuthor()))
            .description(source.getDescription())
            .reservationId(reservationId)
            .build();
    }

    private BookDtoMapper() {}
}
```

4.2.8. Repository sloj

Repository sloj služi za komunikaciju između backend-a i baze podataka. Repository sučelje se označava @Repository anotacijom. Sva repository sučelja (tipično) nasljeđuju JpaRepository sučelje, koje nude neke obične metode, kao što su save() za pohranjivanje novog zapisa u bazu, ili spremanje izmijenjenog zapisa, te findAll() koji vraća sve zapise iz baze. Također se nude opcije paginacije, i pretraživanja po „primjeru“ (Example), odnosno kreiranje krnjeg objekta preko kojeg se nalaze slični zapisi u bazi.

Osim navedenih i sličnih metoda, nudi se i opcija kreiranja proizvoljnih upita bez pisanja SQL query-ja, kroz definiranje metoda koje u svojem nazivu opisuju željeni query. Te metode nije potrebno implementirati, već Spring Boot generira prikladan query automatski. Slijedi primjer definicije UserRepository-ja, koji u sebi ima opisane metode:

```
 @Repository
public interface UserRepository extends JpaRepository<User, Long> {

    Optional<User> findByUsername(String username);
    Optional<User> findByEmail(String email);
}
```

Alternativa tom pristupu je @Query anotacija, koja omogućava pisanje SQL query-ja. Slijedi primjer ReservationRepository sučelja, u kojem su definirane metode s SQL query-jima:

```
public interface ReservationRepository extends JpaRepository<Reservation,
Long> {

    @Query(value = "select * from book_reservation br " +
                   "where br.book_id = :bookId " +
                   "order by id desc limit 1;", nativeQuery = true)
    Reservation findLatestReservation(@Param("bookId") Long bookId);

    @Query(value = "select br.id from book_reservation br " +
        "where br.book_id = :bookId " +
        "and br.returned = false ", nativeQuery = true)
    Long findUnreturnedReservationId(@Param("bookId") Long bookId);
}
```

4.2.9. Service sloj

Service klase definiraju poslovnu logiku aplikacije. Naznačuju se s @Service anotacijom. Njihova uloga je posredovanje između controller i repository slojeva. Ovaj sloj sadrži klase koje najviše nalikuju „tipičnim“ klasama koje bi bilo očekivano vidjeti u nekoj običnoj Java aplikaciji koja se ne koristi frameworcima poput Spring Boota, i drugim frameworcima sličnog opsega.

Slijedi primjer metode borrowBook(), koja sadrži u sebi poslovnu logiku potrebnu za posuđivanje knjige. Ova metoda prvo provjerava posjeduje li korisnik članstvo koje nije isteklo. Nakon toga provjerava je li knjiga dostupna za posudbu, te nakon toga kreira rezervaciju, te ju mapira na DTO objekt te ju vraća na controller sloj:

```
public ReservationDto borrowBook(Long bookId) {

    User currentUser = currentUserService.getLoggedInUser();

    // check if book exists
    Book book = bookRepository.findById(bookId).orElseThrow(() -> new
NotFoundException("Book not found."));

    // check if user's membership is valid
    if (!membershipService.isMembershipActive(currentUser.getId())) {
        throw new MembershipExpiredException("Your membership has expired.");
    }

    // check if book is available to borrow
    if (!isBookAvailableToBorrow(bookId)) {
        throw new BookAlreadyBorrowedException("This book is already
borrowed.");
    }

    // reserve book
    Reservation reservation = Reservation.builder()
        .book(book)
        .borrower(currentUser)
        .borrowedDate(LocalDate.now())
        .dueDate(LocalDate.now().plusDays(DAYS_TO_RETURN))
        .returned(false)
        .build();
```

```
    reservation = reservationRepository.save(reservation);

    return ReservationDtoMapper.map(reservation);
}
```

4.2.10. Controller sloj

Spring Boot „hvata“ HTTP zahtjeve, te ih nakon toga prosljeđuje na prikladnu controller klasu. U tim klasama su definirane metode koje su mapirane na određene requestove. Također u njima mogu biti definirani parametri i/ili tijela zahtjeva koje očekujemo. Controller klase se naznačuju s @RestController anotacijom.

Slijedi primjer metode koja je mapirana na „/api/book“, prima GET zahtjeve, te prima neobavezne parametre; naslov knjige koju tražimo i žanr koji tražimo:

```
@GetMapping
public ResponseEntity<ApiResponse> getAllBooks(@RequestParam(required = false) final String title,
                                               @RequestParam(required = false) final Long genre) {
    return new ResponseEntity<>(new ApiResponse(bookService.getAllBooks(title, genre)), HttpStatus.OK);
}
```

4.2.11. Exception Advice

Ponekad dolazi do iznimki u izvršavanju poslovne logike. U tim slučajevima, odgovor bi trebao poprimiti neki od response code-ova od 400 na dalje, te vratiti poruku koja objašnjava iznimku. Kako bi se spriječilo „zagađivanje“ controller klasa s try/catch blokovima, koristi se ExceptionAdvice klasa, koja na sebi ima @RestControllerAdvice anotaciju. U njoj se nalaze metode s @ExceptionHandler anotacijama, koje Spring Boot poziva kad programski kod poslovne logike baci neku iznimku, te one vraćaju prikladan response frontend-u.

Slijedi metoda iz ExceptionAdvice klase koja hvata probleme kod validacije podataka primljenih u HTTP zahtjevu:

```
@ExceptionHandler(MethodArgumentNotValidException.class)
public ResponseEntity<ApiResponse>
handleValidationExceptions(MethodArgumentNotValidException e) {
    Map<String, String> errors = new HashMap<>();
    e.getBindingResult().getAllErrors().forEach(error -> {
```

```
        String fieldName = ((FieldError) error).getField();
        String errorMessage = error.getDefaultMessage();
        errors.put(fieldName, errorMessage);
    });

    return new ResponseEntity<>(new ApiResponse(errors, "Validation failed."),
HttpStatus.BAD_REQUEST);
}
```

4.3. Frontend

4.3.1. Dockerfile

Dockerfile koji opisuje build proces frontend dijela projekta je također podijeljen na dva stadija. U prvom (build), kopira se izvorni kod aplikacije, te se pokreće npm run build naredba, koja gradi verziju aplikacije koja je minimizirana te optimizirana, čime postaje spremna za rad na produkcijskom okruženju:

```
FROM node:erbium-alpine as build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY ./ .
RUN npm run build
```

Nakon toga, preuzima se nginx image koji će služiti za serviranje aplikacije:

```
FROM nginx as production
RUN mkdir /app
COPY --from=build /app/dist /app
COPY nginx.conf /etc/nginx/nginx.conf
```

4.3.2. Router

Za navigaciju po single-page aplikaciji se koristi router. Na taj način, stranica se ne mora osvježiti sadržaj, nego dohvaća relevantne podatke, te izmijeni jedan dio stranice na kojoj se odvija rad aplikacije. Vue-ov router se zove VueRouter. Neke značajke VueRouter-a su mogućnost ugnježđivanja ruta, mogućnost korištenja raznih parametara i općenita modularnost, postignuta oslanjanjem na komponente.

Slijedi programski kod metode router.beforeEach, iz router/index.js datoteke, čija logika navodi aplikaciju na rute u ovisnosti o valjanosti JWT tokena korisnika. U slučaju da je JWT token istekao, aplikacija odlazi na Login view:

```
router.beforeEach((to, from, next) => {
  if (to.matched.some(record => record.meta.requiresAuth)) {
    if (store.getters.user.id == null) {
      next({
        name: RouteNames.LOGIN,
        params: { nextUrl: to.fullPath }
      });
    } else {
      const decoded = jwt_decode(store.getters.user.token);
      const exp = decoded.exp * 1000;
      if (new Date().getTime() > exp) {
        store.dispatch('setUser', {
          id: null,
          username: null,
          email: null,
          token: null
        });
        next({
          name: RouteNames.LOGIN,
          params: { nextUrl: to.fullPath }
        });
      }
      next();
    }
  } else if (to.matched.some(record => record.meta.guest)) {
    if (store.getters.user.token == null) {
      next();
    } else {
      next({
        name: RouteNames.HOME
      });
    }
  } else {
    next();
  }
});
```

Moguće rute su opisane u routes.js datoteci. Slijedi primjer JSON objekta koji opisuje rutu koja vodi na view koji prikazuje detaljni pregled jedne knjige:

```
{
   path: '/book/:id',
   name: routeNames.BOOK,
```

```
    component: Book,
    meta: {
       requiresAuth: true
    }
},
```

4.3.3. Store

U PWA aplikacijama često dolazi do promjene stanja. Primjer promjene stanja bi bila tipka koja mijenja izgled stranice iz svijetlog načina u tamni. Takve promjene stanja je potrebno pohraniti u neki univerzalni spremnik – store. Vuex biblioteka pruža store komponentu, koju možemo koristiti u aplikaciji. Možemo u nju dodavati razne metode koje opisuju moguće promjene (mutacije) do kojih može doći tijekom rada aplikacije, te akcije koje dovode do tih mutacija. Također možemo navesti getter metode koje dohvaćaju spremljeno stanje, ili daju informacije dobivene u ovisnosti o trenutnom stanju. Slijedi primjer opisanih aspekata vezanih za upravljanje stanjem članstva u knjižnici trenutnog korisnika:

```
export default {
  state: {
    membership: null
  },
  mutations: {
    setMembership(state, payload) {
      state.membership = payload;
    }
  },
  actions: {
    setMembership({ commit }, payload) {
      commit("setMembership", payload);
    }
  },
  getters: {
    membership: state => state.membership,
    validMembership: state => {
      if (store.getters.user.role === AUTH_ROLE.ROLE_ADMIN) {
        return true;
      } else {
        if (state.membership) {
          return isBefore(
            new Date(),
            new Date(state.membership.validUntil)
          );
        }
```

```
        return false;
      }
    }
  },
}
```

4.3.4. Viewovi

Glavni razlog zašto bi netko koristio Vue.js, ili neki od srodnih frontend frameworkova, kao što su Angular, React ili Svelte, je olakšavanje razvoja izgleda i funkcionalnosti web aplikacije. Svi navedeni frameworkovi pružaju mogućnost opisivanja stranica uz pomoć predložaka (template-ova), iz kojih se potom može generirati ekvivalentni HTML. Ovaj proces se zove Client-side rendering.

Vue.js koristi viewove, u kojima web programer može definirati predložak izgleda stranice koristeći gotove komponente. Vuetify je primjer biblioteke koja pruža bogat izbor gotovih komponenata koje su jednostavne za koristiti i prilagođavati. Osim predloška, u view-u se može definirati i CSS stil koji određuje izgled stranice. Ovdje također ide script element, u kojem se opisuju metode koje se pozivaju kod učitavanja stranice, korisnikove interakcije sa stranicom, i tako dalje.

Slijedi primjer predloška koji definira tablicu korištenu u Membership view-u, u kojem se ispisuje popis članstva svih korisnika knjižnice, te se u njemu pruža mogućnost produžetka članstva od strane administratora:

```
<template>
  <div>
    <v-data-table
        :headers="headers"
        :items="users"
        class="elevation-4"

        hide-default-footer
    >
      <template #item.username="{ item }">
        {{ item.username }}
      </template>
      <template #item.actions="{ item }">
        <v-btn v-if="isAdmin"
               small
```

```
                color="success"
                class="mr-2"
                @click="extendMembership(item.membership.id)"
            >
              {{ "Extend membership" }}
            </v-btn>
          </template>
        </v-data-table>
      </div>
    </template>
```

U ovom seminaru nije prikazan prateći script dio koji je potreban za ispravno funkcioniranje ovog predloška. Slijedi izgled tablice koja je definirana gornjim programskim kodom:

Username	Valid until	
admin	2021-12-19	EXTEND MEMBERSHIP
user	2021-12-19	EXTEND MEMBERSHIP
neno	2021-12-25	EXTEND MEMBERSHIP
luka	2021-12-25	EXTEND MEMBERSHIP
jakov	2021-12-25	EXTEND MEMBERSHIP
matija	2021-12-25	EXTEND MEMBERSHIP

4.3.5. Servisi

Servisi u frontend-u služe kao krajnji sloj frontend-a s kojeg se rade HTTP zahtjevi. U tu svrhu koristi se axios biblioteka. Uz poziv je moguće slanje parametara i tijela zahtjeva.

Slijedi primjer metode koja radi HTTP POST zahtjev za kreiranje nove knjige:

```
async createBook({ title, genreId, authorId, description }) {
  return await axios.post("/book", {
    title,
    genreId,
    authorId,
    description
  });
},
```

4.3.6. Mixins

Mixini pružaju prilagodljiv način za dijeljenje funkcionalnosti koja nije specifična nekoj jednoj komponenti između različitih Vue komponenti. Kad u komponentu uključimo mixin, sve moguće opcije opisane u mixinu će biti dodane u opcije komponente.

Slijedi primjer membershipMixina, koji nudi opcije vezane za članstvo, što je potrebno u raznim komponentama aplikacije:

```
export default {
  computed: {
    ...mapGetters(["membership", "validMembership"])
  },
  methods: {
    ...mapActions(["setMembership"])
  }
}
```

5. Zaključak

Prije početka rada na ovom projektu, imao sam nizak nivo znanja o razvoju frontend dijela web aplikacije, odnosno nisam znao puno o frontend frameworkovima. Kroz obrazovanje sam naučio HTML, CSS i ponešto o JavaScriptu, ali osim korištenja Angulara na jednom fakultetskom predmetu, kao i sitno izmjenjivanje funkcionalnosti jednostavne React aplikacije na poslu, nisam imao nikakvog iskustva s modernim frontend rješenjima. Odlučio sam se za Vue.js upravo iz razloga što nisam imao nikakvog iskustva s njime, što se na kraju isplatilo, jer sam naučio puno toga.

Također, prije rada na projektu, moj jedini doticaj s Dockerom i docker-compose-om je bio pokretanje nekih gotovih Docker image-a. Drago mi je što mogu reći da sam kroz rad na projektu naučio dovoljno o toj tehnologiji da mogu pouzdano izjaviti da ju znam koristiti u neke osnovne svrhe, te vjerujem da ću ju od sad koristiti i u budućim projektima.

6. Literatura

[1] Trello, »What is Trello?,« Trello, 5. veljača 2021. [Mrežno]. Available: https://help.trello.com/article/708-what-is-trello. [Pokušaj pristupa 30. studeni 2021].

[2] IBM Cloud Education, »What is Docker?,« IBM, 23. lipanj 2021. [Mrežno]. Available: https://www.ibm.com/cloud/learn/docker. [Pokušaj pristupa 30. studeni 2021].

[3] Fireship, »Learn Docker in 7 Easy Steps - Full Beginner's Tutorial,« Fireship, 24. kolovoz 2020. [Mrežno]. Available: https://youtu.be/gAkwW2tuIqE. [Pokušaj pristupa 30. studeni 2021].

[4] Docker, »Overview of Docker Compose,« Docker, [Mrežno]. Available: https://docs.docker.com/compose/. [Pokušaj pristupa 30. studeni 2021].

[5] DB-Engines, »DB-Engines ranking,« DB-Engines, studeni 2021. [Mrežno]. Available: https://db-engines.com/en/ranking. [Pokušaj pristupa 30. studeni 2021].

[6] P. M. Kouate, »Some Important Key Concepts to start using PostgreSQL,« Medium, 23 lipanj 2020. [Mrežno]. Available: https://towardsdatascience.com/some-important-key-concepts-to-start-using-postgresql-c6de63ab683f. [Pokušaj pristupa 30. studeni 2021].

[7] M. Mulders, »What is Spring Boot?,« Stackify, [Mrežno]. Available: https://stackify.com/what-is-spring-boot/. [Pokušaj pristupa 30. studeni 2021].

[8] Liquibase, »Get Started | How Liquibase Works,« Liquibase, [Mrežno]. Available: https://www.liquibase.org/get-started/how-liquibase-works. [Pokušaj pristupa 30. studeni 2021].

[9] S. Azam, »What is Vue.js, and Why is it Cool?,« Linuxhint, 2020. [Mrežno]. Available: https://linuxhint.com/about_vue_js/. [Pokušaj pristupa 30. studeni 2021].

[10] M. Wanyoike, »How to Get Started with Vuetify,« Sitepoint, 26. lipanj 2019. [Mrežno]. Available: https://www.sitepoint.com/get-started-vuetify/. [Pokušaj pristupa 30. studeni 2021].

[11] M. Hammer, »Reengineering Work: Don't Automate, Obliterate,« 1990. [Mrežno]. Available: https://hbr.org/1990/07/reengineering-work-dont-automate-obliterate. [Pokušaj pristupa 2021].

[12] Business Model Toolbox, »Airbnb Business Model - Business Model Toolbox,« 2021. [Mrežno]. Available: https://bmtoolbox.net/stories/airbnb/. [Pokušaj pristupa 24. studeni 2021].

[13] E. Dure, »Airbnb Adds Heat to the IPO Boom,« Investopedia, 10. prosinca 2020. [Mrežno]. Available: https://www.investopedia.com/airbnb-adds-heat-to-the-ipo-boom-5091845. [Pokušaj pristupa 24. studeni 2021].

[14] T. Nath, »How Airbnb Makes Money,« Investopedia, 10. rujan 2021. [Mrežno]. Available: https://www.investopedia.com/articles/investing/112414/how-airbnb-makes-money.asp. [Pokušaj pristupa 24. studeni 2021].

[15] Fortune, »Airbnb Company Profile, News, Rankings | Fortune,« Fortune, 10. listopad 2021. [Mrežno]. Available: https://fortune.com/company/airbnb/. [Pokušaj pristupa 24. studeni 2021].

[16] Business Models Inc, »How Airbnb's Exponential Business Model Works,« Business Models Inc, 2021. [Mrežno]. Available: https://www.businessmodelsinc.com/exponential-business-model/airbnb/. [Pokušaj pristupa 24. studeni 2021].

[17] T. Sonnemaker, »Airbnb is worth more than the 3 largest hotel chains combined after its stock popped 143% on its first day of trading,« Business Insider, 11. prosinac 2020. [Mrežno]. Available: https://www.businessinsider.com/airbnb-ipo-valuation-tops-three-hotel-chains-combined-opening-day-2020-12. [Pokušaj pristupa 24. studeni 2021].