# WEB STRANICA ZA PREGLEDAVANJE RECEPATA

**Krželj, Antonio**

**SVEUČILIŠTE U SPLITU**

**SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE**

Stručni prijediplomski studij Računarstvo

**Antonio Krželj**

# Z A V R Š N I  R A D

# Web stranica za pregledavanje recepata

Split, rujan 2024.

**SVEUČILIŠTE U SPLITU**
**SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE**

Stručni prijediplomski studij Računarstvo

**Predmet:** Odabrani alati i naredbe u Linuxu

# Z A V R Š N I  R A D

**Kandidat:** Antonio Krželj

**Naslov rada:** Web stranica za pregledavanje recepata

**Mentor:** Nikola Grgić, viši predavač

Split, rujan 2024.

# Sadržaj

# Sažetak

Gourmet Gems je inovativna web aplikacija koja funkcionira kao društvena mreža za ljubitelje kuhanja, omogućujući korisnicima stvaranje, pregledavanje i pretraživanje recepata. Korisnici mogu pristupiti detaljima recepata, ostavljati ili brisati recenzije te dodavati recepte u listu omiljenih. *Frontend* aplikacije razvijen je korištenjem NextJS i TailwindCSS, dok je *backend* implementiran pomoću Java Spring Boota. Podaci su pohranjeni u bazi podataka PostgreSQL, čime se osigurava sigurno i učinkovito upravljanje informacijama. Razvoj aplikacije podijeljen je u nekoliko faza, uključujući dizajn baze podataka, razvoj korisničkog sučelja te integraciju *backend* i *frontend* dijelova. Ovaj završni rad detaljno opisuje sve aspekte razvoja, od odabira tehnologija do implementacije ključnih funkcionalnosti aplikacije, s ciljem pružanja cjelovitog pregleda procesa stvaranja ove web platforme.

**Ključne riječi**: Java Spring Boot, NextJS, PostgreSQL, recepti, TailwindCSS.

# Summary

## Web application for storing recipes

**Gourmet Gems** is an innovative web application that functions as a social network for cooking enthusiasts, allowing users to create, view, and search for recipes. Users can access recipe details, leave or delete reviews, and add recipes to their list of favorites. The frontend of the application was developed using NextJS and TailwindCSS, while the backend was implemented using Java Spring Boot. The data is stored in a PostgreSQL database, ensuring secure and efficient information management. The application development was divided into several phases, including database design, user interface development, and the integration of backend and frontend components. This thesis provides a detailed description of all aspects of development, from the selection of technologies to the implementation of key application functionalities, aiming to offer a comprehensive overview of the process of creating this web platform.

**Keywords**: Java Spring Boot, NextJS, PostgreSQL, recipes, TailwindCSS.

# 1. Uvod

U današnjem digitalnom dobu, tehnologija se neprestano razvija, omogućujući pristup raznovrsnim informacijama i sadržajima na načine koji su prije nekoliko desetljeća bili nezamislivi. Među tim sadržajima, gastronomija zauzima posebno mjesto, budući je hrana neizostavan dio ljudskog života, kulture i društvenih interakcija. Razvoj web aplikacija u domeni kulinarstva pruža nove mogućnosti ljubiteljima kuhanja za dijeljenje svojih kreacija, otkrivanje novih recepata i povezivanje s istomišljenicima širom svijeta.

Cilj ove aplikacije je stvoriti dinamičnu platformu koja će korisnicima omogućiti ne samo pristup receptima, već i sudjelovanje u interaktivnom procesu stvaranja i evaluacije kulinarskih sadržaja. U središtu ove aplikacije nalazi se ideja o zajednici koja se okuplja oko zajedničke strasti prema kuhanju i dijeljenju gastronomskog znanja. Takva platforma pruža prostor za kreativnost, učenje i razmjenu iskustava, a istovremeno potiče korisnike na razvijanje svojih kulinarskih vještina i istraživanje različitih kuhinja.

Razvoj ove web aplikacije zahtijevao je pažljivo planiranje i implementaciju suvremenih tehnologija kako bi se osigurala funkcionalnost, sigurnost i lakoća korištenja. Korištenjem alata kao što su NextJS i TailwindCSS za *frontend* te Java Spring Boot za *backend*, postignuta je visoka razina performansi i skalabilnosti sustava. Istovremeno, baza podataka PostgreSQL osigurava pouzdano i sigurno pohranjivanje podataka, što je ključno za rad aplikacije.

Ovaj rad se bavi opisom svih faza razvoja aplikacije, od inicijalnog planiranja i dizajna baze podataka, preko razvoja korisničkog sučelja, pa sve do integracije svih komponenti u jednu cjelinu. Kroz analizu izazova i rješenja s kojima smo se susretali tijekom razvoja, pružit će se uvid u kompleksnost procesa stvaranja moderne web aplikacije usmjerene na zajednicu korisnika s posebnim interesima. U konačnici, cilj je prikazati kako se tehnologija može iskoristiti za stvaranje inovativnih rješenja koja olakšavaju i obogaćuju svakodnevni život korisnika.

# 2. Korištene tehnologije

U razvoju web aplikacije Gourmet Gems korišten je niz suvremenih tehnologija koje su omogućile stvaranje funkcionalne i estetski privlačne platforme za ljubitelje kulinarstva. Ovdje su ukratko opisane ključne tehnologije koje su korištene tijekom razvoja.

## 2.1. Visual Studio Code

Visual Studio Code (VS Code) je popularno razvojno okruženje koje nudi podršku za širok raspon programskih jezika i alata. Omiljen među programerima zbog svoje fleksibilnosti, VS Code omogućuje integraciju s različitim proširenjima i alatima koji olakšavaju rad na složenim projektima. U razvoju *frontend* dijela aplikacije Gourmet Gems, VS Code je korišten zbog svoje podrške za HTML, TypeScript, i CSS, te zbog mogućnosti prilagodbe putem ekstenzija koje olakšavaju pisanje i organizaciju koda. Njegove značajke kao što su IntelliSense, ugrađeni terminal i moćno *debuggiranje* čine ga idealnim alatom za razvoj moderne web aplikacije.

## 2.2. IntelliJ

IntelliJ IDEA je razvojno okruženje koje nudi snažne alate za razvoj u Javi. Korišten je za *backend* razvoj aplikacije Gourmet Gems zbog svojih naprednih mogućnosti za debuggiranje, refaktoring koda, i integraciju s alatima za verzioniranje kao što je Git. IntelliJ omogućava jednostavno upravljanje velikim projektima, nudi podršku za razne razvojne okvire uključujući Spring Boot, te omogućava jednostavnu integraciju s bazama podataka. Njegove značajke kao što su pametno automatsko dovršavanje koda, detekcija grešaka u stvarnom vremenu i napredni alati za analizu koda, čine ga neophodnim alatom za svakog Java developera.

## 2.3. NextJS

NextJS je React razvojni okvir koji omogućava izradu dinamičkih web aplikacija s podrškom za prikazivanje na strani poslužitelja i statičku generaciju stranica. Ova tehnologija poboljšava performanse aplikacije i SEO optimizaciju, što je posebno važno za aplikaciju poput Gourmet Gems koja se oslanja na visoku dostupnost i brzo učitavanje stranica. NextJS omogućava razvoj modularnih i ponovno upotrebljivih komponenti, što značajno ubrzava razvoj i održavanje aplikacije. Njegove značajke uključuju automatsko preusmjeravanje,

dinamičko učitavanje modula i jednostavnu integraciju s API-ima, što ga čini idealnim izborom za *frontend* razvoj.

## 2.4. TypeScript

TypeScript je nadskup JavaScript jezika koji dodaje statičku tipizaciju. Korišten je u razvoju *frontend* dijela aplikacije Gourmet Gems kako bi se smanjila mogućnost pogrešaka i povećala održivost koda. TypeScript omogućava programerima definiranje vrsta podataka, što olakšava otkrivanje pogrešaka u ranoj fazi razvoja i poboljšava čitljivost koda. Njegova integracija s modernim alatima za razvoj, kao što su VS Code i NextJS, omogućava jednostavno refaktoring koda, automatsko dovršavanje i bolje razumijevanje koda, što sve zajedno doprinosi većoj produktivnosti i kvaliteti finalnog proizvoda.

## 2.5. TailwindCSS

TailwindCSS je razvojni okvir temeljen na korisničkim klasama koji omogućava brzo i jednostavno stiliziranje elemenata. Korišten je za izradu responzivnog i prilagodljivog dizajna aplikacije Gourmet Gems. TailwindCSS nudi širok raspon gotovih klasa koje olakšavaju stiliziranje bez potrebe za pisanjem prilagođenog CSS-a. Ovaj pristup omogućava brzu iteraciju dizajna, dosljednost u stiliziranju i lakšu održivost koda. TailwindCSS također podržava prilagodbu i proširenje putem konfiguracijskih datoteka (Slika 1), što omogućava stvaranje jedinstvenog i konzistentnog vizualnog identiteta aplikacije.


**Slika 1**: Primjer definiranja responzivnosti za tailwindCSS u datoteci `tailwind.config.ts`

## 2.6. Java Spring Boot

Spring Boot je razvojni okvir za izradu aplikacija u Javi koji pojednostavljuje kreiranje samostalnih, produkcijski spremnih aplikacija. Korišten je za *backend* razvoj aplikacije Gourmet Gems zbog svoje robusnosti i jednostavnosti integracije s drugim tehnologijama. Spring Boot omogućava brzo postavljanje i konfiguraciju aplikacije, nudi podršku za razne načine autentikacije i autorizacije, te jednostavnu integraciju s bazama podataka. Njegove značajke uključuju ugrađene poslužitelje (kao što su Tomcat i Jetty), automatizirano testiranje i lako upravljanje ovisnostima, što ga čini idealnim izborom za razvoj složenih web aplikacija.

## 2.7. PostgreSQL

PostgreSQL je napredan sustav za upravljanje relacijskim bazama podataka poznat po svojoj pouzdanosti, skalabilnosti i naprednim mogućnostima obrade podataka. Korišten je za pohranu podataka aplikacije Gourmet Gems zbog svoje sposobnosti učinkovitog upravljanja velikim količinama podataka i podržava kompleksne upite. PostgreSQL nudi podršku za ACID (engl. *Atomicity, Consistency, Isolation, Durability*) transakcije, što osigurava integritet podataka, te omogućava jednostavno proširenje funkcionalnosti putem vlastitih proširenja. Njegove značajke uključuju napredno indeksiranje, replikaciju i visok stupanj prilagodljivosti, što ga čini idealnim izborom za *backend* pohranu podataka u zahtjevnim aplikacijama.

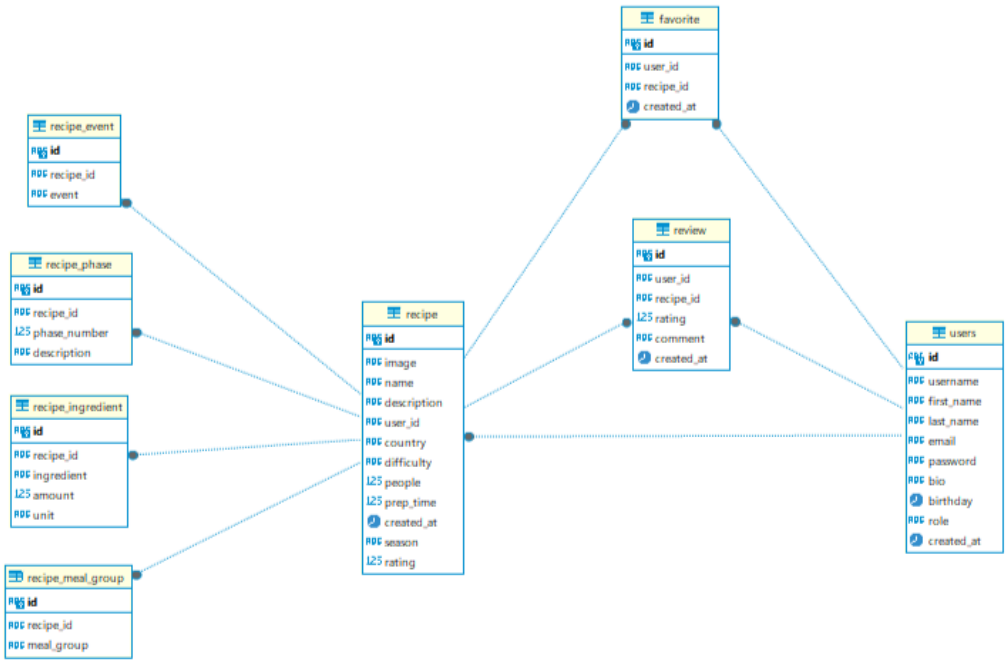
# 3. Baza podataka

Baza podataka je ključni dio aplikacije Gourmet Gems, jer omogućava pohranu i upravljanje svim podacima vezanim za korisnike, recepte, favorite i recenzije. Korištena baza podataka je PostgreSQL, poznata po svojoj stabilnosti, skalabilnosti i naprednim mogućnostima. PostgreSQL podržava ACID transakcije, što osigurava integritet i sigurnost podataka unutar aplikacije.

Baza podataka strukturirana je kako bi omogućila efikasno i logički konzistentno spremanje i dohvaćanje podataka. Sastoji se od više tablica, od kojih svaka ima specifičnu svrhu i sadrži podatke o različitim entitetima kao što su korisnici, recepti, sastojci, faze pripreme, skupine obroka, događaji, favoriti i recenzije.

## 3.1. Entiteti

Entiteti predstavljaju osnovne komponente baze podataka. U aplikaciji Gourmet Gems glavni entiteti uključuju korisnike (`Users`), recepte (`Recipe`), favorite (`Favorite`) i recenzije (`Review`). Svaki entitet je prikazan kroz odgovarajuće tablice u bazi podataka koje čuvaju podatke vezane uz taj entitet (Slika 2).




**Slika 2**: ER dijagram baze podataka za aplikaciju Gourment Gems

### 3.1.1. Entitet `user`

Entitet korisnika predstavlja sve relevantne informacije o korisnicima aplikacije. Tablica `Users` (Ispis 1) sadrži podatke kao što su jedinstveni identifikator korisnika (ID), korisničko ime, ime, prezime, e-mail, lozinku, biografiju, datum rođenja, ulogu korisnika i vrijeme kreiranja profila. Svaki korisnik ima jedinstveni ID koji se koristi kao primarni ključ. Tablica omogućava pohranu svih bitnih informacija potrebnih za autentikaciju i autorizaciju korisnika, kao i za personalizaciju korisničkog iskustva.

```
CREATE TABLE IF NOT EXISTS Users (
    id char(36) NOT NULL PRIMARY KEY DEFAULT (UUID_GENERATE_V4()),
    username VARCHAR(255) NOT NULL,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    bio VARCHAR(255) NOT NULL,
    birthday TIMESTAMP NOT NULL,
    role VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

**Ispis 1**: Struktura tablice `Users` u SQL formatu

### 3.1.2. Entitet `recipe`

Entitet recepta pohranjuje sve informacije vezane za recepte koje korisnici kreiraju. Tablica `Recipe` (Ispis 2) sadrži podatke kao što su jedinstveni identifikator recepta, slika, naziv, opis, identifikator korisnika koji je kreirao recept, zemlja porijekla, sezona, razina težine, broj osoba, vrijeme pripreme, ocjena i vrijeme kreiranja recepta. Svaki recept je povezan s korisnikom putem vanjskog ključa `user_id`, koji referencira tablicu `Users`. Osim osnovnih podataka o receptima, postoje dodatne tablice koje proširuju funkcionalnost recepta.

```sql
CREATE TABLE IF NOT EXISTS Recipe (
    id CHAR(36) NOT NULL PRIMARY KEY DEFAULT (UUID_GENERATE_V4()),
    image VARCHAR(100) NOT NULL,
    name VARCHAR(100) NOT NULL,
    description VARCHAR(500) NOT NULL,
    user_id VARCHAR(100) NOT NULL,
    country VARCHAR(100),
    season VARCHAR(100),
    difficulty VARCHAR(100) NOT NULL,
    people INT NOT NULL,
    prep_time INT NOT NULL,
    rating DECIMAL(10, 2) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES Users(id) ON DELETE CASCADE
);
```

**Ispis 2**: Struktura tablice `Recipe` u SQL formatu

**Povezane tablice:**

**`Recipe_Ingredient`**: Ova tablica (Ispis 3) sadrži informacije o sastojcima potrebnim za pripremu recepta. Svaki zapis povezan je s određenim receptom putem `recipe_id`. Tablica je kreirana kako bi omogućila detaljan prikaz sastojaka potrebnih za recept, uključujući količinu i jedinicu mjere. To omogućava ponovnu upotrebu istih sastojaka u različitim receptima bez dupliciranja podataka, što doprinosi efikasnosti i konzistentnosti baze podataka.

```sql
CREATE TABLE IF NOT EXISTS Recipe_Ingredient (
    id CHAR(36) NOT NULL PRIMARY KEY DEFAULT (UUID_GENERATE_V4()),
    recipe_id VARCHAR(100) NOT NULL,
    ingredient VARCHAR(100) NOT NULL,
    amount DECIMAL(10, 2) NOT NULL,
    unit VARCHAR(50) NOT NULL,
    FOREIGN KEY (recipe_id) REFERENCES Recipe(id) ON DELETE
CASCADE );
```

**Ispis 3**: Struktura tablice `Recipe_Ingredient` u SQL formatu

**`Recipe_Phase`**: Ova tablica (Ispis 4) sadrži detalje o fazama pripreme recepta. Svaka faza pripreme je povezana s određenim receptom putem `recipe_id` i ima opis koji objašnjava korak po korak postupak pripreme. Tablica omogućava detaljan prikaz koraka potrebnih za pripremu recepta.

```
CREATE TABLE IF NOT EXISTS Recipe_Phase (
    id CHAR(36) NOT NULL PRIMARY KEY DEFAULT (UUID_GENERATE_V4()),
    recipe_id VARCHAR(100) NOT NULL,
    phase_number INT NOT NULL,
    description VARCHAR(1000) NOT NULL,
    FOREIGN KEY (recipe_id) REFERENCES Recipe(id) ON DELETE
CASCADE
);
```

**Ispis 4**: Struktura tablice `Recipe_Phase` u SQL formatu

Tablica je kreirana kako bi omogućila detaljno i organizirano pohranjivanje postupka pripreme za svaki recept. S obzirom na to priprema recepta može biti složena i sastojati se od više koraka, ova tablica omogućava razdvajanje i numeriranje tih koraka, čime se korisnicima omogućava lakše praćenje i razumijevanje postupka pripreme.

**`Recipe_Meal_Group`**: Ova tablica (Ispis 5) sadrži podatke o grupama obroka kojima recept pripada, omogućavajući korisnicima pretraživanje recepata prema tipu obroka.

```
CREATE TABLE IF NOT EXISTS Recipe_Meal_Group (
    id CHAR(36) NOT NULL PRIMARY KEY DEFAULT
(UUID_GENERATE_V4()),
    recipe_id VARCHAR(100) NOT NULL,
    meal_group VARCHAR(100) NOT NULL,
    FOREIGN KEY (recipe_id) REFERENCES Recipe(id) ON DELETE
CASCADE
);
```

**Ispis 5**: Struktura tablice `Recipe_Meal_Group` u SQL formatu

**`Recipe_Event`**: Ova tablica (Ispis 6) sadrži podatke o događajima za koje je recept prikladan, kao što su praznici, rođendani, obiteljska okupljanja.

```
CREATE TABLE IF NOT EXISTS Recipe_Event (
    id CHAR(36) NOT NULL PRIMARY KEY DEFAULT (UUID_GENERATE_V4()),
    recipe_id VARCHAR(100) NOT NULL,
    event VARCHAR(100) NOT NULL,
    FOREIGN KEY (recipe_id) REFERENCES Recipe(id) ON DELETE
CASCADE
);
```

**Ispis 6**: Struktura tablice `Recipe_Event` u SQL formatu

Ove dodatne tablice omogućavaju detaljniji prikaz recepata i njihovo bolje organiziranje unutar aplikacije. Svaki recept može imati više tipova sastojaka, a isti sastojci mogu se ponavljati u različitim receptima. To je omogućeno strukturiranjem podataka na način koji osigurava efikasno upravljanje i fleksibilnost u pohranjivanju podataka. Ovakav pristup također poboljšava performanse aplikacije i olakšava održavanje baze podataka.

### 3.1.3. Entitet `favorites`

Entitet favorita omogućava korisnicima pohranu svojih omiljenih recepata. Tablica `Favorite` (Ispis 7) povezuje korisnike s receptima koje su označili kao favorite. Svaki zapis u tablici sadrži jedinstveni identifikator, identifikator korisnika i identifikator recepta, čime se osigurava svakom korisniku mogućnost spremanja više omiljenih recepata.

```
CREATE TABLE IF NOT EXISTS Favorite (
    id CHAR(36) NOT NULL PRIMARY KEY DEFAULT (UUID_GENERATE_V4()),
    user_id VARCHAR(100) NOT NULL,
    recipe_id VARCHAR(100) NOT NULL,
    FOREIGN KEY (user_id) REFERENCES Users(id) ON DELETE CASCADE,
    FOREIGN KEY (recipe_id) REFERENCES Recipe(id) ON DELETE
CASCADE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP );
```

**Ispis 7**: Struktura tablice `Favorite` u SQL formatu

### 3.1.4. Entitet `review`

Entitet recenzija pohranjuje ocjene i komentare koje korisnici ostavljaju na recepte. Tablica `Review` (Ispis 8) sadrži podatke kao što su jedinstveni identifikator recenzije, identifikator korisnika koji je ostavio recenziju, identifikator recepta na koji se recenzija odnosi, ocjenu, komentar i vrijeme kreiranja recenzije.

```
  CREATE TABLE IF NOT EXISTS Review (
1     id CHAR(36) NOT NULL PRIMARY KEY DEFAULT
 2 (UUID_GENERATE_V4()),
 3     user_id VARCHAR(100) NOT NULL,
 4     recipe_id VARCHAR(100) NOT NULL,
 5     FOREIGN KEY (user_id) REFERENCES Users(id) ON DELETE CASCADE,
 6     FOREIGN KEY (recipe_id) REFERENCES Recipe(id) ON DELETE
 7 CASCADE,
 8     rating INT NOT NULL,
 9     comment VARCHAR(500) NOT NULL,
10     created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
  );
```

**Ispis 8**: Struktura tablice `Review` u SQL formatu

Svaka od ovih tablica i entiteta igra ključnu ulogu u omogućavanju funkcionalnosti aplikacije, osiguravajući pravilno pohranjivanje, povezivanje i laku dostupnost svih podataka za potrebe aplikacije Gourmet Gems.

# 4. Komponente aplikacije

*Backend* aplikacije izveden je koristeći Java Spring Boot, pri čemu je aplikacija organizirana u slojevima kako bi se postigla modularnost i lakoća održavanja. U strukturi *backenda* nalaze se modeli, DTO-i (engl. *Data Transfer Objects*), kontroleri, servisi i repozitoriji. Modeli predstavljaju podatkovne strukture koje se koriste u aplikaciji, dok DTO-i omogućavaju prijenos podataka između različitih slojeva aplikacije. Kontroleri upravljaju HTTP zahtjevima i definiraju krajnje točke aplikacije. Servisi sadrže poslovnu logiku, dok repozitoriji omogućuju pristup bazi podataka. Osim ovih slojeva, *backend* aplikacija uključuje i dodatne foldere kao što su `mappers` (za mapiranje između različitih objekata), `utils` (za pomoćne funkcije), `exceptions` (za rukovanje iznimkama) te `config` (gdje je konfiguriran sigurnosni sustav aplikacije).

*Frontend* aplikacije izrađen je u Next.js, pri čemu su datoteke organizirane unutar `app` datoteke koji sadrži stranice (*pages*) aplikacije. Komponente koje se mogu ponovno koristiti smještene su u datoteku components. Datoteka `services` koristi se za dohvaćanje podataka te za njihovo spremanje, brisanje i uređivanje koristeći axios i React Query. `Redux` datoteka sadrži konfiguraciju za React Redux, alat za globalno upravljanje stanjem aplikacije. React Redux omogućava centralizirano pohranjivanje stanja aplikacije, što olakšava dijeljenje podataka između različitih dijelova aplikacije i smanjuje potrebu za *prop-drillingom* (prosljeđivanjem podataka kroz više razina komponenti).

Datoteka `React Query` definira postavke za React Query, alat koji upravlja podatkovnim dohvatima na klijentskoj strani. Budući je Next.js orijentiran na prikazivanje na strani poslužitelja, React Query je prilagođen za rad na klijentskoj strani, omogućavajući efikasno dohvaćanje i *caching* podataka. React Query nudi mnoge funkcionalnosti, uključujući automatsko ponovno dohvaćanje podataka ako se oni promijene, čime se omogućava osvježavanje samo relevantnih dijelova aplikacije umjesto cijele stranice. Ova funkcionalnost značajno poboljšava performanse i korisničko iskustvo, jer korisnici odmah vide ažurirane podatke bez potrebe za ručnim osvježavanjem stranice.

Sve ove komponente zajedno omogućuju izradu robusne i skalabilne aplikacije koja nudi visoku razinu korisničkog iskustva i funkcionalnosti.

## 4.1. `User`

Korisnički sustav aplikacije uključuje osnovne funkcionalnosti kao što su registracija, prijava, pregled profila, uređivanje profila, brisanje korisničkog računa te prikaz liste korisnika. Svaka od ovih funkcionalnosti implementirana je kroz različite komponente i stranice unutar aplikacije.

U *frontend* dijelu aplikacije, podaci se prikupljaju iz forme koristeći biblioteku Formik. Kada korisnik pritisne dugme za prijavu, registraciju, brisanje, uređivanje ili dohvaćanje jednog ili više korisnika, svi uneseni podaci se prikupe i obrade putem Formika. Nakon toga, prikupljeni podaci se šalju na *backend* koristeći `axios.post` (Ispis 9) metodu za prijavu i registraciju, ili odgovarajuće HTTP metode (kao što su `axios.delete`, `axios.put`, ili `axios.get`) za brisanje, uređivanje ili dohvaćanje korisnika. Ako je odgovor poslužitelja pozitivan, korisnik se preusmjerava na odgovarajuću stranicu. U slučaju negativnog odgovora, na ekranu se prikazuje poruka koja korisnika obavještava jesu li uneseni podaci neispravni te ih treba ponovno unijeti. Ovaj pristup omogućava centralizirano i efikasno rukovanje korisničkim podacima u aplikaciji, osiguravajući sve operacije vezane za korisnike budu konzistentno implementirane i lako upravljane.

```
import axios from "axios";

export const userLogin = async (loginData: any) => {
  console.log(loginData);
  try {
    const response = await axios.post(
      "http://localhost:8080/api/v1/user/login",
      loginData,
      {
        headers: { "Content-Type": "application/json" },
      }
    );
    console.log(response.data);
    return response.data;
  } catch (error) {
    console.error(error);
    throw error;
  }
};
```

**Ispis 9**: Primjer slanja podataka s *frontend*a na *backend*

### 4.1.1. Registracija korisnika

Registracija korisnika omogućuje novim korisnicima mogućnost registriranja na platformu. Ova funkcionalnost (Ispis 10) implementirana je na *backendu* koristeći Java Spring Boot. Kontroler za registraciju prima zahtjev POST s podacima o korisniku u obliku `UserDto` objekta. Nakon što je primio objekt šalje ga dalje u servis gdje će se pregledati ispravnost objekta te izvesti sva potrebna logika.

```
    @Override
    @PostMapping("/register")
    public ResponseEntity<UserResDto> register(@RequestBody UserDto
userDto) throws EmailMismatchException, PasswordMismatchException,
AlreadyExistException, InvalidArgumentsException {
        UserResDto user = userService.addUser(userDto);
        return ResponseEntity.status(HttpStatus.OK).body(user);
    }
```

**Ispis 10**: Primjer `register` kontrolera u java spring bootu

U `UserService` klasi, metoda `addUser` (Ispis 11) provjerava ispravnost unesenih podataka. Prvo se provjerava jesu li podaci valjani, zatim se uspoređuju unesene lozinke kako bi se osiguralo podudaranje. Ako korisnik s danim e-mailom već postoji u bazi podataka, baca se iznimka. Ako su svi podaci ispravni, novi korisnik se sprema u bazu podataka.

```
    @Override
    public UserResDto addUser(UserDto userDto) throws
InvalidArgumentsException, PasswordMismatchException,
AlreadyExistException, EmailMismatchException {
        if(userDto == null) throw new
InvalidArgumentsException("Entered arguments are invalid!");
        if(!Objects.equals(userDto.getPassword(),
userDto.getConfirmPassword())) throw new
PasswordMismatchException("Entered passwords doesn't match!");

        Optional<User> existingUser =
userRepository.findByEmail(userDto.getEmail());
```

```
        if(existingUser.isPresent()) throw new
AlreadyExistException("User already exist in database!");
        User newUser =
userRepository.save(userMapper.userDtoToUser(userDto));

        return userMapper.userToUserDto(newUser);
    }
```

**Ispis 11**: Primjer `register` servisa u java spring bootu

`UserRepository` (Ispis 12) koristi JPA (Java Persistence API) za jednostavnu interakciju s bazom podataka. JPA omogućuje definiranje metoda za osnovne CRUD (Create, Read, Update, Delete) operacije bez potrebe za pisanjem SQL upita.

```
@Repository
public interface UserRepository extends JpaRepository<User, String>
{
    Optional<User> findByEmail(String email);
}
```

**Ispis 12**: Primjer `user` jpa repozitorij klase u java spring bootu

### 4.1.2. Prijava korisnika

Prijava korisnika (Ispis 13) omogućava postojećim korisnicima pristup aplikaciji. Kontroler za prijavu prima zahtjev POST s podacima za prijavu, koji se zatim provjeravaju u servisu.

```
@PostMapping("/login")
public ResponseEntity<LoginResDto> login(@RequestBody UserDto
userLoginDto) throws InvalidArgumentsException,
ObjectDoesntExistException, PasswordMismatchException {
    UserResDto user = userService.validateLogin(userLoginDto);
    String jwtToken =
tokenGenerator.generateToken(String.valueOf(user.getRole()),
user.getId());

    LoginResDto loginResDto = LoginResDto.builder()
            .success(true)
```

```
                .token(jwtToken)
                .userResDto(user)
                .build();
        return ResponseEntity.status(HttpStatus.OK).body(loginResDto);
    }
```

**Ispis 13**: Primjer `login` kontrolera u java spring bootu

Nakon provjere valjanosti podataka, generira se JWT (JSON Web Token) token. JWT token je kompaktni način predstavljanja sklopljenih podataka koji se mogu potvrditi i vjerovati zahvaljujući digitalnom potpisu. Token se generira kako bi se autenticirali korisnici i osigurala sigurna komunikacija između klijenta i poslužitelja. Kada se korisnik uspješno prijavi, poslužitelj generira JWT token koji sadrži korisničke podatke i druge relevantne informacije. Ovaj token se zatim šalje klijentu, koji ga pohranjuje i koristi za autorizaciju prilikom budućih zahtjeva prema poslužitelju. Na taj način, svaki put kada korisnik želi pristupiti zaštićenim resursima, klijent šalje token kako bi dokazao svoj identitet i ovlasti.

```
@Component
public class TokenGenerator {
    private final JwtEncoder encoder;

    public TokenGenerator(JwtEncoder encoder) {
        this.encoder = encoder;
    }

    public String generateToken(String authority, String name) {
        Instant now = Instant.now();
        System.out.println("Authority: " + authority);
        System.out.println("User name: " + name);
        JwtClaimsSet claims = JwtClaimsSet.builder()
                .issuer("self")
                .issuedAt(now)
                .expiresAt(now.plus(1, ChronoUnit.HOURS))
                .subject(name)
                .claim("scope", authority)
                .build();
```

```
        return
this.encoder.encode(JwtEncoderParameters.from(claims)).getTokenValue
();
    }
}
```

**Ispis 14**: Primjer klase koja generira token

`TokenGenerator` (Ispis 14) je komponenta koja se koristi za generiranje JWT tokena u aplikaciji. Ova komponenta je odgovorna za stvaranje tokena koji se koriste za autentikaciju i autorizaciju korisnika.

Klasa `TokenGenerator` je označena s anotacijom `@Component` što je čini Spring Beanom, što znači Spring Boot kontejner može automatski upravljati njome i ubrizgavati u druge komponente ili servise koji je trebaju.

U konstruktoru klase `TokenGenerator` ubrizgava se `JwtEncoder` koji je odgovoran za enkodiranje JWT tokena prema specifikacijama. Metoda `generateToken` prima `authority` (ovlasti) i `name` (ime) korisnika te generira JWT token s navedenim podacima. Token sadrži informacije poput izdavača, vremena izdavanja, vremena isteka, subjekta (korisničko ime), i dodatnih podataka poput ovlasti korisnika.

Generirani token se koristi za sigurnu razmjenu podataka između klijenta i poslužitelja. On omogućava korisniku jednokratnu autentikaciju, nakon čega može koristiti taj token za pristup zaštićenim resursima na poslužitelju bez potrebe za ponovnom prijavom. Osim toga, token je digitalno potpisan, što osigurava autentičnost podataka u tokenu i nemogućnost mijenjanja poslije generiranja. Ukratko, `TokenGenerator` klasa je ključna za implementaciju sigurnog sustava autentikacije i autorizacije korisnika u Spring Boot aplikacijama pomoću JWT tokena.

U klasi `UserService`, metoda `validateLogin` (Ispis 15) provjerava postoji li korisnik s danim e-mailom u bazi podataka i uspoređuje unesenu lozinku koristeći BCrypt (biblioteku za šifriranje lozinki).

```
public UserResDto validateLogin(UserDto userLoginDto) throws
InvalidArgumentsException, ObjectDoesntExistException,
PasswordMismatchException {
    if(userLoginDto == null) throw new
InvalidArgumentsException("Sent arguments cannot be null!");
    Optional<User> user =
userRepository.findByEmail(userLoginDto.getEmail());
    if(user.isEmpty()) throw new ObjectDoesntExistException("User
you wanna login with doesn't exist!");
    if(!BCrypt.checkpw(userLoginDto.getPassword(),
user.get().getPassword())) throw new
PasswordMismatchException("Invalid password!");
    return userMapper.userToUserDto(user.get());
}
```

**Ispis 15**: Primjer `validateLogin` servisa u java spring bootu

### 4.1.3. Profil korisnika

Profil korisnika omogućava pregled i dohvaćanje korisničkih podataka za već registrirane korisnike aplikacije. Kontroler za dohvaćanje profila prima zahtjev GET s korisničkim ID-om, koji se zatim obrađuje u servisu za korisnike.

```
@Override
@GetMapping("/{id}")
public ResponseEntity<UserResDto> profile(@PathVariable String id) throws
ObjectDoesntExistException, InvalidArgumentsException {
    User userProfile = userService.findUserById(id);
    return ResponseEntity.ok().body(userMapper.userToUserDto(userProfile));
}
```

**Ispis 16**: Primjer `profile` kontrolera u java spring bootu

Metoda `profile` (Ispis 16) je implementirana kao HTTP zahtjev GET koji služi za dohvaćanje korisničkog profila na temelju unesenog korisničkog ID-a. Ova metoda koristi anotaciju `@GetMapping("/{id}")`, koja označava *endpoint* GET, pri čemu `{id}` predstavlja dinamički dio URL-a koji odgovara korisničkom ID-u.

Pri pozivu ove metode, kao putni parametar (`@PathVariable String id`) unosi se korisnički ID. Metoda vraća objekt `ResponseEntity` koji sadrži DTO korisničkog profila. DTO objekt je prilagođen za prijenos podataka između slojeva aplikacije.

Metoda započinje provjerom unesenog ID-a. Ako je ID valjan, metoda poziva `findUserById` (Ispis 17) iz `userService` klase kako bi dohvatila korisnički profil. Rezultat ovog poziva je objekt tipa `User`, koji predstavlja korisničke podatke. Ovaj objekt se potom mapira u DTO objekt pomoću `userMapper.userToUserDto` metode, kako bi se korisnički podaci prikazali u odgovarajućem formatu za prijenos podataka.

Konačno, metoda vraća objekt `ResponseEntity` s HTTP statusom 200 (OK) i tijelom odgovora koje sadrži DTO korisničkog profila. Ako se dogodi greška, kao što je nepostojanje korisnika ili nevažeći ID, metoda može baciti dvije vrste iznimki kao što su `ObjectDoesntExistException` i `InvalidArgumentsException`.

```
@Override
public User findUserById(String userId) throws
InvalidArgumentsException, ObjectDoesntExistException {
        if(userId == null) throw new
InvalidArgumentsException("Entered user id cannot be null");

        Optional<User> user = userRepository.findById(userId);

        if(user.iEmpty()) throw new ObjectDoesntExistException("User
you are looking for doesn't exist!");

        return user.get();
    }
```

**Ispis 17**: Primjer dohvaćanja korisnika po njihovom id-u u java spring bootu

Metoda `findUserById` služi za dohvaćanje korisničkog profila iz baze podataka na temelju unesenog ID-a. Ova metoda provjerava valjanost ID-a i koristi `userRepository` za pronalazak korisnika.

Na početku metode, provjerava se je li uneseni ID `null`. Ako je ID `null`, baca se `InvalidArgumentsException` s porukom "Entered user id cannot be `null`". Ako je ID valjan, metoda koristi `userRepository` za pronalazak korisnika pomoću metode `findById`, koja vraća `Optional<User>` objekt.

Ako korisnik ne postoji, tj. ako je `Optional` prazan, metoda baca `ObjectDoesntExistException` s porukom "User you are looking for doesn't exist!". Ako korisnik postoji, vraća se objekt tipa `User`.

### 4.1.4. Uređivanje korisnika

Uređivanje korisnika omogućava postojećim korisnicima ažuriranje svojih podataka unutar aplikacije. Kontroler za uređivanje prima zahtjev PUT s korisničkim ID-om i novim podacima za ažuriranje, koji se zatim provjeravaju i obrađuju u servisu.

```
public ResponseEntity<UserResDto> edit(@PathVariable String id,
@RequestBody UserDto userDto) throws InvalidArgumentsException,
ObjectDoesntExistException, AlreadyExistException {
    UserResDto updatedUser = userService.updateUser(id, userDto);
    return ResponseEntity.status(HttpStatus.OK).body(updatedUser);
}
```

**Ispis 18**: Primjer metode u kontroleru za uređivanje korisnika u java spring bootu

Metoda `edit` (Ispis 18) je implementirana kao HTTP zahtjev PUT koji služi za ažuriranje korisničkog profila na temelju unesenog korisničkog ID-a i novih podataka. Ova metoda koristi anotaciju `@PutMapping("/{id}")`, koja označava *endpoint* PUT, pri čemu `{id}` predstavlja dinamički dio URL-a koji odgovara korisničkom ID-u.

Pri pozivu ove metode, kao putni parametar (`@PathVariable String id`) unosi se korisnički ID, a tijelo zahtjeva (`@RequestBody UserDto userDto`) sadrži nove podatke korisnika. Metoda vraća objekt `ResponseEntity` koji sadrži DTO ažuriranog korisničkog profila.

Metoda započinje provjerom unesenog ID-a i novih podataka. Ako su ID ili novi podaci nevažeći, metoda baca `InvalidArgumentsException`. Ako su podaci valjani, metoda poziva `updateUser` (Ispis 19) iz `userService` klase kako bi ažurirala korisnički

profil. Rezultat ovog poziva je objekt tipa `UserResDto`, koji predstavlja ažurirane korisničke podatke.

```
@Override
public UserResDto updateUser(String userId, UserDto userDto) throws
InvalidArgumentsException, ObjectDoesntExistException {
    if(userId == null || userDto == null) throw new
InvalidArgumentsException("Entered user id or user info cannot be
null!");
    User user = findUserById(userId);
    User updatedUser = userRepository.save(updatedUser(user,
userDto));
    return userMapper.userToUserDto(updatedUser);
}
```

**Ispis 19**: Primjer metode u `user` servisu za uređivanje korisnika u java spring bootu

Metoda `updateUser` (Ispis 19) služi za ažuriranje korisničkog profila u bazi podataka na temelju unesenog ID-a i novih podataka. Ova metoda provjerava valjanost ID-a i novih podataka te koristi `userRepository` za spremanje ažuriranih podataka.

Na početku metode, provjerava se jesu li uneseni ID i novi podaci `null`. Ako je bilo koji od njih `null`, baca se `InvalidArgumentsException` s porukom "Entered user id or user info cannot be `null`!". Ako su podaci valjani, metoda koristi `findUserById` za dohvaćanje trenutnih podataka korisnika.

Nakon što su trenutni podaci korisnika dohvaćeni, metoda koristi `userRepository` za spremanje ažuriranih podataka korisnika pomoću metode `save`. Ažurirani korisnički objekt se potom mapira u DTO objekt pomoću `userMapper.userToUserDto` metode, kako bi se podaci prikazali u odgovarajućem formatu za prijenos podataka.

### 4.1.5. Brisanje korisnika

Brisanje korisnika omogućava uklanjanje postojećih korisnika iz aplikacije. Kontroler za brisanje prima zahtjev DELETE s korisničkim ID-om, koji se zatim provjerava i obrađuje u servisu za korisnike.

```
@DeleteMapping("/{id}")
public ResponseEntity<Boolean> delete(@PathVariable String id)
throws ObjectDoesntExistException, AlreadyExistException,
InvalidArgumentsException {
    userService.deleteUser(id);
    return ResponseEntity.status(HttpStatus.OK).body(true);
}
```

**Ispis 20**: Primjer metode u `user` kontroleru za brisanje korisnika u java spring bootu

Metoda `delete` (Ispis 20) je implementirana kao HTTP zahtjev DELETE koji služi za brisanje korisničkog profila na temelju unesenog korisničkog ID-a. Ova metoda koristi anotaciju `@DeleteMapping("/{id}")`, koja označava *endpoint* DELETE, pri čemu `{id}` predstavlja dinamički dio URL-a koji odgovara korisničkom ID-u.

Pri pozivu ove metode, kao putni parametar (`@PathVariable String id`) unosi se korisnički ID. Metoda vraća objekt `ResponseEntity` koji sadrži boolean vrijednost koja označava uspješnost brisanja korisnika.

Metoda započinje provjerom unesenog ID-a. Ako je ID nevažeći, metoda baca `InvalidArgumentsException`. Ako je ID valjan, metoda poziva `deleteUser` iz `userService` klase kako bi obrisala korisnički profil. Ako brisanje korisnika uspije, metoda vraća objekt `ResponseEntity` s HTTP statusom 200 (OK) i tijelom odgovora koje sadrži vrijednost `true`. Ako se dogodi greška, kao što je nepostojanje korisnika ili nevažeći ID, metoda može baciti `ObjectDoesntExistException` ili `InvalidArgumentsException`.

```
public void deleteUser(String userId) throws
InvalidArgumentsException, ObjectDoesntExistException {
    if(userId == null) throw new InvalidArgumentsException("Entered
user id cannot be null");

    userRepository.delete(findUserById(userId));
}
```

**Ispis 21**: Primjer metode u `user` servisu za brisanje korisnika u java spring bootu

Metoda `deleteUser` (Ispis 21) služi za brisanje korisničkog profila iz baze podataka na temelju unesenog ID-a. Ova metoda provjerava valjanost ID-a te koristi `userRepository` za pronalazak i brisanje korisnika.

Na početku metode, provjerava se je li uneseni ID `null`. Ako je ID `null`, baca se `InvalidArgumentsException` s porukom "Entered user id cannot be `null`". Ako je ID valjan, metoda koristi `findUserById` za dohvaćanje trenutnih podataka korisnika.

Nakon što su trenutni podaci korisnika dohvaćeni, metoda koristi `userRepository` za brisanje korisničkog profila pomoću metode `delete`. Ako korisnik ne postoji, metoda baca `ObjectDoesntExistException`.

## 4.2. Recipe

Sustav za upravljanje receptima unutar aplikacije omogućava korisnicima kreiranje, uređivanje i brisanje recepata. Ove funkcionalnosti su dizajnirane kako bi osigurale samo registriranim korisnicima mogućnost dodavanja novih recepata, dok mogućnost uređivanja i brisanja recepata imaju samo vlasnici tih recepata ili administratori sustava.

Kreiranje i uređivanje recepata implementirano je korištenjem biblioteka Formik i Yup. Formik olakšava rad s formama, pružajući jednostavno upravljanje stanjima forme i validaciju podataka, dok Yup omogućava definiranje i provedbu pravila za validaciju unesenih podataka. Kada korisnik popuni formu za kreiranje ili uređivanje recepta, podaci se prikupljaju i validiraju pomoću Formika i Yupa. Ako su svi podaci ispravni, oni se šalju na *backend* poslužitelj koristeći odgovarajuće HTTP metode – `axios.post` za kreiranje novog recepta, `axios.put` za uređivanje postojećeg recepta, te `axios.delete` za brisanje recepta.

Kako bi se osigurala sigurnost i integritet podataka, samo registrirani korisnici mogu kreirati recepte, dok mogućnost uređivanja i brisanja recepata imaju samo vlasnici tih recepata ili administratori. Ova ograničenja implementirana su putem autorizacijskih mehanizama koji provjeravaju prava pristupa korisnika.

```
const router = useRouter();
const { userId } = params;
useEffect(() => {
  if (!authenticateState.success) {
    router.push("/Login");
    return;
  }
  if (
    authenticateState.user.role !== "ADMIN" &&
    authenticateState.user.id !== userId
  ) {
    router.push("/Home");
    return;
  }
}, [authenticateState, userId]);
```

**Ispis 22**: Primjer autorizacije korisnika na stranicu na *frontendu*

Kôd za autorizaciju korisnika (Ispis 22) implementiran je koristeći React *hook* `useEffect`, kako bi se osiguralo samo ovlaštenim korisnicima mogućnost pristupa određenim stranicama koje su vezane za recepte. Funkcija `useEffect` se izvršava svaki put kada se promijene vrijednosti `authenticateState` ili `userId`, čime se provjerava valjanost pristupa korisnika.

Korištenje *hooka* `useRouter` iz Next.js biblioteke omogućava pristup routeru, čime se omogućuje preusmjeravanje korisnika na druge stranice. Na početku se iz parametara URL-a izvlači `userId`, koji predstavlja ID korisnika koji je vlasnik recepta.

Prvo se provjerava je li korisnik uspješno autenticiran. Ako autentikacija nije uspješna, korisnik se preusmjerava na stranicu za prijavu (`/Login`). Ova provjera se obavlja pomoću stanja `authenticateState.success`. Ako ovo stanje nije istinito, funkcija `router.push("/Login")` preusmjerava korisnika na stranicu za prijavu i time zaustavlja daljnje izvršavanje koda.

Sljedeći korak je provjera korisničke uloge i vlasništva nad receptom. Ako korisnik nije administrator (provjerava se pomoću `authenticateState.user.role !== "ADMIN"`) i ako korisnik nije vlasnik recepta (`authenticateState.user.id !== userId`), korisnik se preusmjerava na početnu stranicu (`/Home`). Ova provjera osigurava pristup stranici isključivo administratoru ili vlasniku recepta, dok se svi ostali korisnici preusmjeravaju na početnu stranicu.

Ovaj pristup omogućava sigurnu i pouzdanu provjeru korisničkih prava pristupa na *frontend* dijelu aplikacije, osigurava se mogućnost mijenjanja sadržaja recepata samo za ovlaštene korisnike. Korištenjem *hooka* `useEffect`, ove provjere se automatski izvršavaju svaki put kada se promijene relevantni parametri, čime se dodatno osigurava dinamičko i točno upravljanje pristupom unutar aplikacije.

### 4.2.1. Kreiranje recepata

Kreiranje recepata u aplikaciji predstavlja ključnu funkcionalnost koja omogućava registriranim korisnicima unos novih kulinarskih kreacija. Ovaj proces uključuje više koraka,

od prijenosa podataka i slike recepta do spremanja tih podataka u bazu podataka. U nastavku su detaljno opisane metode koje se koriste za realizaciju ove funkcionalnosti.

```
    @PostMapping(value = "/create", consumes = {"multipart/form-
data"})
    public ResponseEntity<Boolean>
addNewRecipe(@RequestPart("recipe") RecipeDto recipeDto,
@RequestPart("image") MultipartFile image) throws IOException,
InvalidArgumentsException {

            recipeDto.setImage(image);
            recipeService.createRecipe(recipeDto);

            return ResponseEntity.ok().build();

    }
```

**Ispis 23**: Primjer metode u `recipe` kontroleru za dodavanje recepata u java spring bootu

Metoda `addNewRecipe` (Ispis 23) je implementirana kao HTTP zahtjev POST koji služi za dodavanje novog recepta. Ova metoda koristi anotaciju `@PostMapping(value = "/create", consumes = {"multipart/form-data"})`,što označava prihvaćanje podataka u formatu multipart/form-data. Ovaj format je potreban kako bi se mogli prenijeti kompleksni podaci koji uključuju tekstualne podatke (detalje o receptu) i binarne podatke (sliku).

Pri pozivu ove metode, kao dijelovi zahtjeva (`@RequestPart`) unosi se objekt `RecipeDto`, koji sadrži sve informacije o receptu, i objekt `MultipartFile`, koji predstavlja sliku recepta. Metoda prvo postavlja sliku u objekt `RecipeDto` pomoću `recipeDto.setImage(image)`. Ako je kreiranje recepta uspješno, metoda vraća objekt `ResponseEntity` s HTTP statusom 200 (OK).

```
public Boolean createRecipe(RecipeDto recipeDto) throws
InvalidArgumentsException, IOException {
    if(recipeDto == null) throw new InvalidArgumentsException("Sent
recipe argument data cannot be null!");
```

```
    imageUtil.save("../../coolinarka-gui-next/public/uploads",
recipeDto.getImage().getOriginalFilename(), recipeDto.getImage());

    Recipe recipe =
recipeRepository.save(recipeMapper.recipeDtoToRecipe(recipeDto));

    recipeDto.getIngredients().forEach(recipeIngredientDto -> {
        RecipeIngredient recipeIngredient =
recipeIngredientMapper.recipeIngredientDtoToRecipeIngredient(recipeI
ngredientDto);
        recipeIngredient.setRecipe(recipe);
        recipeIngredientRepository.save(recipeIngredient);
    });

    recipeDto.getPhases().forEach(recipePhaseDto -> {
        RecipePhase recipePhase =
recipePhaseMapper.recipePhaseDtoToRecipePhase(recipePhaseDto);
        recipePhase.setRecipe(recipe);
        recipePhaseRepository.save(recipePhase);
    });

    recipeDto.getEvents().forEach(event ->
recipeEventRepository.save(RecipeEvent.builder().recipe(recipe).even
t(event).build()));

    recipeDto.getMealGroup().forEach(mealGroup ->
recipeMealGroupRepository.save(RecipeMealGroup.builder().mealGroup(m
ealGroup).recipe(recipe).build()));

    return true;
}
```

**Ispis 24**: Primjer metode u `recipe` servisu za dodavanje recepata u java spring bootu

Metoda `createRecipe` (Ispis 24) nalazi se u `recipeService` klasi i obavlja stvarni posao kreiranja recepta. Ova metoda prima `RecipeDto` objekt kao argument i baca iznimke `InvalidArgumentsException` i `IOException` u slučaju pogrešaka.

Metoda započinje provjerom je li `RecipeDto` objekt `null`. Ako je `null`, baca se `InvalidArgumentsException` s porukom "Sent recipe argument data cannot be `null`!". Nakon toga, metoda koristi `imageUtil.save` kako bi spremila sliku recepta na lokalno spremište.

Nakon što je slika uspješno spremljena, metoda koristi `recipeRepository.save` za spremanje recepta u bazu podataka. Ova metoda koristi JPA (Java Persistence API) za interakciju s bazom podataka, čime se osigurava jednostavno i efikasno upravljanje podacima.

Metoda zatim prolazi kroz listu sastojaka (`ingredients`) iz `RecipeDto` objekta i sprema svaki sastojak u bazu podataka pomoću `recipeIngredientRepository.save`. Slično tome, prolazi se kroz listu faza pripreme (`phases`) i svaka faza se sprema pomoću `recipePhaseRepository.save`.

Dodatno, metoda sprema događaje (`events`) i grupe obroka (`mealGroup`) povezane s receptom koristeći odgovarajuće repozitorije (`recipeEventRepository` i `recipeMealGroupRepository`). Ako su svi ovi koraci uspješno izvršeni, metoda vraća `true`, čime se označava uspješno kreiran recept.

### 4.2.2. Stranica sa svim detaljima recepta

Stranica sa svim detaljima recepta omogućava korisnicima pregled specifičnih informacija o pojedinom receptu. Ovaj proces uključuje dohvaćanje recepta iz baze podataka prema unesenom ID-u i prikazivanje tih informacija korisnicima u odgovarajućem formatu.

```
@GetMapping("/{id}")
public ResponseEntity<RecipeResponseDto> recipePage(@PathVariable
String id) throws ObjectDoesntExistException {
    RecipeResponseDto recipeResponseDto =
recipeService.filterById(id);
    return ResponseEntity.ok().body(recipeResponseDto);
}
```

**Ispis 25**: Primjer metode u `recipe` kontroleru za dohvaćanje pojedinog recepta

Metoda `recipePage` (Ispis 25) implementirana je kao HTTP zahtjev GET koji služi za dohvaćanje detalja određenog recepta na temelju unesenog ID-a. Ova metoda koristi anotaciju `@GetMapping("/{id}")`, što označava *endpoint* GET, pri čemu `{id}` predstavlja dinamički dio URL-a koji odgovara ID-u recepta.

Pri pozivu ove metode, kao putni parametar (`@PathVariable String id`) unosi se ID recepta. Metoda vraća objekt `ResponseEntity` koji sadrži DTO recepta. DTO objekt je prilagođen za prijenos podataka između slojeva aplikacije.

Metoda poziva `filterById` (Ispis 26) iz `recipeService` klase kako bi dohvatila recept prema unesenom ID-u. Rezultat ovog poziva je objekt tipa `RecipeResponseDto`, koji predstavlja sve relevantne podatke o receptu. Konačno, metoda vraća objekt `ResponseEntity` s HTTP statusom 200 (OK) i tijelom odgovora koje sadrži DTO recepta.

```
@Override
public RecipeResponseDto filterById(String recipeId) throws
ObjectDoesntExistException {
    Optional<Recipe> recipe = recipeRepository.findById(recipeId);

    if(recipe.isEmpty()) throw new
ObjectDoesntExistException("Recipe you are looking for doesn't
exist");

    return recipeResponseMapper(recipe.get());
}
```

**Ispis 26**: Primjer metode u `recipe` servisu za dohvaćanje recepta po njegovom ID-u

Metoda `filterById` nalazi se u `recipeService` klasi i obavlja stvarni posao dohvaćanja detalja recepta iz baze podataka. Ova metoda prima ID recepta kao argument i baca iznimku `ObjectDoesntExistException` u slučaju recept s danim ID-om ne postoji.

Metoda prvo koristi `recipeRepository.findById` za pronalaženje recepta prema unesenom ID-u. Ako recept nije pronađen, metoda baca `ObjectDoesntExistException` s

porukom "Recipe you are looking for doesn't exist". Ako je recept pronađen, metoda koristi `recipeResponseMapper` za mapiranje objekta tipa `Recipe` u `RecipeResponseDto`, koji sadrži sve relevantne podatke o receptu u prikladnom formatu za prijenos podataka.

### 4.2.3. Izlistavanje svih ili filtriranih recepata

Izlistavanje recepata omogućava korisnicima pregled svih dostupnih recepata u aplikaciji ili filtriranje recepata prema određenim kriterijima kao što su naziv, država, sezona, korisnik, težina pripreme i sastojci. Ovaj proces uključuje obradu HTTP zahtjeva GET, prikupljanje i obradu parametara filtriranja te dohvaćanje recepata iz baze podataka.

```
@Override
@GetMapping("/filter")
public ResponseEntity<RecipePageDto> filterList(@RequestParam
Map<String, String> allParams) {
    RecipePageDto recipePageDto =
recipeService.filterRecipes(allParams);
    return ResponseEntity.ok().body(recipePageDto);
}
```

**Ispis 27**: Primjer metode u `recipe` kontroleru za dohvaćanje liste recepta

Metoda `filterList` (Ispis 27) implementirana je kao HTTP zahtjev GET koji služi za dohvaćanje svih recepata ili filtriranje recepata prema zadanim kriterijima. Ova metoda koristi anotaciju `@GetMapping("/filter")`, što označava *endpoint* koji prima zahtjev na putanji `/filter`.

Metoda koristi `@RequestParam Map<String, String> allParams` kako bi prikupila sve parametre filtriranja iz URL-a. Zatim poziva `recipeService.filterRecipes(allParams)` kako bi dobila rezultate filtriranja u obliku `RecipePageDto` objekta, koji sadrži popis recepata i informacije o stranici (paginaciji).

Konačno, metoda vraća objekt `ResponseEntity` s HTTP statusom 200 (OK) i tijelom odgovora koje sadrži `RecipePageDto` objekt.

```
public RecipePageDto filterRecipes(Map<String, String> allParams) {
```

```
    int page = allParams.get("page") != null ?
Integer.parseInt(allParams.get("page")) - 1 : 0;
    int size = allParams.get("size") != null ?
Integer.parseInt(allParams.get("size")) : 6;

    String sort = allParams.get("sort");
    if (StringUtils.isEmpty(sort)) {
        sort = "id";
    }
    Pageable pageable = PageRequest.of(page, size,
Sort.by(Sort.Direction.DESC, sort));

    String name = allParams.get("name");
    String country = allParams.get("country");
    String season = allParams.get("season");
    String userId = allParams.get("userId");
    String difficulty = allParams.get("difficulty");
    List<String> ingredients = allParams.get("ingredients") != null
&& !allParams.get("ingredients").isEmpty()
            ?
Arrays.asList(allParams.get("ingredients").toLowerCase().split(","))
            : null;

    long ingredientCount = ingredients != null ? ingredients.size()
: 0;

    Page<Recipe> recipePage = recipeRepository.findRecipesByFilters(
            name != null && !name.isEmpty() ? name.toLowerCase() :
null,
            country != null && !country.isEmpty() ?
country.toLowerCase() : null,
            season != null && !season.isEmpty() ?
season.toLowerCase() : null,
            difficulty != null && !difficulty.isEmpty() ?
difficulty.toLowerCase() : null,
            userId != null && !userId.isEmpty() ? userId : null,
            ingredients,
```

```
            ingredientCount,
            pageable);

    List<Recipe> recipeList = recipePage.getContent();

    return RecipePageDto.builder()

.recipePage(recipeList.stream().map(this::recipeResponseMapper).toLi
st())
            .pageNum(page)
            .pageSize(size)
            .lastPage(recipePage.getTotalPages())
            .build();
}
```

**Ispis 28**: Primjer metode u `recipe` servisu za dohvaćanje liste recepta

Metoda `filterRecipes` (Ispis 28) nalazi se u `recipeService` klasi i obavlja stvarni posao filtriranja recepata. Ova metoda prima mapu svih parametara kao argument i koristi različite kriterije za filtriranje recepata.

Prvo se definiraju parametri za paginaciju, poput broja stranice (`page`) i veličine stranice (`size`). Ako ovi parametri nisu zadani, koriste se zadane vrijednosti. Zatim se definiraju parametri za sortiranje (`sort`), naziv recepta (`name`), država (`country`), sezona (`season`), ID korisnika (`userId`), težina pripreme (`difficulty`) i sastojci (`ingredients`). Ako ovi parametri nisu zadani ili su prazni, postavljaju se na `null`. Na temelju ovih parametara, metoda poziva `recipeRepository.findRecipesByFilters` kako bi dohvatila filtrirane recepte iz baze podataka. Rezultat je stranica (`Page<Recipe>`) recepata koja sadrži popis recepata i informacije o paginaciji.

Metoda zatim stvara `RecipePageDto` objekt, koji sadrži popis recepata (mapiranih u DTO format), broj trenutne stranice, veličinu stranice i ukupni broj stranica.

```
@Query("SELECT r FROM Recipe r WHERE "
        + "(:name IS NULL OR :name = '' OR (LENGTH(:name) >= 3 AND
LOWER(r.name) LIKE LOWER(CONCAT('%', :name, '%')))) "
```

```
        + "AND (:country IS NULL OR LOWER(r.country) LIKE
%:country%) "
        + "AND (:season IS NULL OR LOWER(r.season) LIKE %:season%) "
        + "AND (:difficulty IS NULL OR LOWER(r.difficulty) LIKE
%:difficulty%) "
        + "AND (:userId IS NULL OR r.user.id = :userId) "
        + "AND (:ingredients IS NULL OR ("
        + "    SELECT COUNT(DISTINCT LOWER(ri.ingredient)) "
        + "    FROM RecipeIngredient ri "
        + "    WHERE ri.recipe.id = r.id AND LOWER(ri.ingredient) IN
:ingredients) = :ingredientCount)")
Page<Recipe> findRecipesByFilters(
        @Param("name") String name,
        @Param("country") String country,
        @Param("season") String season,
        @Param("difficulty") String difficulty,
        @Param("userId") String userId,
        @Param("ingredients") List<String> ingredients,
        @Param("ingredientCount") long ingredientCount,
        Pageable pageable);
```

**Ispis 29**: Primjer metode u `recipe` repozitoriju za dohvaćanje liste recepta iz baze podataka

Upit `findRecipesByFilters` (Ispis 29) je definiran u `recipeRepository` klasi koristeći JPQL (Java Persistence Query Language). Ovaj upit filtrira recepte na temelju parametara kao što su naziv, država, sezona, težina pripreme, ID korisnika i sastojci.

Upit koristi različite uvjete za filtriranje recepata. Ako je određeni parametar `null` ili prazan, taj uvjet se zanemaruje. Upit također koristi `Pageable` objekt kako bi omogućio paginaciju rezultata.

### 4.2.4. Uređivanje recepta

Uređivanje recepata omogućava korisnicima ažuriranje postojećih recepata u aplikaciji. Ovaj proces uključuje obradu HTTP zahtjeva PUT za ažuriranje podataka o receptu, spremanje promjena u bazu podataka i rukovanje slikama koje su povezane s receptom.

```
@PutMapping(value = "/{id}", consumes = {"multipart/form-data"})
public ResponseEntity<Boolean> editRecipe(@PathVariable String id,
@RequestPart("recipe") RecipeDto recipeDto, @RequestPart("image")
MultipartFile image) throws ObjectDoesntExistException, IOException,
InvalidArgumentsException {
    recipeDto.setImage(image);
    recipeService.updateRecipe(id, recipeDto);
    return ResponseEntity.status(HttpStatus.OK).body(true);
}
```

**Ispis 30**: Primjer metode u `recipe` kontroleru za uređivanje recepta

Metoda `editRecipe` (Ispis 30) implementirana je kao HTTP zahtjev PUT koji služi za ažuriranje recepta na temelju unesenog ID-a recepta. Ova metoda koristi anotaciju `@PutMapping("/{id}")`, koja označava *endpoint* koji prima zahtjev na putanji koja sadrži ID recepta. Metoda koristi `@RequestPart` anotaciju kako bi prihvatila dva dijela podataka: `RecipeDto` objekt koji sadrži podatke o receptu i `MultipartFile` objekt koji predstavlja sliku recepta. Nakon što se podaci prikupe, metoda poziva `recipeService.updateRecipe(id, recipeDto)` kako bi ažurirala recept u bazi podataka. Konačno, metoda vraća objekt `ResponseEntity` s HTTP statusom 200 (OK) i tijelom odgovora koje potvrđuje uspješno ažuriranje recepta.

```
@Transactional
@Override
public RecipeResponseDto updateRecipe(String recipeId, RecipeDto
recipeDto)
        throws ObjectDoesntExistException, IOException,
InvalidArgumentsException {

    if (recipeDto == null) {
        throw new InvalidArgumentsException("Sent recipe argument
data cannot be null!");
    }

    Optional<Recipe> optionalRecipe =
recipeRepository.findById(recipeId);
    if (optionalRecipe.isEmpty()) {
        throw new ObjectDoesntExistException("Recipe with given ID
does not exist!");
    }

    Recipe recipe = optionalRecipe.get();

    // Handle image update
    if (recipeDto.getImage() != null) {
        imageUtil.save("../../coolinarka-gui-next/public/uploads",
```

```
                recipeDto.getImage().getOriginalFilename(),
recipeDto.getImage());
        recipe.setImage(recipeDto.getImage().getOriginalFilename());
    }

    // Update basic details
    recipe.setName(recipeDto.getName());
    recipe.setDescription(recipeDto.getDescription());
    recipe.setCountry(recipeDto.getCountry());
    recipe.setPrepTime(recipeDto.getPrepTime());
    recipe.setPeople(recipeDto.getPeople());
    recipe.setDifficulty(recipeDto.getDifficulty());
    recipe.setSeason(recipeDto.getSeason());

    // Remove old ingredients
    recipeIngredientRepository.deleteByRecipeId(recipeId);

    // Save new ingredients
    recipeDto.getIngredients().forEach(recipeIngredientDto -> {
        RecipeIngredient recipeIngredient = recipeIngredientMapper

.recipeIngredientDtoToRecipeIngredient(recipeIngredientDto);
        recipeIngredient.setRecipe(recipe);
        recipeIngredientRepository.save(recipeIngredient);
    });

    // Remove old phases
    recipePhaseRepository.deleteByRecipeId(recipeId);

    // Save new phases
    recipeDto.getPhases().forEach(recipePhaseDto -> {
        RecipePhase recipePhase = recipePhaseMapper
                .recipePhaseDtoToRecipePhase(recipePhaseDto);
        recipePhase.setRecipe(recipe);
        recipePhaseRepository.save(recipePhase);
    });

    // Remove old events and meal groups
    recipeEventRepository.deleteByRecipeId(recipeId);
    recipeMealGroupRepository.deleteByRecipeId(recipeId);

    // Save new events
    recipeDto.getEvents().forEach(event -> {
        RecipeEvent recipeEvent = RecipeEvent.builder()
                .recipe(recipe)
                .event(event)
                .build();
        recipeEventRepository.save(recipeEvent);
    });

    // Save new meal groups
    recipeDto.getMealGroup().forEach(mealGroup -> {
        RecipeMealGroup recipeMealGroup = RecipeMealGroup.builder()
                .mealGroup(mealGroup)
                .recipe(recipe)
                .build();
```

```
            recipeMealGroupRepository.save(recipeMealGroup);
        });

        // Save the updated recipe
        Recipe updatedRecipe = recipeRepository.save(recipe);
        return null;
    }
```

**Ispis 31**: Primjer metode u `recipe` servisu za uređivanje recepta

Metoda `updateRecipe` (Ispis 31) nalazi se u `recipeService` klasi i obavlja stvarni posao ažuriranja recepta. Ova metoda prima ID recepta i `RecipeDto` objekt kao argumente te koristi različite korake kako bi ažurirala recept.Prvo, metoda provjerava je li `recipeDto` objekt prazan i baca iznimku ako jest. Zatim dohvaća postojeći recept iz baze podataka na temelju ID-a. Ako recept ne postoji, metoda baca iznimku `ObjectDoesntExistException`. Ako je slika recepta prisutna u `recipeDto` objektu, metoda sprema novu sliku pomoću `imageUtil` klase i ažurira polje slike u objektu recepta. Metoda zatim ažurira osnovne podatke recepta, kao što su naziv, opis, država, vrijeme pripreme, broj osoba, težina pripreme i sezona. Sljedeći korak je uklanjanje starih sastojaka i spremanje novih sastojaka povezanih s receptom. Ovo se postiže pomoću `recipeIngredientRepository` i `recipeIngredientMapper` objekata. Isto tako, metoda uklanja stare faze pripreme i sprema nove faze pomoću `recipePhaseRepository` i `recipePhaseMapper` objekata. Na kraju, metoda uklanja stare događaje i grupe jela te sprema nove događaje i grupe jela pomoću `recipeEventRepository` i `recipeMealGroupRepository` objekata. Nakon što su svi podaci ažurirani, metoda sprema ažurirani recept u bazu podataka i vraća ažurirani `RecipeResponseDto` objekt.

### 4.2.5. Brisanje recepta

Brisanje recepta omogućava korisnicima uklanjanje postojećih recepata iz aplikacije. Ova funkcionalnost koristi HTTP zahtjev DELETE kako bi identificirala recept za brisanje i obrisala ga iz baze podataka zajedno s pripadajućom slikom.

```
@Override
@DeleteMapping("/{id}")
public ResponseEntity<Boolean> deleteRecipe(@PathVariable String id)
throws ObjectDoesntExistException {
    recipeService.deleteRecipe(id);
    return ResponseEntity.status(HttpStatus.OK).body(true);
```

```
}
```

**Ispis 32**: Primjer metode u `recipe` kontroleru za brisanje recepta

Metoda `deleteRecipe` (Ispis 32) implementirana je kao HTTP zahtjev DELETE koji služi za brisanje recepta na temelju unesenog ID-a recepta. Ova metoda koristi anotaciju `@DeleteMapping("/{id}")`, koja označava *endpoint* koji prima zahtjev DELETE na putanji koja sadrži ID recepta.

Metoda koristi `@PathVariable` anotaciju kako bi prihvatila ID recepta kao ulazni parametar. Nakon što se ID recepta dobije, metoda poziva `recipeService.deleteRecipe(id)` (Ispis 33) kako bi obrisala recept iz baze podataka. Konačno, metoda vraća objekt `ResponseEntity` s HTTP statusom 200 (OK) i tijelom odgovora koje potvrđuje uspješno brisanje recepta.

```
@Override
public void deleteRecipe(String recipeId) throws
ObjectDoesntExistException {
    Optional<Recipe> recipeOptional =
recipeRepository.findById(recipeId);

    if (recipeOptional.isEmpty()) throw new
ObjectDoesntExistException("Recipe you want to delete doesn't
exist");

    Recipe recipe = recipeOptional.get();
    imageUtil.delete(recipe.getImage());

    recipeRepository.delete(recipe);
}
```

**Ispis 33**: Primjer metode u `recipe` servisu za brisanje recepta

## 4.3. `Favorites`

Korisnici koji su prijavljeni mogu koristiti funkcionalnost favorita kako bi dodali ili uklonili recepte iz svoje liste omiljenih. Ova mogućnost omogućava korisnicima brz pristup svojim najdražim receptima te organiziraju recepte prema vlastitim preferencijama. Implementacija favorita osigurava korisnicima personalizirano iskustvo unutar aplikacije, olakšavajući im pronalaženje i spremanje omiljenih recepata za kasniju uporabu.

### 4.3.1. Dodavanje recepta u favorite

Dodavanje recepta u favorite omogućava prijavljenim korisnicima spremanje omiljenih recepata za brzi pristup u budućnosti. Ova funkcionalnost je implementirana putem HTTP zahtjeva POST koji omogućuje korisniku slanje zahtjeva za dodavanje recepta u svoju listu favorita.

```
@PostMapping("/")
public ResponseEntity<Boolean> addToFavorites(@RequestBody FavoritesDto
favoritesDto) throws InvalidArgumentsException, ObjectDoesntExistException,
AlreadyExistException {
    favoriteService.addToFavorites(favoritesDto);
    return ResponseEntity.status(HttpStatus.OK).body(true);
}
```

**Ispis 34**: Primjer metode u `favorites` kontroleru za dodavanje recepata u omiljene

Metoda `addToFavorites` (Ispis 34) implementirana je kao HTTP zahtjev POST koji prima FavoritesDto objekt putem `@RequestBody` anotacije. Ova metoda koristi `@PostMapping("/")` kako bi definirala *endpoint* za dodavanje recepta u favorite.

Metoda prvo poziva `favoriteService.addToFavorites(favoritesDto)` kako bi obradila logiku dodavanja recepta u favorite. Ako je operacija uspješna, metoda vraća `ResponseEntity` s HTTP statusom 200 (OK) i tijelom odgovora koje potvrđuje uspješno dodavanje recepta u favorite.

```
public void addToFavorites(FavoritesDto favoritesDto) throws
InvalidArgumentsException, AlreadyExistException,
ObjectDoesntExistException {
```

```
    User user = userService.findUserById(favoritesDto.getUser());
    RecipeResponseDto recipeResponseDto =
recipeService.filterById(favoritesDto.getRecipe());
    if (user == null) throw new ObjectDoesntExistException("User
that want to add recipe in favorites doesn't exist in db");
    if (recipeResponseDto == null) throw new
ObjectDoesntExistException("Recipe that want to be added in
favorites doesn't exist in db");


favoriteRepository.save(Favorites.builder().recipe(recipeRepository.
findById(favoritesDto.getRecipe()).get()).user(user).build());
}
```

**Ispis 35**: Primjer metode u `favorites` servisu za dodavanje recepata u omiljene

Servisna metoda `addToFavorites` u `favoriteService` klasi (Ispis 35) obavlja stvarni posao dodavanja recepta u favorite. Ova metoda prima `FavoritesDto` objekt kao argument i koristi različite korake kako bi obradila zahtjev. Metoda prvo dohvaća korisnika putem `userService.findUserById`. Ako korisnik ne postoji, baca se iznimka `ObjectDoesntExistException`. Zatim dohvaća recept putem `recipeService.filterById(favoritesDto.getRecipe())`. Ako recept ne postoji, također se baca iznimka `ObjectDoesntExistException`. Ako korisnik i recept postoje, metoda stvara novi `Favorites` objekt koji povezuje korisnika i recept, te ga sprema u `favoriteRepository`. Time se recept dodaje u korisnikovu listu favorita.

### 4.3.2. Uklanjanje recepta iz favorita

Uklanjanje recepta iz favorita omogućava korisnicima jednostavno upravljanje svojim listama favorita, omogućujući im brisanje recepata koje više ne žele čuvati kao favorite. Ova funkcionalnost je implementirana putem HTTP zahtjeva DELETE koji omogućava korisniku slanje zahtjeva za uklanjanje određenog recepta iz svoje liste favorita.

```
@DeleteMapping("/recipe/{id}/user/{uid}")
public ResponseEntity<Boolean>
removeFromFavorites(@PathVariable("id") String favoritesId,
```

```
@PathVariable("uid") String userId) throws
ObjectDoesntExistException, InvalidArgumentsException {
    favoriteService.remove(favoritesId, userId);
    return ResponseEntity.status(HttpStatus.OK).body(true);
}
```

**Ispis 36**: Primjer metode u `favorites` kontroleru za uklanjanje recepata iz omiljenih

Metoda `removeFromFavorites` (Ispis 36) implementirana je kao HTTP zahtjev DELETE koji prima putne parametre za `recipeId` i `userId` putem `@PathVariable` anotacija. Ova metoda koristi `@DeleteMapping("/recipe/{id}/user/{uid}")` kako bi definirala *endpoint* za uklanjanje recepta iz favorita.

Metoda prvo poziva `favoriteService.remove(favoritesId, userId)` kako bi obradila logiku uklanjanja recepta iz favorita. Ako je operacija uspješna, metoda vraća `ResponseEntity` s HTTP statusom 200 (OK) i tijelom odgovora koje potvrđuje uspješno uklanjanje recepta iz favorita.

```
@Override
public void remove(String favoriteId, String userId) throws
ObjectDoesntExistException, InvalidArgumentsException {
    if (favoriteId == null) {
        throw new InvalidArgumentsException("Sent arguments cannot
be null!");
    }
    Optional<Favorites> favorites =
favoriteRepository.findByRecipeIdAndUserId(favoriteId, userId);

    if (favorites.isEmpty()) {
        throw new ObjectDoesntExistException("Recipe you want to
delete from favorites doesn't exist!");
    }

    favoriteRepository.delete(favorites.get());
}
```

**Ispis 37**: Primjer metode u `favorites` servisu za uklanjanje recepata iz omiljenih

Servisna metoda `remove` u `favoriteService` klasi (Ispis 37) obavlja stvarni posao uklanjanja recepta iz favorita. Ova metoda prima `favoriteId` i `userId` kao argumente i koristi ih kako bi pronašla i uklonila odgovarajući zapis favorita iz baze podataka. Metoda prvo provjerava jesu li `favoriteId` i `userId` valjani. Ako nije, baca se iznimka `InvalidArgumentsException`. Zatim se koristi `favoriteRepository.findByRecipeIdAndUserId` metoda kako bi pronašla zapis favorita koji povezuje određeni recept s određenim korisnikom. Ako zapis ne postoji, baca se iznimka `ObjectDoesntExistException`. Nakon uspješnog pronalaska, zapis se briše iz baze podataka pomoću `favoriteRepository.delete`.

## 4.4. `Review`

Recenzije su ključni dio interakcije korisnika s aplikacijom, omogućujući im dijeljenje mišljenja i iskustava o receptima koje su isprobali. Svaki korisnik može ostaviti samo jednu recenziju na svaki recept, koja se sastoji od ocjene i komentara na tu ocjenu. Ovo poglavlje detaljno opisuje kako su implementirane funkcionalnosti za dodavanje, uklanjanje i pregled recenzija.

### 4.4.1. Dodavanje recenzije na recept

Dodavanje recenzije na recept omogućuje korisnicima izražavanje svojih dojmova i ocjena recepta koji su isprobali. Ova funkcionalnost osigurava ostavljanje samo jedne recenzije po receptu, koja uključuje ocjenu i komentar.

```
@PostMapping("/create")
public ResponseEntity<ReviewResDto> createReview(@RequestBody
ReviewDto reviewDto) throws ObjectDoesntExistException,
InvalidArgumentsException, AlreadyExistException {
    ReviewResDto review = reviewService.createReview(reviewDto);
    return ResponseEntity.status(HttpStatus.OK).body(review);
}
```

**Ispis 38**: Primjer metode u `review` kontroleru za ostavljanje recenzije na recept

Kontroler sadrži HTTP POST metodu `createReview` (Ispis 38), koja prima podatke o recenziji putem JSON formata u tijelu zahtjeva. Nakon provjere valjanosti podataka, poziva

se odgovarajuća metoda u servisnom sloju za obradu recenzija. Metoda `createReview` u kontroleru prvo provjerava valjanost proslijeđenih podataka, a zatim poziva odgovarajuću metodu `createReview` u servisnom sloju.

Servisna metoda `createReview` (Ispis 39) u servisnom sloju upravlja logikom za dodavanje nove recenzije. Provjerava se postojanje korisnika i recepta u bazi podataka, te se osigurava da korisnik nije već ocijenio isti recept. Nakon što se nova recenzija spremi u bazu podataka, ažurira se prosječna ocjena recepta.

```
public ReviewResDto createReview(ReviewDto reviewDto) throws
InvalidArgumentsException, ObjectDoesntExistException,
AlreadyExistException {
    if (reviewDto == null) {
        throw new InvalidArgumentsException("Proslijeđeni argumenti
recenzije ne mogu biti null!");
    }

    User user = userService.findUserById(reviewDto.getUser());
    if (user == null) {
        throw new ObjectDoesntExistException("Korisnik koji želi
ostaviti recenziju ne postoji u bazi podataka.");
    }

    Optional<Recipe> recipe =
recipeRepository.findById(reviewDto.getRecipe());
    if (recipe.isEmpty()) {
        throw new ObjectDoesntExistException("Recept koji želite
ocijeniti ne postoji u bazi podataka.");
    }

    // Provjera je li korisnik već ocijenio ovaj recept
    Review existingReview =
reviewRepository.findByUserIdAndRecipeId(reviewDto.getUser(),
reviewDto.getRecipe());
    if (existingReview != null) {
        throw new AlreadyExistException("Korisnik je već ocijenio
ovaj recept.");
```

```
    }

    // Spremanje nove recenzije u bazu podataka
    Review review = reviewRepository.save(Review.builder()
            .recipe(recipe.get())
            .user(user)
            .rating(reviewDto.getRating())
            .comment(reviewDto.getComment())
            .build());

    // Ažuriranje prosječne ocjene recepta
    recipe.get().setRating(getRecipeRating(reviewDto.getRecipe()));
    recipeRepository.save(recipe.get());

    // Priprema odgovora s podacima o novoj recenziji
    return ReviewResDto.builder()
            .id(review.getId())
            .createdAt(review.getCreatedAt())
            .comment(review.getComment())
            .rating(review.getRating())
            .build();
}
```

**Ispis 39**: Primjer metode u `review` servisu za ostavljanje recenzije na recept

Metoda `getRecipeRating` (Ispis 40) izračunava prosječnu ocjenu recepta na temelju svih recenzija koje su ostavljene za taj recept.

```
public double getRecipeRating(String recipeId) throws
ObjectDoesntExistException {
    Optional<Recipe> recipe = recipeRepository.findById(recipeId);
    if (recipe.isEmpty()) {
        throw new ObjectDoesntExistException("Recept koji želite
ocijeniti ne postoji u bazi podataka.");
    }
```

```
    List<Review> reviews =
reviewRepository.findByRecipeId(recipeId);

    if (reviews.isEmpty()) {
        return 0.0; // Ako nema recenzija, prosječna ocjena je 0.0
    }

    // Izračun prosječne ocjene
    double averageRating = reviews.stream()
            .mapToDouble(Review::getRating)
            .average()
            .orElse(0.0);

    // Zaokruživanje na dvije decimale
    return Math.round(averageRating * 100.0) / 100.0;
}
```

**Ispis 40**: Metoda izračuna prosječne ocjene recepta

### 4.4.2. Uklanjanje recenzije s recepta

Uklanjanje recenzije s recepta omogućuje korisnicima brisanje svoje prethodno ostavljene recenzije. Svaki korisnik može ukloniti samo svoju recenziju, a uklanjanjem se također ažurira prosječna ocjena recepta. Kontroler sadrži HTTP DELETE metodu `deleteReview` (Ispis 41), koja prima ID recenzije koju korisnik želi izbrisati.

```
@DeleteMapping("/{id}")
public ResponseEntity<Boolean> deleteReview(@PathVariable String id)
throws ObjectDoesntExistException {
    reviewService.deleteReview(id);
    return ResponseEntity.status(HttpStatus.OK).body(true);
}
```

**Ispis 41**: Primjer metode u `review` kontroleru za uklanjanje recenzije s recepta

Metoda `deleteReview` u kontroleru (Ispis 41) jednostavno poziva odgovarajuću metodu u servisnom sloju za obradu recenzija.

Servisna metoda `deleteReview` u servisnom sloju (Ispis 42) provjerava postojanje recenzije s zadanim ID-om. Nakon brisanja recenzije, ažurira se prosječna ocjena recepta.

```
@Override
public void deleteReview(String reviewId) throws
ObjectDoesntExistException {
    Optional<Review> review = reviewRepository.findById(reviewId);
    if (review.isEmpty()) {
        throw new ObjectDoesntExistException("Recenzija koju želite
izbrisati ne postoji!");
    }

    // Pronalaženje ID-a recepta kako bi se mogla ažurirati
prosječna ocjena
    String recipeId = review.get().getRecipe().getId();

    // Brisanje recenzije iz baze podataka
    reviewRepository.delete(review.get());

    // Ažuriranje prosječne ocjene recepta nakon brisanja recenzije
    Optional<Recipe> recipe = recipeRepository.findById(recipeId);
    if (recipe.isPresent()) {
        recipe.get().setRating(getRecipeRating(recipeId));
        recipeRepository.save(recipe.get());
    }
}
```

**Ispis 42**: Primjer metode u `review` servisu za uklanjanje recenzije s recepta

# 5. Zaključak

Web aplikacija Gourmet Gems predstavlja inovativnu društvenu mrežu namijenjenu ljubiteljima kulinarstva. Njena glavna svrha je omogućiti korisnicima jednostavno kreiranje, pregledavanje i pretraživanje recepata, kao i interakciju s drugim kulinarskim entuzijastima. Uvođenjem ove aplikacije, korisnici mogu ne samo dijeliti svoje kulinarske kreacije, već i otkrivati nove recepte te sudjelovati u kulinarskoj zajednici putem recenzija i omiljenih recepata.

Tehnička strana aplikacije oslanja se na moderni razvojni pristup. *Frontend* aplikacije izgrađen je korištenjem NextJS, React razvojnog okvira koji omogućava prikazivanje na strani poslužitelja i statičku generaciju stranica, čime se poboljšavaju performanse aplikacije i SEO optimizacija. NextJS, zajedno s TailwindCSS-om, omogućava kreiranje responzivnog i estetski privlačnog grafičkog sučelja. TailwindCSS, kao *utility-first* CSS razvojni okvir, omogućava brzu i jednostavnu stilizaciju elemenata, što je ključno za dosljednost u dizajnu i lako održavanje koda.

*Backend* aplikacije razvijen je u Java Spring Boot okruženju, koje je poznato po svojoj robusnosti i jednostavnosti integracije s različitim tehnologijama. Spring Boot pojednostavljuje kreiranje produkcijski spremnih aplikacija, nudi podršku za razne načine autentikacije i autorizacije, te omogućava jednostavnu integraciju s bazama podataka. Korištenje PostgreSQL baze podataka osigurava pouzdanu i sigurnu pohranu podataka, s podrškom za ACID transakcije koje garantiraju integritet podataka.

Kroz razvoj aplikacije, poseban naglasak stavljen je na korisničko iskustvo. Neregistrirani korisnici mogu pretraživati i filtrirati recepte, dok registrirani korisnici imaju proširene mogućnosti koje uključuju kreiranje vlastitih recepata, uređivanje profila, ostavljanje recenzija i dodavanje recepata u favorite. Administrator aplikacije ima dodatne ovlasti, uključujući brisanje i uređivanje recepata, korisničkih profila i recenzija, čime se osigurava kontrola kvalitete sadržaja na platformi.

Struktura baze podataka dizajnirana je za fleksibilno i učinkovito upravljanje podacima. Glavni entiteti uključuju korisnike (`Users`), recepte (`Recipe`), favorite (`Favorites`) i recenzije (`Reviews`). Svaki entitet ima odgovarajuću tablicu u bazi podataka koja omogućava pohranu i dohvaćanje svih relevantnih informacija. Povezane tablice, poput

`Recipe_Ingredient`, omogućavaju detaljnu pohranu informacija o sastojcima potrebnim za pripremu recepata, čime se dodatno poboljšava funkcionalnost aplikacije.

Gourmet Gems uspješno kombinira društvene i tehničke aspekte kako bi stvorila korisnički orijentiranu platformu za dijeljenje kulinarskih iskustava. Implementacija naprednih tehnologija, fokus na korisničku interakciju i pažljivo dizajnirana baza podataka čine ovu aplikaciju vrijednim alatom za sve ljubitelje kuhanja. Ova aplikacija poboljšava iskustvo dijeljenja i otkrivanja recepata te potiče stvaranje i jačanje kulinarske zajednice.

# Literatura

[1] VS code, „Visual Studio Code documentation“, https://code.visualstudio.com/docs (posjećeno 05.07.2024.).

[4] PostgreSQL, „PostgreSQL documentation“, https://www.postgresql.org/docs/ (posjećeno 05.07.2024.).

[3] Java spring boot, „Java spring boot documentation“, https://docs.spring.io/spring-boot/documentation.html (posjećeno 07.07.2024.).

[4] NextJS , „NextJS documentation“, https://nextjs.org/docs (posjećeno 07.07.2024.).