

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 448

Mobilna aplikacija za upravljanje receptima

Marko Tunjić

Zagreb, lipanj 2022.

SADRŽAJ

| | |
|--|-----------|
| 1. Uvod | 1 |
| 2. Zahtjevi | 3 |
| 2.1. Korisnički zahtjevi | 3 |
| 2.2. Funkcionalni zahtjevi | 3 |
| 2.3. Nefunkcionalni zahtjevi | 4 |
| 2.4. Zahtjevi domene primjene | 4 |
| 3. Baza podataka | 5 |
| 3.1. Entiteti | 5 |
| 3.2. Relacije | 5 |
| 4. Oblikovanje rješenja | 8 |
| 5. Arhitektura | 13 |
| 5.1. Poslužiteljska strana | 13 |
| 5.1.1. GraphQL | 13 |
| 5.1.2. Autentifikacija i autorizacija | 16 |
| 5.1.3. Pristup podacima | 20 |
| 5.1.4. Kontinuirana isporuka | 22 |
| 5.2. Korisnička strana | 23 |
| 5.2.1. Flutter | 24 |
| 5.2.2. Ekrani | 26 |
| 6. Upute za instalaciju i pokretanje rješenja | 30 |
| 7. Zaključak | 31 |
| Literatura | 32 |

1. Uvod

Internet i web stranice su već dugo vremena dio ljudske svakodnevice no zbog današnje potrebe za brzinom i efikasnošću sve popularnije su mobilne aplikacije. Naime one su vrlo jednostavne za koristiti jer je potrebno samo izvaditi mobitel kliknuti na aplikaciju i sve je spremno. Upravo zato danas puno web aplikacija također ima i svoju mobilnu inačicu.

Problem kod mobilnih uređaja je što jednom napisani programski kod u jezicima kao na primjer: java, kotlin ili swift neće biti moguće pokretati na različitim operacijskim sustavima. Tako se do danas pojavilo nekolicina jezika i radnih okvira za pisanje programskog koda koji će se moći pokretati na većini operacijskih sustava, a neki od poznatijih su flutter i react native. Takav pristup izradi mobilnih aplikacija se naziva nativni (*eng. native*) pristup.

Unatoč tranziciji sa preglednika na mobilne uređaje nije nestala potreba za poslužiteljskim dijelom i bazom podataka, jer takva arhitektura omogućava komunikaciju i dijeljenje sadržaja među ljudima širom svijeta. Pa se tako na poslužiteljskoj strani i dalje koriste Spring, ASP.NET i slični okviri, a za bazu podataka upravitelji po izboru na primjer PostgreSQL, MSSQL, MySQL... Stoga je još uvijek popularan oblikovni obrazac model-pogled-upravljač (*eng. Model-View-Controller*) koji propisuje oblikovanje arhitekture na način da: model ima ulogu spremanja podataka, pogled ulogu prikaza podataka, a upravljač ulogu dohvata podataka.

Radi jednostavnosti se mobilne aplikacije ne koriste starijim protokolima kao na primer SOAP i XML formatom podatka nego se koriste novijim načinom prijenosa podataka između poslužiteljskog dijela aplikacije i korisničkog dijela aplikacije. To je JSON format podataka s odgovarajućim JSON zahtjevima prema poslužitelju. Uz JSON se koristi neki od arhitekturnih stilova kao REST ili GraphQL koji su alternativa za SOAP protokol i s njime mogu i web i mobilne aplikacije uspješno i jednostavno

uspostavljati komunikaciju.

Zbog potrebe da poslužiteljska strana i baza podataka moraju biti uvijek dostupni pojavio se "*oblak*" i usluge u oblaku (*eng. Cloud, Cloud Services*). Uz pomoć tih usluga vrlo je lako pokrenuti instancu potrebne baze i poslužitelja u "*oblaku*" koje će biti globalno i neprestano dostupane. Kao pružatelji takvih usluga ističu se: AWS, Heroku, Azure i tako dalje.

Cilj ovoga rada bio je izraditi mobilnu aplikaciju koja će korisnicima olakšati: pristup, spremanje, dijeljenje i pretraživanje recepata. Sadržaj je podijeljen u 4 dijela u kojima će se opisati zahtjevi, način izrade svakog sloja aplikacije (baza podataka, poslužiteljski i korisnički dio), korištene tehnologije i konačni proizvod.

2. Zahtjevi

U ovom poglavlju biti će opisani svi zahtjevi koje konačni proizvod mora ispunjavati. Zahtjevi su podijeljeni na korisničke koji opisuju mogućnosti korisnika, funkcionalne koji opisuju usluge koje sustav mora pružati, nefunkcionalne koji definiraju ograničenja na funkcionalne zahtjeve, i zahtjeve domene primjene koji proizlaze iz načina primjene sustava.

2.1. Korisnički zahtjevi

1. **Korisnici aplikacije:** postoje 3 vrste korisnika: anonimni, prijavljeni i administratori, a ovisno o vrsti korisnika kojoj pripada, svaki korisnik ima različite ovlasti i uloge.
2. **Anonimni korisnici:** mogu samo pregledavati i pretraživati postojeće recepte.
3. **Prijavljeni korisnici:** uz funkcionalnosti anonimnih korisnika, prijavljeni korisnici mogu također i dodavati vlastite recepte, komentirati, ocjenjivati ocjenama od 0 do 5 i dodavati željene recepte u listu omiljenih recepata.
4. **Administratori:** njihova uloga je održavanje aplikacije primjerenom, zato će recepti prije negoli postanu javno vidljivi morati biti odobreni od strane administratora.

2.2. Funkcionalni zahtjevi

1. **Registracija:** korisnik će pri registraciji morati predati neku vlastitu postojeću e-mail adresu na koju će primati liste za kupnju od recepata koje dodaju u svoje favorite.
2. **Lista za kupnju:** prilikom dodavanja recepta u favorite korisnik na vlastiti e-mail dobije poruku s listom potrebnih sastojaka koja može poslužiti kao lista za

kupnju.

3. **Pretraživanje recepata:** po nazivu i/ili po sastojcima pri čemu se pretraga po sastojcima može obaviti po sastojcima koje recept ne smije sadržavati (što je bitno zbog alergija) ili po sastojcima koje korisnik ima pri ruci.
4. **Upravljanje receptima:** ako korisnici smatraju da su objavili komentar ili recept koji nisu htjeli objaviti moći će ih obrisati. Također ako smatraju da je neki komentar na njihovom receptu neprikladan i njega će moći obrisati.
5. **Upravljanje ponašanjem korisnika:** ako administratori smatraju da je bilo koji recept ili komentar potpuno neprikladan mogu ga obrisati.

2.3. Nefunkcionalni zahtjevi

1. **Podaci koji moraju biti priloženi prilikom izrade recepta:** naziv, opis, koraci pripreme, sastojci. Također postoji mogućnost da se priloži više fotografija i/ili videozapisa koji pobliže opisuju recept i pomažu korisnicima pri kuhanju.
2. **Veličina podataka:** duljina raznih podataka koje korisnik šalje će biti određena dizajnom baze podataka. Veličina fotografija i videozapisa će biti određena HTTP poslužiteljem.
3. **Ekran dobrodošlice:** iz estetskih razloga je potrebno napraviti ekran dobrodošlice
4. **Ekrani za pregled favorita:** taj ekran je koristan jer olakšava pronađak svih favorita prijavljenim korisnicima.
5. **Ekran za pregled neodobrenih recepata:** taj ekran je koristan jer olakšava pronađak svih neodobrenih recepata administratoru koji je trenutno aktivan.

2.4. Zahtjevi domene primjene

1. **Upravljanje korisnicma:** neki od korisnika često imaju namjeru upropoštavanja aplikacije zlouporabom vlastitih ovlasti zato će administratori imati mogućnost suspendiranja takvih korisnika.

3. Baza podataka

Kao upravitelj bazom podataka korišten je PostgreSQL i u svrhu lakšeg razvoja također je korištena aplikacija sa korisničkim sučeljem za izvođenje upita: PgAdmin. Sama baza je relacijskog tipa i modelirana je entitetima i vezama između njih. Za modeliranje je korišten web alat ERDPlus. Za upogonjavanje baze je korišten AWS (*Amazon Web Services*) točnije njihov *RDS (Relational Database Service)* servis koji se koristi upravo za upogonjavanje baza podataka u oblaku.

3.1. Entiteti

Entiteti su modelirani prema zahtjevima zadatka, a veze su modelirane prema stvarnim situacijama. Postojeći entiteti su: recept, sastojak, korak pripreme, korisnik, uloga, videozapis i fotografija. Dakle postoji glavni entitet koji je središte aplikacije, a to je recept njega opisuju njegov identifikacijski broj, ime, opis, trajanje pripreme te na kraju parametar koji pokazuje je li recept odobren od strane administratora ili nije. Iz zahtjeva da svaki recept mora sadržavati korake pripreme i sastojke izrađena su dva entiteta koji upravo njih modeliraju. Također postoje entiteti za fotografije i videozapise jer svaki recept može imati više videozapisa ili fotografija koji će uređivati korisničko sučelje i pomoći korisnicima pri odabiru prikladnog recepta. Korisnici su modelirani kao jedan od entiteta jer im je omogućena interakcija sa receptima na razne načine, a te interakcije su opisane vezama kao što su komentiranje, dodavanje u favorite, ocjenjivanje... Slika modela koji je izgrađen iz prethodnog opisa (*eng. Entity relationship diagram*) je prikazana na slici 3.1

3.2. Relacije

Iz prethodno opisanog modela pretvorbom iz entiteta u relacijsku shemu, što omogućava ERDPlus alat, dobiveno je 10 tablica. One odgovaraju entitetima i vezama između njih, tako se pojavljuju tablice koje ne predstavljaju ni jedan entitet nego veze

a to su: komentari, favoriti i ocjene. Da bi tablice bile dostupne aplikaciji pa tako i svim korisnicima bilo je potrebno upogoniti bazu u oblaku i dodati tablice odgovarajućim naredbama koje su vidljive u odlomku 3.2. Relacijska shema je prikaza slikom 3.2.

SQL naredbe za stvaranje tablica

```

CREATE TABLE role
(
    id SERIAL,
    role_name VARCHAR(20) NOT NULL,
    UNIQUE(role_name),
    PRIMARY KEY (id)
);

CREATE TABLE users
(
    e_mail VARCHAR(100) NOT NULL,
    password CHAR(60) NOT NULL,
    username VARCHAR(50) NOT NULL,
    id SERIAL,
    profile_picture VARCHAR(500) NOT NULL,
    is_banned BOOLEAN NOT NULL,
    role_id INT NOT NULL,
    is_confirmed BOOLEAN NOT NULL,
    confirmation_code char(30) NOT NULL,
    UNIQUE(username,e_mail),
    PRIMARY KEY (id),
    FOREIGN KEY (role_id) REFERENCES Role(id)
);

CREATE TABLE recipe
(
    cover_picture VARCHAR(500) NOT NULL,
    recipe_name VARCHAR(50) NOT NULL,
    description VARCHAR(500) NOT NULL,
    id SERIAL,
    is_approved BOOLEAN NOT NULL,
    cooking_duration INT NOT NULL,
    user_id INT NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (user_id) REFERENCES Users(id)
);

CREATE TABLE recipe_step
(
    id SERIAL,
    step_description VARCHAR(500) NOT NULL,
    order_number INT NOT NULL,
    recipe_id INT NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (recipe_id) REFERENCES Recipe(id)
);

CREATE TABLE image
(
    id SERIAL,
    order_number INT NOT NULL,
    link VARCHAR(500) NOT NULL,
    recipe_id INT NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (recipe_id) REFERENCES Recipe(id)
);

CREATE TABLE ingredient
(
    id SERIAL,
    ingredient_name VARCHAR(50) NOT NULL,
    quantity INT NOT NULL,
    measure VARCHAR(50) NOT NULL,
    recipe_id INT NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (recipe_id) REFERENCES Recipe(id)
);

CREATE TABLE video
(
    id SERIAL,
    order_number INT NOT NULL,
    link VARCHAR(500) NOT NULL,
    recipe_id INT NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (recipe_id) REFERENCES Recipe(id)
);

CREATE TABLE favorite
(
    user_id INT NOT NULL,
    recipe_id INT NOT NULL,
    PRIMARY KEY (user_id, recipe_id),
    FOREIGN KEY (user_id) REFERENCES Users(id),
    FOREIGN KEY (recipe_id) REFERENCES Recipe(id)
);

CREATE TABLE rating
(
    rating_value INT NOT NULL,
    user_id INT NOT NULL,
    recipe_id INT NOT NULL,
    PRIMARY KEY (user_id, recipe_id),
    FOREIGN KEY (user_id) REFERENCES Users(id),
    FOREIGN KEY (recipe_id) REFERENCES Recipe(id)
);

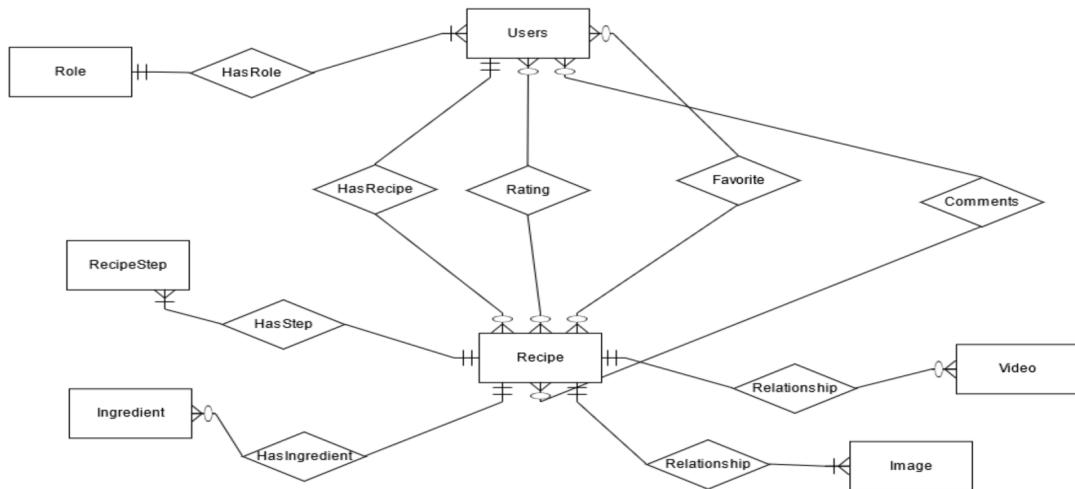
CREATE TABLE comments
(
    id SERIAL,
    user_id INT NOT NULL,
    recipe_id INT NOT NULL,
    comment_text TEXT NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (user_id) REFERENCES Users(id),
    FOREIGN KEY (recipe_id) REFERENCES Recipe(id)
);

```

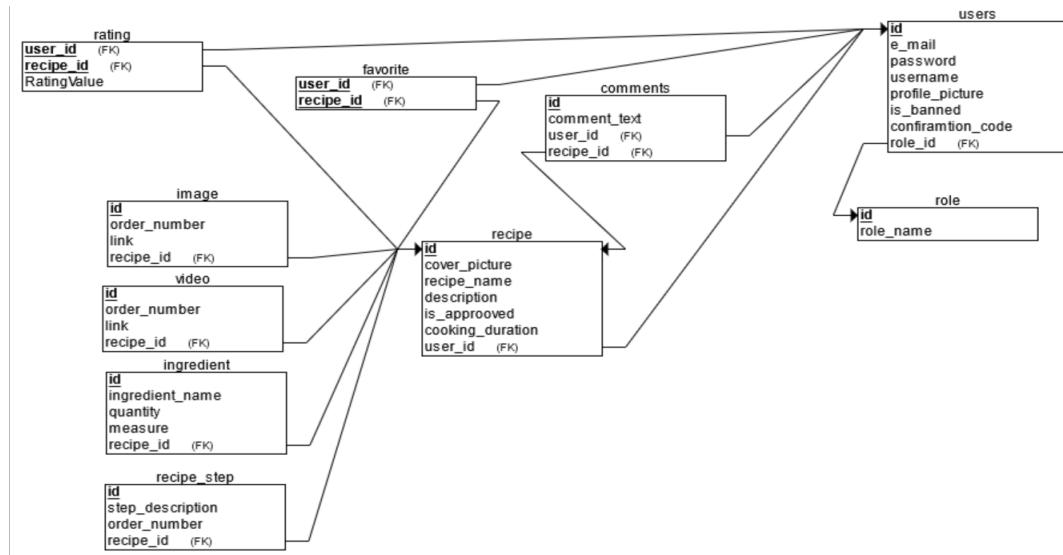
```

id SERIAL,
comment_text VARCHAR(200) NOT NULL,
user_id INT NOT NULL,
recipe_id INT NOT NULL,
PRIMARY KEY (id),
FOREIGN KEY (user_id) REFERENCES Users(id),
FOREIGN KEY (recipe_id) REFERENCES Recipe(id)
);

```



Slika 3.1: ERDiagram



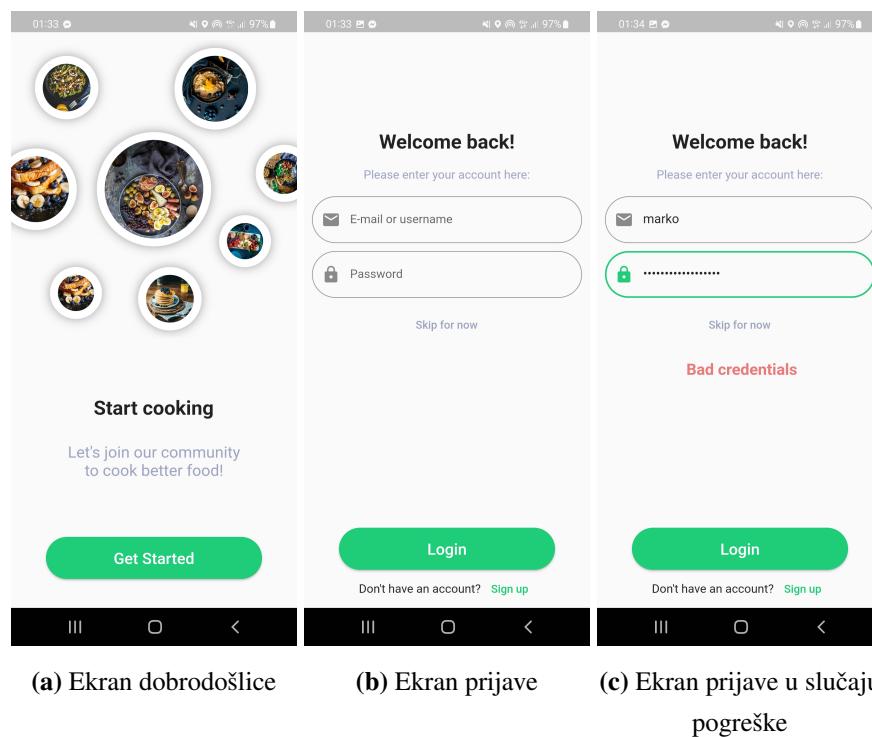
Slika 3.2: Relacijska shema

4. Oblikovanje rješenja

Za ispunjavanje svih zahtjeva potrebno je 9 ekrana. Svaki ekran pristupa različitoj usluzi na poslužiteljskoj strani gdje će se provjeravati smije li trenutni korisnik pristupati tim uslugama.

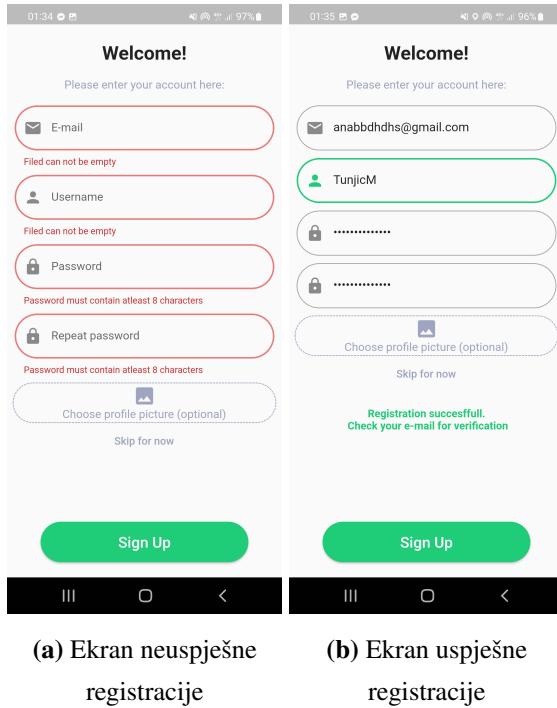
Prvi ekran je ekran dobrodošlice koji spada u nefunkcionalne zahtjeve i dodan je kao estetski detalj te ne šalje nikakve upite (slika 4.1 (a)).

Drugi ekran namijenjen je za prijavu u aplikaciju te pristupa usluzi za prijavu u sustav na poslužitelju (slika 4.1 (a) i (b)).



Slika 4.1: Ekran dobrodošlice i prijave

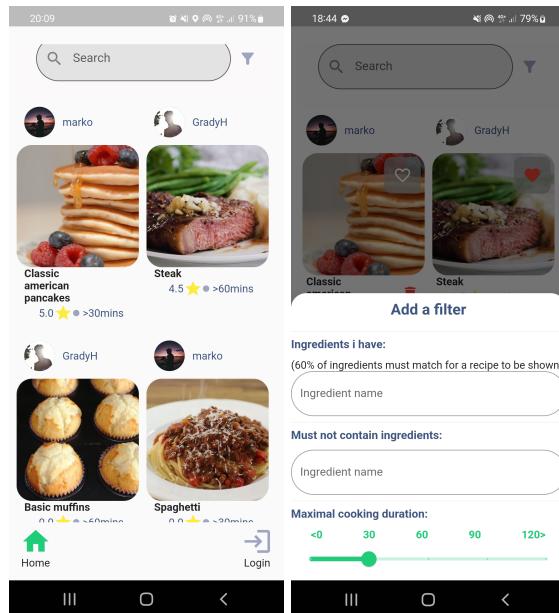
Treći ekran služi za registraciju u sustav. Na njemu će korisnik unositi zahtijevane podatke i poslati ih poslužitelju. Poslužitelj validira podatke i odgovara porukom koja može biti potvrđna i zahtijevati od korisnika potvrdu identiteta putem elektroničke pošte ili može biti poruka pogreške koja zahtijeva promjenu unesenih podataka (slika 4.2).



Slika 4.2: Ekrani za registraciju

Na četvrtom ekranu se nalaze svi recepti koji postoje u bazi podataka i elementi koji služe za filtriranje recepata i dodavanje recepata u favorite. Ovaj ekran od poslužitelja zahtijeva sve recepte koje zadovoljavaju odabrani filter. Taj filter može biti prazan u slučaju zahtijevanja svih recepata ili popunjen zahtijevanim nazivom i/ili sastojcima koji smiju odnosno ne smiju biti sadržani u receptu (slika 4.3).

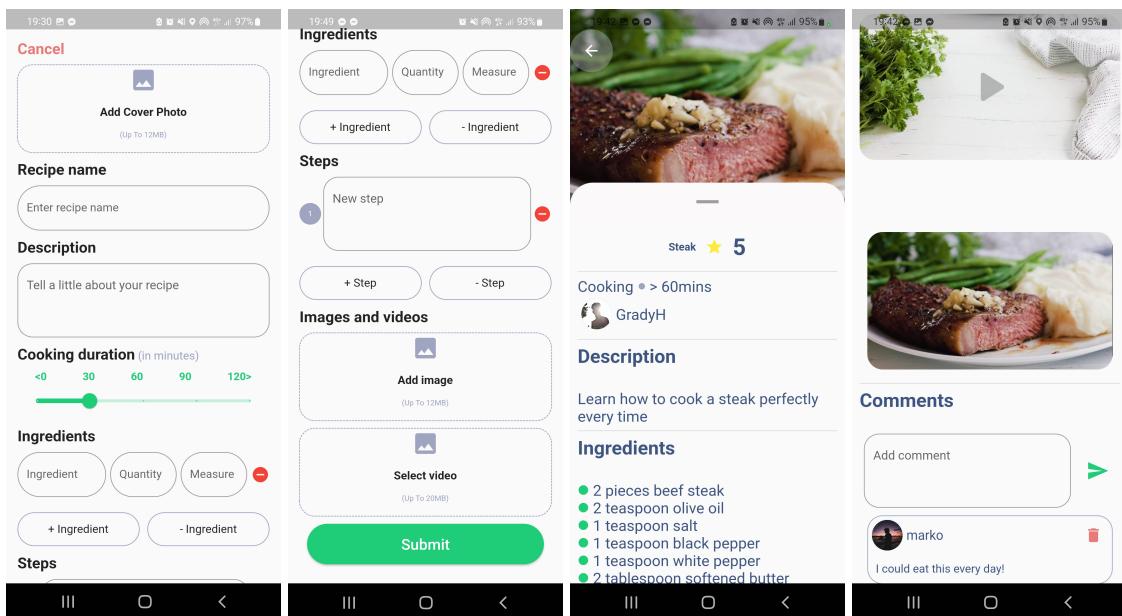
Peti ekran je zapravo formular za dodavanje novog recepta na kojem postoje elementi za dodavanje obaveznih podataka o receptu kao na primjer naziv, opis, koraci i slično. Također postoje elementi za odabir fotografija i/ili videozapisa. Ti podaci će isto kao i za registraciju biti poslani poslužitelju koji ih validira i odgovara porukom o uspješnosti dodavanja recepta (slika 4.4 (a) i (b)).



(a) Ekran sa svim receptima
(b) Dijalog za filtriranje

Slika 4.3: Ekran sa receptima

Na sljedećem ekranu se nalaze svi podaci o pojedinačnom receptu. Do tog ekrana se dolazi klikom na recept sa četvrtog ekrana. Uz sve podatke o receptu mora postojati i prostor za ostavljanje komentara i ocjene (slika 4.4 (a) i (b)). Kada je administrator pristupio receptu mora postojati gumb za odobravanje to jest poništavanje odobrenja recepta (slika 4.5 (b)). Da bi se omogućile te funkcionalnosti mora se pristupati uslugama: dohvaćanja odabranog recepta, ostavljanju komentara, ocjenjivanja i odobravanja recepta na poslužitelju.

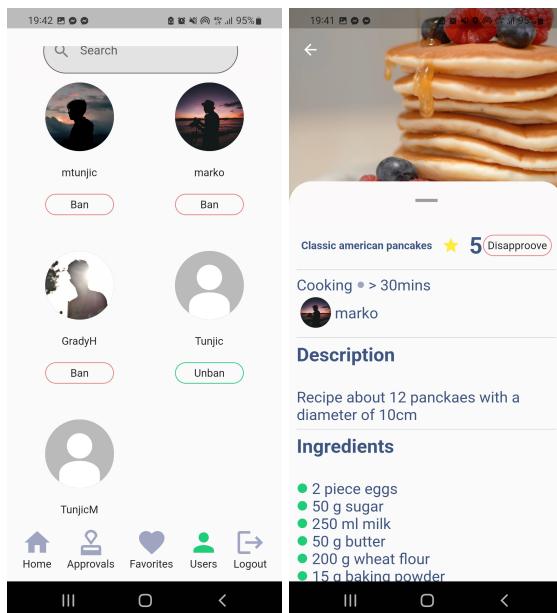


(a) Prvi dio ekrana za dodavanje recepta (b) Drugi dio ekrana za dodavanje recepta (c) Prvi dio ekrana za detalje o receptu (d) Drugi dio ekrana za detalje o receptu

Slika 4.4: Ekran za dodavanje recepta i pregled recepta

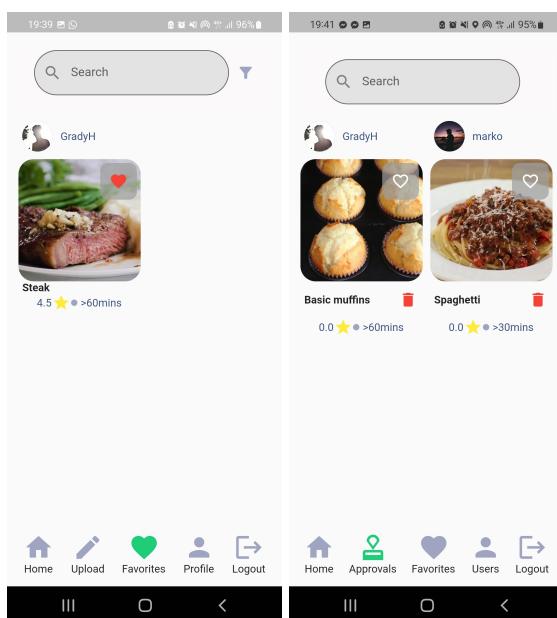
Sedmi ekran je specifičan samo za administratore i na njemu se nalaze svi registrirani korisnici i elementi koji će omogućiti slanje zahtijva za suspendiranje korisnika (slika 4.5 (a)).

Osmi i deveti ekrani su jednaki kao i četvrti ekran. Razlikuju se samo u receptima koji su prikazani. Na osmom ekranu moraju biti prikazani samo recepti koji se nalaze u favoritima za trenutno prijavljenog korisnika, a na devetom samo neodbreni recepti (slika 4.6). To se postiže postavljanjem odgovarajućeg atributa u filter i slanjem zahtjeva na poslužitelj.



(a) Ekran za upravljanje korisnicima (b) Ekran za odobravanje recepta

Slika 4.5: Ekrani za administratore



(a) Ekran sa favoritima (b) Ekran sa neodobrenim receptima

Slika 4.6: Ekran sa favoritima i ekran sa neodobrenim receptima

5. Arhitektura

U ovom poglavlju je opisana arhitektura, implementacija i korištene tehnologije na poslužiteljskoj i korisničkoj strani aplikacije.

5.1. Poslužiteljska strana

Poslužiteljska strana je pisana u programskom jeziku Java verzije 11 jer je najveća podržana verzija u odabranoj usluzi oblaka upravo java 11. Korišteni radni okvir je SpringBoot koji je namijenjen upravo za izradu poslužiteljske strane web i mobilnih aplikacija. Kao arhitekturni stil korišten je GraphQL koji predstavlja upitni jezik koji je specifično dizajniran za korisničku stranu aplikacije. Tako GraphQL omogućava korisničkoj strani da od poslužiteljske strane zatraži i dobije samo one podatke koji su potrebni. Projekt je strukturiran kao maven projekt koji omogućava dohvati i korištenje biblioteka koje nisu dio standardnog javinog paketa. Za verzioniranje projekta je korišten alat *git* koji nudi razne mogućnosti, ali jedna od bitnijih je da je vrlo lako vratiti projekt na staru verziju ukoliko se dogodi veća pogreška u pisanju programskog koda. Za upogonjavanje se kao i za bazu podataka koristi AWS, ali drugi servis: *ElasticBeanstalk* servis. U svrhe razvoja korišteno je okruženje VSCode.

5.1.1. GraphQL

Povijest

Jedan od većih problema dizajniranja web poslužitelja pa tako i samih web aplikacija u REST arhitekturnom stilu je problem prekomjernog dohvata podataka odnosno nedovoljnog dohvata podataka (*eng. overfetching and underfetching*). Ako bi se taj problem pokušao riješiti sa stvaranjem više krajinjih točaka (*eng. endpoint*) za dohvat specifičnog skupa podataka, a uz to i više prijenosnih objekata. Uloga takvih objekata je spremanje i slanje samo onih podataka koji su potrebni za ispunjavanje zahtjeva.

Tako je tvrtka Facebook došla do ideje GraphQL-a. Naime razvojni programeri su tada radili s velikim brojem podataka koji su bili međusobno ugnježđenji i povezani na razne načine i da bi aplikacija radila dovoljno brzo i efikasno bilo je nužno dohvatiti samo one podatke koji se uistinu i koriste, a da se pritom dohvaćaju sa što manje zahtjeva.

Definicija

GraphQL je upitni jezik koji korisničkoj strani dopušta definiranje formata i oblika podataka koji će zahtijevati od poslužiteljske strane. Također omogućava da se upiti šalju na samo jednu krajnju točku. U GraphQL-u postoje 2 vrste upita: upiti i mutacije (*eng. query and mutation*), razlika je u tome što upiti služe za dohvaćanje podataka, a mutacije za mijenjanje podataka.

Implementacija

Unatoč tome što GraphQL koji je u usporedbi sa REST-om dosta različit, implementacija nije otežana. U okviru ovoga rada u *pom.xml* datoteku u kojoj se dodaju ovisnosti (*eng. dependencies*) bilo je potrebno dodati 2 nove ovisnosti: *graphql-spring-boot-starter*, *graphql-java-tools* i po potrebi treću ovisnost koja nudi mogućnost testiranja sučelja, a ona je *graphiql-spring-boot-starter* (odlomak 5.1.1). Nakon toga ove ovisnosti zahtijevaju izradu jedne datoteke u kojoj će pisati sve moguće vrste podataka koje se mogu tražiti, i sve moguće upite koji se mogu primiti. Takva datoteka se najčešće imenuje *schema.graphqls*. U ovom radu podaci u shemi su bili ujedno i relacijski modeli iz baze podataka. Upiti postoje razni i bili su dodavani jedan po jedan ovisno o potrebi aplikacije (odlomak 5.1.1). U odlomku koji prikazuje *graphql* upite (5.1.1) postoji velik broj mutacija i upita, ali to i dalje ne mijenja činjenicu da postoji samo jedna krajnja točka koja je u ovom slučaju */graphql*. Iz priloženog primjera jednog GraphQL tipa podatka u odlomku 5.1.1 je moguće primjetiti da je tip podatka upravo jedan od relacijskih modela iz baze podataka. Razlika je u tome da postoje neki dodatni podaci koji rješavaju problem nedovoljnog dohvata podataka. Bitan detalj je da se imena i tipovi polja iz Javine klase i tipa podatka iz *graphql* sheme moraju podudarati jer se koristi refleksija za dohvat tih polja.

GraphQL ovisnosti

```
<dependency>
    <groupId>com.graphql-java</groupId>
    <artifactId>graphql-spring-boot-starter</artifactId>
    <version>5.0.2</version>
</dependency>
<dependency>
    <groupId>com.graphql-java</groupId>
    <artifactId>graphql-java-tools</artifactId>
    <version>5.2.4</version>
</dependency>
<dependency>
    <groupId>com.graphql-java</groupId>
    <artifactId>graphiql-spring-boot-starter</artifactId>
    <version>5.0.2</version>
</dependency>
```

GraphQL upiti

```
schema{
    query: Query
    mutation: Mutation
}

type Query{
    recipes(filter: Filter!): Recipes!
    singleRecipe(recipeId: ID!): Recipe!
    userForId(userId: ID!): Users!
    users(filter: Filter!): UsersResponse!
    notApprovedRecipes(filter: Filter!): Recipes!
    favorites(userId: ID!, filter: Filter!): Recipes!
}

type Mutation{
    login(identifier: String!, password: String!): LoginResponse!
    register(payload: RegisterRequest!): Users!
    addRecipe(payload: RecipePayload!): Recipe!
    editFavorite(userId: ID!, recipeId: ID!, state: Boolean!): Boolean!
    deleteRecipe(recipeId: ID!): Boolean!
    addComment(userId: ID!, recipeId: ID!, commentText: String!): Comments
    addRating(userId: ID!, recipeId: ID!, ratingValue: Int!): Boolean!
    deleteComment(commentId: ID!): Boolean!
    changeBanStatus(userId: ID!, banStatus: Boolean!): Boolean!
    changeApprovedStatus(recipeId: ID!, isApproved: Boolean!): Boolean!
}
```

GraphQL tip podatka

```
type Recipe{
    id: ID!
    coverPicture: String!
    recipeName: String!
    description: String!
    isApproved: Boolean!
```

```

        cookingDuration: Int!
        user: Users!
        recipeSteps: [RecipeStep] @relation
        ingredients: [Ingredient] @relation
        images: [Image] @relation
        videos: [Video] @relation
        ratings: [Rating] @relation
        comments: [Comments] @relation
        favoriteTo: [Favorite] @relation
        averageRating: Float
        isLikedByCurrentUser: Boolean
        ratingFromCurrentUser: Int!
    }
}

```

Nakon dodavanja sheme i ovisnosti potrebno je dodati razrješivače (eng. *resolvere*). U ovom projektu postoje 3 vrste razrješivača: QueryResolver, MutationResolver i obični Resolver, oni su u prethodno navedenim ovisnostima definirani kao sučelja: GraphQLQueryResolver, GraphQLMutationResolver, GraphQLResolver. Prvi služi za prihvatanje upita, drugi služi za prihvatanje mutacija, a treći služi za rješavanje problema prethodno spomenutih dodatnih parametara koji su umetnuti u GraphQL shemu. To se radi na način da se zasebno definira način dohvata tih podataka. Također se može implementirati sučelje GraphQLErrorHandler koji će presretati greške s poslužiteljske strane i vraćati odgovarajuću poruku korisničkoj strani. Naime GraphQL ima jednu manu, a ta mana je da se na korisničku stranu uvijek vraća HTTP status kod 200 OK, pa se rukovanje greškama obavlja na drukčiji način. Taj način je da se u odgovor u slučaju greške doda parametar *errors* koji će sadržavati sve greške koje su se dogodile (slika 5.1). Ako se ne implementira sučelje GraphQLErrorHandler poslužitelj će uvjek vraćati predefiniranu pogrešku.

Za kraj nakon implementacije svih navedenih sučelja (slika 5.2) sada se uspješno može izvršiti prvi upit koji je demonstriran kroz sučelje *GraphiQL* i prikazuje strukturu upita i odgovora (slika 5.3).

5.1.2. Autentifikacija i autorizacija

JWT

U području autentifikacije i autorizacije se vidi jedna velika razlika između moderne izrade mobilne i web aplikcije, a to je odsutnost sesije. Jedna od namjena sesije je identificiranje trenutno prijavljenog korisnika na temelju nekog atributa koji je jedinstven za tog korisnika. Problem kod implementacije sesija je uporaba kolačića koji se

```

1 v query Recipes($filter: Filter!){
2   recipes(filter: $filter)
3 v   {
4 v     recipes{
5 v       user{
6 v         id,
7 v         username,
8 v         profilePicture
9 v       }
10 v      id,
11 v      averageRating,
12 v      cookingDuration,
13 v      coverPicture,
14 v      recipeName,
15 v      isLikedByCurrentUser
16 v    }
17 v    number_of_pages,
18 v    current_index
19 v  }
20 }

QUERY VARIABLES
1 v {
2   "filter": {
3     "index": -1
4   }
5 }

```

```

{
  "data": null,
  "errors": [
    {
      "message": "Exception while fetching data (/recipes) : Page index can not be less than 0!",
      "path": [
        "recipes"
      ],
      "exception": {
        "cause": null,
        "stackTrace": [
          {
            "classLoaderName": null,
            "moduleName": null,
            "moduleVersion": null,
            "methodName": "getRecipesForFilter",
            "fileName": "RecipeService.java",
            "lineNumber": 175,
            "className": "hr.fer.zpr.marko_tunjic.zavrsni_rad.services.RecipeService",
            "nativeMethod": false
          },
          {
            "classLoaderName": null,
            "moduleName": null,
            "moduleVersion": null,
            "methodName": "invoke",
            "fileName": "<generated>",
            "lineNumber": -1,
            "className": "hr.fer.zpr.marko_tunjic.zavrsni_rad.services.RecipeService$$FastClassBySpringCGLIB$5$c5f5b4ed",
            "nativeMethod": false
          }
        ]
      }
    }
  ]
}

```

Slika 5.1: Primjer greške ako postoji GraphQLErrorHandler



Slika 5.2: Struktura GraphQL paketa

ne koriste u mobilnim aplikacijama. Zato se mobilne aplikacije koriste drugim mehanizmom, a to su tokeni. U ovoj aplikaciji se koriste tokeni pod imenom *JWT (JSON Web Token)*, takvi tokeni u sebi sadrže podatke koji se koriste za uspostavu autentifikacije korisnika. Da bi se ti podaci mogli sigurno prenositi između poslužiteljske i korisničke strane koriste se kriptografski algoritmi (kao na primjer HMACSHA256) za enkripciju tokena. Pa će se u takvom tokenu nalaziti korisničko ime, istek tokena, korišteni algoritmi i slični podaci koji će omogućiti autentifikaciju i autorizaciju korisnika aplikacije.

Autentifikacija

Da bi se ovo ostvarilo prvo je bilo potrebno dodati ovisnost prema SpringSecurity-u i prema jjwt (odlomak 5.1.2). Nakon toga da bi aplikacija mogla doći do podataka trebalo je implementirati dva sučelja: *UserDetails* i *UserDetailsService*.

The screenshot shows a GraphQL query in the left panel and its JSON response in the right panel.

```

query Recipes($filter: Filter!){
  recipes(filter: $filter)
  {
    recipes{
      user{
        id,
        username,
        profilePicture
      }
      id,
      averageRating,
      cookingDuration,
      coverPicture,
      recipeName,
      isLikedByCurrentUser
    }
    number_of_pages,
    current_index
  }
}

```

QUERY VARIABLES

```

{
  "filter": {
    "index": 1
  }
}

```

RESULTS

```

{
  "data": {
    "recipes": [
      {
        "user": {
          "id": "51",
          "username": "marko",
          "profilePicture": "https://firebasestorage.googleapis.com/v0/b/finalbscthesis.appspot.com/o/marko_profilePicture.png?alt=media"
        },
        "id": "52",
        "averageRating": 5,
        "cookingDuration": 30,
        "coverPicture": "https://firebasestorage.googleapis.com/o/recipeImage_1815604001941510692.png?alt=media",
        "recipeName": "Classic american pancakes",
        "isLikedByCurrentUser": false
      },
      {
        "user": {
          "id": "74",
          "username": "GradyH",
          "profilePicture": "https://firebasestorage.googleapis.com/v0/b/finalbscthesis.appspot.com/o/GradyH_profilePicture.png?alt=media"
        },
        "id": "75",
        "averageRating": 4.5,
        "cookingDuration": 60,
        "coverPicture": "https://firebasestorage.googleapis.com/o/recipeImage_1832260928432270555.png?alt=media",
        "recipeName": "Steak",
        "isLikedByCurrentUser": false
      }
    ]
  }
}

```

Slika 5.3: Primjer ispravnog upita

vice. Prvo sučelje služi za omotavanje osnovnih podataka o korisniku, kao na primjer korisničko ime, lozinka, je li korisnik suspendiran, je li korisnik potvrdio svoj identitet, koje su korisnikove uloge (administrator ili korisnik) i slično, a drugo sučelje služi za dohvati tih podataka na osnovu korisničkog imena jer je to jedinstven atribut za svakog korisnika. U ovoj aplikaciji sučelje *UserDetails* omotava podatke iz baze podataka, a sučelje *UserDetailsService* dohvaca podatke iz baze i spremi ih u prvo sučelje.

Ovisnosti prema bibliotekama potrebnih za implementaciju autentifikacije i autorizacije

```

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt</artifactId>
  <version>0.9.1</version>
</dependency>

```

Nadalje da bi se ti podaci mogli dohvatiti u bilo kojem trenutku potrebno ih je spremi "negdje" u "nekom" trenutku. Zato je bilo potrebno napraviti jedan *Filter*. *Filteri* su posebni objekti u web aplikaciji koji presreću bilo koji zahtjev koji stigne na poslužitelj, obrade taj zahtjev i onda ga proslijede dalje prema web aplikaciji. Filter za autentifikaciju će provjeriti postoji li HTTP zaglavje pod imenom: *Authorization* i ako postoji pretpostaviti će da se unutra nalazi token kojeg će pokušati isparsirati i iz

njega dohvatiti korisničko ime. Na osnovu tog korisničkog imena i uz pomoć sučelja *UserDetailsService* vrlo jednostavno se dohvati sučelje *UserDetails*. Da bi podaci bili dostupni cijelom poslužiteljskom dijelu aplikacije, filter će spremiti bitne podatke u klasu koja se zove *SecurityContextHolder* koja će pružiti mogućnost dohvata trenutnog korisnika u bilo kojem trenutku.

Jedan od mogućih scenarija je da će podaci poslani u tokenu biti ili nevažeći ili isteknuti zato je potrebno napraviti ulaznu točku koja će (eng. *EntryPoint*) u slučaju pogrešne autorizacije korisniku vratiti odgovarajuću poruku. Ta ulazna točka se u implementaciji naziva *AuthenticationEntryPoint* koja putem HTTP odgovora šalje pogrešku u svojoj metodi *commence*. Struktura svih implementiranih sučelja u projektu prikazana je na slici 5.4



Slika 5.4: Struktura paketa za autentifikaciju i autorizaciju

Autorizacija

Na kraju je potrebno definirati autorizaciju i spojiti s autentifikacijom, a to se obavlja tako da se proširi klasa *WebSecurityConfigurerAdapter*. U njoj je definirano koja će se ulazna točka koristiti u slučaju pogreške, koji algoritam se koristi za šifriranje lozinki, kada se poziva filter za autentifikaciju i kada se obavlja autorizacija. U ovoj aplikaciji se koristi prethodno definirana ulazna točka, za šifriranje lozinke se koristi *BCrypt* algoritam, filter se poziva prije svakog zahtjeva, a autorizacija je omogućena da se definira za svaku metodu zasebno. To se obavlja uz pomoć anotacije *@PreAuthorize* (slika 5.5). Ona se koristi na način da se u tijelo anotacije stave sve uloge koje smiju pristupati anotiranoj metodi (slika 5.6). One se najčešće stavljaju ili u *QueryResolver-u* ili u *MutationResolver-u* i tako će se ograničiti pristup neautorizirani korisnicima.

```

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

@Override
protected void configure(HttpSecurity http) throws Exception {
    http.cors().and().csrf().disable()
        .exceptionHandling().authenticationEntryPoint(unauthorizedHandler).and()
        .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()
        .authorizeRequests()
        .anyRequest().permitAll();
    http.addFilterBefore(authenticationJwtTokenFilter(), beforeFilter: UsernamePasswordAuthenticationFilter.class);
}

```

Slika 5.5: Izgled sigurnosne konfiguracije

```

@PreAuthorize("hasAuthority('USER') or hasAuthority('MODERATOR') or isAnonymous()")
public Recipes getRecipes(Filter filter) {
    return recipeService.getRecipesForFilter(filter);
}

```

Slika 5.6: Izgled metode s anotacijom @PreAuthorize

5.1.3. Pristup podacima

Za pristup podacima se koriste dva sloja prvi sloj je sloj za perzistenciju, a drugi sloj je sloj usluge. Sloj za perzistenciju modeliran je uz pomoć *ORM-a (Object Relational Mapper)* čija je zadaća preslikavanje relacija iz baze podataka u Javine objekte. Postoji više vrsta različitih ORM-ova, a u ovom projektu se koristi *Hibernate*.

U sloju usluge se nalazi sva logika potrebna za pristup i obradu podatka. Sloj usluge je implementiran da bi se razdvojila odgovornost obrade podataka i dohvata podataka od odgovornosti obrade zahtjeva. Za sve to je bilo potrebno dodati ovisnost prema odgovarajućim *JPA* i *JDBC* ovisnostima (odlomak 5.1.3).



(a) Sve usluge

Slika 5.7: Izgled paketa za sloj usluge

Ovisnosti potrebne za pristupanje podacima

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jdbc</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-mail</artifactId>
</dependency>
<dependency>
    <groupId>com.google.firebaseio</groupId>
    <artifactId>firebase-admin</artifactId>
    <version>7.0.1</version>
</dependency>
```

Sloj za perzistenciju

U programskom jeziku java, *hibernate* se koristi kroz *JPA* (Java Persistence API) programsko sučelje. JPA zahtijeva da se način preslikavanja iz relacije u objekt definira ili anotacijama ili u posebnoj konfiguracijskoj datoteci, u ovom projektu je to učinjeno preko anotacija, a sve takve klase prikazane su na slici 5.8 (a). Anotacije i način anotiranja koji se mora poštivati prikazan je na slici 5.8 (b). U izvornom kodu implementacije vidljivo je da u modelima ovoga projekta nisu anotirane sve veze koje su određene samom bazom podataka jer je to riješeno uz pomoć razrješivača iz *GraphQL-a*. Nakon kreiranja svih modela potrebno je napraviti sučelja koja nasljeđuju sučelje *JPARepository* preko kojeg se pristupa samim podacima, a u tom sučelju su definirane metode kao na primjer metoda za dohvaćanje svih relacija, metoda za dohvaćanje relacija po identifikacijskom atributu, ili nekom drugom atributu i slične metode. Također iz slike 5.8 (b) na kojoj je prikazan model koji preslikava tablicu *image* u javinu klasu, može se primjetiti da se slika u bazu spremi kao hiperveza na samo sliku. To je moguće jer se za spremanje slika koristi *firebase*. Za pristup *firebase-u* je bilo potrebno dodati ovisnost (odломak 5.1.3) i jednu datoteku u kojoj su zapisani podaci za pristup bazi koja iz sigurnosnih razloga nije priložena. Nakon je toga je vrlo lako uz pomoć metode *create* primljenu sliku "dignuti" na *firebase oblak*.

```

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

    @Column(length = 100, nullable = false, unique = false)
    private String link;

    @Column(nullable = false, unique = false)
    private Integer orderNumber;

    @ManyToOne
    @JoinColumn(name = "recipe_id")
    private Recipe recipe;

```

(a) Primjer anotiranja klase

Slika 5.8: JPA konfiguracija

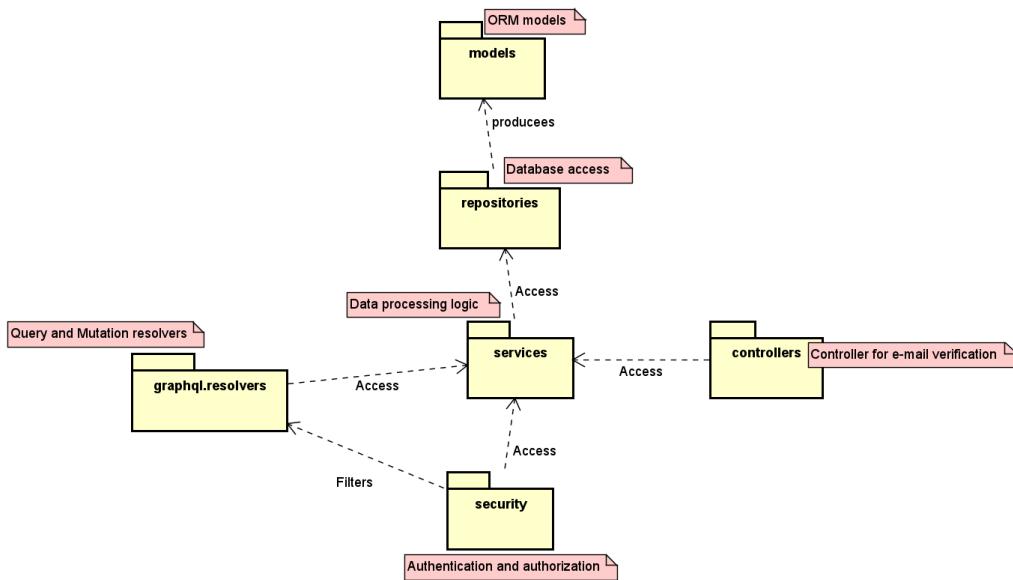
Sloj usluge

Jedno od SOLID načela programiranja ("*Single responsibility*") nalaže da jedna klasa ima samo jednu odgovornost zato je sloj za perzistenciju od razrješivača odvojen slojem usluge.

Također osim same obrade podataka u sloju usluge se nalazi i klasa koja služi za slanje električne pošte. Ta usluga se koristi za slanje poruke za potvrdu identiteta i za slanje liste za kupnju koja se šalje ako neki korisnik doda recept u svoje favorite. Tijelo poruke se gradi kao HTML dokument, a šalje se uz pomoć *Java Mail API-a* za koji je bilo potrebno dodati ovisnost prikazanu u odlomku 5.1.3. I konačno sve usluge su prikazane u odlomku 5.1.3 (b). Prikaz svih paketa (slojeva) i veza između njih prikazana je na slici (5.9)

5.1.4. Kontinuirana isporuka

Kontinuirana isporuka predstavlja proces prevođenja programskog koda i upogonjavaњa na predviđeno mjesto (u ovom slučaju je to "oblak"). Ona se obavlja uz pomoć *github actions*. To je zapravo samo jedan pogamski kod (slika 5.10) uz pomoć kojeg se izvorni kod prevodi u .jar arhivu i podiže na oblak u *AWS ElasticBeanstalk* servis. Jedini problem je bio što nakon upogonjanja poslužitelj može primati relativno male



Slika 5.9: Dijagram paketa

zahtjeve, a za ovu aplikaciju su potrebni poprilično veliki zahtjevi jer se šalju i fotografije i videozapisi. Razlog iz kojeg se fotografije i videozapisi šalju sa korisničke na poslužiteljsku stranu pa tek onda na *firebase* je sigurnosni razlog. Naime krajnji korisnici koji imaju loše namjere mogu poslati maliciozne datoteke umjesto prepostavljenih slika i videozapisa pa je prije prihvatanja datoteke potrebno provjeriti radi li se zaista o slici. Ova funkcionalnost nije implementirana u ovom radu jer postoji mogućnost odobravanja recepta, ali unatoč tome to nije potpuno riješen problem. Dakle da bi se omogućilo slanje fotografija i videozapisa potrebno je uz .jar arhivu dodati i konfiguracijsku datoteku za *nginx* poslužitelj koja će povećati prepostavljenu maksimalnu veličinu zahtjeva. To je postignuto uz pomoć maven-ovog "antrun plugin-a" (slika 5.11) koji će .jar arhivu i konfiguracijsku datoteku spremiti u .zip arhivu i proslijediti to dalje.

5.2. Korisnička strana

Korisnička strana je pisana u programskom jeziku dart, pritom koristeći Flutter radni okvir. Za komunikaciju s poslužiteljskom stranom korišten je paket flutter-graphql. Za upogonjavanje aplikacije korištena je github-ova mogućnost stvaranja izdanja, a sam proces stvaranja je obavljen uz pomoć *github actions* (slika 5.12). U svrhe razvoja korišteno je okruženje VSCode.

```

build_and_deploy_backend:
  name: Build jar and deploy
  runs-on: ubuntu-latest
  steps:
    - name: Checkout source code
      uses: actions/checkout@v2
    - name: Generate deployment package
      working-directory: ./backend/zavrsni_rad
      run: mvn clean package
    - name: Deploy to EB
      uses: einaregilsson/beanstalk-deploy@v2
      with:
        wait_for_environment_recovery: 60
        aws_access_key: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws_secret_key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        application_name: zavrsni_rad
        environment_name: Zavrsnirad-env
        version_label: ${{ github.run_number }}
        region: us-east-1
        deployment_package: backend/zavrsni_rad/target/zavrsni_rad-0.0.1-SNAPSHOT.zip

```

Slika 5.10: Github actions zadatak za kontinuiranu isporuku

```

<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-antrun-plugin</artifactId>
  <executions>
    <execution>
      <id>prepare</id>
      <phase>package</phase>
      <configuration>
        <target>
          <copy todir="${project.build.directory}/${project.build.finalName}"/>
            <fileset dir="/" includes=".ebextensions/**" />
            <fileset dir="${project.build.directory}" includes="*.jar" />
          </copy>
          <copy todir="${project.build.directory}/${project.build.finalName}"/>
            <fileset dir="/" includes=".platform/**" />
            <fileset dir="${project.build.directory}" includes="*.jar" />
          </copy>
          <zip destfile="${project.build.directory}/${project.build.finalName}.zip" basedir="${project.build.directory}/${project.build.finalName}" />
        </target>
      </configuration>
      <goals>
        <goal>run</goal>
      </goals>
    </execution>
  </executions>
</plugin>

```

Slika 5.11: Plugin za stvaranje zip arhive za upogonjavanje

5.2.1. Flutter

Flutter je googleov javno dostupni (*eng. Open Source*) projekt koji je namijenjen za stvaranje lijepih i brzih mobilnih, web ili računalnih. Karakteristika *fluttera* je da se aplikacija sa samo jednom napisanim programskim kodom može pokrenuti na svim navedenim uređajima i na većini operacijskih sustava. Također jedna od prednosti tog radnog okvira je da je vrlo lako napraviti dinamične aplikacije jer se koristi upravljanje stanjem (*eng. state management*). Postoji više načina kako pristupiti ovoj funkcionalnosti, a u ovom radu korištem je *BLoC* obrazac koji je ujedno i googleov preporučeni način.

Također se sadržaj dinamički mijenja u ovisnosti o upitima na poslužiteljsku stranu i ta dinamičnost se postiže uz pomoć komponenti namijenjenih za slanje *graphql* upita

```

build_appbundle:
  name: Build Flutter (Android)
  runs-on: ubuntu-latest
  steps:
    - uses: actions/checkout@v3
    - uses: actions/setup-java@v2
      with:
        distribution: 'zulu'
        java-version: '11'
    - uses: subosito/flutter-action@v2
      with:
        flutter-version: '2.10.5'
    - run: flutter pub get
      working-directory: ./frontend/zavrsni_rad
    - run: flutter test
      working-directory: ./frontend/zavrsni_rad
    - run: flutter build apk
      working-directory: ./frontend/zavrsni_rad
    - run: flutter build appbundle
      working-directory: ./frontend/zavrsni_rad
    - name: Push to Releases
      uses: ncipollo/release-action@v1
      with:
        artifacts: "frontend/zavrsni_rad/build/app/outputs/apk/release/*"
        tag: v1.0.${{ github.run_number }}

```

(a) GitHub actions za stvaranje nove verzije aplikacije

Slika 5.12: Osnovne karakteristike korisničke strane

koji se zovu *Query i Mutation*. Navedene komponente imaju jednu funkciju koja se poziva svaki put kada se promjeni stanje upita i zove se *builder*. Toj funkciji se kao ulazni parametar šalje rezultat i stanje izvođenja upita to jest mutacije, i dužnost joj je vratiti komponentu koja će se prikazati na ekranu.

BLoC

BLoC (Business Logic Components) je obrazac za upravljanjem stanjem aplikacije. Karakteristika ovog obrasca je da međusobno razdvaja prikaz komponente, upravljanje njenim stanjem i logiku iza upravljanja stanjem. Srž obrasca je načelo da bi sve asinkrone operacije u aplikaciji trebalo modelirati kao tok događaja. To se radi na način da jedna komponenta na osnovu interakcije s okolinom kreira događaje i šalje ih u ulazni tok događaja, a svaka komponenta koja ovisi o tom toku događaja će kroz izlazni tok primiti bitne podatke i na osnovu njih ažurirati svoje stanje. U implementaciji se to postiže uz pomoć 3 komponente: prvo su potrebne klase koje predstavljaju događaje (u njima se mogu prenositi potrebni podaci) (slika 5.13 (a)), drugo *BlocProvider* (slika 5.13 (a)) i treće *BlocBuilder* (slika 5.13 (b)). *BlocProvider* predstavlja dio *BLoC-a* koji

```

import 'package:bloc/bloc.dart';

abstract class ImageEvent {}

class AddImage extends ImageEvent {
    final File image;
    AddImage({required this.image}) : super();
}

class RemoveImage extends ImageEvent {
    final int index;
    RemoveImage({required this.index}) : super();
}

class BlocImages extends Bloc<ImageEvent, List<File>> {
    BlocImages() : super([]) {
        on<AddImage>((event, emit) {
            state.add(event.image);
            emit([ ... state]);
        });
        on<RemoveImage>((event, emit) {
            state.removeAt(event.index);
            emit([ ... state]);
        });
    }
}

```

```

BlocBuilder<BlocImages, List<File>>(
    builder: ((context, state) {
        newRecipe.images = state
            .map((e) => base64Encode(e.readAsBytesSync()))
            .toList();
        return ImagesInputWidget(images: state);
    }),
), // BlocBuilder

```

(a) BLoCProvider i modeli događaja

(b) Primjer BLoCBuilder-a

Slika 5.13: Implementacija BLoC-a

se bavi presretanjem svakog ulaznog događaja, obrađivanjem i slanjem u izlazni tok, a *BlocBuilder* služi za izgradnju komponente na osnovu trenutnog događaja u izlaznom toku događaja.

5.2.2. Ekrani

U radnom okviru flutter ne postoji razlika između ekrana (eng *screen*) i na primjer teksta jer je sve definirano kao komponenta (eng. "*widget*"), ali iz semantičkih razloga svaku skupinu komponenti koja se prikazuje u istome trenutku nazivamo ecran. Često se broj ekrana uzima kao stupanj složenosti aplikacije. Tako ovaj rad ima 9 ekrana, ali oni nisu dostupni svim korisnicima nego su podijeljeni po ulogama.

Ekran dobrodošlice i prijave

Ekran dobrodošlice (slika 4.1 (a)) nema nekakvu posebnu namjenu osim estetske. Karakteristika ekrana dobrodošlice je da se prikazuje samo jednom. Prikazivanje je mogće samo kada se aplikacija prvi puta pokreće na trenutnom uređaju. To svojstvo se postiže podizanjem zastavice u memoriji uređaja uz pomoć sučelja dijeljenih tajni (eng. "*SharedPreferences*"). Ekran za prijavu sadrži 2 polja za unos i 3 gumba (slika 4.1 (b)). Prvo polje je polje za unos korisničkog imena ili e-pošte, drugo polje je polje za unos pripadajuće lozinke. Ako je uneseno krivo korisničko ime ili lozinka prikazuje se odgovarajuća poruka (slika 4.1 (c)). Preostali gumbovi su: gumb za preskakanje

prijave i prelazak na stranicu sa svim receptima, gumb za potvrdu prijave koji šalje zahtjev za prijavu na poslužitelj te gumb za prelazak na ekran za registraciju.

Ekran za registraciju

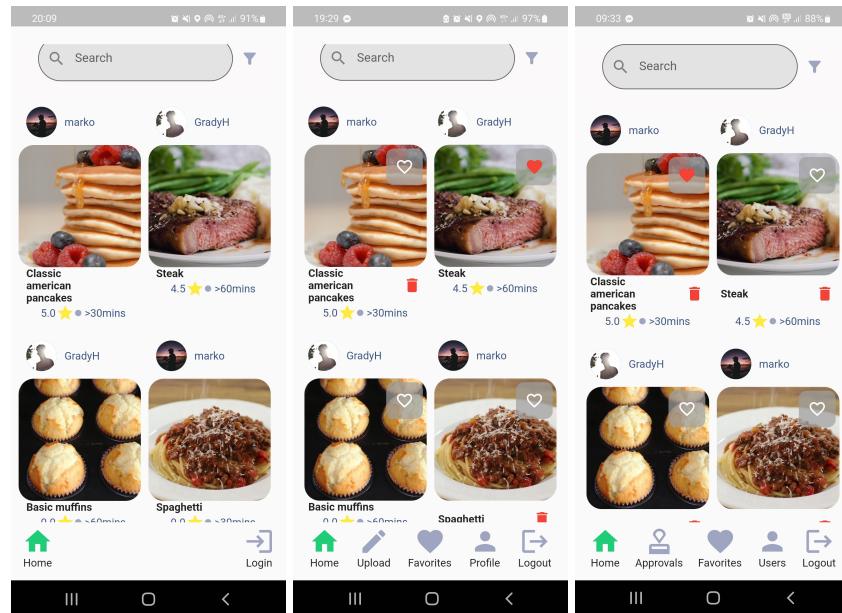
Ekran za registraciju sadrži 5 polja za unos: unos računa elektroničke pošte, unos korisničkog imena, unos lozinke, potvrda lozinke i odabir profilne slike. Profilna slika je za razliku od ostalih polja opcionalno polje. Karakteristika polja za lozinku je da ima posebne validacijske kriterije. Ti kriteriji zahtijevaju da lozinka mora sadržavati barem 8 znakova, jedno veliko slovo i jedan specijalni znak. Nakon uspješne registracije prikaže se poruka da je registracija uspješno izvršena (slika 4.2 (b)) i da je potrebno izvršiti potvrdu identiteta putem e-mail račun-a koji je bio unesen. Ukoliko je registracija nevaljana prikaže se odgovarajuća poruka (slika 4.2 (a)).

Ekran za pregled svih recepata

Ekran za pregled svih recepata će uvijek biti prvi ekran koji će se prikazati ako postoji prijavljeni korisnik, a to se također postiže uz pomoć dijeljenih tajni (*eng. Shared Preferences*). Ekran za pregled svih recepata se ovisno o ulozi koju trenutni korisnik ima razlikuje u navigacijskoj traci. Naime traka sadržava samo one gume koji vode do ekrana koji su dozvoljeni trenutno prijavljenom korisniku (slika 5.14).

Trenutno prijavljeni korisnik se određuje na temelju dijeljenjih tajni. U njih se spremaju podaci o trenutnom korisniku kao na primjer: korisničko ime, identifikacijski broj i *JWT token* za autentifikaciju i autorizaciju. To je bitno jer će se prije svakog upita, koji se obavlja dalnjom interakcijom korisnika i aplikacije, u HTTP zaglavlj staviti spomenuti token. Uz pomoć tog tokena će poslužiteljska strana obaviti autentifikaciju i autorizaciju. Ako korisnik ne bude uspješno autoriziran biti će odjavljen i preusmjeren na ekran za prijavu s odgovarajućom porukom.

Na ekranu za pregled svih recepata moguće je vidjeti sve recepte koji sa nalaze u bazi podataka. Recepti su podijeljeni u stranice, a svaka stranica sadrži 10 recepata. Ako postoji više od 10 recepata na dnu ekrana će se pojavit jedna posebna komponenta koja služi za prelazak na sljedeću, prethodnu ili neku određenu stranicu. Za svaki recept naveden je autor i profilna slika autora, pozadinskom slikom recepta, naziv, prosječna ocjena i vrijeme potrebno za pripremu. Klikom na bilo koji opisnik recepta se otvara novi ekran za pregled pojedinog recepta. U gornjem desnom kutu opisnika



(a) Traka za neprijavljenog korisnika (b) Traka za prijavljenog korisnika (c) Traka za administratora

Slika 5.14: Različite trake za različite korisnike

postoji gumb za dodavanje recepta u favorite. U donjem desnom kutu opisnika postoji gumb koji služi za brisanje recepta koji je prikazan samo u slučaju da recept pripada trenutno prijavljenom korisniku ili je trenutno prijavljeni korisnik zapravo administrator.

Također na samom vrhu ekrana postoji komponenta koja služi za filtriranje recepata po nazivu, a pored njega se nalazi jedan gumb koji prilikom dodira otvara dijalog kojim se mogu filtrirati recepti po sastojcima i vremenu pripreme.

Ekran za pregled pojedinog recepta

Ekran za pregled pojedinog recepta će prikazivati sve podatke o receptu, a to su: sastojci, koraci pripreme, svi videozapisi i fotografije koje se nalaze u skrolabilnoj listi s kojom se interagira pokretima lijevo i desno. Dodatno ako trenutno prijavljeni korisnik ima dovoljne ovlasti moći će kliknuti na gumb za ocjenjivanje i vidjeti polje za unos komentara. Ako je prijavljeni korisnik administrator moći će kliknuti na gumb za dobrovanja to jest poništavanje odobrenja recepta (slika 4.4 (c) i (d)).

Ekran za dodavanje novog recepta

Ekran za dodavanje novog recepta je dostupan samo prijavljenim korisnicima koji nisu administratori. Ovaj ekran se sastoji od velikog broja polja za unos i odabir videozapisa i fotografija te gumba za potvrdu. Karakteristika svakog polja je da ne smije biti prazno i ne smije sadržavati više znakova od broja koji je određen bazom podataka. Posebne sekcije su sekcije za unos sastojaka, koraka pripreme, videozapisa i fotografija jer su sve te sekcije modelirane BLoC obrascem. To je bilo potrebno jer nije moguće unaprijed znati koliko takvih objekata korisnik želi dodati pa postoji opcija za dinamičko dodavanje polja za unos (slika 4.4 (a) i (b)).

Ekran za upravljanje korisnicima

Posljednji ekran je ekran za upravljanje korisnicima, one je dostupan samo administratorima. Na ovoj stranici se nalazi popis svih korisnika koji, a straničenje funkcioniра na potpuno jednak način kao i straničenje za recepte. Za svakog korisnika navedeno je korisničko ime, profilna slika i gumb za suspendiranje to jest poništavanje suspenzije. Nema opcije za brisanje korisnika jer bi to značilo da bi pojedini recepti ostali bez autora ili bi bili obrisani iz baze što nije poželjna funkcionalnost. Zato je korisniku koji je suspendiran samo onemogućena prijava u sustav (slika 4.5 (a)).

6. Upute za instalaciju i pokretanje rješenja

Preduvjet za instalaciju aplikacije je posjedovanje android uređaja jer je aplikacija namijenjena samo za takve uređaje. Za instalaciju potreban je instalacijski paket koji je moguće preuzeti iz jednog od izdanja na sljedećoj stranici: [github](#). Prije pokretanja procesa instalacije potrebno je u postavkama mobitela odobriti instalaciju iz nepoznatih izvora jer aplikacija nije odobrena od strane *Google play-a*. Kada instalacija završi aplikacija je spremna za uporabu.

Poslužiteljsku stranu i bazu podataka nije potrebno pokretati jer su već pokrenute u *"oblaku"*.

7. Zaključak

Konačni proizvod ovoga rada je potpuno funkcionalna mobilna aplikacija koja je globalno dostupna u svakom trenutku. Aplikacija se sastoji od 3 dijela: baze podataka, poslužiteljski dio i korisnički dio. Svaki od njih funkcionira potpuno samostalno i neovisno o ostalim dijelovima. Svaki dio ima neke prednosti i nedostatke koje je potrebno nadograditi u budućim radovima.

Prednost baze podataka je što se slike i videozapisi spremaju kao hiperveze jer to održava veličinu baze najmanje mogućom. To je postignuto na način da se fotografije i videozapisi spremaju u "oblak" točnije *firebase*. Nedostatak je taj što se ne koriste pohranjene procedure koje bi bilo korisno implementirati u budućim radovima jer bi to dodatno ubrzalo proces dohvata podataka.

Poslužiteljska strana ima prednost jer je implementirana GraphQL arhitekturnim stilom koji smanjuje gomilanje prijenosnih objekata i krajnjih točaka. Uz to rješava problem prekomjernog i nedovoljnog dohvata podataka. Mana je ta što se ne obavlja provjera pristiglih fotografija i videozapisa prije slanja na *firebase* i time je sigurnost aplikacije narušena. To obavezno mora biti popravljeno u budućim verzijama rada.

Velika prednost korisničke strane je što se za upravljanje stanjem koristio *BLoC* obrazac koji je brz i razdvaja logiku obrade događaja od prikaza komponenti. Time je aplikacija prikladno uslojena i oblikovana po SOLID načelima. Nedostatak je taj što nisu implementirane obavijesti. Dakle korisnici neće primati obavijesti ukoliko je netko ocjenio ili komentirao njihov recept, a administratori neće primiti obavijest ukoliko neki korisnik doda novi recept. Ova funkcionalnost bi bila korisna za implementirati jer bi poboljšala iskustvo prilikom korištenja aplikacije.

LITERATURA

- Fikayo Adepoju. *Introduction to GraphQL*, 2020. URL https://circleci.com/blog/introduction-to-graphql/?utm_source=google&utm_medium=sem&utm_campaign=sem-google-dg--emea-en-dsa-maxConv-auth-nb&utm_term=g__c_dsa_&utm_content=&gclid=Cj0KCQjwvqeUBhCBARIoAODt45ZUJuzzRmEx2aRQn8G-i4gLvvFSojTfyb_smxhWuP182ihPCci_06kaAkTLEALw_wcB.
- bezkoder. *Spring Boot Token based Authentication with Spring Security and JWT*, 2022. URL <https://www.bezkoder.com/spring-boot-jwt-authentication/>.
- Sardor Islomov. *Getting Started with the BLoC Pattern*, 2022. URL <https://www.raywenderlich.com/31973428-getting-started-with-the-bloc-pattern>.
- Dimitri Mestdagh. *Securing your GraphQL API with Spring Security*, 2020. URL <https://dimitr.im/graphql-spring-security>.
- Marko Čupić. *Programiranje u Javi*. Marko Čupić, 2015. URL <http://java.zemris.fer.hr/nastava/opjj/book-2015-09-30.pdf>.

Mobilna aplikacija za upravljanje receptima

Sažetak

U ovom radu napravljena je mobilna aplikacija koja će krajnjim korsinicima olakšati spremanje i pretraživanje recepata. Aplikacija se sastoji od 3 dijela: baze podataka, poslužiteljske strane i korisničke strane. Da bi aplikacija svima i u svakom trenutku bila dostupna korištene su usluge u oblaku od tvrtke Amazon. Aplikaciji se može pristupiti kao anonimni korisnik, prijavljeni korisnik ili kao administrator.

Baza podataka je implementirana kao relacijska baza podataka, a kao upravitelj je korišten PostgreSQL. Baza je modelirana entitetima i vezama između njih. Glavna karakteristika baze podataka je što se fotografije i videozapisi spremaju u "*oblak*", a u bazu se spremaju samo hiperveza na "*oblak*". Takva implementacija održavanja veličinu baze podataka najmanje mogućom.

Za razvoj poslužiteljske strane korišten je radni okvir SpringBoot. Kao arhitekturni stil je korišten GraphQL. GraphQL predstavlja upitni jezik namijenjen korisničkoj strani za dohvatanje samo onih podataka koji su uistinu potrebni. GraphQL omogućavanje smanjenje gomilanja prijenosnih objekata i krajnjih točaka, a uz to rješava problem prekomjernog i nedovoljnog dohvata podataka. Za implementaciju je bilo potrebno definirati shemu koja definira format podataka i upita te razrješivače koji na temelju sheme omogućuju prihvatanje upita. Taj proces je vrlo jednostavan u korištenom radnom okviru. Također radni okvir omogućava vrlo jednostavnu implementaciju autentifikacije i autorizacije uz pomoć sučelja *SpringSecurity* i JWT tokena.

Korisnička strana je implementirana u flutter radnom okviru. Jedna od prednosti tog radnog okvira je da se temelji na nativnom (eng. native) pristupu razvoju aplikacija. Takav pristup omogućava pisanje samo jednog izvornog koda koji će se moći pokrenuti na većini operacijskih sustava. Također omogućava i izradu dinamičnih i responzivnih aplikacija. Jedan od načina kako izraditi takvu aplikaciju je uz pomoć *BLoC* obrasca. Taj obrazac razdvaja logiku obrade događaja od dizajniranja komponente i tako omogućava prikladno uslojavanje aplikacije i smanjenje međuvisnosti programskog koda. Za pristup poslužiteljskoj strani koriste se posebne komponente koje su oblikovane specifično za razmjenu poruka sa poslužiteljem implementiranog GraphQL arhitekturnim stilom. **Ključne riječi:** Mobilna aplikacija, baza podataka, poslužiteljska strana,

korisnička strana, oblak, PostgreSQL, SpringBoot, GraphQL, Flutter, Native, AWS .

Mobile application for recipe management

Abstract

The result of this project is a mobile application that will help the end users to store and search for recipes. The application is divided in three parts: database, backend and frontend. In order for the application to be globally and constantly accessible cloud services from the company Amazon were used. Three kinds of users can access this application: anonymous users, logged in users or administrators.

The database is implemented as a relational database, and PostgreSQL was used as the database manager. The database was modeled with entities and relations between them. The main characteristic of the database is that images and videos are stored on the "*cloud*", and the database saves only the hyperlink to that "*cloud*". Such implementation keeps the database size minimal.

The framework for developing the backend is SpringBoot. The architectural style that was used is GraphQL. GraphQL is a query language meant for the frontend to be able to access only the needed data. GraphQL also minimizes code duplication that occurs by creating a lot of data transfer objects and endpoints, and solves the problem of overfetching and underfetching. For GraphQL to work firstly a schema must be defined which contains the format of the data and queries. After that follows the implementation of resolvers which based on the schema allow requests to be received. That process is simple with the usage of the SpringBoot framework. Also that framework provides a simple way of implementing authentication and authorization with the help of the SpringSecurity interface and JWT tokens.

The frontend was implemented in the flutter framework. One of the benefits of that framework is that it uses native approach for development. Such approach enables writing only one source code that can be run on almost every operating system. Flutter also provides a way of creating dynamic and responsive application. One way of creating such an application is by using the BLoC pattern. That pattern splits the logic of handling events from the logic of designing widgets and that allows the application to be appropriately layered. For accessing the backend special widgets are used which are specifically designed for accessing servers that are implemented with GraphQL.

Keywords: Mobile application, database, backend, frontend, cloud, PostgreSQL, SpringBoot, GraphQL, Flutter, Native, AWS.