Web-aplikacija za praćenje prehrambenih navika i zdrave prehrane

Ćurković, Marko

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1585

WEB-APLIKACIJA ZA PRAĆENJE PREHRAMBENIH NAVIKA I ZDRAVE PREHRANE

Marko Ćurković

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1585

WEB-APLIKACIJA ZA PRAĆENJE PREHRAMBENIH NAVIKA I ZDRAVE PREHRANE

Marko Ćurković

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

ZAVRŠNI ZADATAK br. 1585

Pristupnik:	**Marko Ćurković (0036541445)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Boris Milašinović

Zadatak: **Web-aplikacija za praćenje prehrambenih navika i zdrave prehrane**

Opis zadatka:

Konzumiranu hranu često je potrebno precizno evidentirati uslijed prepisanih strogih dijeta. Za završni rad potrebno je izraditi web-aplikaciju koja bi prijavljenom korisniku omogućila odabir namirnice i unos količine nakon čega bi aplikacija evidentirala kalorije, proteine i/ili druge relevantne informacije koje bi korisnik kasnije mogao pregledati i preuzeti u obliku jednostavnih izvještaja. Unos konzumirane hrane može biti temeljem slobodnog odabira ili posljedica neke od prepisanih dijeta, odnosno preporučenih jelovnika. Jelovnike može unositi svaki korisnik za sebe ili mu ih može unijeti neki drugi korisnik (npr. nutricionist) unoseći e-mail adresu korisnika kojem je jelovnik namijenjen. Prijavu korisnika implementirati koristeći Googleovu uslugu prijave korisnika. U slučaju neaktivnosti korisnika, potrebno je poslati korisniku podsjetnik da nije unio namirnice ili ih da nije unio dovoljno za pojedini dan.

Rok za predaju rada: 14. lipnja 2024.

Sadržaj

1. Uvod

Prehrana, kao esencijalni proces unosa hrane u organizam, ključna je za održavanje zdravlja svakog pojedinca. U suvremenom ubrzanom načinu života, često karakteriziranom sjedilačkim poslovima, izazovno je usvojiti i održavati kvalitetne prehrambene navike.

Uz sveprisutan stres i brojne obveze, mnogi ljudi nalaze teškoću u praćenju zdrave prehrane. Prema podacima Hrvatskog zavoda za javno zdravstvo [1], 65 % odraslih osoba u Hrvatskoj ima prekomjernu težinu, što nas svrstava na vrh ljestvice pretilosti u Europi. Ovaj trend je posljedica modernog životnog stila i tehnološkog napretka koji karakterizira 21. stoljeće, gdje mnogi provode do sedam sati dnevno za računalom. U takvim uvjetima, ljudi često posežu za brzom i nezdravom hranom, što dodatno pogoršava problem.

Stoga, kako olakšati praćenje prehrane i potaknuti ljude da se aktivno bave svojim prehrambenim navikama? Iz ovih potreba proizašla je ideja o razvoju aplikacije *MealMate* za praćenje prehrane. Cilj aplikacije *MealMate*, čija je izrada opisana u radu, je omogućiti korisnicima da jednostavno kreiraju željeni tjedni jelovnik i prate ga u bilo kojem trenutku putem interneta. Aplikacija nudi i mogućnost praćenja konzumiranih sastojaka i jela te količina kalorija, proteina i ugljikohidrata kroz vizualno privlačne stupčaste dijagrame. Korisnici koji ne mogu samostalno sastaviti svoj jelovnik mogu ga brzo i jednostavno preuzeti od prijatelja ili nutricionista. Dodatno, aplikacija omogućava preuzimanje izvještaja u CSV ili PDF formatu, što korisnicima olakšava pregled i analizu unesenih podataka o prehrani.

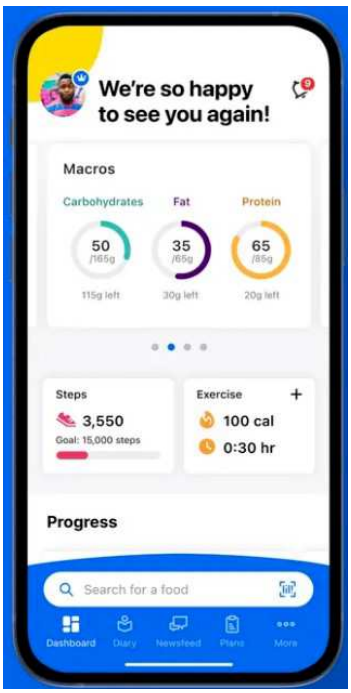
U ovom radu su najprije prikazana slična rješenja aplikacije, a zatim su opisane funkcionalnosti koje sustav treba zadovoljiti. Četvrto poglavlje opisuje model baze podataka, a potom su navedene i opisane sve tehnologije korištene pri izradi aplikacije. U šestom poglavlju predstavljena je arhitektura aplikacije koja detaljno opisuje serverski i klijentski sloj aplikacije i prikaz rješenja. Sedmo poglavlje bavi se prikazom rješenja te, konačno, rad završava s prikazom uputa za pokretanje aplikacije.

2. Pregled sličnih rješenja

Danas postoji mnogo aplikacija za praćenje prehrane, no one koje najviše odgovaraju opisanim potrebama u ovom radu su MyFitnessPal i Yazio. Ove aplikacije su iznimno popularne i služe kao izvrstan uzor za daljnji razvoj i poboljšanje web-aplikacije.

2.1 MyFitnessPal

MyFitnessPal [2] je najpopularnija aplikacija za praćenje prehrane i tjelovježbe. Omogućava korisnicima praćenje dnevnog unosa hrane i tjelesne aktivnosti. Osnovna ideja aplikacije je pomoć korisnicima da ostvare svoje ciljeve vezane za prehranu i tjelesnu aktivnost.



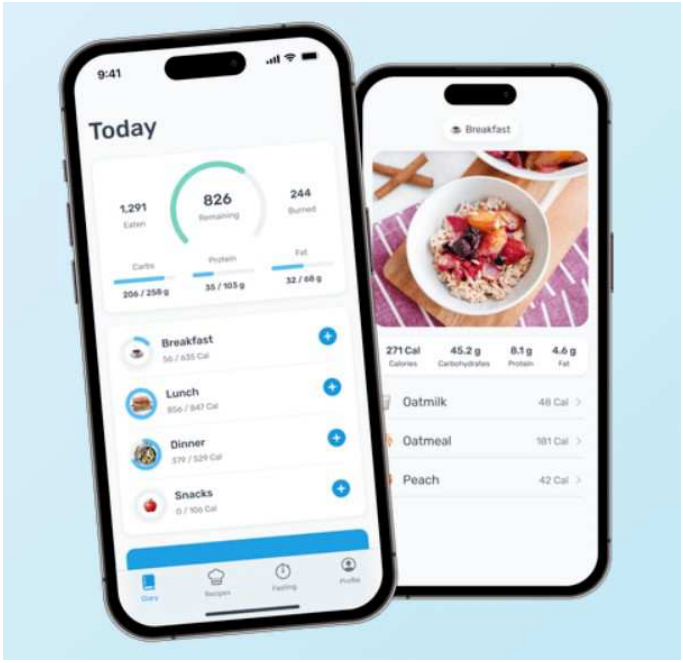
Slika 2.1 Izgled MyFitnessPal aplikacije [2]

Aplikacija sadrži bazu podataka s milijun sastojaka, što korisniku olakšava odabir hrane koju želi dodati kao obrok. Korisnici imaju mogućnost skenirati barkodove proizvoda kako bi brzo unijeli informacije o hrani, te imaju mogućnost integracije

aplikacije s *fitness* uređajima, omogućujući sveobuhvatno praćenje tjelesnog zdravlja i aktivnosti.

2.2 Yazio

Yazio [3] je mobilna aplikacija za praćenje prehrane i kalorija koja korisnicima omogućava da planiraju i prate svoje obroke i aktivnosti s ciljem postizanja prehrambenih ciljeva. Izrađena je s ciljem kako bi korisnicima pružila alat za bolje razumijevanje zdrave prehrane i poticanje zdravijih životnih navika.



Slika 2.2 Izgled Yazio aplikacije [3]

Aplikacija nudi različite dijetalne planove za korisnike prilagođene specifičnim ciljevima poput mršavljenja, održavanja težine ili izgradnje mišića. Korisnici mogu jednostavno unositi podatke o konzumiranoj hrani i pićima, uz automatsko izračunavanje unesenih kalorija, proteina, masti i ugljikohidrata. Također su dostupni brojni članci, savjeti i recepti koji pomažu korisniku u stvaranju zdravijih prehrambenih navika. Yazio sadrži funkcije za postavljanje ciljeva i izazova, motivirajući korisnike da ostvare svoje ciljeve kroz igru.

2.3 Usporedba rješenja

Postojeće aplikacije kao što su MyFitnessPal i Yazio usmjerene su na motivaciju korisnika za ostvarivanje ciljeva vezanih uz unos kalorija, proteina i ostalih nutrijenata. Web-aplikacija *MealMate* namijenjena je za praćenje specifičnih tjednih jelovnika, omogućavajući korisnicima da lako prate i biraju obroke za svaki dan u tjednu. Aplikacija također pojednostavljuje suradnju između nutricionista ili trenera i korisnika. Osoba zadužena za prehranu može u svakom trenutku ažurirati jelovnik korisnika.

Prednosti postojećih rješenja MyFitnessPal i Yazio je što sadrže veliku bazu podataka hrane i korisnici mogu na brži i praktičniji način bilježiti unesene obroke i sastojke uz pomoću skeniranja barkoda namirnice. Dodatna prednost aplikacije MyFitnessPal je praćenje tjelesne aktivnosti korisnika. Povezivanjem podataka o prehrani i fizičkoj aktivnosti pruža korisnicima potpuni uvid u njihove zdravstvene navike, što olakšava održavanje zdravog načina života.

Prednost predstavljene aplikacije *MealMate* je što korisnik na jednostavan način može kreirati tjedni jelovnik i dodavati obroke tijekom dana pomoću nekoliko klikova. Također, korisnik može preuzeti izvještaje konzumiranih obroka te pregledati kako napreduje njegova prehrana.

3. Specifikacija programske potpore

3.1 Funkcionalni zahtjevi

Funkcionalni zahtjevi definiraju softversku funkcionalnost i očekivano ponašanje aplikacije. Odgovaraju na pitanje što se može napraviti koristeći aplikaciju.

UC1 – Prijava korisnika putem Google autentifikacije

- Glavni Sudionik: Neprijavljeni korisnik
- Cilj: Prijava u aplikaciju koristeći Google račun
- Preduvjet: Korisnik mora imati aktivan Google račun
- Opis osnovnog tijeka:
    1. Korisnik odabire opciju za prijavu putem Google-a
    2. Aplikacija preusmjerava korisnika na Google stranicu za prijavu
    3. Korisnik unosi svoje korisničko ime i lozinku (ako već nije prijavljen)
    4. Korisnik odobrava aplikaciji pristup potrebnim informacijama s njegovog Google računa
    5. Aplikacija preusmjerava korisnika na osnovni zaslon ako su podaci ispravni
- Opis mogućih odstupanja:
    1. Korisnik unosi neispravne podatke za prijavu ili odbija dati potrebne dozvole aplikaciji
    2. Google servis nije dostupan zbog tehničkih problema

UC2 – Dodavanje i uređivanje obroka

- Glavni sudionik: Prijavljeni korisnik
- Cilj: Dodavanje novih obroka i uređivanje postojećih unosa u aplikaciji
- Preduvjet: Korisnik je prijavljen u aplikaciju

- Opis osnovnog tijeka:
    1. Korisnik dolazi do sekcije za upravljanje obrocima
    2. Za dodavanje obroka korisnik odabire opciju za dodavanje obroka
    3. Korisnik odabire željenu opciju želi li dodati sastojak ili jelo
    4. Korisnik odabire sastojak ili jelo (nakon toga ima uvid u nutritivne vrijednosti sastojka ili jela)
    5. Aplikacija provjerava unesene podatke i sprema novi obrok u bazu
    6. Za uređivanje postojećeg obroka, korisnik odabire opciju za uređivanje
    7. Korisnik mijenja željene podatke i potvrđuje izmjene
    8. Aplikacija ažurira podatke o obroku u bazi
- Opis mogućih odstupanja:
    1. Korisnik unosi nepotpune ili neispravne podatke prilikom dodavanja ili uređivanja obroka

UC3 – Pregled nutritivnih vrijednosti obroka

- Glavni Sudionik: Prijavljeni korisnik
- Cilj: Pregled nutritivnih vrijednosti obroka
- Preduvjet: Korisnik je prijavljen u aplikaciju
- Opis osnovnog tijeka:
    1. Korisnik dolazi do sekcije za upravljanje obrocima
    2. Aplikacija prikazuje naziv i količinu za svaki obrok uz prikaz nutritivnih vrijednosti uključujući kalorije, proteine i ugljikohidrate
- Opis mogućih odstupanja:
    1. Aplikacija ne uspije učitati nutritivne informacije zbog tehničkog problema

UC4 – Kreiranje i uređivanje tjednih jelovnika

- Glavni Sudionik: Prijavljeni korisnik
- Cilj: Kreiranje i uređivanje tjednih jelovnika
- Preduvjet: Korisnik je prijavljen u aplikaciju
- Opis osnovnog tijeka:
    1. Korisnik odabire opciju za kreiranje ili uređivanje jelovnika
    2. Za kreiranje novog jelovnika korisnik upisuje ime jelovnika
    3. Aplikacija korisniku prikazuje ekran sa tjednim kalendarom i formom za dodavanje sastojaka ili jela
    4. Korisnik odabire sastojak ili jelo, dan i vrstu obroka
    5. Korisnik dodaje sastojak ili jelo u jelovnik
    6. Aplikacija sprema promjene u bazu
    7. Za uređivanje postojećeg jelovnika, korisnik odabire jelovnik i koristi opciju za uređivanje
    8. Korisnik dodaje, briše i mijenja željene sastojke ili jela unutar jelovnika
    9. Aplikacija ažurira informacije o jelovniku
- Opis mogućih odstupanja:
    1. Korisnik pokušava dodati namirnicu u jelovnik bez svih potrebnih informacija

UC5 – Dijeljenje jelovnika

- Glavni Sudionik: Prijavljeni korisnik
- Cilj: Dijeljenje jelovnika s drugim korisnicima
- Preduvjet: Korisnik je kreirao jelovnik i prijavljen je u aplikaciju
- Opis osnovnog tijeka:
    1. Korisnik odabire jelovnik koji želi podijeliti i odabire opciju za dijeljenje jelovnika

2. Korisnik unosi e-mail adresu korisnika s kojim želi podijeliti jelovnik
3. Aplikacija provjerava postoji li korisnički račun povezan s unesenim podacima
4. Ako korisnik postoji, aplikacija omogućava mu dopušta pristup jelovniku

- Opis mogućih odstupanja:
  1. Korisnik unosi e-mail adresu koje ne postoji u bazi podataka

UC6 – Upravljanje primljenim jelovnicima

- Glavni Sudionik: Korisnik koji je primio jelovnik
- Cilj: Upravljanje jelovnicima koje je korisnik primio od drugih korisnika
- Preduvjet: Korisnik je prijavljen u aplikaciju i dobio je pristup jelovniku od drugog korisnika
- Opis osnovnog tijeka:
  1. Korisnik odabire jelovnik s kojim želi upravljati
  2. Korisnik bira opciju za kloniranje jelovnika s kojom stvara kopiju jelovnika u svojim osobnim jelovnicima
  3. Aplikacija kreira kopiju jelovnika, omogućujući korisniku da uredi ovaj jelovnik kao vlastiti
  4. Ako korisnik želi obrisati podijeljeni jelovnik, odabire opciju za brisanje jelovnika
  5. Aplikacija briše podijeljeni jelovnik iz korisnikovih jelovnika

UC7 – Pregled statistika unosa

- Glavni Sudionik: Prijavljeni korisnik
- Cilj: Pregleda statistika o unosu kalorija, proteina i ugljikohidrata po datumima
- Preduvjet: Korisnik je prijavljen u aplikaciju

- Opis osnovnog tijeka:
    1. Korisnik odabire opciju za pregled statistika
    2. Aplikacija prikazuje detaljne statistike u obliku stupčastih dijagrama o ukupnom unosu kalorija, proteina i ugljikohidrata.
    3. Korisnik ima mogućnost biranja datuma i broj dana za koji želi prikaz dijagrama

UC8 – Slanje molbe za dodavanje sastojka

- Glavni Sudionik: Prijavljeni korisnik
- Cilj: Slanje molbe za dodavanje novog sastojka u bazu podataka aplikacije
- Preduvjet: Korisnik je prijavljen u aplikaciju
- Opis osnovnog tijeka:
    1. Korisnik odabire opciju za slanje molbe administratoru
    2. Korisnik u formu upisuje naziv sastojka za koju želi da bude dodana
    3. Korisnik šalje molbu pomoću aplikacije
    4. Aplikacija sprema molbu u bazu podataka gdje administrator može pregledati željene namirnice korisnika

UC9 – Preuzimanje izvještaja

- Glavni Sudionik: Prijavljeni korisnik
- Cilj: Preuzimanje izvještaja o unesenim obrocima
- Preduvjet: Korisnik je prijavljen u aplikaciju
- Opis osnovnog tijeka:
    1. Korisnik odabire opciju za generiranje izvještaja
    2. Korisnik ima mogućnost biranja datuma između kojih želi prikaz obroka u izvještaju
    3. Korisnik bira CSV ili PDF format izvještaja
    4. Aplikacija generira izvještaj na temelju zadanih parametara

UC10 – Podsjetnik o neaktivnosti

- Glavni Sudionik: Aplikacija
- Cilj: Potaknuti korisnika za korištenje aplikacije slanjem podsjetnika na e-mail
- Preduvjet: Korisnik nije unio dovoljan broj obroka u danu
- Opis osnovnog tijeka:
    1. Aplikacija provjerava aktivnost korisnika
    2. Ako aplikacija vidi da korisnik nije unio dovoljan broj obroka, aplikacija automatski pokreće slanje podsjetnika na korisnikov e-mail
    3. Aplikacija generira e-mail podsjetnik i šalje ga korisniku

UC11 – Kreiranje vlastitog jela

- Glavni Sudionik: Prijavljeni korisnik
- Cilj: Kreiranje novih jela od strane korisnika
- Preduvjet: Korisnik je prijavljen u aplikaciju
- Opis osnovnog tijeka:
    1. Korisnik dolazi do sekcije za upravljanje jelima
    2. Aplikacija prikazuje popis svih trenutnih korisnikovih jela
    3. Korisnik odabire opciju za kreiranje novog jela i aplikacija otvara formu za unos podataka
    4. Korisnik unosi naziv jela, dodaje potrebne sastojke i aplikacija automatski pokazuje nutritivne vrijednosti svakog sastojka i jela sveukupno
    5. Korisnik odabire opciju za spremanje jela
    6. Aplikacija sprema novo jela u bazu podataka
- Opis mogućih odstupanja:

1. Korisnik pokušava dodati namirnicu u jelovnik bez svih potrebnih informacija

UC12 – Upravljanje vlastitim jelima

- Glavni Sudionik: Prijavljeni korisnik
- Cilj: Upravljanje vlastitim jelima
- Preduvjet: Korisnik je prijavljen u aplikaciju
- Opis osnovnog tijeka:
  1. Korisnik dolazi do sekcije za upravljanje jelima
  2. Aplikacija prikazuje popis svih trenutnih korisnikovih jela koje korisnik ima mogućnost uređivati i obrisati
  3. Korisnik odabire opciju za uređivanje
  4. Aplikacija korisniku prikazuje formu za uređivanje te korisnik ima mogućnost urediti svoje jelo
  5. Korisnik odabire opciju za brisanje jela
  6. Aplikacija briše jelo i sprema promjene u bazu podataka
- Opis mogućih odstupanja:
  1. Korisnik pokušava spremiti uređeno jelo bez svih potrebnih informacija

3.2 Nefunkcionalni zahtjevi

Nefunkcionalni zahtjevi opisuju kriterije koje aplikacija mora zadovoljavati. Odgovaraju na pitanje kako dobro sustav radi i najčešće su posljedica ugovora ili pravila, opisa vanjskih sučelja, zahtjeva na performanse i ograničenja na dizajn.

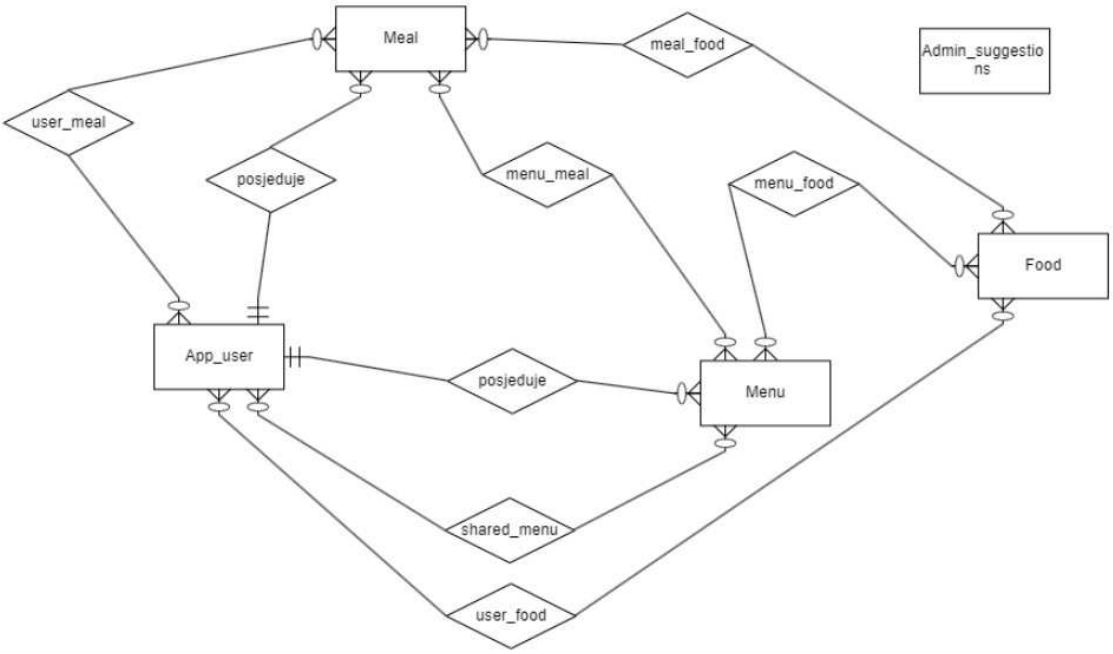
Aplikacija treba omogućiti korisnicima prijavu pomoću Google računa. Također, korisnici bi trebali imati mogućnost preuzimanja izvještaja u PDF i CSV formatima. Aplikacija treba biti razvijena korištenjem Spring Boot-a za serverski dio i React-a za klijentski dio aplikacije.

4. Model baze podataka

Za implementaciju ovog rada koristi se relacijska baza podataka, čija je glavna svrha olakšati modeliranje stvarnog svijeta. Osnovni element baze podataka je relacija (tablica) koja je definirana imenom i atributima. U bazi podataka se nalaze sljedeći entiteti:

- *Menu*
- *Food*
- *App_user*
- *Shared_menu*
- *Menu_food*
- *User_food*
- *User_Meal*
- *Meal*
- *Meal_food*
- *Menu_Meal*
- *Admin_suggestions*

4.1 Dijagram baze podataka



Slika 4.1 Konceptualni model baze podataka

4.2 Opis dijagrama baze podataka

Entiteti *Food* i *App_user* mogu se smatrati osnovnim entitetima u predstavljenom dijagramu baze podataka. Oni predstavljaju temelj baze i svi ostali entiteti ovise o njima. Entitet *Food* opisuje sastojke koje korisnik ima priliku koristiti i unositi kao konzumirane. Svaki sastojak karakterizira se imenom, količinom izraženom u gramima, brojem kalorija, proteinima i ugljikohidratima. Glavni ključ entiteta *Food* je identifikacijski broj (*id*) koji baza podataka automatski generira prilikom spremanja instance entiteta.

Entitet *App_user* predstavlja svakog korisnika koji se barem jednom prijavio u aplikaciju. Korisnik se definira email adresom, imenom i prezimenom. Glavni ključ, slično kao kod entiteta *Food*, automatski se generira prilikom spremanja instance entiteta *App_user.*

Entitet *Meal* predstavlja jela koja korisnik stvara u aplikaciji. Svako jelo sastoji se od više sastojaka, čime se korisniku omogućuje opisivanje većih obroka bez potrebe za

ponovnim unosom velikog broja sastojaka. Jelo je definirano imenom i primarnim ključem koji služi kao identifikacijski broj. Zbog mnogo prema mnogo (*Many-to-Many*) relacije između entiteta *Meal* i *Food*, stvorena je pomoćna tablica *Meal_food*. Ova tablica predstavlja sastojke koje korisnik dodaje u jelo. *Meal_food* je definiran identifikacijskim brojem i količinom odabranog sastojka u gramima.

Entitet *User_food* je spojna tablica koja proizlazi iz relacije mnogo prema mnogo (*Many-to-Many*) između entiteta *App_user* i *Food. User_food* služi kao evidencija obroka, koji su sastojci, koje korisnik unosi tijekom dana. Tijekom dana korisnik može unijeti neograničen broj sastojaka, koji se smatraju konzumirani tog dana. Korištenjem entiteta *User_food*, izračunavaju se nutritivne vrijednosti unesenih sastojaka, što omogućuje njihov prikaz na ekranu. *User_food* sastoji se od datuma, količine namirnice u gramima te dva strana ključa: identifikacijskog broja entiteta *Food* i entiteta *App_user.* Glavni ključ entiteta *User_food* formira se od datuma i dva strana ključa, omogućujući time razlikovanje dnevnih unosa namirnica po korisnicima.

Sličnu ulogu kao *User_food* i tablica *User_meal*, koja proizlazi iz relacije mnogo prema mnogo (*Many-to-Many*) između entiteta *App_user* i *Meal*. *User_meal* služi za evidenciju obroka u slučaju kada korisnik odabere jelo kao svoj obrok. *User_Meal* se sastoji od datuma unosa obroka i dva strana ključa: identifikacijskog broja entiteta *App_user* i entiteta *Meal.* Primarni ključ u entitetu *User_meal* čine sve tri navedene varijable zajedno.

Entitet *Menu* predstavlja jelovnike koje korisnici kreiraju unutar aplikacije. *Menu* je povezan s entitetom *App_user* preko relacije jedan prema mnogo (*One-to-Many*), što omogućuje svakom korisniku stvaranje više jelovnika. Svaka instanca entiteta *Menu* sadrži strani ključ na entitet *App_user*, što omogućava pronalazak korisnika koji je stvorio jelovnik. *Menu* se sastoji od imena jelovnika, broja proteina, ugljikohidrata i kalorija unutar jelovnika i email adrese korisnika koji ga je stvorio. U *Menu* je moguće dodavati sastojke i jela koje korisnik odabire i raspoređuje prema željenom vremenu dana za konzumaciju. Iz tog razloga postoje entiteti *Menu_food* i *Menu_meal.*

Entitet *Menu_food* je rezultat mnogo prema mnogo (*Many-to-Many*) relacije između entiteta *Menu* i *Food.* Glavni ključ entiteta *Menu_food* sastoji se od dva strana ključa na entitete *Menu* i *Food*, imena dana i vrste jela, gdje je vrsta jela, nazvana *Meal_type*, realizirana kao enumeracija. Slično tome, entitet *Menu_meal* rezultat je mnogo prema mnogo (*Many-to-Many*) relacije između *Menu* i *Meal*. *Menu_meal* sadrži iste atribute

kao *Menu_food*, osim što umjesto stranog ključa na *Food* sadrži strani ključ na entitet *Meal*.

Entitet *Shared_menu* opisuje sve jelovnike koje korisnik dijeli s drugim korisnicima. Entitet je rezultat mnogo prema mnogo (*Many-to-Many*) relacije između entiteta *Menu* i *App_user*. Svaki korisnik ima mogućnost podijeliti neograničen broj jelovnika. *Shared_menu* sastoji se samo od dva strana ključa i glavnog ključa identifikacijskog broja, koji baza podataka automatski stvara prilikom spremanja instance.

Entitet *Admin_suggestion* koristi se za pohranu imena sastojaka koje korisnici imaju želju koristiti, ali trenutno nisu u bazi podataka. Korisnik ima mogućnost zabilježiti ime sastojka u bazu, zatim administrator pregleda prijedloge i odluči hoće li dodane sastojke uključiti u bazu podataka. Ovaj mehanizam omogućava prilagodbu baze podataka prema stvarnim potrebama korisnika.

4.3 Opis tablica

Food – sastojci koje korisnik ima mogućnost koristiti

Id	INT	Jedinstveni identifikator, *Glavni ključ*
Calories	INT	Broj kalorija sastojka, *Obavezno polje*
Carbs	INT	Broj ugljikohidrata sastojka, *Obavezno polje*
Grams	INT	Količina sastojka u gramima, *Obavezno polje*
Protein	INT	Broj proteina sastojka, *Obavezno polje*
Food	VARCHAR(255)	Ime sastojka, *Obavezno polje, Jedinstveno*

App_user – predstavlja korisnika aplikacije

Id	INT	Jedinstveni identifikator, *Glavni ključ*

Email	VARCHAR(255)	Email adresa korisnika, *Obavezno polje, Jedinstveno*
Name	VARCHAR(255)	Ime korisnika, *Obavezno polje*
Surname	VARCHAR(255)	Prezime korisnika, *Obavezno polje*

Menu – tjedni jelovnici koje korisnik stvara

Id	INT	Jedinstveni identifikator, *Glavni ključ*
Calories	INT	Ukupan broj kalorija u jelovniku
Carbs	INT	Ukupan broj ugljikohidrata u jelovniku
Name	VARCHAR(255)	Ime jelovnika, *Obavezno polje*
Proteins	INT	Ukupan broj proteina u jelovniku
User_id	INT	Strani ključ na korisnika

Meal - jela koja sadrže više sastojaka koje korisnik ima mogućnost stvoriti

Id	INT	Jedinstveni identifikator, *Glavni ključ*
Name	VARCHAR(255)	Ime jela, *Obavezno polje*
User_id	INT	Strani ključ na korisnika

Meal_food – sastojci koje je korisnik dodao u jelo

Id	INT	Jedinstveni identifikator, *Glavni ključ*
Size	INT	Količina sastojka u gramima, *Obavezno polje*

Food_id	INT	Strani ključ na sastojak
Meal_id	INT	Strani ključ na jelo

Menu_meal – jela koja su dodana u jelovnik

- Svi atributi zajedno čine glavni ključ

Day	VARCHAR(255)	Dan za koji je jelo dodano, *Obavezno polje*
Meal_type	INT	Vrsta obroka u koji je jelo stavljeno, *Obavezno polje*
Meal_id	INT	Strani ključ na jelo
Menu_id	INT	Strani ključ na jelovnik

Menu_food – namirnice koje su dodane u jelovnik

- Day, Meal_type, Menu_id i Food_id čine glavni ključ

Day	VARCHAR(255)	Dan za koji je sastojak dodano, *Obavezno polje*
Meal_type	INT	Vrsta obroka u koji je sastojak stavljen, *Obavezno polje*
Food_id	INT	Strani ključ na sastojak
Menu_id	INT	Strani ključ na jelovnik
Size	INT	Količina sastojka, *Obavezno polje*

User_meal – jelo koje je korisnik unio i smatra se kao obrok

- Sva tri atributa zajedno čine glavni ključ

Date	Date	Datum unosa obroka, *Obavezno polje*
User_id	INT	Strani ključ na korisnika
Meal_id	INT	Strani ključ na jelo

User_food – pojedinačni sastojak koji je korisnik unio i smatra se kao obrok

- Date, Food_id i User_id čine glavni ključ

Date	Date	Datum unosa obroka, *Obavezno polje*
Size	INT	Količina unesenog sastojka, *Obavezno polje*
Food_id	INT	Strani ključ na sastojak
User_id	INT	Strani ključ na korisnika

Shared_menu – podijeljeni jelovnici između korisnika

Id	INT	Jedinstveni identifikator, *Glavni ključ*
Menu_id	INT	Strani ključ na jelovnik
User_id	INT	Strani ključ na korisnika

Admin_suggestions – predloženi sastojci od strane korisnika

Id	INT	Jedinstveni identifikator, *Glavni ključ*
Name	VARCHAR(255)	Ime predloženog sastojka

5. Tehnologije korištene u radu

5.1 Spring Boot

Za razvoj serverske strane aplikacije korišten je Spring Boot okvir koji omogućava jednostavnu konfiguraciju i pokretanje Spring aplikacija. Okvir Spring Boot temelji se na programskom jeziku Java i stvoren je s ciljem kako bi se programeri mogli fokusirati na razvoj funkcionalnosti aplikacije umjesto na upravljanje često kompliciranim konfiguracijama. Spring povezuje sve potrebne komponente aplikacije u jedno nudeći automatsku konfiguraciju aplikacije. [4]

Java je objektno-orijentiran programski jezik koji obuhvaća bogatu kolekciju standardnih biblioteka, omogućavajući razvoj softverskih rješenja. Programi pisani u Javi izvode se na Java virtualnom stroju, što programerima pruža bezbrižnost o pokretanju programa na platformi, bitno je samo da platforma podržava Java virtualni stroj. Programski jezik Java dodatno olakšava razvoj aplikacija programerima omogućavajući im korištenje višedretvenosti i automatsko upravljanje memorijom. Automatsko upravljanje memorijom sprječava problem s curenjem memorije, ali također povećava efikasnost i stabilnost aplikacija. [5]

Maven je alat koji se koristi za automatizaciju izrade projekata. Najčešće se koristi za programski jezik Java, ali također pruža mogućnosti za ostale programske jezike kao što su Ruby i C#. Pri izradi ove aplikacije korišten je Maven za upravljanje zavisnostima i konfiguraciju aplikacije. Prednost korištenja Maven-a je njegovo upravljanje bibliotekama što programeru olakšava posao. Jedini zadatak za programera je definirati biblioteku unutar *pom.xml* datoteke koja služi kao središnje mjesto za konfiguraciju projekta. [6]

Tijekom izrade aplikacije, korištene su brojne biblioteke, koje se mogu kategorizirati u dvije glavne skupine: biblioteke koje su izravno povezane uz Spring Boot i nezavisne biblioteke.

Biblioteke izravno povezane uz Spring Boot:

- Spring Boot Starter Data JPA

    - Ključna biblioteka koja omogućava aplikaciji rad s bazom podataka koristeći *Java Persistence API (JPA)*. Koristeći ovu biblioteku programeri mogu na jednostavan i brz način kreirati svoje entitete, repozitorije i transakcije bez ručnog podešavanja konfiguracija

- Spring Boot Starter Mail
    - Biblioteka koja omogućava aplikaciji slanje mailova
    - U ovom radu se koristi za slanje podsjetnika neaktivnim korisnicima
- Spring Boot Starter Web
    - Ključna biblioteka koja omogućava programeru brz i jednostavan razvoj web aplikacija. Biblioteka automatizira sve konfiguracije potrebne za web razvoj i dolazi s ugrađenim web serverom Tomcat. Koristeći ovu biblioteku olakšava se posao programeru u efikasnoj izradi web aplikacije koristeći Spring MVC (Model-Pogled-Upravljač) arhitekturu
- Spring Boot Devtools
    - Biblioteka koja pruža automatsko primjenjivanje promjena u kodu i omogućava programerima vidljivost promjena gotovo odmah bez potrebe za ponovnim pokretanjem aplikacije
- Spring Boot Starter OAuth2 Client
    - Biblioteka koja pruža programerima jednostavnu implementaciju OAuth2 autorizacijskih protokola unutar aplikacije
    - U ovom radu korištena je za prijavu putem vanjskog pružatelja usluga Google-a

Nezavisne biblioteke:

- PDFBox
    - Biblioteka koja omogućava kreiranje i upravljanje PDF dokumentima unutar Java programa
    - U ovoj aplikaciji koristi se za generiranje izvješća o obrocima korisnika u obliku PDF dokumenta

- PostgreSQL Driver
    - Biblioteka koja omogućava aplikaciji povezivanje i korištenje PostgreSQL baze podataka
- H2 Database
    - Biblioteka koja omogućava korištenje H2 baze podataka koja je idealna za vrijeme razvoja i testiranja aplikacije zbog svoje jednostavne konfiguracije
    - H2 baza podataka je u ovom radu korištena kako bi korisnici mogli na jednostavan način pokretati web aplikaciju pomoću Docker tehnologije
- Lombok
    - Biblioteka koja omogućava automatsko generiranje koda. Najčešće se koristi za generiranje *gettera*, *settera*, konstruktora, *toString* i *equals* metoda
    - Olakšava programerima posao i smanjuje količinu koda o kojem programeri moraju brinuti

5.2 PostgreSQL

Za bazu podataka u aplikaciji korištena je objektno-relacijska baza PostgreSQL. PostgreSQL korištena je za upravljanje i pohranu podataka iz aplikacije. Ova baza podataka vrlo je često odabir za veliku količinu programera zbog svoje podrške za standardni SQL, transakcije i kompleksne upite. [7]

5.2.1 Povezivanje Spring Boota s PostgreSQL-om

Povezivanje Spring Boot dijela aplikacije s bazom podataka predstavlja serverski dio aplikacije. Integracija te dvije komponente omogućava aplikaciji brzo upravljanje i odrađivanje zahtjeva koji dolaze s klijentske strane aplikacije. Konfiguracija za povezivanje komponenata odrađuje se unutar *application.properties* datoteke koja se nalazi na Spring Boot komponenti. Datoteka *application.properties* služi kao središnje mjesto za

konfiguraciju Spring Boot aplikacije na kojem se mogu definirati različite postavke za ponašanje aplikacije.

U izradi ove aplikacije korištene su tri datoteke za konfiguraciju iz razloga kako bi se aplikacija mogla jednostavno pokretati na računalu svakog korisnika bez potrebe za skidanjem svih potrebnih alata. Konfiguracijske datoteke koje su korištene su:

- *application.properties*
- *application-docker.properties*
- *application-local.properties*

```
spring.application.name=zavrsni
spring.jpa.hibernate.ddl-auto=update

spring.security.oauth2.client.registration.google.client-id=
spring.security.oauth2.client.registration.google.client-secret=

frontend.url=http://localhost:3000

spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=posterranger@gmail.com
spring.mail.password=pvrsktmfzacjmcba
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

Programski odsječak 5.1 Primjer izvedbe *application.properties* datoteke

Ova datoteka služi kao osnovna konfiguracija. Sadrži postavke koje su zajedničke i ostalim datotekama pa se stavljaju u zajedničku datoteku. Ovdje se stavljaju osnovne postavke koje ne zahtijevaju česte promjene. U datoteci je postavljeno ime aplikacije, postavka da se sa svakim pokretanjem aplikacije obnovi baza podataka, konfiguracija za OAtuh2 autentifikaciju gdje se *google.client* i *google.client-secret* koriste za autentifikaciju s Google poslužiteljem, URL klijentskog dijela aplikacije i konfiguracija za email komunikaciju.

```
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.datasource.url=jdbc:postgresql://db:5432/zavrsni
spring.datasource.username=postgres
spring.datasource.password=postgres
```

Programski odsječak 5.2 Primjer izvedbe *application-docker.properties* datoteke

Ova datoteka služi za povezivanje aplikacije s PostgreSQL bazom podataka unutar Docker okruženja. Konfiguracija je prilagođena kako bi aplikacija i baza podataka mogli komunicirati preko Docker mreže.

```
1. spring.h2.console.enabled = true
2. spring.datasource.url = jdbc:h2:mem:testdb
3. spring.datasource.driverClassName=org.h2.Driver
```

Programski odsječak 5.3 Primjer izvedbe *application-local.properties* datoteke

Datoteka služi kako bi se aplikacija mogla povezati na H2 bazu podataka u memoriji. Konfiguracija baze podataka sa Spring Boot-om je vrlo jednostavna i može se vrlo brzo povezati s aplikacijom. U ovoj aplikaciji služi kako bi korisnik na jednostavan način mogao kompilirati Java projekt pomoću kojeg će moći pokrenuti aplikaciju s Docker-om.

5.3 React

Za kreiranje korisničkih sučelja i klijentske strane aplikacije korišten je React. React je Javascript biblioteka koja omogućava brz razvoj sučelja web aplikacija, pritom omogućavajući dinamičko prikazivanje podataka bez potrebe za ponovnim učitavanjem stranice. Glavna prednost React biblioteke je korištenje komponenata. Komponente služe za izdvajanje koda koji će biti iskorišten za ponovnu upotrebu. Korištenje komponenata smanjuje redundanciju i ponavljanje istog koda. React se samostalno brine o ažuriranju komponenata kada se dogodi promjena u njihovom stanju. [8]

Poznat stil razvoja u Reactu je *data down, actions up* što predstavlja razvoj u kojem se podaci prenose od komponente prema roditeljskoj komponenti, a interakcije korisnika s komponentom (klik gumba) mogu imati utjecaj ažuriranja stanja cijele aplikacije.

5.3.1 React Vite

Uz osnovne biblioteke React-a korišten je i Vite. Vite je alat koji omogućava React-u brže pokretanje i ažuriranje aplikacije tijekom razvoja. Dodavanjem Vite-a u projekt s React-om osigurava se da je aplikacija konfigurirana i spremna za suvremen razvoj web aplikacije. Također uvođenjem Vite-a programeri se mogu usredotočiti na kreiranje koda, a ne na konfiguriranje alata za izgradnju. [9]

Na ovom radu konfiguracijska datoteka koja postavlja postavke kako Vite treba upravljati projektom je *vite.config.js*

```
1. import { defineConfig } from 'vite'
2. import react from '@vitejs/plugin-react'
3.
5. export default defineConfig({
6.   plugins: [react()],
7. })
```

Programski odsječak 5.4 Primjer izvedbe *vite.config.js* datoteke

5.3.2 Tailwind CSS

Uz React na klijentskoj strani korišten je i Tailwind CSS za stiliziranje aplikacije. Tailwind CSS omogućava brzo i jednostavno stiliziranje uz pomoću klasa. Tailwind daje korisniku na korištenje veliki broj klasa koje se mogu koristiti direktno na HTML elementima. Ova vrsta stiliziranja aplikacija olakšava posao programeru sa inače često složenim CSS kodom s predefiniranim klasama koje su lako pamtljive i jednostavne za korištenje. Tailwind se može na vrlo jednostavan način konfigurirati unutar React projekta. Potrebno je uključiti Tailwind stilove unutar glavne CSS datoteke u projektu. U ovom radu glavna CSS datoteka je *index.css*, a uključeni su stilovi koji omogućavaju korištenje Tailwinda. [10]

5.4 Docker

Kako bi svaki budući korisnik mogao pokrenuti aplikaciju na svojem računalu bez brige imaju li sve potrebne programe korišten je Docker. Docker je platforma za spremanje i pokretanje aplikacija unutar izoliranih okruženja koja se nazivaju kontejneri. Glavni koncepti Docker-a su slika i kontejner. Unutar slike se pakira aplikacija i sve njene potrebne ovisnosti (biblioteke i datoteke), a kontejner je stvoren uz pomoću Docker slike i pokrenut je kao potpuno izoliran proces od operacijskog sustava na kojem je pokrenut.

Slika se stvara uz pomoću *Dockerfile* datoteke u kojoj se nalaze sve potrebne naredbe za izgradnju slike. Izrada dobrog *Dockerfile-a* je jako bitna kako bi aplikacija radila jednako u svim okruženjima. [11]

Unutar ovog rada postoje 2 *Dockerfile-a*:

- *Dockerfile* za stvaranje slike Spring Boot dijela aplikacije

```
FROM openjdk:17-slim

ENV TZ=Europe/Zagreb
EXPOSE 8080
WORKDIR /app
RUN apt-get update && apt-get install -y \
    fontconfig \
    fonts-dejavu \
    && rm -rf /var/lib/apt/lists/*
COPY ./target/*.jar app.jar
ENV SPRING_PROFILES_ACTIVE=docker
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Programski odsječak 5.5 Primjer izvedbe Dockerfile datoteke

U ovoj datoteci postavlja se osnovna slika na kojoj će se Docker slika graditi za koju se uzima osnovna slika Java OpenJDK verzije 17. Postavlja se i port na kojem će kontejner slušati i radni direktorij kontejnera. Na kraju postavljaju se naredbe za instalaciju svih potrebnih biblioteka kako bi aplikacija radila i uspješno se pokrenula. Također postavlja se profil pod nazivom *docker*, kako bi se koristila potrebna konfiguracijska datoteka za povezivanje s PostgreSQL bazom podataka.

- Dockerfile za stvaranje React dijela aplikacije

```
FROM node:16 AS build
WORKDIR /app
COPY package.json package-lock.json ./
RUN npm ci
COPY . .
RUN npm run build
FROM nginx:stable-alpine
COPY --from=build /app/dist /usr/share/nginx/html
RUN rm /etc/nginx/conf.d/default.conf
COPY nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 80
```

Programski odsječak 5.6 Primjer izvedbe Dockerfile datoteke

Slično kao u prvoj datoteci postavlja se osnovna slika, radni direktorij i postavljaju se sve naredbe za instalaciju biblioteka. Razlika je što se u ovoj datoteci koristi i druga slika „nginx“ koja je korisna za web server unutar Docker kontejnera. Također postavlja se port na kojem kontejner sluša zahtjeve.

Docker Compose služi kako bi se uspješno pokretale Docker aplikacije koje sadrže više kontejnera. Za rad ove aplikacije koriste se tri kontejnera: za serversku stranu, klijentsku stranu i bazu podataka. Za konfiguriranje aplikacije i kontejnera koristi se YAML datoteka. U ovom radu koristi se *docker-compose.yml* datoteka:

```
version: '3.8'

services:
  db:
    image: postgres
    environment:
```

```
      POSTGRES_DB: zavrsni
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
    ports:
      - "5432:5432"
    volumes:
      - pgdata:/var/lib/postgresql/data

  backend:
    build: ./backend
    ports:
      - "8080:8080"
    environment:
      - SPRING_PROFILES_ACTIVE=docker
    depends_on:
      - db

  frontend:
    build: ./frontend
    ports:
      - "3000:80"

volumes:
  pgdata: {}
```

Programski odsječak 5.7 Primjer izvedbe docker-compose.yml datoteke

Docker-compose.yml sastoji od tri komponente koje čine kontejnere:

- Baza podataka – *db*
- Serverski dio aplikacija – *backend*
- Klijentski dio aplikacije – *frontend*

U prvom dijelu postavlja se *db* kontejner koji služi kao baza podataka. Postavlja se slika *postgres* koja se koristi, varijable za konfiguraciju baze i putanja direktorija za zadržavanje podataka PostgreSQL baze izvan životnog vijeka kontejnera.

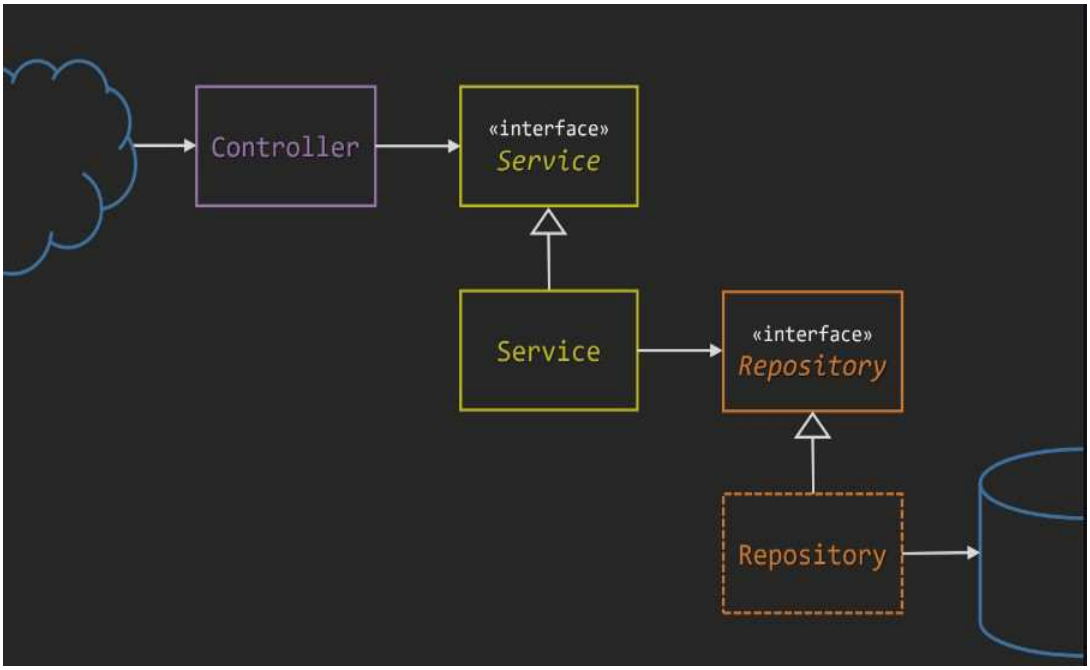
U drugom dijelu postavlja se kontejner koji služi kao serverska strana aplikacije pod nazivom *backend* . Postavlja se naredba Docker-u za izgradnju slike pomoću Dockerfile-a unutar direktorija *./backend*. Postavlja se varijabla za određivanje konfiguracija pri pokretanju Spring Boot aplikacije i ovisnost kontejnera *backend* o kontejneru *db* što će odrediti redoslijed stvaranja kontejnera.

U zadnjem dijelu postavlja se direktorij gdje Docker treba stvoriti sliku za kontejner pod nazivom *frontend* koji služi kao klijentski dio aplikacije. *Volumes* se koristi za zadržavanje podataka u bazi PostgreSQL.

6. Arhitektura aplikacije

6.1 Serverski sloj aplikacije

Serverski dio aplikacije strukturiran je kroz nekoliko slojeva iz razloga kako bi se razdvojila odgovornost komponenti zbog lakšeg razvoja i održavanja aplikacije. Struktura slojeva prati princip MVC (*Model-Pogled-Upravljač*) arhitekture. Sloj pogled nije ostvaren na tradicionalan način da prikazuje podatke korisniku nego je zamijenjen odgovorima prema klijentu u JSON formatu.



Slika 6.1 Arhitektura serverskog sloja [12]

Osnovni slojevi arhitekture su:

- Upravljač (Controller)
- Servisni sloj (Service)
- Repozitorij (Repository)

Svaki od slojeva ima svoje posebne odgovornosti koje služe jasnom razdvajanju funkcionalnosti i modularnosti aplikacije.

6.1.1 Upravljač

Glavna odgovornost upravljača je primanje HTTP zahtjeva od strane klijenta, njihova obrada, te prosljeđivanje servisnom sloju za daljnje upravljanje. U skladu s REST arhitekturom, upravljač ima mogućnost obrade četiri vrste HTTP zahtjeva:

- GET - Koriste se za dohvat podataka iz baze podataka
- POST - Koriste se za stvaranje novih instanci i njihovo spremanje u bazu podataka
- PUT - Koriste se za ažuriranje postojećih instanci
- DELETE - Koriste se za brisanje instanci iz baze podataka

Prije prosljeđivanja zahtjeva na servisni sloj, upravljač provjerava ispravnost ulaznih podataka osiguravajući kompletnost primljenih zahtjeva.

```
1. @PostMapping("/share")
 2.     public ResponseEntity<?> shareMenu(@RequestParam Long menuId, @RequestParam String
sharedWithEmail) {
 3.         try {
 4.             Optional<Menu> menu = menuService.findById(menuId);
 5.             Optional<AppUser> sharedWithUser =
appUserService.findByEmail(sharedWithEmail);
 6.             String message = sharedMenuService.shareMenu(menu.get(),
sharedWithUser.get());
 7.             return ResponseEntity.ok(message);
 8.         } catch (Exception e) {
 9.             return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error
sharing menu: " + e.getMessage());
10.         }
11.     }
12.
13.     @GetMapping("/shared")
14.     public ResponseEntity<?> getSharedMenus(@RequestParam String email) {
15.         try {
16.             Optional<AppUser> user = appUserService.findByEmail(email);
17.             List<MenuDTO> sharedMenus =
sharedMenuService.findSharedMenusByUser(user.get());
18.             return ResponseEntity.ok(sharedMenus);
19.         } catch (Exception e) {
20.             return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error
fetching shared menus: " + e.getMessage());
21.         }
22.     }
```

Programski odsječak 6.1 Primjer izvedbe upravljača

6.1.2 Servisni sloj

Servisni sloj ključna je komponenta u arhitekturi aplikacije i služi kao posrednik između upravljača i repozitorija. Ovaj sloj sadrži poslovnu logiku aplikacije, što ga čini

središtem za obradu podataka. Servisni sloj obrađuje podatke dobivene od upravljača, provjerava njihovu valjanost i provjerava jesu li svi uvjeti za daljnji proces zadovoljeni. Nakon toga servisni sloj komunicira s repozitorijem kako bi dohvatio podatke iz baze podataka ili pohranio napravljene izmjene u bazu.

Korištenje sučelja servisnog sloja odvojeno od implementacije servisnog sloja koristi se iz razloga odvajanja definicije „što servisni sloj radi“ i konkretne implementacije servisnog sloja. Ta vrsta korištenja omogućava lake promjene u aplikaciji ako u međuvremenu dođe do promjene zahtjeva projekta, te se implementacije mogu mijenjati ili zamijeniti bez utjecaja na ostatak aplikacije. Sučelja imaju važnu ulogu u implementaciji umetanja ovisnosti i inverzije kontrole koje su jako bitne u modernim okvirima, a posebno u Spring-u.

6.1.3 Repozitorij

Glavna uloga repozitorija je dohvaćanje, spremanje i upravljanje podacima iz baze podataka. Repozitorij povezuje poslovnu logiku aplikacije s bazom podataka. Prednost je što sav kod za pristup bazi je centraliziran unutar repozitorija s čime se olakšava upravljanje kodom u slučaju greške ili dodavanja funkcionalnosti.

```
 1. public interface SharedMenuRepository extends JpaRepository<SharedMenu, Long> {
 2.     List<SharedMenu> findBySharedWithUser(AppUser user);
 3.     List<SharedMenu> findByMenuId(Long menuId);
 4.
 5.     Optional<SharedMenu> findByMenuAndSharedWithUser(Menu menu, AppUser sharedWithUser);
 6.
 7.     @Modifying
 8.     @Transactional
 9.     @Query("DELETE FROM SharedMenu sm WHERE sm.menu = :menu AND sm.sharedWithUser =
:appUser")
10.     void deleteByMenuAndAppUser(Menu menu, AppUser appUser);
11. }
```

Programski odsječak 6.2 Primjer izvedbe repozitorija

U izradi aplikacije koristio sam sučelje *JpaRepository* iz biblioteke Spring Dana JPA koja pojednostavljuje posao programera i ima predefinirane osnovne metode, što omogućava programeru da se fokusira na poslovnu logiku dok standardne operacije automatski postoje. U slučaju da metoda nije unutar sučelja jednostavno se dodaje uz pomoću SQL upita.

6.1.4 Inverzija kontrole

Inverzija kontrole je programski princip u kojem kontrolu nad tijekom izvršenja programa preuzima radna okolina, a ne glavni program. Ovaj princip se najčešće ostvaruje kroz upotrebu sučelja, gdje komponente aplikacije ne stvaraju i ne upravljaju svojim ovisnostima (npr. objektima koje koriste). Umjesto toga, ove ovisnosti se injektiraju kroz radno okruženje, što u slučaju ovog rada predstavlja Spring. Spring se brine o kreiranju objekata, njihovim ovisnostima, te upravljanju njihovim životnim ciklusom.

Ključna korist ovog principa je smanjenje međuovisnosti među komponentama, što omogućuje veću modularnost i fleksibilnost u aplikaciji. U slučaju promjene zahtjeva, pojedine komponente se mogu lako zamijeniti drugim implementacijama koje ispunjavaju iste ugovore definirane sučeljem, bez potrebe za izmjenama ostalih dijelova aplikacije. [13]

6.1.5 Umetanje ovisnosti

Umetanje ovisnosti je obrazac koji doprinosi smanjenju ovisnosti između komponenata aplikacije. Ovaj obrazac promiče upotrebu apstrakcija (sučelja) umjesto konkretnih implementacija, čime se odvajaju komponente visoke razine (koje obavljaju ključne operacije) od komponenata niže razine (koje pružaju specifične funkcionalnosti). [13]

Apstrakcija se ostvaruje kroz implementaciju sučelja, a polimorfizam igra ključnu ulogu omogućavajući da se konkretne implementacije mogu zamijeniti drugima koje također implementiraju isto sučelje. U kontekstu ovog rada, inverzija ovisnosti implementirana je tako da su sve veze između komponenata uspostavljene preko sučelja, što olakšava zamjenu, testiranje i održavanje komponenata. Primjer umetanja ovisnosti u ovom radu:

```
1. @RestController
2. @RequestMapping("/api/food")
3. @AllArgsConstructor
4. public class FoodController {
5.
6.     private final FoodService foodService;
7.     private final AppUserService appUserService;
8.     private final MealService mealService;
9.
10.    @GetMapping("")
11.    public List<Food> listFood(){
12.        return  foodService.getAllFood();
13.    }
14.    @GetMapping("/search")
15.    public ResponseEntity<List<Food>> searchFoods(@RequestParam String query) {
```

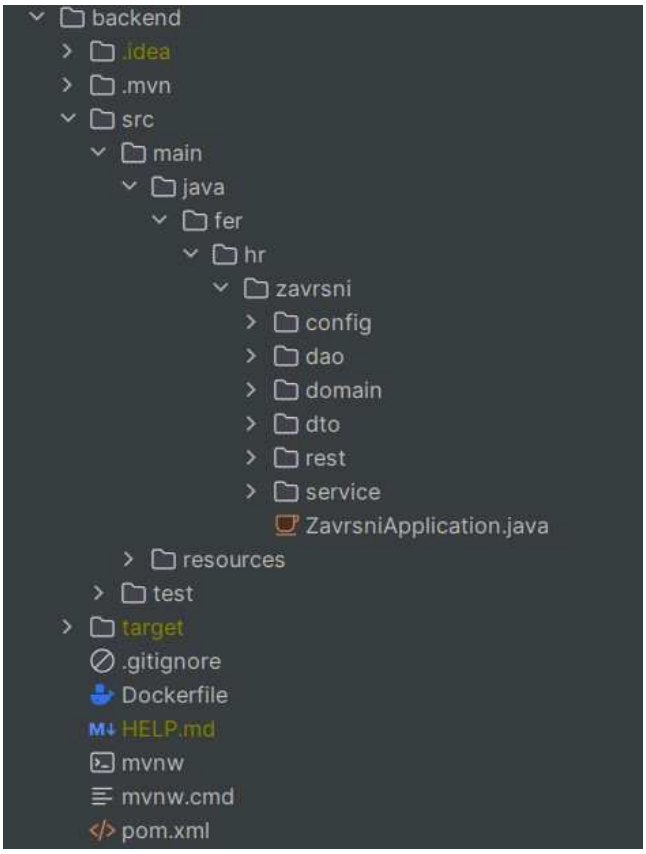
```
16.         List<Food> foods = foodService.findFoodsByQuery(query);
17.         return ResponseEntity.ok(foods);
18.     }
```

Programski odsječak 6.3 Primjer izvedbe umetanja ovisnosti

Razred *FoodController* dobiva tražene ovisnosti *FoodService*, *AppUserService* i *MealService* preko konstruktora klase pomoću umetanja ovisnosti. Ovaj primjer umetanja naziva se konstruktorsko umetanje koje je najčešće korišteno u Spring-u jer osigurava da su sve ovisnosti postavljene prije korištenja objekta. Konstruktor se uz pomoću Lombok anotacije *@AllArgsConstructor* automatski generira uključujući sve finalne atribute klase. Takva vrsta anotacije smanjuje količinu koda u programu i pojednostavljuje kod.

6.1.6 Organizacija serverskog koda

Serverski kod strukturiran je na način koji jasno razdvaja odgovornosti svakog direktorija i omogućava jasnu snalažljivost svakog programera. Svaki direktorij ima definiranu ulogu što olakšava daljnje razvijanje i održavanje koda.



Slika 6.2 Organizacija serverskog koda

Projekt je strukturiran u više direktorija. Direktorij *resources* sadrži konfiguracije datoteke i sve što nije Java programski kod. Konfiguracije datoteke su bitne za postavljanje i komunikaciju aplikacije s ostalim slojevima.

Glavni direktoriji:

- Config - Sadrži konfiguracijske klase koje postavljaju sigurnosne postavke, konfiguraciju OAuth2 autentifikacije i CORS konfiguraciju koja omogućava komunikaciju klijentskog i serverskog sloja
- Dao - Sadrži klase koje upravljaju interakcijama s bazom podataka. U ovom radu te klase predstavljaju repozitoriji
- Domain - Sadrže klase koje definiraju entitete i njihove veze. Veze između entiteta stvorene su koristeći JPA anotacije za mapiranje. Entitet jelo je ovdje povezano s korisnikom i sastojcima koji čine jelo uz pomoću anotacija @OneToMany i @ManyToOne

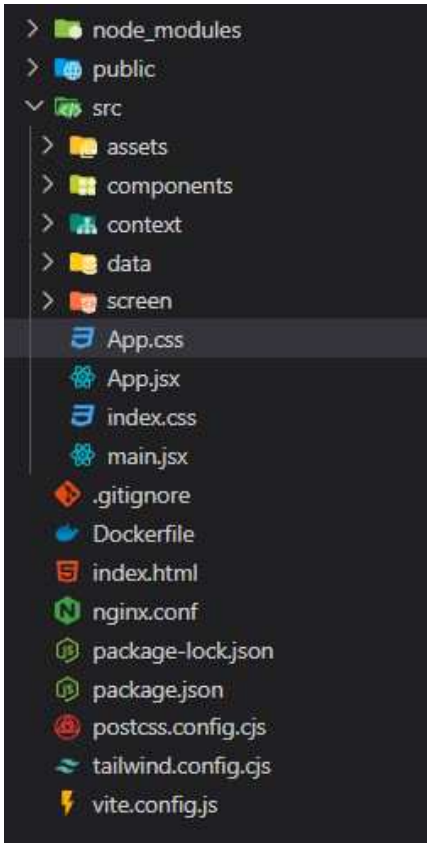
```
1. @Entity
2. @Table(name = "meal")
3. @Data
4. public class Meal {
5.     @Id
6.     @GeneratedValue(strategy = GenerationType.IDENTITY)
7.     private int id;
8.
9.     private String name;
10.
11.    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)
12.    @JoinColumn(name = "meal_id") // This column is in MealFood table
13.    private Set<MealFood> foods;
14.
15.    @ManyToOne
16.    @JoinColumn(name = "user_id")
17.    private AppUser user;
18. }
```

Programski odsječak 6.4 Primjer izvedbe entiteta Meal

- Dto - Sadrži klase koje se koriste za prijenos podataka između serverskog i klijentskog sloja
- Rest - Sadrži klase koje predstavljaju upravljače. Zaduženi su za primanje zahtjeva od klijentskog sloja
- Service - Sadrži klase servisnog sloja čija je uloga implementiranje poslovne logike aplikacije

6.2 Klijentski sloj aplikacije

Klijentski sloj aplikacije strukturiran je s ciljem da se osigura jednostavnost daljnjeg razvoja i održavanja, pritom zadržavajući visoku razinu modularnosti. Direktoriji unutar projekta jasno su organizirani prema odgovornostima, što omogućava brzo snalaženje i razumijevanje strukture aplikacije. Imenovanje direktorija je intuitivno i reflektira ulogu svake komponente unutar klijentskog sloja. Ovako planirana organizacija ne samo da poboljšava efikasnost razvoja, već i pojednostavljuje proces dodavanja novih funkcionalnosti.



Slika 6.3 Organizacija klijentskog koda

Glavni direktorij gdje se nalazi izvorni kod klijentskog sloja i ostali direktoriji je */src*. Bitni direktoriji unutar */src* direktorija su:

- Assets - Služi kao spremište za statičke resurse potrebne aplikaciji

- Components - Služi kao spremište za sve React komponente koje se koriste u više različitih dijelova aplikacije. Ovaj pristup omogućava jednostavnost u razvoju i smanjuje ponavljanje koda
- Context - Sadrži komponente koje koriste React Context API za upravljanje globalnim stanjem aplikacije. Takav način korištenja omogućava ostalim komponentama da pristupe dijeljenim podacima
- Data - Sadrži konfiguracije datoteke ili skripte za upravljanje podacima
- Screen - Sadrži komponente koje se koriste kao glavni zasloni ili rute unutar aplikacije. Najčešće se sastoje od više komponenata iz direktorija *Components*.

Datoteka *App.jsx* je glavna datoteka koja upravlja svim ostalim komponentama za prikazivanje i rutama klijentskog sloja aplikacije.

```
1. function App() {
2.
3.   return (
4.     <AuthProvider>
5.     <Router>
6.     <Routes>
7.         <Route path="/" element={<LandingPage />} />
8.         <Route path="/signin" element={<SignIn />} />
9.         <Route path="/menus" element={<PersonalMenu />} />
10.        <Route path="/menucreate" element={<MenuCreator />} />
11.        <Route path="/todaymeals" element={<TodayMeals />} />
12.        <Route path="/menu-detail/:menuId" element={<MenuUse />} />
13.        <Route path="/dashboard" element={<Dashboard/>} />
14.        <Route path="/report" element={<Reports/>} />
15.        <Route path="/admin-suggest" element={<AdminSuggest />} />
16.        <Route path="/meals" element={<Meals/>} />
17.      </Routes>
18.    </Router>
19.  </AuthProvider>
20.  )
21. }
22.
23. export default App
```

Programski odsječak 6.5 Primjer izvedbe *App.jsx* datoteke

Komponenta `AuthProvider` služi kao kontekst koji omogućava ostalim komponentama unutar njega da saznaju informacije o statusu prijave korisnika. Korištenjem ovog konteksta, komponente mogu pristupiti podacima o tome je li korisnik prijavljen ili nije, što im omogućava da prilagode svoje ponašanje i prikazane sadržaje na temelju korisnikovog statusa. Ovaj pristup centralizira upravljanje statusom korisnika i olakšava njihovu distribuciju kroz aplikaciju.

Komponenta `Router` ključna je za upravljanje navigacijom unutar aplikacije. `Router` sluša promjene u URL-u te na temelju definiranih ruta (`Route`) odlučuje koja komponenta

će se prikazati. To znači da kad korisnik unese određeni URL ili je usmjeren na link unutar aplikacije, `Router` upravlja što će se renderirati bez potrebe za učitavanjem nove stranice s poslužitelja.

`Routes` je kontejner koji obuhvaća sve `Route` komponente, djelujući kao centralno mjesto za definiranje veza između URL putanja i pripadajućih React komponenti. Svaka `Route` komponenta unutar `Routes` kontejnera specificira putanju i element koji će se renderirati kada se ta putanja posjeti. Ovaj pristup omogućava organizirano i lako upravljanje rutama u aplikaciji.

6.2.1 React Context

```
1. const AuthContext = createContext();
 2.
 3. export const useAuth = () => useContext(AuthContext);
 4.
 5. export const AuthProvider = ({ children }) => {
 6.     const [isLoggedIn, setIsLoggedIn] = useState(false);
 7.
 8.     const refreshAuthStatus = async () => {
 9.          try {
10.             const response = await axios.get('http://localhost:8080/api/user/status', {
withCredentials: true, headers: { "Access-Control-Allow-Credentials": "true" }});
11.             setIsLoggedIn(response.data.isLoggedIn);
12.         } catch (error) {
13.             console.error("Greška pri provjeri statusa prijave", error);
14.             setIsLoggedIn(false);
15.         }
16.     };
17.
18.     useEffect(() => {
19.         refreshAuthStatus();
20.     }, []);
21.
22.     return (
23.         <AuthContext.Provider value={{ isLoggedIn, setIsLoggedIn, refreshAuthStatus }}>
24.             {children}
25.         </AuthContext.Provider>
26.     );
27. };
28.
```

Programski odsječak 6.6 Primjer izvedbe AuthContext.jsx datoteke

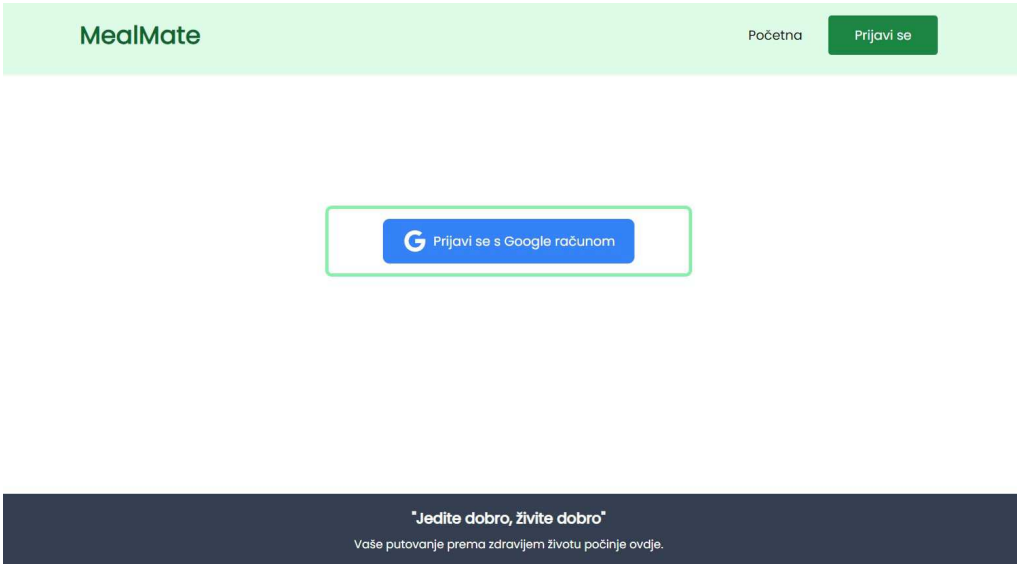
Na početku se definira novi kontekst uz pomoć funkcije iz Reacta i omogućava pristup kontekstu ostalim komponentama korištenjem Custom Hook-a `useContext`. Zatim se definira komponenta koja upravlja stanjem prijave korisnika uz pomoć stanja `isLoggedIn` i sadrži funkciju za dohvat statusa korisnika uz koju ostale komponente mogu saznati jeli korisnik prijavljen ili nije. [14]

7. Prikaz rješenja

Na početku korisniku se prikazuje početni zaslon s osnovnim informacijama o web aplikaciji i s gumbom za daljnju prijavu u aplikaciju.

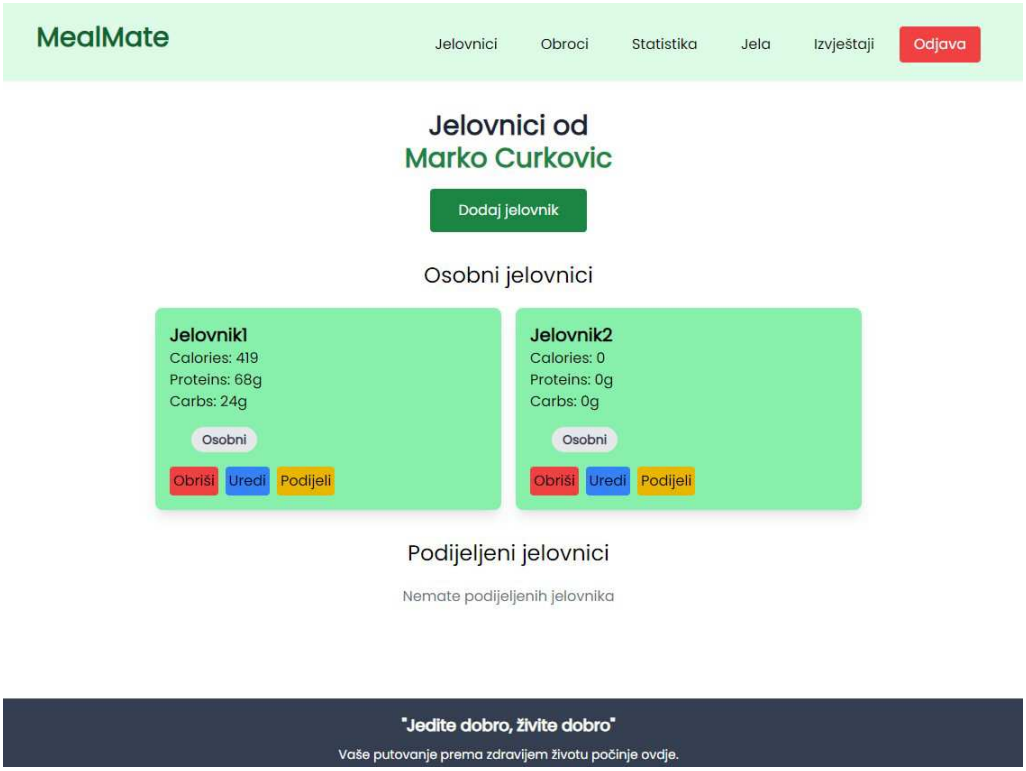
Slika 7.1 Početni zaslon

Nakon pritiska gumba za prijavu pojavljuje se korisniku opcija za nastavak prijave uz pomoć Google-a.



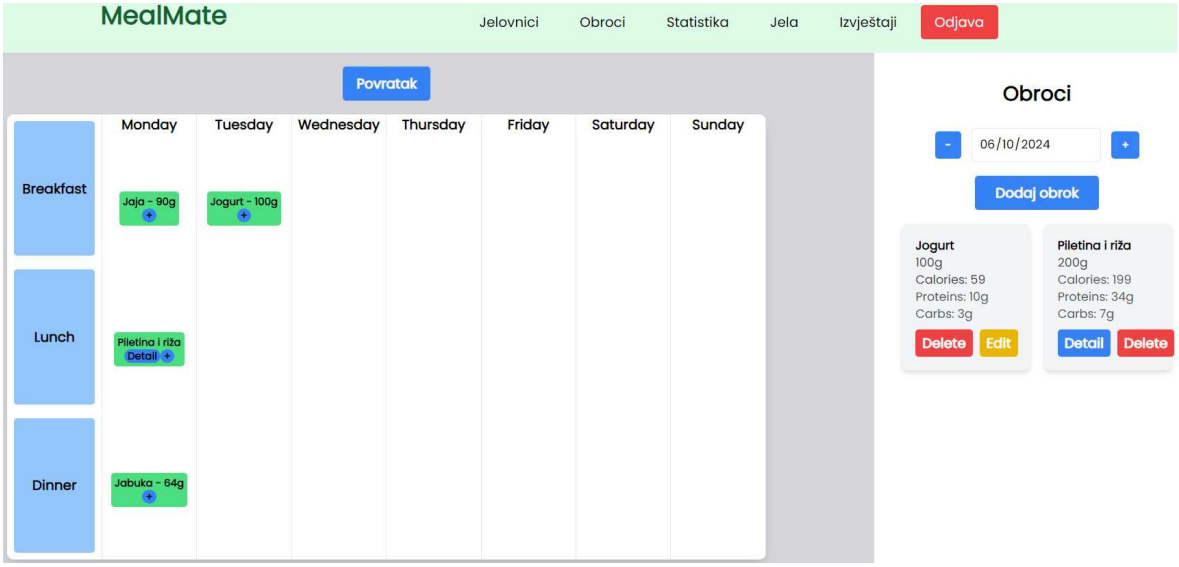
Slika 7.2 Google prijava

Nakon uspješne prijave korisniku se prikazuju njegovi osobni i podijeljeni jelovnici. Također prikazuju mu se i opcije za dodavanje jelovnika i navigacijska traka.

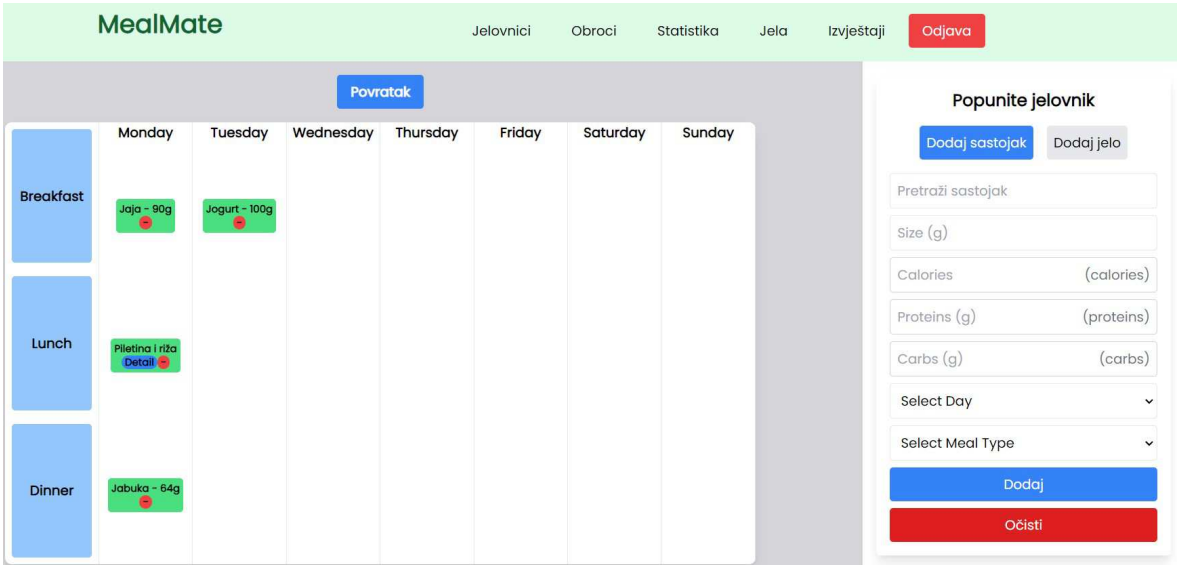


7.3 Stranica s jelovnicima

Kada korisnik stisne na jedan od jelovnika ima opciju dodavati jela i sastojke u svoje obroke. Na kartici jelovnika ima opcije za brisanje jelovnika, uređivanje gdje onda može dodavati, brisati i uređivati jela i sastojke te na kraju i podijeliti odabrani jelovnik s drugim korisnikom.

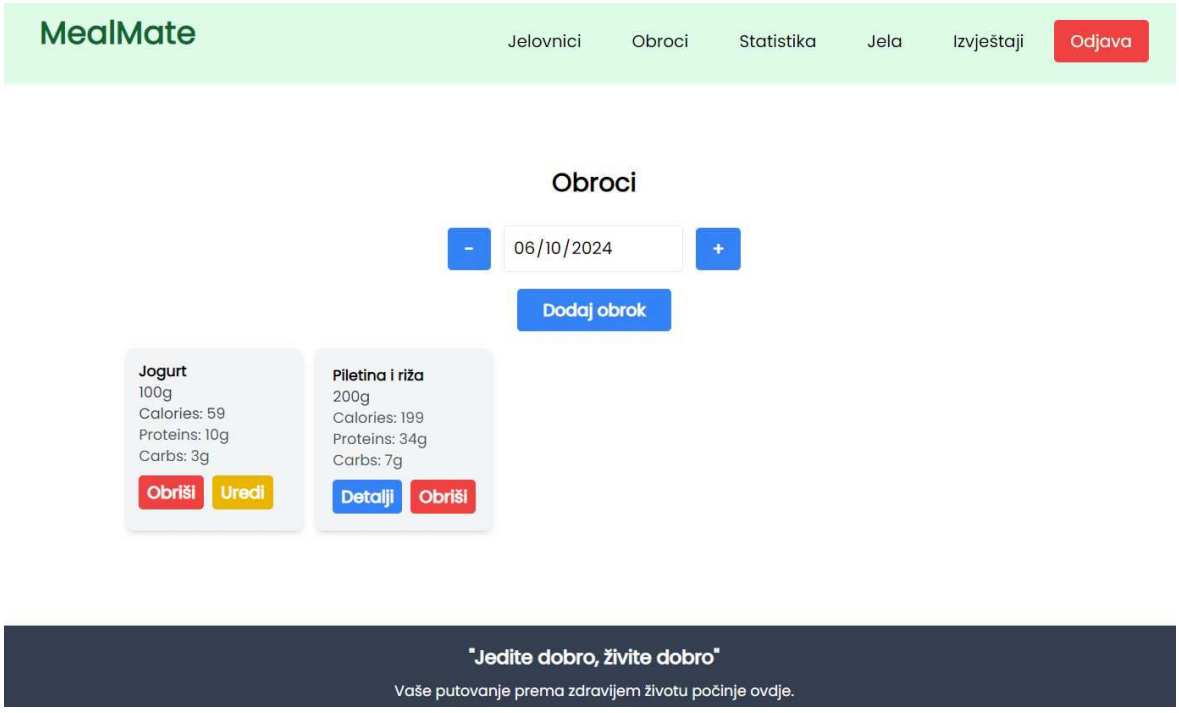


Slika 7.4 Stranica za korištenje jelovnika



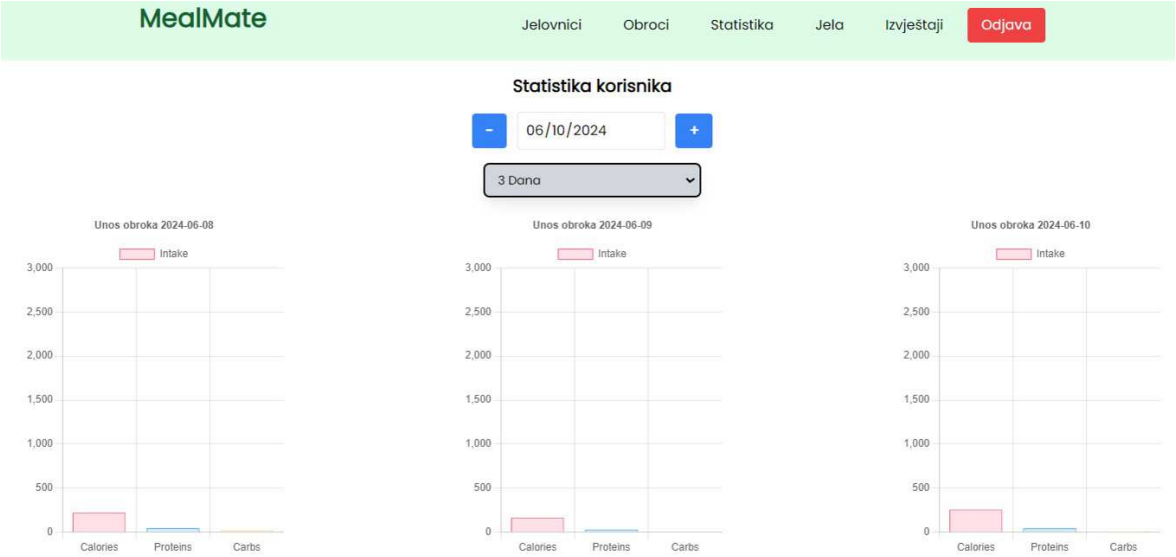
Slika 7.5 Stranica za uređivanje jelovnika

Odabirom pojma *Obroci* na navigacijskoj traci korisniku se prikazuju osobni obroci koje je dodao i ima mogućnost birati dan koji želi pregledati.



Slika 7.6 Stranica s obrocima

Odabirom pojma *Statistika* korisniku se prikazuju dijagrami s numeričkim vrijednostima unesenih kalorija, proteina i ugljikohidrata po danima. Korisnik ima mogućnost odabira prikaza više dijagrama u isto vrijeme.

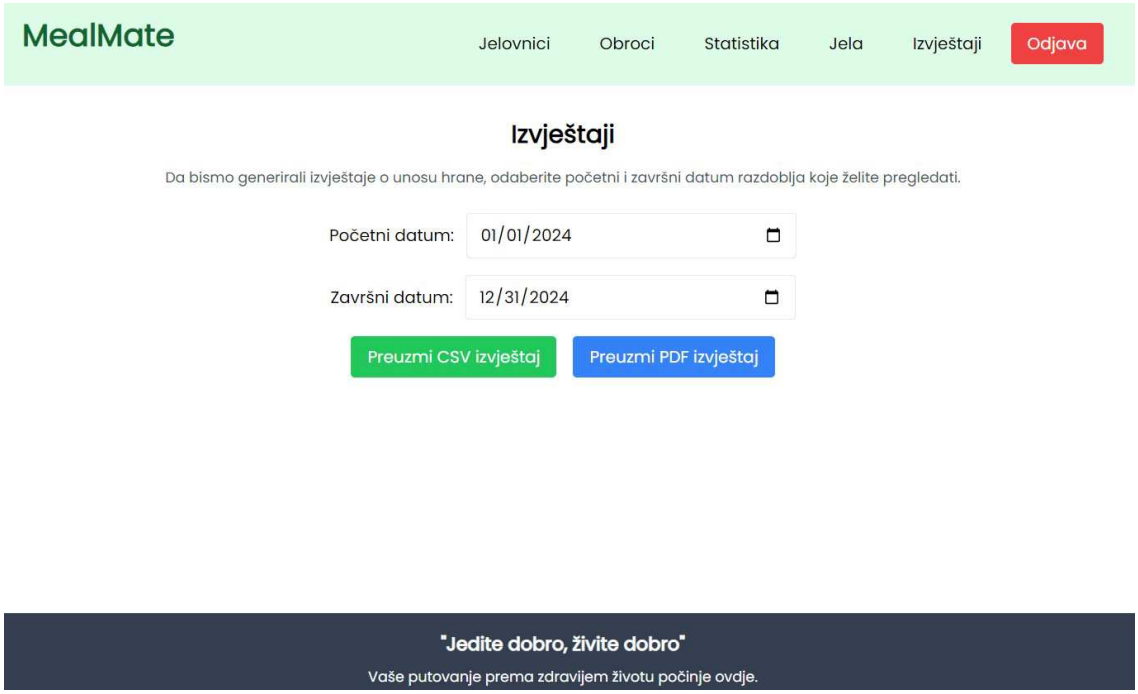


Slika 7.7 Statistika obroka

Odabirom pojma *Jela* korisnik može pregledati sva svoja jela koja je stvorio i ima mogućnost koristiti. Na istom zaslonu ima mogućnost stvaranja novih jela i uređivanje postojećih.

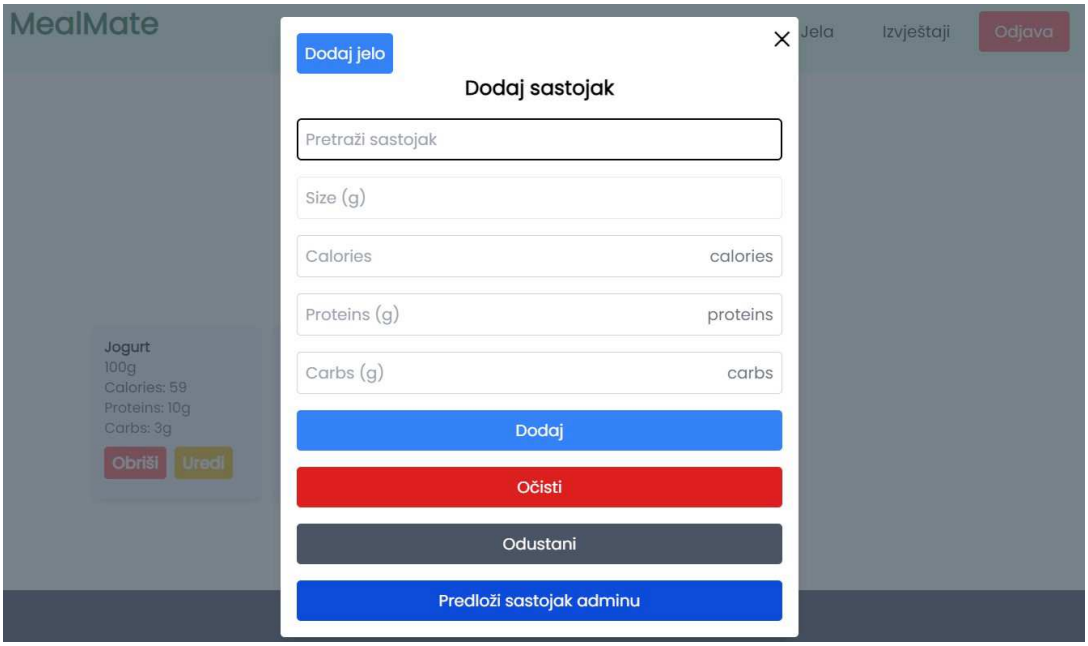
Slika 7.8 Prikaz jela

Odabirom na opciju *Izvještaji* korisnik ima mogućnost preuzimanja jednostavnih CSV i PDF izvještaja o dodanim obrocima i jelima. Također ima mogućnost biranja datuma između kojih želi da se izvještaj generira.

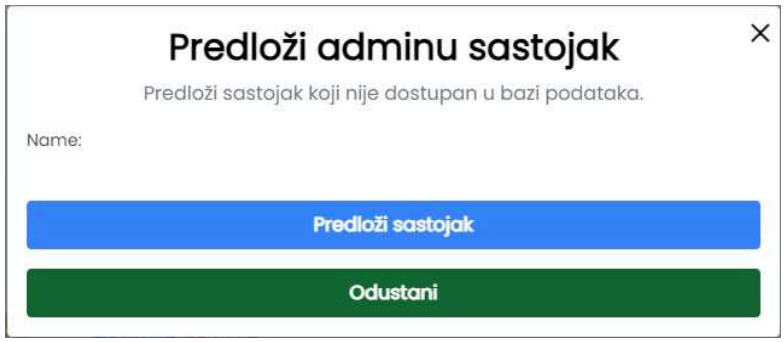


Slika 7.9 Prikaz stranice za izvještaje

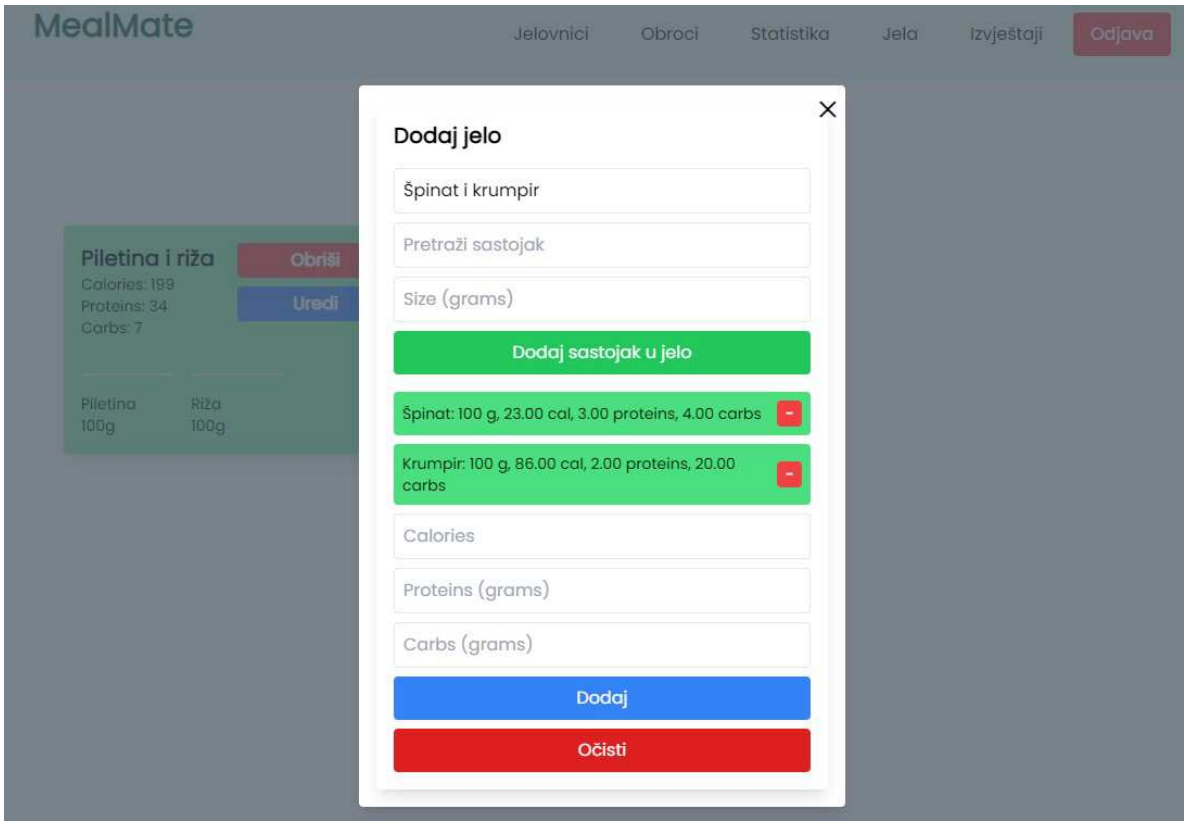
Pri dodavanju novih obroka korisnik ima mogućnost predlaganja novih sastojaka administratoru gdje mu se zatim otvara nova forma za upis imena željenog sastojka.



Slika 7.10 Forma za unos novih obroka



Slika 7.11 Forma za predlaganje novog sastojka



Slika 7.12 Stvaranje novih jela

Kod unosa novih jela korisnik upisuje ime jela, zatim dodaje željene sastojke u jelo i na kraju stvara željeno jelo.

8. Upute za pokretanje

Ovo poglavlje objašnjava postupak pokretanja web aplikacije na lokalnom računalu. Kako bi pokretanje bilo uspješno, računalo mora imati instalirano dva ključna alata: **Maven** i **Docker.** Maven je alat za automatizaciju izrade projekta, koji se u ovom pokretanju koristi za generiranje izvršne .jar datoteke Java programa. Jar datoteka je potrebna za stvaranje Docker slike koja omogućava pokretanje Java aplikacije unutar kontejnera. Docker je potreban za pokretanje stvorenih kontejnera. Kontejnerizacija omogućava rad aplikacije izolirano od ostalih aplikacija na računalu. Nakon dva ključna alata, prvi korak je odabrati direktorij u kojem će se nalaziti izvorni kod aplikacije. Zatim je bitno otvoriti naredbeni redak u odabranom direktoriju i pokrenuti naredbu:

```
git clone https://github.com/Curko15/ZavrsniRad.git
```

Nakon toga direktorij će sadržavati izvorni kod aplikacije. Izvorni kod se sadrži od glavnog direktorija *ZavrsniRad* u kojem se nalaze YAML datoteka i direktoriji *backend* i *frontend*. Potrebno je otvoriti naredbeni redak unutar direktorija backend i pokrenuti naredbu:

```
mvn clean package -Dspring.profiles.active=local
```

koja će stvoriti izvršnu .jar datoteku i postaviti postavku da se Spring Boot pokreće s *application-local.properties* konfiguracijskom datotekom kako bi se mogla izvršna datoteka stvoriti uz pomoć H2 baze podataka. Kada je izvršna datoteka uspješna stvorena potrebno se vratiti u naredbenom retku na glavni direktorij *ZavrsniRad* gdje se nalaze direktoriji i *docker-compose.yml* datoteka te pokrenuti naredbu:

```
docker-compose up –build
```

koja će stvoriti Docker slike i pokrenuti kontejnere. Nakon uspješno stvorenih i pokrenutih kontejnera treba otići u tražilicu i upisati *localhost:3000* te bi se aplikacija trebala prikazati. Sažetak uputa:

1. U željeni direktorij pokrenuti naredbu:

   ```
   git clone https://github.com/Curko15/ZavrsniRad.git
   ```

2. Unutar direktorija *backend* pokrenuti naredbu:

   ```
   mvn clean package -Dspring.profiles.active=local
   ```

3. Unutar direktorija *ZavrsniRad* gdje se nalazi *docker-compose.yml* datoteka pokrenuti naredbu:

   ```
   docker-compose up --build
   ```

4. Pretražiti u tražilici *localhost:3000*

Zaključak

U ovom radu prikazan je jedan od načina izrade web aplikacije za praćenje prehrane. Odluka za korištenje Spring Boot-a i React-a pokazala se kao vrlo dobra zbog jednostavnosti korištenja i mogućnosti lako povezivanja komponenti. Prednost React-a je u tome što su komponente podijeljene prema odgovornostima, što olakšava lociranje grešaka i njihovo ispravljanje.

Serverski sloj napravljen u Spring Boot-u služi kao programsko sučelje i može se lako prilagoditi za izradu drugih klijentskih slojeva, poput mobilne aplikacije za praćenje prehrane. Prednost Spring Boot-a leži u njegovoj sposobnosti jednostavnog upravljanja podacima, kao i automatskoj konfiguraciji, što omogućava programerima da se fokusiraju na ispunjavanje specifičnih zahtjeva aplikacije.

Aplikacija nudi mnoge mogućnosti za proširenje, a trenutni rad predstavlja odličnu osnovu za daljnji razvoj. Mogu se dodati prijave s drugim vanjskim poslužiteljima kao što je *Facebook*, te omogućiti skeniranje barkodova sastojaka kako bi korisnici mogli brže pretraživati i dodavati sastojke.

Izrada web aplikacije uz korištenje Spring Boot-a, React-a, PostgreSQL-a i Docker-a pružila mi je detaljan uvid u stvaranje web aplikacija i iskustvo u korištenju različitih tehnologija koje će mi biti korisne u budućim projektima.

Literatura

[1] *Hrvatski zavod za javno zdravstvo*, Poveznica: https://www.hzjz.hr/ (zadnji pristup: lipanj 2024.)

[2] *MyFitnessPal*, Poveznica: https://www.myfitnesspal.com/ (zadnji pristup: lipanj 2024.)

[3] *Yazio*, Poveznica: https://www.yazio.com/en/calorie-counter (zadnji pristup: lipanj 2024.)

[4] *Spring Boot*, Poveznica: https://spring.io/projects/spring-boot (zadnji pristup: lipanj 2024.)

[5] *Java*, Poveznica: https://www.ibm.com/topics/java (zadnji pristup: lipanj 2024.)

[6] Maven, Poveznica: https://maven.apache.org/what-is-maven.html (zadnji pristup: lipanj 2024.)

[7] *PostgreSQL*, Poveznica: https://www.postgresql.org/about/ (zadnji pristup: lipanj 2024.)

[8] *React*, Poveznica: https://legacy.reactjs.org/docs/getting-started.html (zadnji pristup: lipanj 2024.)

[9] *Vite*, Poveznica: https://vitejs.dev/guide/ (zadnji pristup: lipanj 2024.)

[10] *Tailwind*, Poveznica: https://v2.tailwindcss.com/docs (zadnji pristup: lipanj 2024.)

[11] *Docker*, Poveznica: https://docs.docker.com/guides/ (zadnji pristup: lipanj 2024.)

[12] Slika preuzeta iz prezentacije, Poveznica: https://gitlab.com/hrvojesimic/progi-project-teams-backend/-/tree/master?ref_type=heads (zadnji pristup: lipanj 2024.)

[13] *Dependency injection and inversion of control*, Poveznica: https://www.geeksforgeeks.org/spring-difference-between-inversion-of-control-and-dependency-injection/ (zadnji pristup: lipanj 2024.)

[14] *Context*, Poveznica: https://legacy.reactjs.org/docs/context.html (zadnji pristup: lipanj 2024.)

Sažetak

Naslov:

Web-aplikacija za praćenje prehrambenih navika i zdrave prehrane

Sažetak:

U sklopu završnog rada izrađena je web-aplikacija za praćenje prehrambenih navika i zdrave prehrane. Aplikacija je realizirana korištenjem Java okvira Spring Boot i React Javascript biblioteke. Za bazu podataka je upotrijebljen PostgreSQL, a aplikacija je kontejnerizirana uz pomoć Docker-a radi lakšeg pokretanja.

Web-aplikacija omogućava korisnicima prijavu putem Google računa, kreiranje vlastitih jelovnika i jela, dodavanje obroka, pregled nutritivnih vrijednosti unesenih obroka te preuzimanje jednostavnih izvještaja o prehrambenim unosima. Ova aplikacija nudi sveobuhvatan alat za praćenje i poboljšanje prehrambenih navika korisnika.

Ključne riječi: web-aplikacija, praćenje prehrane, Spring Boot, React, PostgreSQL, Docker

Summary

Title:

A web application for monitoring food habits and healthy eating

Summary:

As part of this thesis, a web-application for monitoring food habits and healthy eating was developed. The application is implemented using the Java Spring Boot framework and the React Javascript library. PostgreSQL is used for the database, and the application is containerized with Docker for easier deployment.

The web application allows users to log in with Google account, create their own menus and dishes, add meals, review the nutritional values of the entered meals, and download simple reports on their dietary intake. This application provides a comprehensive tool for monitoring and improving users dietary habits.

Key words: web-application, nutrition tracking, Spring Boot, React, PostgreSQL, Docker