Razvoj klijentskog dijela web aplikacije za fitness

Pušeljić, Mirna

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

MIRNA PUŠELJIĆ

RAZVOJ KLIJENTSKOG DIJELA WEB APLIKACIJE ZA FITNESS

ZAVRŠNI RAD

Pula, rujan 2024. godine

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

MIRNA PUŠELJIĆ

RAZVOJ KLIJENTSKOG DIJELA WEB APLIKACIJE ZA FITNESS

ZAVRŠNI RAD

JMBAG: 0303103382, redoviti student
Studijski smjer: Informatika
Kolegij: Web aplikacije
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: doc. dr. sc. Nikola Tanković

Pula, rujan 2024. godine

IZJAVA O KORIŠTENJU AUTORSKOGA DJELA

Ja, Mirna Pušeljić, dajem odobrenje Sveučilištu Jurja Dobrile u Puli, nositelju prava korištenja, da moj završni rad pod nazivom „Razvoj klijentskog dijela web aplikacije za fitness“ upotrijebi da tako navedeno autorsko djelo objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te preslika u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu sa Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

Potpis

Mirna Pušeljić

U Puli, rujan 2024. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Mirna Pušeljić, kandidat za prvostupnika informatike ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljeni način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student
Mirna Pušeljić

U Puli, rujan 2024. godine

SAŽETAK

Ovaj završni rad bavi se razvojem klijentskog dijela web aplikacije za fitness pod nazivom „ReadySetGym“. Cilj aplikacije je pružiti korisnicima intuitivan i koristan alat za poboljšanje njihove tjelesne forme, praćenja napretka u vježbanju te promicanje zdravih životnih navika. Aplikacija omogućuje personalizaciju treninga, praćenje prehrane te mogućnost interakcija s drugim korisnicima aplikacije. Razvoj klijentskog dijela obuhvaća korištenje tehnologija poput Vue3 s Composition API, Pinia Store, Firebase za pohranu, te GitHub za verzioniranje. Kroz detaljan prikaz procesa razvoja, uključujući dizajn, implementaciju i testiranje, ovaj rad pruža uvid u izazove i rješenja pri razvoju modernih web aplikacija za fitness.

Ključne riječi: fitness, web aplikacija, Vue3, Composition API, Pinia Store, Firebase, praćenje napretka, peronalizacija treninga

ABSTRACT

This thesis focuses on the development of the client-side of a fitness web application named „ReadySetGym“. The aim of the application is to provide users with an intuitive and useful tool for improving their physical fitness, tracking exercise progress, and promoting healthy lifestyle habits. The application allows for workout personalization, nutrition tracking, and participation in an online community. The development of the client-side involves using technologies such as Vue3 with Composition API, Pinia Store, Firebase for storag, and GitHub for version control. Through a detailed overview of the development process, including design, implementation, and testing, this thesis provides insight into the challenges and solutions in developing modern fitness web applications.

Keywords: fitness, web application, Vue3, Composition API, Pinia Store, Firebase, progress tracking, workout personalization

SADRŽAJ

1. UVOD

Razvoj klijentskog dijela web aplikacije za fitness predstavlja ključan korak u kreiranju sveobuhvatne digitalne platforme koja podržava unapređenje tjelesne forme i zdravlja korisnika. Ovaj završni rad fokusira se na izradu klijentskog dijela web aplikacije pod nazivom „ReadySetGym“. Cilj aplikacije je omogućiti korisnicima intuitivan i efikasan alat za poboljšanje njihove fizičke kondicije, praćenje napretka u vježbanju i promicanje zdravih životnih navika.

„ReadySetGym“ korisnicima nudi personalizirane treninge za specifične dijelove tijela, uz detaljne upute za izvođenje vježbi. Ključna značajka aplikacije je mogućnost prilagodbe treninga individualnim sposobnostima i ciljevima korisnika, uz mogućnost pohrane personaliziranih planova za kasniju upotrebu. Osim personaliziranih treninga, aplikacija uključuje alate za praćenje prehrane, pružajući korisnicima preporuke za uravnoteženu prehranu koja će podržavati njihove fitness ciljeve. Jedna od ključnih značajki „ReadySetGym“ bit će chat s ostalim korisnicima unutar aplikacije. Korisnici će moći dijeliti svoja iskustva, postignuća i međusobno se motivirati, stvarajući tako podršku i osjećaj zajedništva u svom fitness putovanju.

Razvoj klijentskog dijela aplikacije izveden je koristeći napredne tehnologije poput Vue3 s Composition API, Pinia Store za upravljanje stanjem aplikacije, te Firebase za pohranu multimedijskih sadržaj. Verzioniranje koda omogućeno je putem GitHub platoforme, dok se za testiranje koristi jedinično testiranje i integracijski testovi.

Ovaj rad detaljno prikazuje sve faze razvoja klijentskog dijela „ReadySetGym“ aplikacije, uključujući izbor tehnologija, dizajn korisničkog sučelja, implementaciju funkcionalnosti, te testiranje i optimizaciju. Cilj je pružiti dublji uvid u kompleksnost i izazove povezane s razvojem ovakvih aplikacija te ponuditi praktična rješenja za njihovo uspješno ostvarenje.

Poslužiteljski dio ove aplikacije izradio je student Luka Benković s Fakulteta informatike u Puli također u sklopu završnog rada.

2. IZRADA PROTOTIPA

U fazi izrade prototipa za web aplikaciju „ReadySetGym“ korišten je alat Figma, koji omogućava brzo i efikasno kreiranje vizualnih prikaza korisničkog sučelja. Figma je odabrana zbog svojih naprednih funkcionalnosti za kolaboraciju, jednostavnog korisničkog sučelja i mogućnosti kreiranja interaktivnih prototipova.

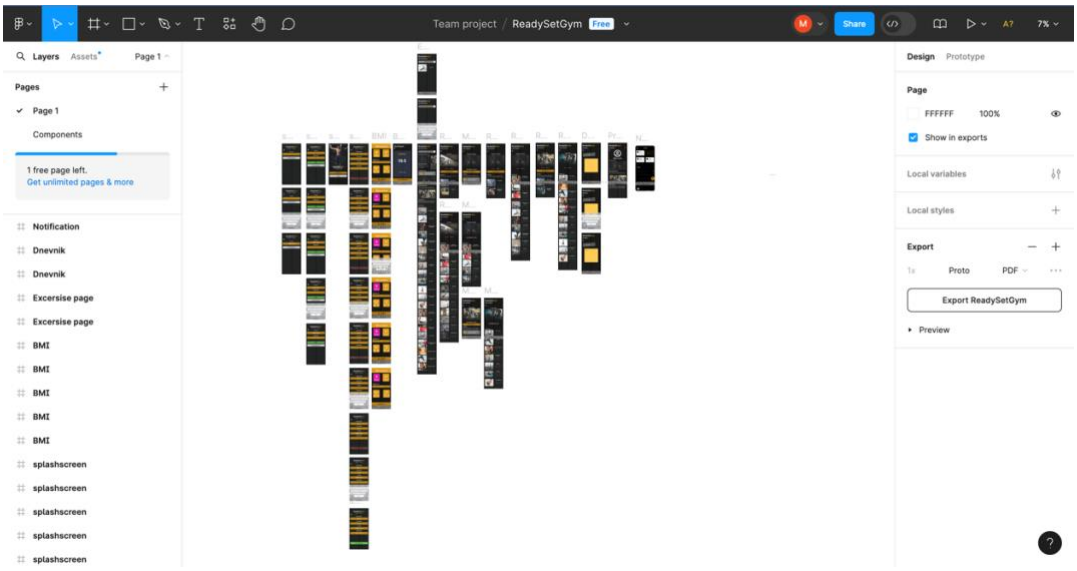
Cilj izrade prototipa bio je razviti vizualno privlačan i funkcionalan prikaz svih ključnih elemenata aplikacije prije početka implementacije. Prototip je poslužio kao vodič za daljnji razvoj i omogućio testiranje korisničkog iskustva (UX) kako bi se identificirali eventualni problemi i unaprijedili dizajn prije pisanja koda.

2.1. KORACI U IZRADI PROTOTIPA

1. Analiza zahtjeva: Prvi korak je definiranje ciljeva aplikacije, ključnih značajki i korisničkih potreba.
2. Wireframe: Izrađeni su osnovni wireframe-ovi kako bi se definirala struktura stranica i raspored elemenata. Wireframe-ovi su pomogli u planiranju informacijske arhitekture i korisničkih tokova.
3. Vizualni dizajn: U ovoj fazi dodani su grafički elementi, boje, tipografija i drugi vizualni elementi koji doprinose estetskom izgledu aplikacije.
4. Interaktivni prototip: Kreiran je interaktivni prototip koji simulira stvarno korisničko iskustvo. Ovaj prototip omogućava testiranje navigacije i interakcija unutar aplikacije, čime se osigurava da su svi korisnički tokovi intuitivni i funkcionalni.

Svi koraci u izradi prototipa preuzeti su iz literaturnog izvora [1].

Na sljedećoj slici prikazan je cjelokupni prototip aplikacije „ReadySetGym“ izrađen u Figmi



Slika 1. Izgled korisničkog sučelja u Figmi

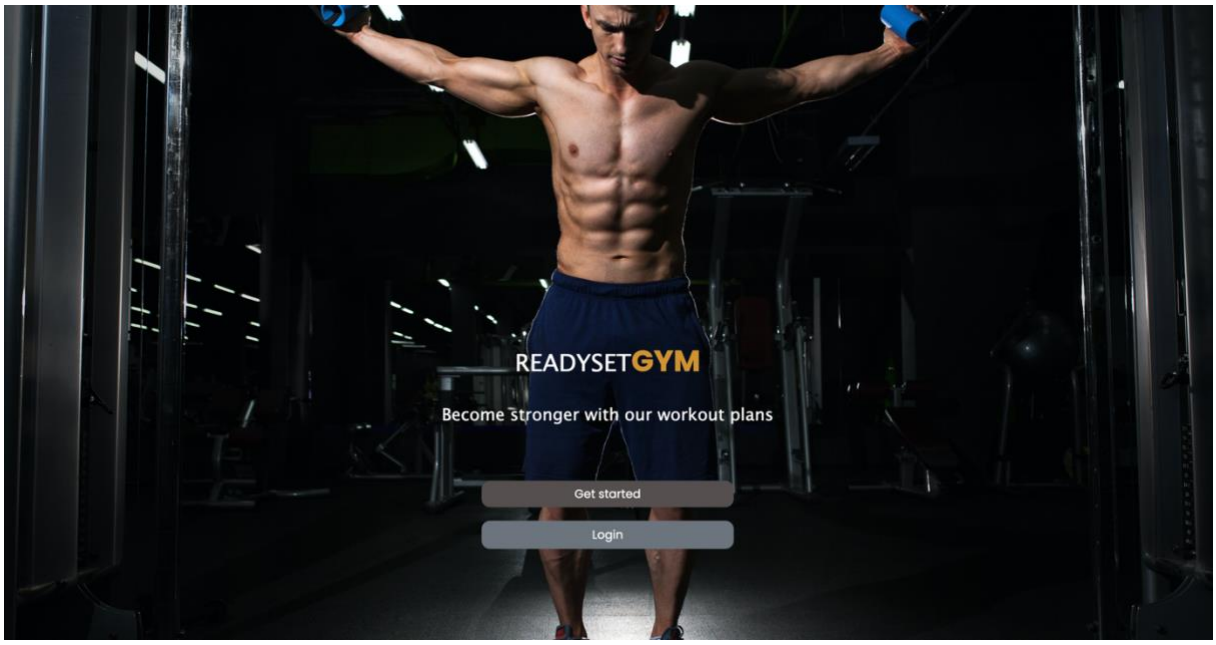
Izrada prototipa je bila ključna faza u razvoju aplikacije, omogućujući vizualizaciju konačnog proizvoda i osiguravajući da su svi dizajnerski elementi usklađeni s korisničkim očekivanjima. Kroz iterativni proces dizajna i testiranja, prototip je postao čvrsta osnova za daljnju implementaciju aplikacije.

3. PRIKAZ FUNKCIONALNOSTI

Aplikacija sadrži nekoliko funkcionalnosti koje omogućuju korisnicima registraciju, prijavu i navigaciju kroz aplikaciju.

3.1. POČETNA STRANICA

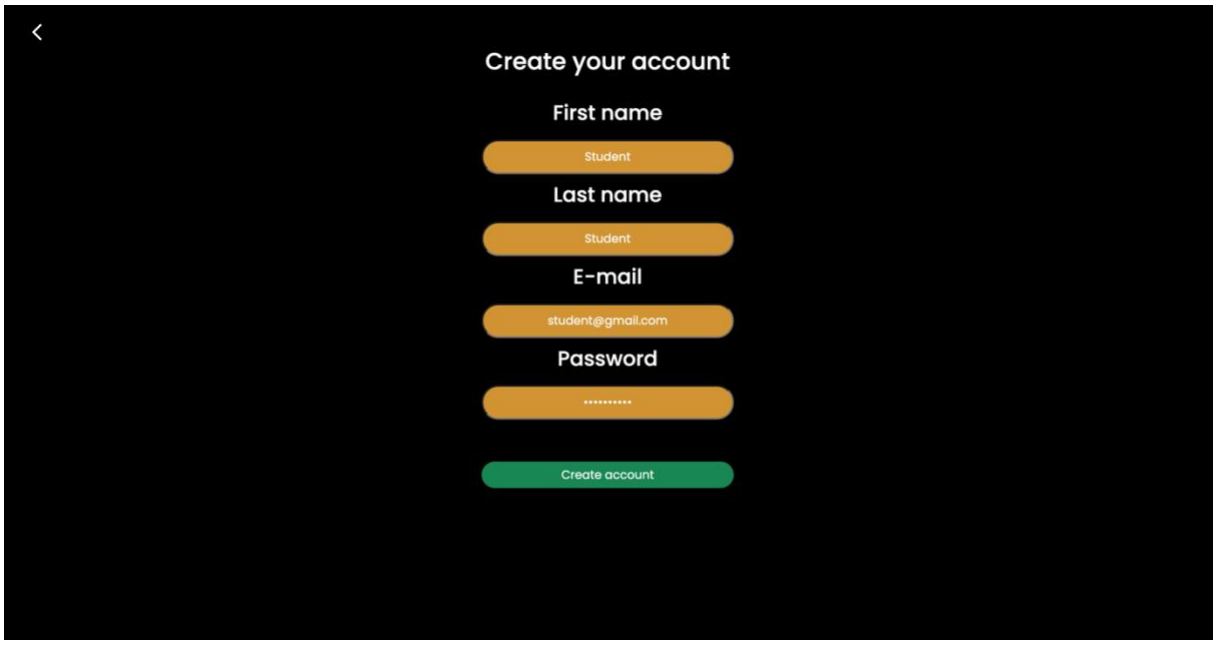
Početna stranica je prva stranica koju korisnici vide kada pristupe aplikaciji. Dizajnirana je kako bi korisnicima pružila atraktivan i jednostavan ulaz u aplikaciju. Korisnicima su na raspolaganju dva gumba: jedan za registraciju i drugi za prijavu postojećih korisnika.



Slika 2. Početna stranica

3.2. REGISTRACIJA KORISNIKA

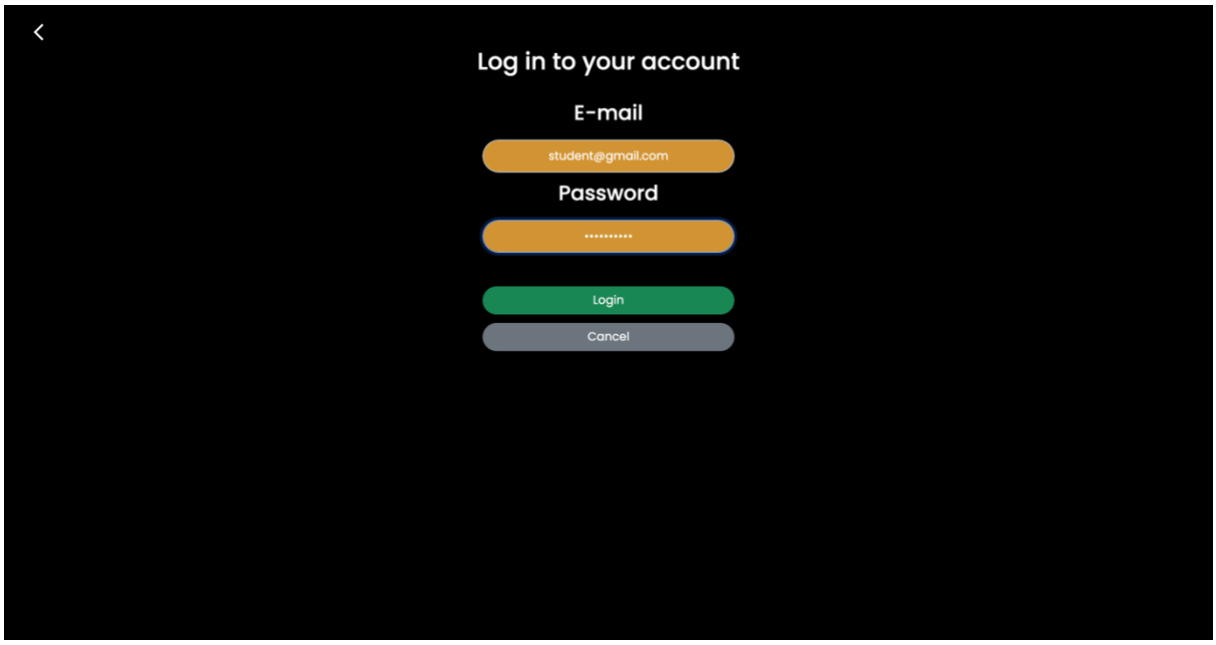
Stranica za registraciju omogućuje novim korisnicima da kreiraju račun. Korisnici unose svoje osobne podatke, uključujući ime, prezime, email adresu i lozinku. Nakon unosa svih potrebnih podataka, korisnici mogu kliknuti na gumb za kreiranje računa. Uspješnom registracijom, korisnici se preusmjeravaju na stranicu za prijavu.



Slika 3. Stranica registracije

3.3. PRIJAVA KORISNIKA

Stranica za prijavu omogućuje postojećim korisnicima da pristupe svojim računima. Korisnici unose svoju email adresu i lozinku te se prijavljuju na gumb za prijavu. Ako su podaci točni, korisnici se preusmjeravaju na glavnu stranicu aplikacije. U slučaju neuspješne prijave, korisnici dobivaju poruku o grešci koja ih obavještava o problemu s prijavom.

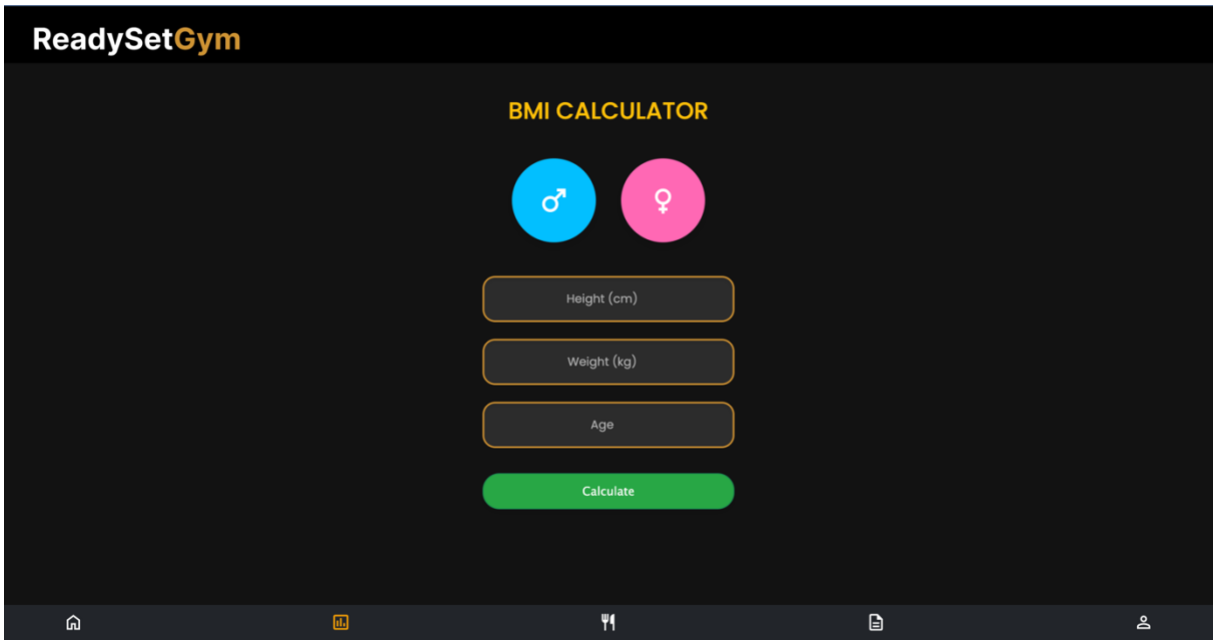


Slika 4. Stranica za prijavu korisnika

3.4. BMI KALKULATOR

Funkcionalnost BMI kalkulatora omogućuje korisnicima da izračunaju svoj indeks tjelesne mase na temelju unesenih podataka. Korisnici najprije odabiru svoj spol, a zatim unose osnovne podatke kao što su tjelesna težina, visina i dob. Nakon unosa svih potrebnih informacija, korisnici mogu kliknuti na gumb za izračunavanje BMI-a.

Rezultat izračuna prikazuje indeks tjelesne mase korisnika, koji se potom uspoređuje s međunarodno prihvaćenih kategorijama (npr. pothranjenost, normalna težina, prekomjerna težina). Na temelju dobivenog BMI rezultata, korisnici dobivaju odgovarajuću povratnu informaciju koja ih obavještava o njihovoj kategoriji tjelesne mase. Ova povratna informacija može poslužiti kao smjernica za daljnje korake prema očuvanju ili poboljšanju zdravlja korisnika.

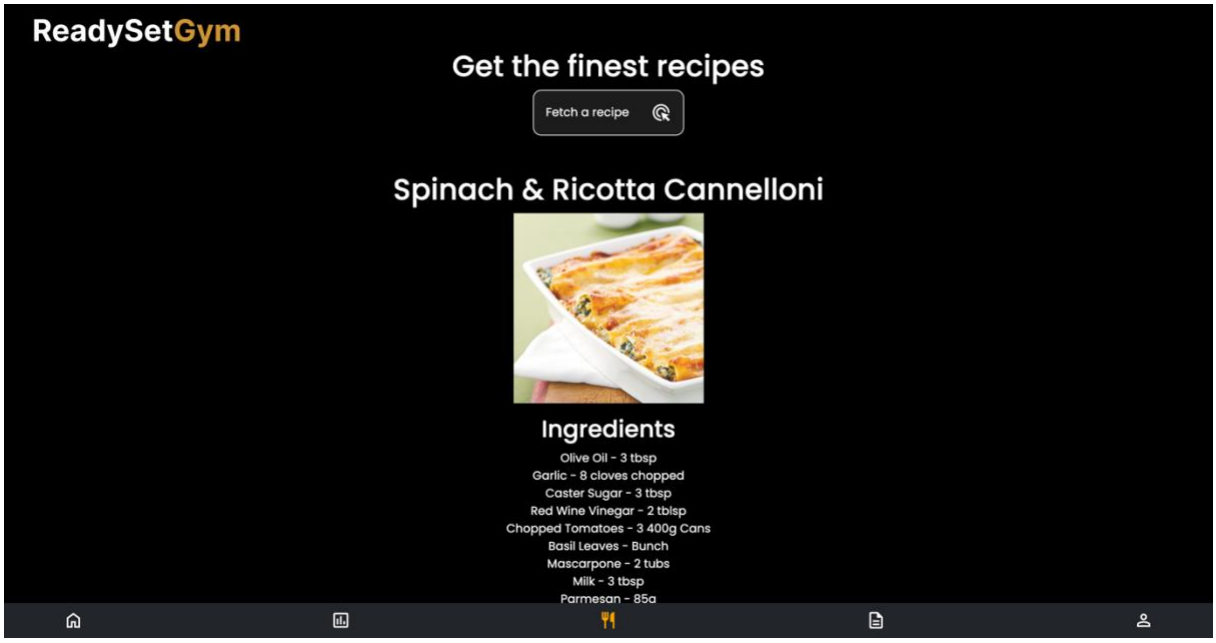


Slika 5. Stranica za izračun BMI-a

3.5. PRIKAZ RECEPATA

Funkcionalnost za prikaz recepata omogućuje korisnicima pregled kulinarskih recepata unutar aplikacije. Svaki recept uključuje nekoliko ključnih elemenata: naziv obroka, fotografiju, popis sastojaka s pripadajućim mjerama, detaljne upute za pripremu obroka te poveznice na YouTube videozapise koji prikazuju postupak pripreme obroka.

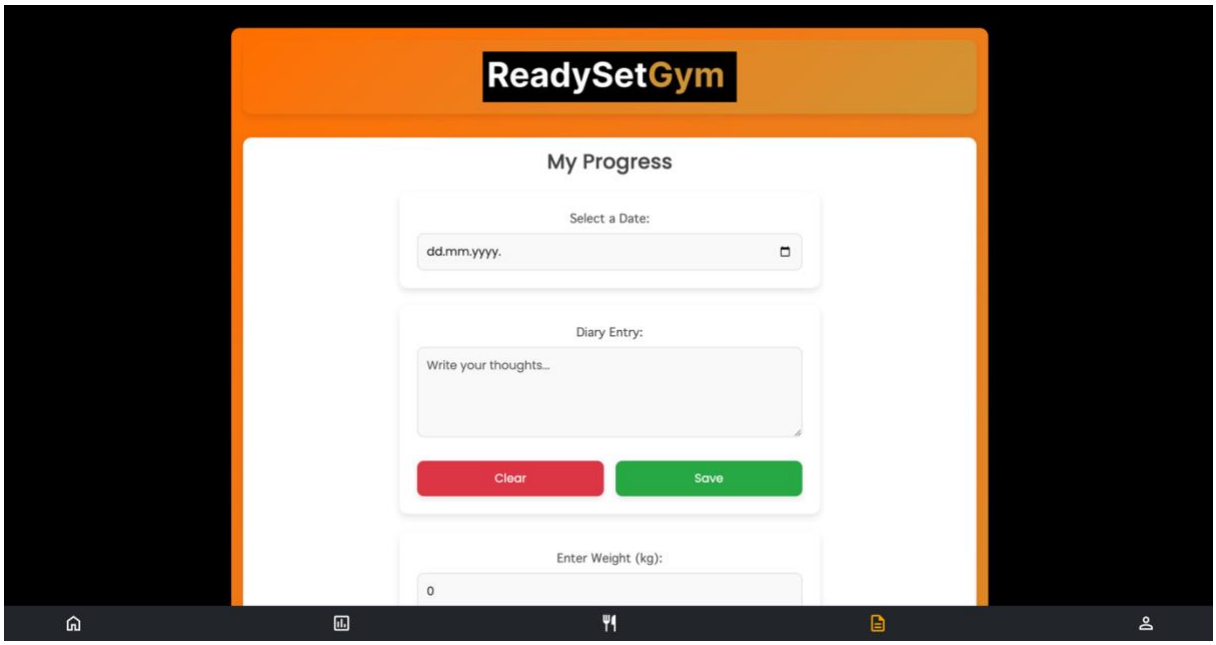
Korisnici mogu pregledavati recepte, detaljno proučavati sve navedene informacije i slijediti upute kako bi pripremili odabrani obrok. Na dnu stranice nalazi se opcija za spremanje recepta. Kada korisnik odabere ovu opciju, recept se dodaje na njegov profil. Spremljeni recepti mogu se pregledavati i upravljati njima putem korisničkog profila, omogućujući jednostavan pristup i organizaciju omiljenih obroka.



Slika 6. Stranica za prikaz recepata

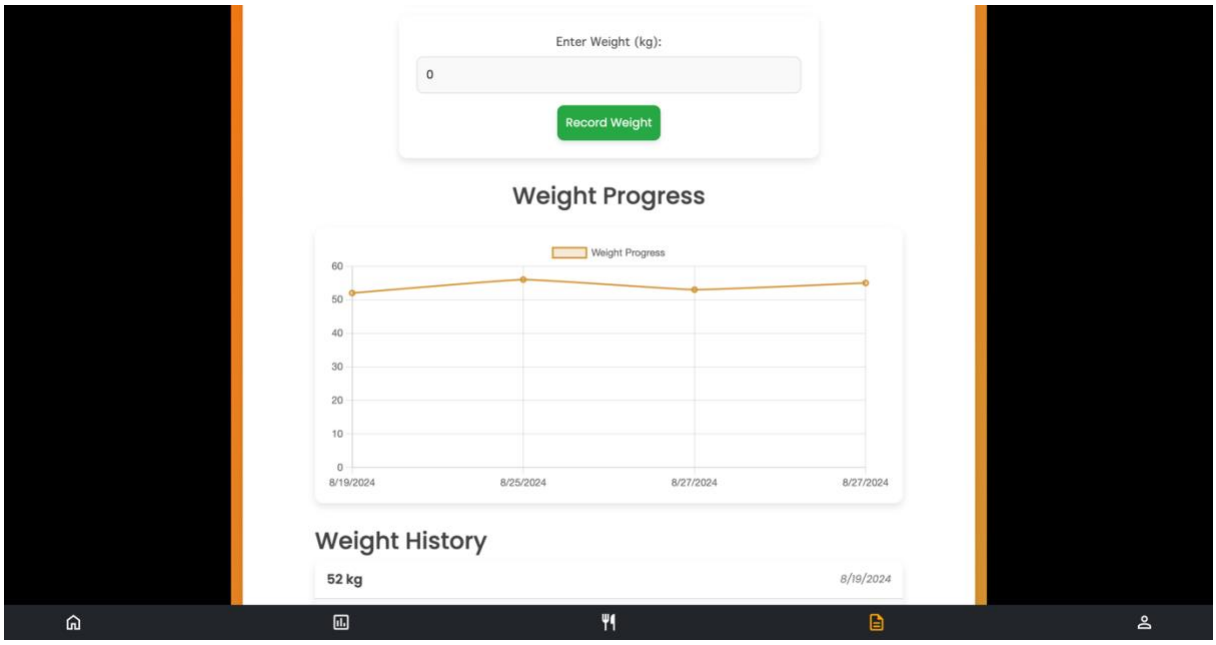
3.6. ZAPISIVANJE DNEVNIKA I PRAĆENJE PROMJENA TJELESNE TEŽINE

Korisnici imaju mogućnost unositi tekstualne bilješke koje se spremaju uz određeni datum. Da bi zapisali dnevnik, korisnik odabire datum unosa i upisuje željeni tekst u predviđeno polje. Nakon što unesu sadržaj, korisnici mogu spremiti dnevnik koji se automatski pohranjuje na njihov osobni profil. Zapisane bilješke dostupne su korisnicima za pregled putem njihovog profila.



Slika 7. Stranica za zapisivanje dnevnika

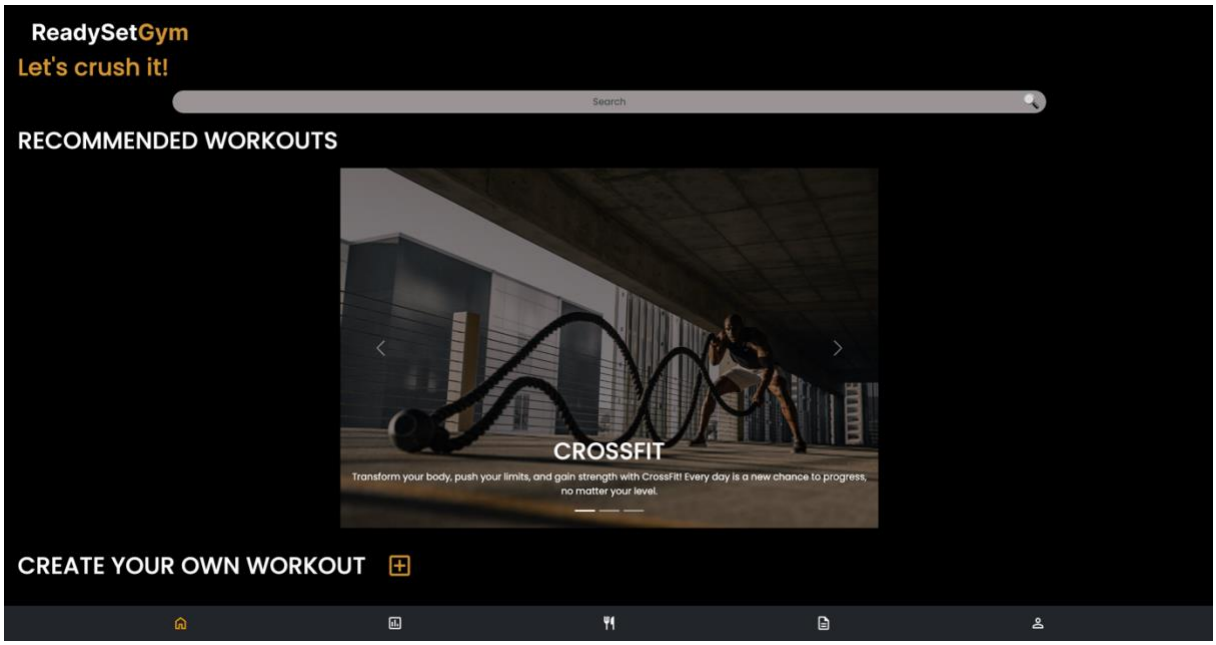
Također, funkcionalnost za praćenje promjena tjelesne težine omogućuje korisnicima da redovito unose svoju tjelesnu težinu u aplikaciju. Na temelju unesenih podataka, aplikacija generira grafički prikaz promjena tjelesne težine tijekom vremena. Grad se automatski ažurira s novim unosima, omogućujući korisnicima vizualizaciju trenda promjena tjelesne težine i lakše praćenje napretka prema ciljevima tjelesne kondicije.



Slika 8. Praćenje promjena tjelesne težine

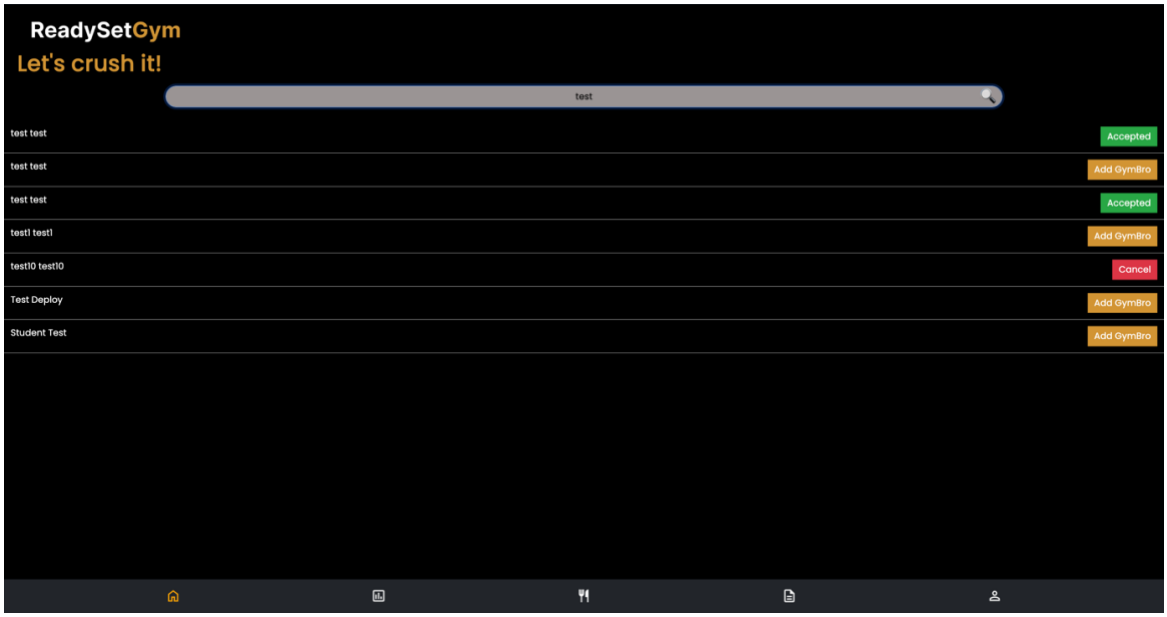
3.7. HOMEPAGE

Na homepage-u aplikacije nalaze se različite funkcionalnosti koje poboljšavaju korisničko iskustvo i omogućuju interakciju s različitim sadržajima i opcijama.



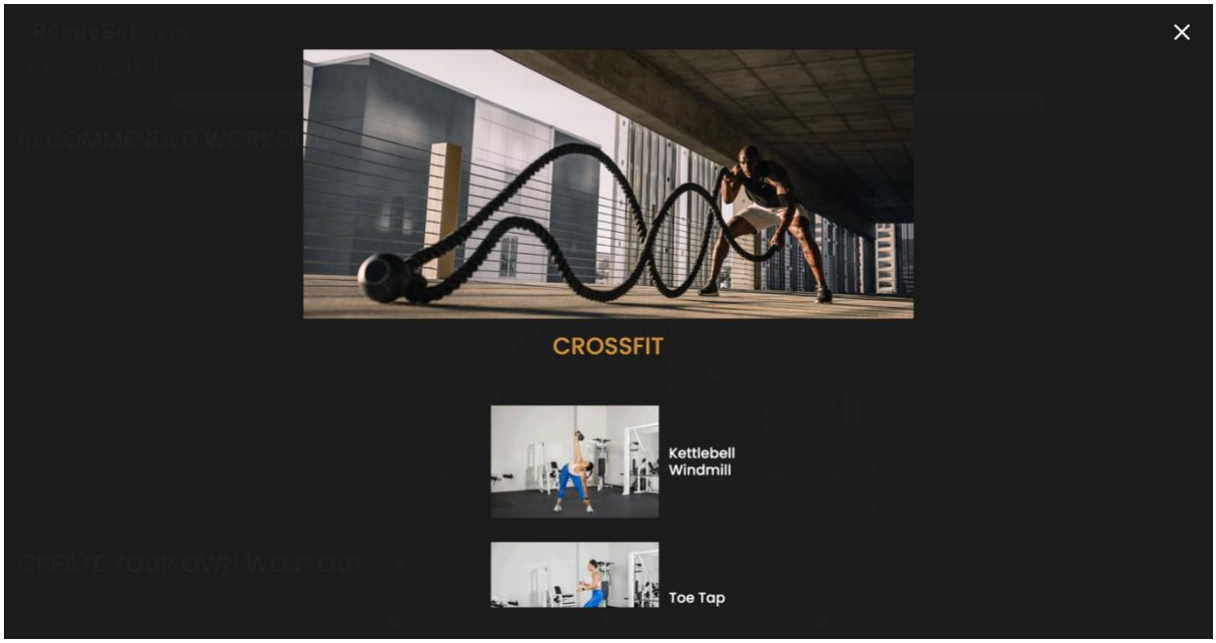
Slika 9. Prikaz homepage-a

Korisnici imaju mogućnost pretraživanja drugih korisnika putem pretraživača. Funkcionalnost uključuje pretraživanje svih korisnika koji se koriste aplikacijom i slanje zahtjeva što omogućava korisnicima širenje mreže kontakata unutar aplikacije.



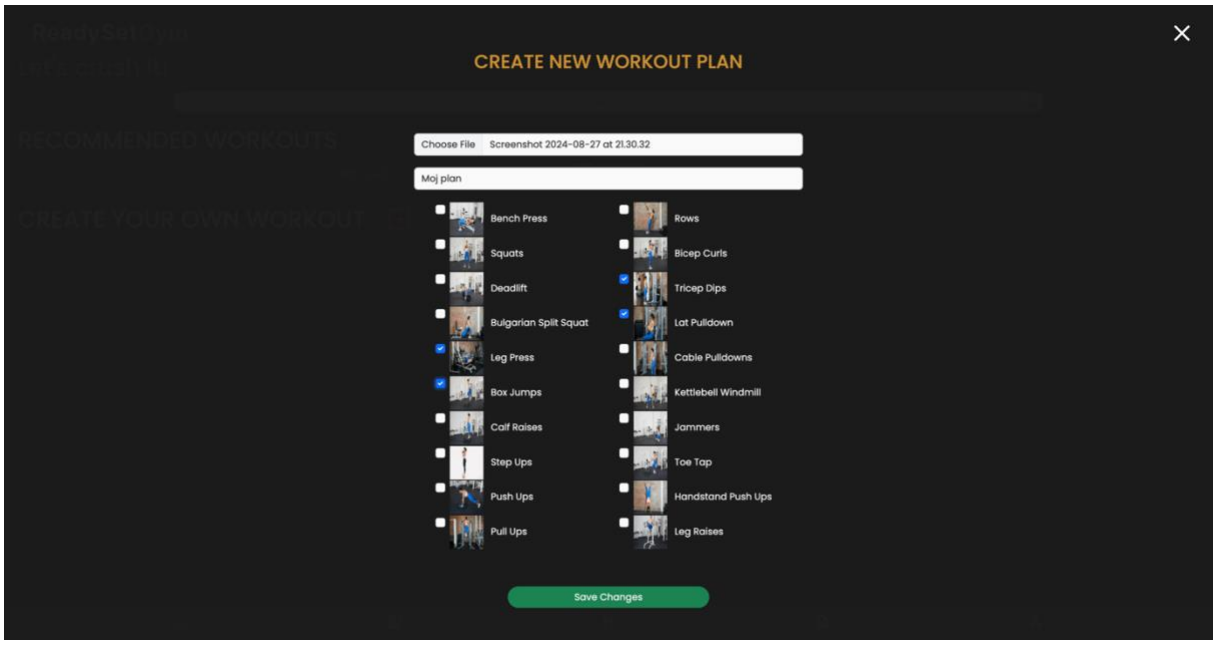
Slika 10. Pretraživanje drugih korisnika

Također, korisnici mogu pregledavati gotove planove u obliku carousela. Ovaj carousel omogućuje vizualizaciju različitih planova u preglednom formatu, a korisnici mogu pregledati detalje svakog plana otvaranjem modala. Modal sadrži sve relevantne informacije o odabranom planu, uključujući naziv plana, ime vježbi te kratak gif koji prikazuje pravilno izvođenje vježbi.

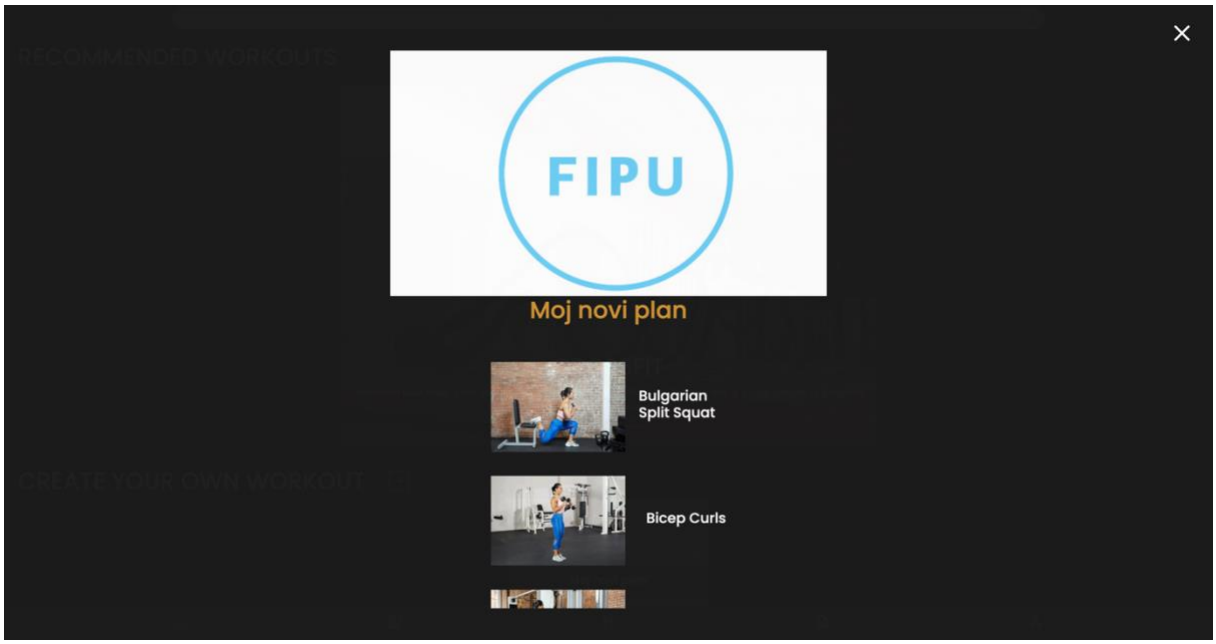


Slika 11. Prikaz modala gotovog plana

Korisnici imaju mogućnost kreiranja i dodavanje svojih personaliziranih planova. Ova funkcionalnost omogućuje korisnicima da odaberu sliku za plan, unesu naziv plana te odaberu vježbe koje će biti uključene. Kreirani personalizirani planovi se prikazuju u obliku carousela, čime korisnici mogu lako pristupiti i pregledavati svoje planove. Također, postoji opcija za brisanje personaliziranih planova, čime se omogućuje korisnicima da upravljaju svojim sadržajem prema vlastitim potrebama i preferencijama.



Slika 12. Kreiranje personaliziranih planova

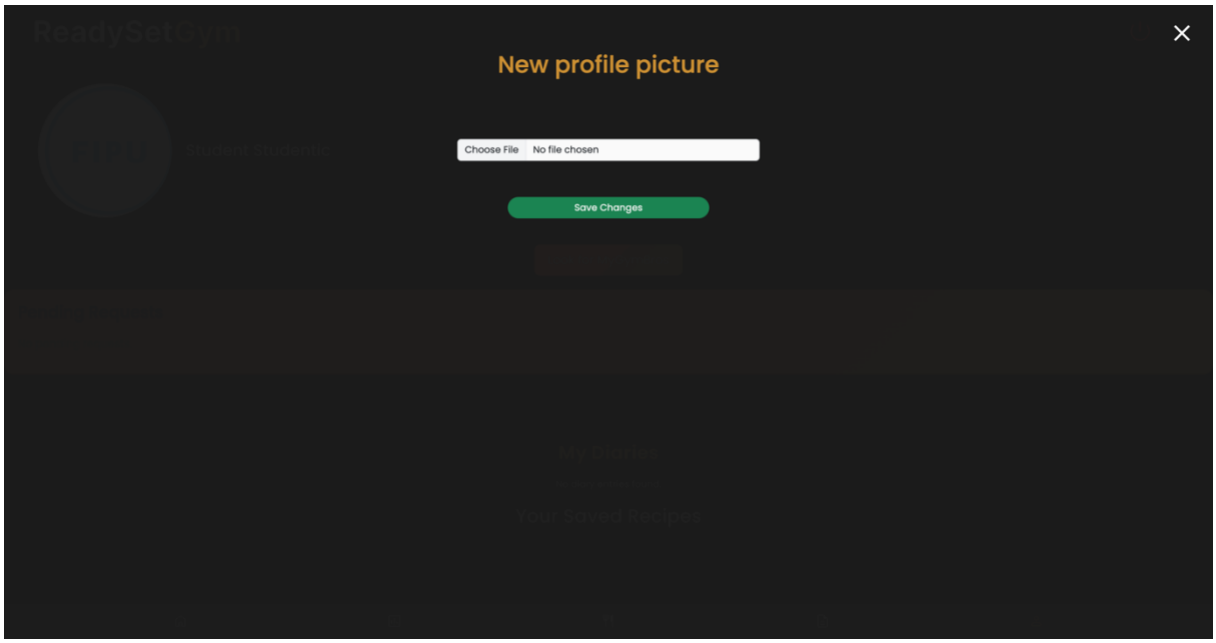


Slika 13. Prikaz personaliziranih planova

3.8. KORISNIČKI PROFIL

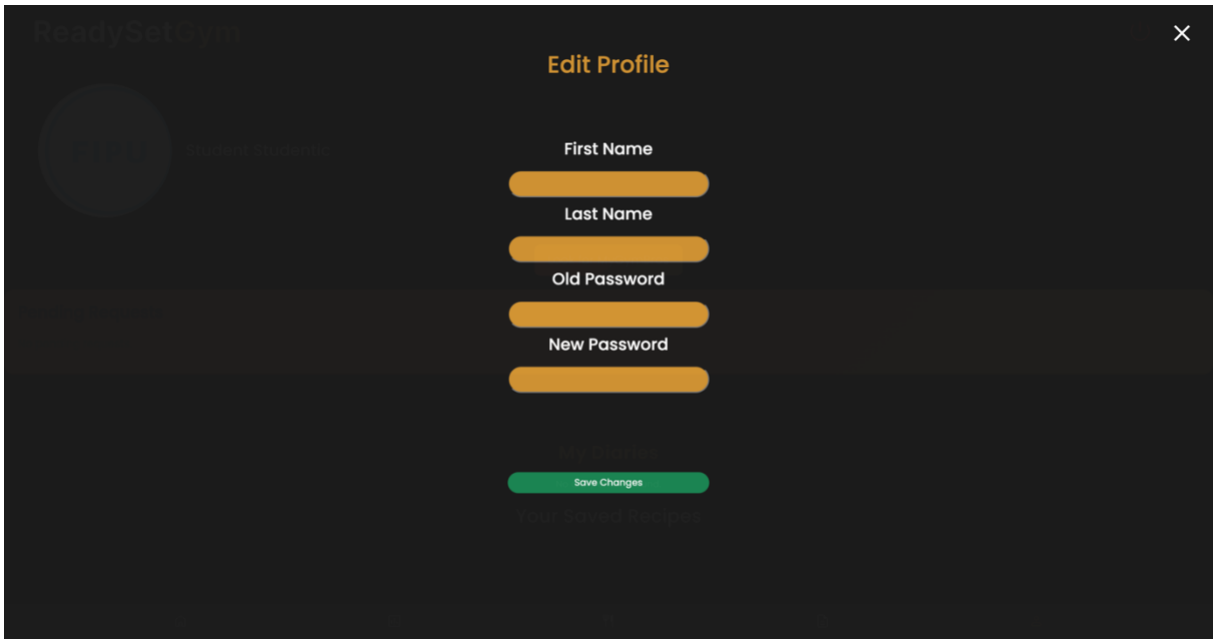
Profil korisnika pruža nekoliko ključnih funkcionalnosti koje omogućuju personalizaciju, interakciju s drugim korisnicima i upravljanje sadržajem.

Korisnici mogu promijeniti svoju profilnu sliku putem opcije koja otvara modal za odabir nove slike. Nakon odabira slike, promjena se automatski ažurira na korisnikovom profilu.



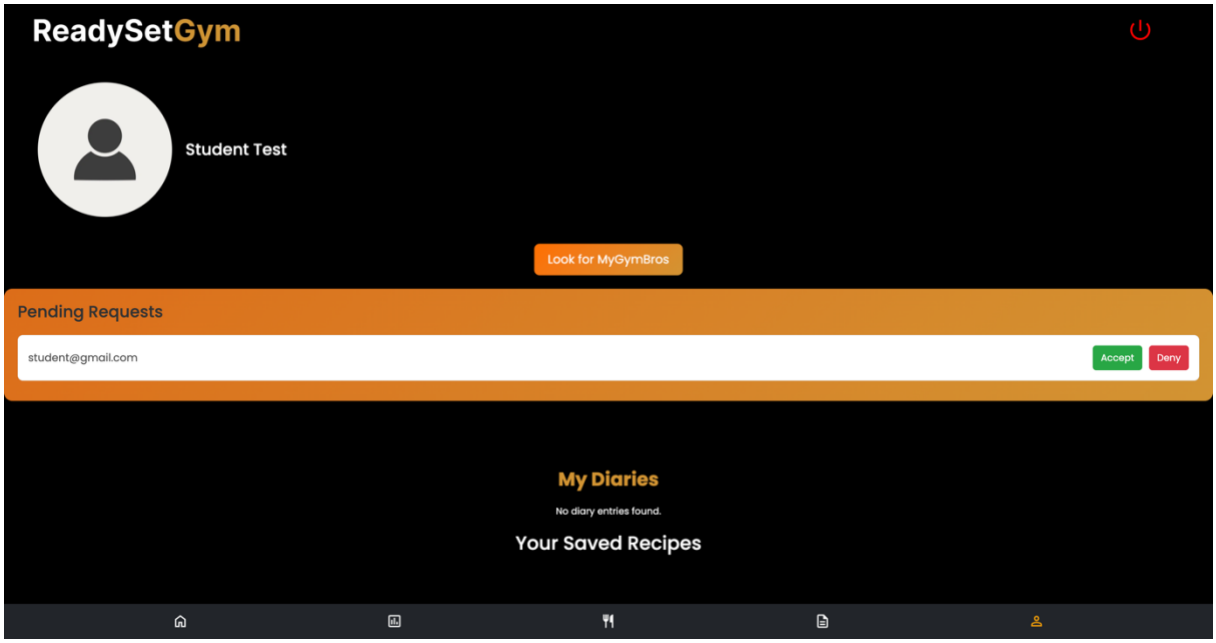
Slika 14. Prikaz modala za mijenjanje profilne slike

Na profilu korisnici mogu ažurirati svoje osobne podatke, uključujući ime, prezime i lozinku. Ova funkcionalnost omogućuje korisnicima da održavaju točnost svojih informacija i osiguraju sigurnost svojih računa.

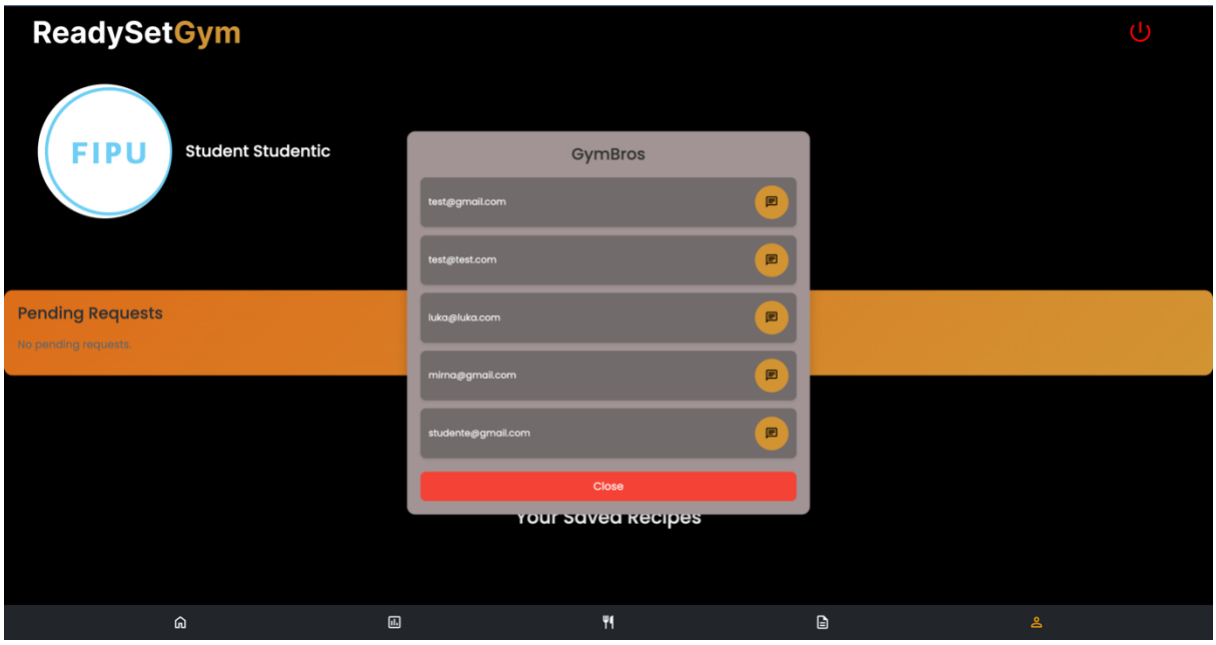


Slika 15. Prikaz modala za mijenjanje podataka

Ako drugi korisnici pošalju zahtjev za prijateljstvo, korisnici mogu pregledati te zahtjeve na svom profilu. Postoji opcija za prihvaćanje ili odbijanje zahtjeva. Ako korisnik prihvati zahtjev, na profilu postoji opcija „Look for MyGymBros“, koja prikazuje popis svih prijatelja korisnika.



Slika 16. Prikaz zahtjeva

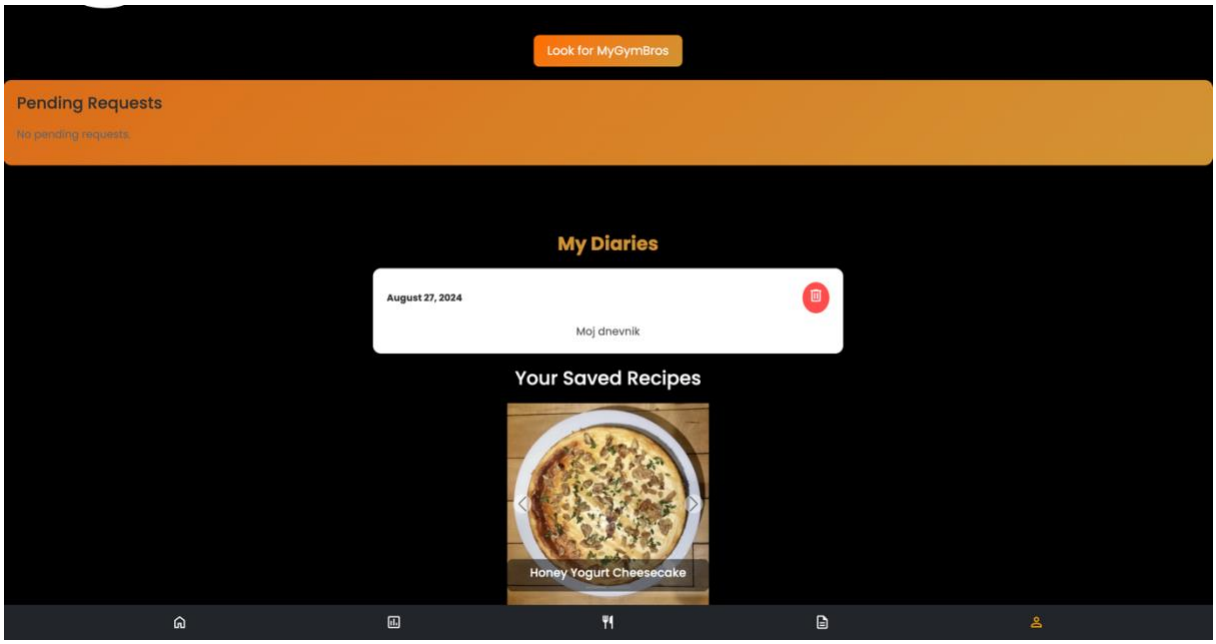


Slika 17. Prikaz popisa prijatelja

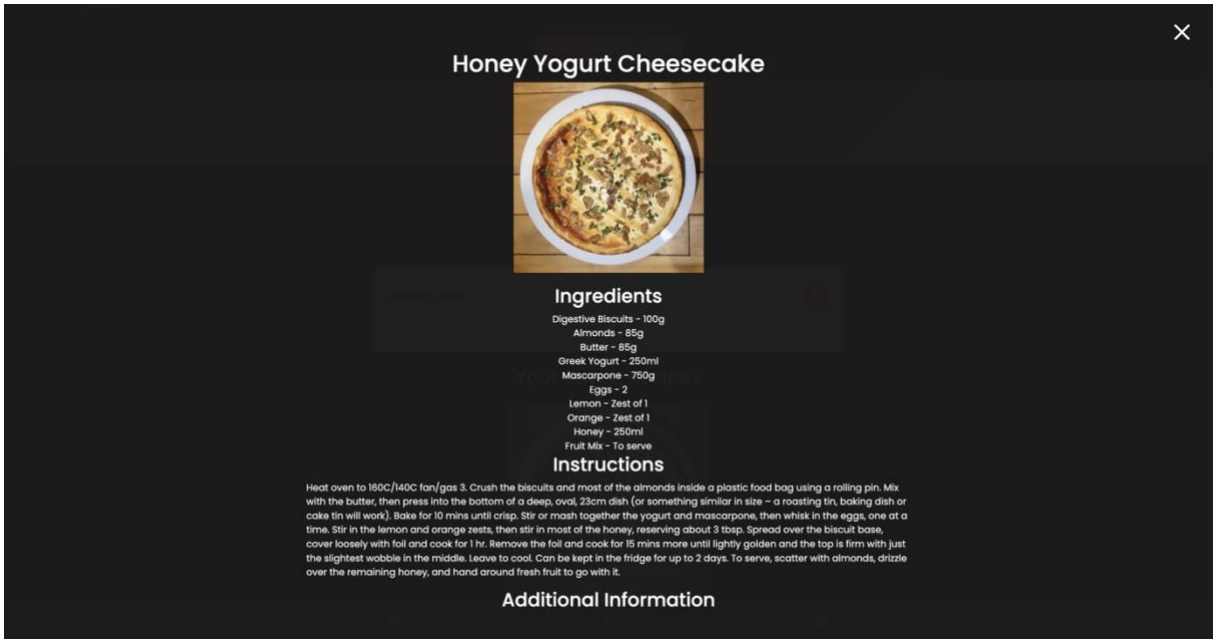
Nakon prihvaćanja zahtjeva za prijateljstvo, korisnici mogu kontaktirati svoje prijatelje putem chat sustava. Ova funkcionalnost omogućava korisnicima da komuniciraju i razmjenjuju poruka s prijateljima unutar aplikacije.

Slika 18. Prikaz chat-a

Korisnici mogu pregledavati spremljene bilješke na svom profilu, s mogućnošću brisanja postojećih unosa. Također, korisnici mogu pregledavati spremljene recepte koji su prikazani u obliku caruousela. Za detaljnije informacije o pojedinom receptu, korisnici mogu otvoriti modal koji pruža sve relevantne informacije o receptu.



Slika 19. Prikaz spremljenih bilješki i recepata



Slika 20. Prikaz detaljnih informacija o spremljenom receptu

4. KORIŠTENI ALATI I TEHNOLOGIJE

U razvoju aplikacije korišten je niz modernih tehnologija i alata koji su omogućili učinkovit i skalabilan razvoj. U nastavku su detaljno opisane ključne tehnologije i alati korišteni tijekom izrade klijentskog dijela aplikacije.

4.1. VUE 3 COMPOSITION API

Vue 3 je najnovija verzija progresivnog JavaScript frameworka Vue.js, koji je dizajniran za razvoj interaktivnih korisničkih sučelja. Jedna od ključnih novosti u Vue 3 je Composition API, koji predstavlja novi način organizacije i ponovne upotrebe logike unutar komponenti.

Tradicionalno, Vue.js koristi Options API, gdje je logika organizirana prema opcijama kao što su `data`, `methods`, i `computed`. Nasuprot tome, Composition API omogućuje grupiranje povezanih logičkih dijelova unutar jedne funkcije, čime se postiže bolja čitljivost i organizacija koda. Na primjer, logika vezana uz upravljanje stanjem, efektima i drugim aspektima komponente može se grupirati unutar jedne funkcije koja koristi `reactive` ili `ref` objekte [2].

Ova fleksibilnost je bila ključna za razvoj složenijih funkcionalnosti aplikacije, omogućujući bolju ponovnu upotrebu koda, lakšu organizaciju i održavanje, te veću skalabilnost.

4.2. PINIA STORE

Za centralizirano upravljanje stanjem aplikacije korišten je Pinia Store, koji je službeni state management alat za Vue 3. Pinia je jednostavnija i modernija zamjena za Vuex, koji je prethodno bio standardni alat za upravljanje stanjem u Vue aplikacijama. Pinia koristi Composition API i pruža intuitivniji i jednostavniji način definiranja i korištenja globalnog stanja unutar aplikacije.

Pinia omogućuje definiranje `stores`, koji predstavljaju centralna mjesta gdje se čuvaju podaci i poslovna logika aplikacije. Ovi `stores` mogu se koristiti u različitim dijelovima aplikacije, omogućujući komponentama da lako dijele i ažuriraju podatke bez potrebe za složenim prolaskom podataka kroz komponente. Pinia također podržava asinkronu logiku i ima ugrađenu podršku za Vue DevTools, što dodatno olakšava debugiranje aplikacije [3].

Korištenje Pinia Store-a omogućilo je održavanje dosljednosti podataka kroz cijelu aplikaciju, olakšalo upravljanje složenim stanjima i omogućilo bolju modularnost koda.

4.3. FIREBASE

Firebase je platforma za razvoj mobilnih i web aplikacija koju je razvio Google. Firebase pruža niz usluga, uključujući real-time baze podataka, autentifikaciju korisnika, pohranu datoteka, hosting i još mnogo toga [4]. U ovom projektu, Firebase je korišten za pohranu multimedijskih sadržaja te za testiranje autentifikacije korisnika koje je kasnije prebačeno da se korisnici spremaju u MongoDB.

Firebase Storage omogućuje sigurno pohranjivanje i dohvaćanje multimedijskih datoteka kao što su slike, videozapisi i dokumenti. Ova funkcionalnost bila je ključna za omogućavanje korisnicima da pohranjuju i dijele svoje sadržaje unutar aplikacije.
Firebase Authentication omogućava jednostavnu i sigurnu autentifikaciju korisnika putem e-maila, društvenih mreža i drugih metoda. Ova usluga osigurava siguran pristup aplikaciji, te omogućava personalizirano korisničko iskustvo.
Firebase Realtime Database omogućuje pohranjivanje i sinkronizaciju podataka u stvarnom vremenu. Ova baza podataka koristi se za pohranu korisničkih podataka, bilješki i drugih važnih informacija koje trebaju biti dostupne i ažurirane u stvarnom vremenu [4].

Integracija Firebase-a omogućila je skalabilnost i sigurnost aplikacije, kao i jednostavno upravljanje podacima i korisnicima.

4.4. BOOTSTRAP

Bootstrap je najpopularniji framework za izradu responzivnih i mobilno-prilagodljivih web stranica. Razvijen od strane Twittera, Bootstrap pruža unaprijed definirane stilove i komponente, kao što su gumbi, obrasci, navigacijski elementi i mrežni sustavi, što ubrzava razvoj korisničkog sučelja [5].

U ovom projektu, Bootstrap je korišten za brzu izradu estetskih ugodnih i responzivnih elemenata sučelja, kao što su navigacijski izbornici, forme i gumbi. Korištenje Bootstrap-a omogućilo je ujednačen izgled aplikacije na različitim uređajima i veličinama ekrana bez potrebe za prilagođavanjem CSS koda za svaki pojedini slučaj. Osim toga, Bootstrap je

integriran s Vue 3, čime je olakšana implementacija komponenti i stilova unutar Vue komponenti, čime se dodatno ubrzao proces razvoja.

4.5. JEDINIČNO TESTIRANJE I INTEGRACIJSKI TESTOVI

Jedinično testiranje i integracijski testovi dvije su ključne metode u procesu osiguravanja kvalitete softvera. Njihova primjena omogućuje rano otkrivanje grešaka, bolju stabilnost sustava te osiguranje da različiti dijelovi aplikacije rade ispravno, kako zasebno tako i zajedno.

4.5.1. JEDINIČNO TESTIRANJE

Jedinično testiranje usmjereno je na provjeru ispravnosti pojedinačnih dijelova koda, obično funkcija ili klasa, u potpunoj izolaciji. Svaka jedinica koda testira se samostalno kako bi se osiguralo da radi prema očekivanjima bez utjecaja drugih dijelova sustava. Važan aspekt jediničnog testiranja je izolacija, koja omogućuje testiranje komponenti bez ikakvih nuspojava ili ovisnosti o drugim dijelovima sustava [6].

Jedinično testiranje obično se izvodi brzo i ne uključuje interakciju s vanjskim resursima kao što su baze podataka, mrežni sustavi ili datotečni sustavi. Korištenje tehnika poput „mockinga“ omogućuje zamjenu stvarnih ovisnosti lažnim objektima koji simuliraju ponašanje pravih komponenata, čime se dodatno povećava izolacija i pouzdanost testova.

Međutim, jedinično testiranje ima svoja ograničenja. Na primjer, privatne funkcije u jezicima s modifikatorima pristupa često nisu dostupne za testiranje, što može zahtijevati promjene u kodu ili korištenje posebnih kompajlerskih zastavica kako bi se omogućilo testiranje tih funkcija.

4.5.2. INTEGRACIJSKI TESTOVI

Integracijsko testiranje usmjereno je na provjeru ispravnosti interakcije između različitih komponenti sustava. Dok jedinično testiranje izolira pojedinačne dijelove koda, integracijsko testiranje ispituje kako ti dijelovi rade zajedno kada su međusobno povezani [6].

Za razliku od jediničnog testiranja, integracijski testovi obuhvaćaju i nuspojave koje proizlaze iz interakcije između komponenti, što može uključivati pristup bazi podataka, mrežne pozive

ili druge vanjske resurse. Ovi testovi su važni za otkrivanje problema koji se mogu pojaviti kada različiti dijelovi aplikacije komuniciraju, a koji možda nisu bili očiti prilikom jediničnog testiranja.
U integracijskom testiranju može se koristiti stvarne i zamjenske (mock) resurse, ovisno o svrsi testiranja. Na primjer, može se koristiti stvarna baza podataka za provjeru ispravnosti upita i promjena u podacima, ili se mogu koristiti mock objekti za simuliranje tih interakcija bez potrebe za stvarnim pristupom bazi.

5. PROGRAMSKO RJEŠENJE I IMPLEMENTACIJA

Ovo poglavlje predstavlja detaljan prikaz procesa razvoja klijentskog dijela web aplikacije za fitness, koja je predmet ovog završnog rada. Klijentski dio aplikacije ključan je za interakciju korisnika s aplikacijom te osigurava pristup svim funkcionalnostima kroz intuitivno i responzivno korisničko sučelje. U nastavku će se analizirati struktura aplikacije, korištene tehnologije, implementacija glavnih funkcionalnosti te način na koji su ovi elementi integrirani kako bi se osigurala optimalna korisnička iskustva.

U procesu implementacije korištene su moderne web tehnologije koje omogućuju razvoj dinamičkih i interaktivnih komponenti unutar aplikacije. Poseban naglasak stavljen je na upotrebu Vue 3 frameworka, koji je korišten za izgradnju reaktivnih korisničkih sučelja, kao i na integraciju s backend servisima putem REST API-ja. Za slanje HTTP zahtjeva i rad s vanjskim API-jem korišten je Axios, popularna biblioteka za jednostavno upravljanje HTTP zahtjevima [7]. Osim toga, u ovom poglavlju bit će razmotrene ključne odluke vezane uz organizaciju koda, upravljanje stanjem aplikacije te osiguranje responzivnosti i pristupačnosti aplikacije na različitim uređajima.

Tijekom razvoja klijentskog dijela aplikacije, bilo je potrebno suočiti se s različitim izazovima, poput optimizacije performansi, održavanja sigurnosti podataka korisnika i osiguravanja konzistentnosti dizajna kroz cijelu aplikaciju. U ovom poglavlju bit će prikazani načini na koje su ti izazovi riješeni, kao i strategije koje su primijenjene za postizanje skalabilnog i održivog rješenja.

Prikazat će se ključni dijelovi programskog koda koji demonstriraju implementaciju osnovnih funkcionalnosti, kao što su registracija korisnika, prijava korisnika, te interakcija s bazom

podataka i vanjskim API-jem. Osim toga, detaljno će se opisati i testiranje klijentskog dijela aplikacije, s posebnim fokusom na jedinično i integracijsko testiranje, koje su korištene za osiguranje kvalitete i pouzdanosti konačnog rješenja.

5.1. ARHITEKTURA I DIZAJN APLIKACIJE

Arhitektura klijentskog dijela web aplikacije za fitness temelji se na modularnom pristupu koji omogućuje lako upravljanje, održavanje i proširivanje sustava. Klijentski dio aplikacije razvijen je koristeći Vue.js, progresivni JavaScript framework, što omogućuje izgradnju reaktivnih i visoko responzivnih korisničkih sučelja. Vue.js je odabran zbog svoje fleksibilnosti, jednostavnosti učenja, te podrške za komponente i modularnu arhitekturu, što je ključno za složenije web aplikacije.

Ovaj dio rada temelji se na informacijama iz Vue.js dokumentacije [8].

5.1.1. ORGANIZACIJA KOMPONENTI

Aplikacija je podijeljena u niz komponenti, od kojih svaka predstavlja zasebnu funkcionalnost ili dio korisničkog sučelja. Komponente su organizirane hijerarhijski, s glavnom komponentom (App.vue) koja služi kao korijenska komponenta aplikacije. U nastavku je opis strukture i organizacije komponenti u aplikaciji.

Folder 'components' sadrži sve osnovne Vue komponente koje čine korisničko sučelje aplikacije. Ove komponente su dizajnirane da budu modularne i ponovljive, što omogućuje jednostavno upravljanje i ponovnu upotrebu koda.

Komponente u folderu 'components':

1. calculateBMIComponent.vue

```
computed: {
  bmiMessage() {
    if (this.bmi === null) return "";
    if (this.bmi < 18.5)
      return "You are undernourished. Consult your doctor to increase body weight.";
    if (this.bmi > 24.9)
      return "You have excess body weight. Consult a doctor to reduce it.";
    return "You have a normal body weight. Eat healthily and engage in physical activity to maintain it.";
```

```
  },
 },
 methods: {
  calculateBMI() {
   if (this.height && this.weight) {
    const heightInMeters = this.height / 100;
    this.bmi = (this.weight / heightInMeters ** 2).toFixed(2);
   } else {
    this.bmi = null;
   }
  },
  selectGender(gender) {
   this.male = gender === "male";
   this.female = gender === "female";
  },
 },
```

Komponenta omogućava unos visine, težine, dobi i spola korisnika. Metoda `calculateBMI` koristi unesene vrijednosti visine i težine kako bi izračunala BMI (indeks tjelesne mase). Rezultat je zaokružen na dvije decimale i pohranjen u varijabli `bmi`. Kroz `bmiMessage` computed property, komponenta generira odgovarajuću poruku korisniku na temelju izračunatog BMI-a. Poruka varira ovisno o tome je li korisnikova težina ispod, iznad ili u normalnom rasponu. Metoda `selectGender` omogućava korisniku da odabere svoj spol, što može utjecati na daljnje funkcionalnosti komponente.

2. chatPage.vue

```
async created() {
  webSocketManager.onMessage((message) => {
   this.messages.push(message);
   this.$nextTick(() => {
    this.scrollToBottom();
   });
  });
 },
 methods: {
  async getChatLogs() {
   const response = await this.chatLogsCollectionStore.getUsersChatLogs(
    this.userEmail
```

```
    );
    this.messages = response.map((message) => {
      return {
        ...message,
        isMine: message.sender !== this.userEmail,
      };
    });
  },
  async getGymBrosData() {
    const response = await this.usersCollectionStore.fetchGymBrosData(
      this.userEmail
    );
    this.recipientName = `${response.firstName} ${response.lastName}`;
    this.recipientImage = response.imagePath
      ? response.imagePath
      : require("@/assets/profile.jpg");
  },
  sendMessage() {
    if (this.newMessage.trim() === "") return;

    const message = {
      recipient: this.userEmail,
      content: this.newMessage,
      timestamp: new Date(),
      isMine: true,
    };
    webSocketManager.sendMessage(message);
    this.messages.push(message);
    this.newMessage = "";
    this.$nextTick(() => {
      this.scrollToBottom();
    });
  },
```

Komponenta upravlja prikazom razgovora i poruka između korisnika. Sadrži funkcionalnost za filtriranje i odabir razgovora. Koristi `webSocketManager` za slanje i primanje poruka u stvarnom vremenu. `onMessage` metoda služi za dodavanje primljenih poruka i automatsko pomicanje prozora prema dolje. Metode `getChatLogs` i `getGymBrosData` dohvaćaju

povijest razgovora i podatke o korisniku putem API poziva. Učitani podaci koriste se za inicijalizaciju prikaza i pripremu za daljnju komunikaciju.

3. diaryComponent.vue

```
const clearForm = () => {   };

const saveEntry = async () => {
  const res = await userDiaryCollectionStore.saveDiaryEntry(
    diaryContent.value,
    date.value
  );
  if (res) {
    clearForm();
    eventBus.emit("diaryEntrySaved");
    eventBus.emit("success", "Diary entry added successfully!");
  }
};

const addWeight = async () => {
  weights.value.push({
    weight: weightInput.value,
    date: new Date().getTime(),
  });
  await weightCollectionStore.recordWeight(weightInput.value);
  updateChart();
  clearForm();
  eventBus.emit("success", "Weight recorded successfully!");
};

const removeWeight = async (date) => {
  try {
    await weightCollectionStore.deleteWeight(date);
    weights.value = weights.value.filter((weight) => weight.date !== date);
    updateChart();
    eventBus.emit("success", "Weight removed successfully!");
  } catch (error) {
    eventBus.emit("error", "Failed to remove weight.");
  }
};
```

```
const renderChart = (weights) => {
};

const updateChart = async () => {
  try {
    const response = await weightCollectionStore.updateChart();
    if (Array.isArray(response)) {
      weights.value = response;
      renderChart(weights.value);
    }
  } catch (error) {
    console.error("Error updating chart:", error);
  }
};
```

Komponenta omogućava korisnicima unos dnevničkih zapisa i spremanje istih pomoću `userDiaryCollectionStore`. Nakon uspješnog spremanja, forma se briše, a korisnik dobiva obavijest o uspješnom dodavanju. Korisnici mogu dodavati i uklanjati unose o tjelesnoj masi. Dodani unosi se spremaju u `weightCollectionStore`, a uklonjeni unosi se brišu. Komponenta ažurira prikaz grafa svaki put kada se podaci promijene. `Char.js` se koristi za prikaz grafa tjelesne mase. Komponenta prikazuje napredak tjelesne mase kroz vrijeme koristeći linijski graf. Graf se ažurira svaki put kada se dodaju ili uklone podaci. Metode `renderChart` i `updateChart` koriste `Char.js` za prikaz i ažuriranje grafa na temelju podataka iz `weightCollectionStore`. `renederChart` se poziva nakon što su podaci ažurirani kako bi se graf prikazao na ekranu. `clearForm` briše podatke iz forme nakon uspješnog dodavanja unosa. `addWeight` dodaje novi unos o tjelesnoj masi i ažurira graf, dok `removeWeight` uklanja unesene podatke i ažurira graf.

4. homePageComponent.vue

```
async created() {
  await this.workoutPlansCollectionStore.fetchUserWorkouts();
  await this.friendsStore.fetchRequests();
  this.fetchExerciseList();
  this.fetchUserWorkouts();
  this.fetchNewUserWorkouts();

  eventBus.on("closeModal", (closeModalData) => {  });
```

```
  eventBus.on("planMoved", (planId) => {    },
 methods: {
  openModalEvent(modalType, workoutPlan) {        },
  handleClose() {},
  closeModal() {  },
  openModal() {  },
  fetchUserWorkouts() {   },
  fetchNewUserWorkouts() {        },
  async fetchExerciseList() {        },
  handleInput() { },
  async sendApiRequest() {        },
  async addFriend(email) {        },
  async cancelRequest(email) {    }
  },
 },
 computed: {
  searchResults() {  },
```

Komponenta upravlja različitim modalnim prozorima (`mainModal`, `userWorkoutPlanModalBody`). Koristi `eventBus` za otvaranje i zatvaranje modalnih prozora te za ažuriranje prikaza nakon akcija poput brisanja plana. `openModalEvent` otvara modalni prozor određenog tipa i postavlja trenutni plan vježbanja. `handleClose` i `closeModal` zatvaraju modalni prozor i resetiraju stanje modalnog prikaza. fetchUserWorkouts preuzima korisničke planove vježbanja iz `workoutPlansCollectionStore`. `fetchNewUserWorkouts` ažurira popis planova vježbanja nakon što je uspješno dodan novi plan. `fetchExerciseList` preuzima popis vježbi iz `exerciseListCollectionStore`. `handleInput` implementira debouncing za pretraživanje korisnika. `sendAPIRequest` šalje zahtjev za pretraživanje korisnika na temeljen na `searchText`. `addFriend` šalje zahtjev za prijateljstvo ako još nije poslan. `cancelRequest` odabucuje zahtjev za prijateljstvo ako je u tijeku. `searchResults` povlači rezultate pretraživanja iz `usersCollectionStore`. `currentUserEmail` povlači email trenutnog korisnika iz `usersCollectionStore`.

5. profileComponent.vue

```
const fetchPendingRequests = async () => {  };
const acceptRequest = async (email) => {    };
const denyRequest = async (email) => {      };
```

```
const fetchUserDiaries = async () => {          };
const fetchUserRecipes = async () => {        };
eventBus.on("diaryEntrySaved", fetchUserDiaries);
eventBus.on("recipeSaved", fetchUserRecipes);
eventBus.on("recipeMoved", (recipeId) => {  });
const deleteDiaryEntry = async (diaryId) => {};
const formatDate = (dateString) => {            };
 },
 methods: {
   closeModalAndShowSuccess(message) {
   },
   toggleGymBrosModal() {
   },
   async fetchGymBros() {
   },
   async getUserDiary() {  },
   async getUserProfile() { },
   fetchNewUserData() {   },
   async openModal(modalType) {  },

   async getUserProfile() {   },
   fetchNewUserData() {   },
   async logout() {            },
   handleLogout() {   },
   closeModal() {   },
   showRecipeDetails(modalType, recipe) {  },
   startChat(email) {   },
 created() {
   eventBus.on("closeModal", (data) => {   });
   eventBus.on("recipeMoved", (recipeId) => {   });
   this.getUserProfile();
   this.getUserDiary();
   eventBus.on("updateUserImage", async (newImagePath) => {   });
   eventBus.on("updateUserData", async () => {   });
   eventBus.on("openRecipeModal", (data) => { },
```

Komponenta upravlja nekoliko modalnih prozora (`mainModal, recipeDetailModal, confirmationModal`). Koristi `eventBus` za otvaranje i zatvaranje modalnih prozora te za ažuriranje podataka prikazanih u njima. `fetchUserDiaries` preuzima dnevnike iz

`userDiaryCollectionStore`. `fetchUserRecipes` preuzima korisničke recepte iz `recipesAPIStore` i ažurira podatke recepata. `fetchPendingRequests` preuzima čekajuće zahtjeve za prijateljstvo iz `friendsStore`. `acceptRequest` prihvaća zahtjev za prijateljstvo i ažurira listu čekajućih poziva. `denyRequest` odbacuje zahtjev za prijateljstvo i ažurira listu čekajućih zahtjeva. `deleteDiaryEntry` briše stavku dnevnika i ažurira prikaz. `showRecipeDetails` otvara modalni prozor s detaljima recepta. `formatDate` formatira datum u čitljiv format. `openModal` otvara modalni prozor specifičnog tipa. `closeModalAndShowSuccess` zatvara modalni prozor i prikazuje poruku o uspjehu. `logout` otvara modal za potvrdu odjave. `handleLogout` obavlja odjavu i preusmjerava korisnika na stranicu za prijavu. `getUserProfile` preuzima i prikazuje podatke o korisničkom profilu. `fetchNewUserData` ažurira korisničke podatke nakon uspješne akcije.

6. recipesComponent.vue

```
computed: {
  filteredIngredients() {
    if (!this.recipe) return [];
    return Object.keys(this.recipe)
      .filter((key) => key.startsWith("strIngredient") && this.recipe[key])
      .map((key, index) => ({
        name: this.recipe[key],
        measure: this.recipe["strMeasure" + (index + 1)],
      }));
  },
},
methods: {
  async fetchRecipe() {   },
  async saveRecipe() {
    try {
      await this.recipesAPI.saveRecipe(this.recipe);
      eventBus.emit("recipeSaved");
      this.savedMessage = true;
    } catch (error) {
      console.error("Error saving recipe:", error);
    }
  },
  async loadSavedRecipe() {  },
created() {
```

```
    this.loadSavedRecipe();
  },
```

`fetchRecipe` učitava podatke o receptu koristeći `recipeAPI.fetchRecipeData()`. Postavlja `loading` na `true` dok traje učitavanje i vraća `errorMessage` ako dođe do greške. `saveRecipe` sprema trenutni recept koristeći `recipesAPI.saveRecipe()` i emitira događaj `recipeSaved` putem `eventBus`. Postavlja `savedMessage` na `true` ako recept bude uspješno spremljen. `loadSavedRecipe` učitava spremljeni recept za korisnika koristeći `recipesAPI.fetchUsersRecipes()`. Ako se uspješno pronađe recept, pretvara ga iz stringa u objekt ako je potrebno. `filteredIngredients` filtrira i formatira sastojke recepta. Koristi `Object.keys()` da pronađe sve sastojke i pripremi ih za prikaz. Ako dođe do greške pri preuzimanju ili spremanju recepta, `errorMessage` se postavlja s porukom greške.

Također, unutar foldera 'components' nalazi se folder 'modals'. Folder 'modals' sadrži komponente za različite modale dijaloge koji se koriste u aplikaciji. Modalne komponente omogućuju prikaz informacija ili interakciju s korisnikom u posebnim, privremenim prozorima.

1. addNewWorkoutModalBody.vue

```
const handleImageSelect = (event) => {
  file.value = event.target.files[0];
};

const uploadToStorage = async () => {
  if (!file.value) return;

  loading.value = true;
  const imageRef = storageRef(
    storage,
    `workoutTitleImages/${file.value.name}`
  );

  try {
    await uploadBytes(imageRef, file.value);
    const imageUrl = await getDownloadURL(imageRef);
    await saveNewWorkoutPlan(imageUrl);
  } catch (error) {
```

```
    console.error("Error during upload:", error);
  } finally {
    loading.value = false;
  }
};

const closeModal = () => {   };

const saveNewWorkoutPlan = async (imageUrl) => {
  try {
    const email = localStorage.getItem("userEmail");
    const formData = {
      email,
      name: workoutPlanName.value,
      exercises: selectedExercises.value,
      imageUrl,
    };
    const res = await workoutPlansCollectionStore.saveNewUserWorkoutPlan(
      formData
    );
    if (res.status === 201) {
      eventBus.emit("success", "Workout plan added successfully!");
      closeModal();
    } else {
      console.error("Failed to save workout plan:", res);
    }
  } catch (error) {
    console.error("Error during saveNewWorkoutPlan:", error);
  }
};
const handleSelect = (data) => {   };

const getExercises = async () => {   };
```

Funkcija `handleImageSelect` omogućava korisniku odabir slike s lokalnog uređaja koja će se kasnije uploadati na Firebase Storage. Funkcija `uploadToStorage` vrši upload odabrane slike na Firebase Storage i dohvaća URL slike koji se koristi za pohranu u bazi podataka. Funkcija `closeModal` resetira unesene podatke i zatvara modal koristeći

`eventBus`. Funkcija `saveWorkoutPlan` pohranjuje novi plan treninga u bazu podataka, uključujući naziv, odabrane vježbe i URL slike.

2. confirmationModal.vue

```
props: {
  isVisible: {
    type: Boolean,
    required: true,
  },
  message: {
    type: String,
    default: "Are you sure?",
  },
 },
 methods: {
  confirm() {
    this.$emit("confirm");
    this.close();
  },
  cancel() {
    this.$emit("cancel");
    this.close();
  },
  close() {
    this.$emit("close");
  },
 },
```

Komponenta koristi `isVisible` prop za kontrolu prikaza modala. `message` prop omogućava prilagodbu teksta koji se prikazuje u modalu. Funkcija `confirm` emitira događaj `confirm` kada korisnik potvrdi akciju, zatim zatvara modal. Funkcija `cancel` emitira događaj `cancel` kada korisnik odustane od akcije, također zatvarajući modal. Funkcija `close` emitira događaj `close` za zatvaranje modala.

3. deletePlanModal.vue

```
 props: {
  isVisible: {
    type: Boolean,
    required: true,
  },
```

```
},
methods: {
  onConfirm() {
    this.$emit("confirm");
  },
  onCancel() {
    this.$emit("cancel");
  },
}
```

Koristi `isVisible` prop za kontrolu prikaza modala, određujući je li modal vidljiv ili ne. Potvrda `onConfirm` emitira događaj `confirm` kada korisnik potvrdi brisanje plana. Funkcija `onCancel` emitira događaj `cancel` kada korisnik odustane od brisanja plana.

4. editProfileDataModalBody.vue

```
async changePassword() {
    this.loading = true;
    this.errorMessage = "";
    this.isErrorMessageMoved = false;
    try {
      const formData = {
        firstName: this.firstName,
        lastName: this.lastName,
        old_password: this.oldPassword,
        new_password: this.newPassword,
      };
      const res = await this.usersCollectionStore.updateUserData(formData);
      if (res.data.message !== "Old password doesn't match.") {
        this.loading = false;
        eventBus.emit("success", "Data updated successfully!");
        const closeModalData = {
          closeModal: true,
        };
        eventBus.emit("updateUserData");
        eventBus.emit("closeModal", closeModalData);
      } else {
        this.loading = false;
        this.errorMessage = "Old password doesn't match.";
        this.isErrorMessageMoved = true;
        setTimeout(() => {
```

```
          this.errorMessage = "";
          this.isErrorMessageMoved = false;
        }, 1000);
      }
    } catch (error) {
      console.error(error.response);
    }
  },
```

Komponenta omogućuje korisnicima da izmijene svoje osobne podatke, uključujući ime, prezime i lozinku. Provjerava se stara lozinka korisnika prije nego što se prihvati nova, osiguravajući da se promjene mogu primijeniti samo ako je trenutna lozinka ispravna. Komponenta koristi `usersCollectionStore` za slanje podataka na backend i ažuriranje korisničkih informacija. Uspješno ažuriranje pokreće emitiranje događaja pomoću `eventBus` kako bi se druge komponente obavijestile o promjene i zatvorio modal. Ako stara lozinka ne odgovara, korisniku se prikazuje poruka o pogrešci koja se automatski uklanja nakon kratkog vremena.

5. editProfilePictureModalBody.vue

```
const handleImageSelect = (event) => {
    file.value = event.target.files[0];
  };

  const uploadToStorage = async () => {
    if (!file.value) return;

    loading.value = true;

    const imageRef = storageRef(storage, `profileImages/${file.value.name}`);

    try {
      await uploadBytes(imageRef, file.value);
      const imageUrl = await getDownloadURL(imageRef);
      await saveImagePathToUserDB(imageUrl);
    } catch (error) {
      console.error("Error during upload:", error);
    } finally {
      loading.value = false;
    }
  };
```

```
const saveImagePathToUserDB = async (imagePath) => {
  const email = localStorage.getItem("userEmail");
  const formData = {
    email: email,
    imagePath: imagePath,
  };

  const res = await usersCollectionStore.updateUserProfilePicture(formData);

  if (res.status >= 200 && res.status < 300) {
    eventBus.emit("updateUserImage", imagePath);
    eventBus.emit("closeModal", { closeModal: true });
    eventBus.emit("success", "Profile picture updated successfully!");
  } else {
    console.error("Failed to update image path in DB", res);
```

Korisnici mogu odabrati novu profilnu sliku pomoću `handleImageSelect` koja sprema odabranu datoteku u file referencu. Funkcija `uploadToStorage` upravlja prijenosom odabrane slike na Firebase Storage, kreirajući referencu za novu sliku na temelju imena datoteke. Nakon uspješnog učitavanja slike, funkcija `saveImagePathToUserDB` ažurira korisnikov profil s novom slikom, spremajući put slike u bazu podataka pomoću `usersCollectionStore.` Nakon ažuriranja slike, pomoću `eventBus` šalju se događaji za ažuriranje prikazane slike korisnika, zatvaranje modala i prikaz poruke o uspjehu.

6. recipeDetailModal.vue

```
computed: {
  filteredIngredients() {
    if (!this.recipe || !this.recipe.recipe) {
      return [];
    }
    const recipeData = this.recipe.recipe;
    const ingredients = [];
    for (let i = 1; i <= 12; i++) {
      const ingredient = recipeData[`strIngredient${i}`];
      const measure = recipeData[`strMeasure${i}`] || "";
      if (ingredient) {
        ingredients.push({ name: ingredient, measure: measure });
      }
    }
```

```
    return ingredients;
  },
 },
 methods: {
  closeModal() {  },
  async moveRecipe() {
   try {
    const response = await this.recipesAPI.deleteRecipe(this.recipe._id);
    if (response.message === "Recipe deleted successfully.") {
     eventBus.emit("recipeMoved", this.recipe._id);
     this.closeModal();
    } else {
     console.error("Failed to delete recipe:", response.message);
    }
   } catch (error) {
    console.error("Error deleting recipe:", error);
   }
  },
 },
 created() {
  eventBus.on("openRecipeModal", (data) => {  });
  eventBus.on("closeModal", () => {  });
 },
```

Ova komponenta služi za prikazivanje detalja recepta u modalnom prozoru. Recept se prikazuje na temelju podataka koji su primljeni putem `eventBus-a` nakon emitiranog događaja `openRecipeModal`. Komponenta koristi `filteredIngredients` computed property za filriranje i prikazivanje relativnih sastojaka recepta, uključujući ime i količinu. Funkcija `closeModal` se koristi za zatvaranje modala i resetiranje stanja recepta kada je modalni prozor neaktivan. Funkcija `moveRecipe` omogućuje brisanje recepta putem poziva `deleteRecipe` metode iz `recipesAPIStore-a,` a zatim šalje događaj `recipeMoved` kako bi obavijestila ostale dijelove aplikacije o uspješnom brisanju recepta.

7. recommendedWorkoutModalBody.vue

```
props: {
  workoutPlan: {
   type: String,
   required: true,
  },
 },
```

```
 setup() {
  const recommendedWorkoutsCollectionStore =
    useRecommendedWorkoutsCollectionStore();
  return { recommendedWorkoutsCollectionStore };
 },
 async created() {
  if (this.workoutPlan) {
   this.workoutPlanData =
    await this.recommendedWorkoutsCollectionStore.fetchRecommendedWorkoutsByName(
     this.workoutPlan
    );
  }
 },
```

Ova komponenta prikazuje detalje preporučenog plana vježbanja vježbanja koji se temelji na nazivu plana proslijeđenom putem `workoutPlan` propa. Kada je komponenta kreirana (`created` hook), automatski se poziva metoda `fetchRecommendedWorkoutByName` iz `recommendedWorkoutsCollectionStore-a` kako bi dohvatila podatke o preporučenom planu vježbanja. Dohvaćeni podaci se spremaju u `workoutsPlanData,` što omogućuje njihovo prikazivanje unutar modalnog prozora.

8. userWorkoutPlanModalBody.vue

```
components: {
  deletePlanModal,
 },
 props: {
  workoutPlan: {
   type: String,
   required: true,
  },
 },
 data() {
  return {
   workoutPlanData: null,
   showDeleteModal: false,
  };
 },
 setup() {
  const workoutPlansCollectionStore = useWorkoutPlansCollectionStore();
```

```
  return { workoutPlansCollectionStore };
},
methods: {
  async closeModal() {
    this.workoutPlanData = null;
    this.$emit("close");
  },
  async handleDelete() {
    try {
      await this.workoutPlansCollectionStore.deletePlan(
        this.workoutPlanData._id
      );
      eventBus.emit("success", "Plan deleted successfully");
      this.$emit("close");
      eventBus.emit("refreshWorkoutPlans");
    } catch (error) {
      console.error("Error deleting workout plan:", error);
    } finally {
      this.showDeleteModal = false;
    }
  },
},
```

Ova komponenta prikazuje detalje korisničkog plana vježbanja temeljenoga na proslijeđenom `workoutPlan` prop-u. Kada je vrijednost propa `workoutPlan` promijenjena, automatski se poziva metoda `getUserWorkoutPlanData` iz `workoutPlansCollectionStore` koja dohvaća podatke o specifičnom planu vježbanja i sprema ih u `workoutPlanData`. Korisnik može izbrisati plan vježbanja pomoću metode `handleDelete`, koja poziva `deletePlan` metodu iz `workoutPlansCollectionStore`. Nakon uspješnog brisanja, emitira se event za osvježavanje planova. Komponenta uključuje modalni prozor `deletePlanModal` koji se prikazuje kada korisnik inicira brisanje plana, omogućujući korisniku potvrdu ili otkazivanje brisanja.

Folder 'views' sadrži komponente koje predstavljaju glavne stranice ili prikaze aplikacije.

- FirstPage.vue: Komponenta za prvu stranicu aplikaciju koja omogućuje korisniku odabir između registracije ili prijave korisnika.

- generalView.vue: Komponente koja predstavlja kontrolnu ploču aplikacije. Uključuje navigacijske elemente.
- LogIn.vue: Komponenta za prijavu korisnika. Sadrži formu za unos korisničkog imena i lozinke te povezuje s funkcionalnostima autentifikacije.
- modalBody.vue: Generički modalni prikaz koji se koristi kao osnova za različite modalne dijaloge. Sadrži osnovnu strukturu koja se prilagođava za specifične potrebe.
- SignUp.vue: Komponenta za registraciju novih korisnika. Sadrži formu za unos osobnih podataka i stvaranje korisničkog računa.

5.1.2. IMPLEMENTACIJA I KONFIGURACIJA ROUTERA

U ovom dijelu detaljno će biti opisano kako je implementirana navigacija unutar aplikacije koristeći Vue Router, te kako je osigurano upravljanje autentifikacijom i prijenosom podataka između komponenti.

Vue Router je ključni alat koji omogućuje navigaciju između različitih stranica unutar Vue.js aplikacije. U aplikaciji, router je definiran u datoteci 'index.js', gdje se koriste `createRouter` i `createWebHistory` za kreiranje i konfiguraciju routera.

```
import { createRouter, createWebHistory } from "vue-router";
import SignUp from "../views/SignUp.vue";
import LogIn from "../views/LogIn.vue";
import FirstPage from "../views/FirstPage.vue";
import config from "../../config.json";
import axios from "axios";
import generalView from "@/views/generalView.vue";
import chatPage from "@/components/chatPage.vue";
```

Ovdje su definirane različite rute koje aplikacije koristi, svaka povezana s odgovarajućom komponentom:

- 'SignUp' i 'LogIn' komponente su povezane s rutama '/signup' i '/login', koje korisnicima omogućuju registraciju i prijavu.
- 'FirstPage' je početna stranica aplikacije i povezana je s root rutom '/'.
- 'generalView' je glavna stranica aplikacije nakon prijave, a ruta '/home' ima atribut `meta: {requiresAuth: true}` što označava da je za pristup ovoj ruti potreba autentifikacija

- 'chatPage' omogućuje korisnicima međusobnu komunikaciju putem chata, gdje se podaci o korisniku (email) prenose kao `prop`.

Kako bi se osigurala sigurnost aplikacije, implementirana je funkcija `beforeEach`, koja provjerava je li korisnik autentificiran prije nego što mu se dozvoli pristup određenim stranicama.

```
router.beforeEach(async (to, from, next) => {
  console.log("Navigation triggered to:", to.name);
  if (to.matched.some((record) => record.meta.requiresAuth)) {
    console.log("Route requires authentication. Checking...");
    try {
      const response = await axios.get(`${config.BACKEND_URL}/auth/check`, {
        withCredentials: true,
      });
      console.log("Auth check response:", response.data);
      if (response.data.message === "Authenticated") {
        console.log("User is authenticated.");
        next();
      } else {
        console.log("User is not authenticated. Redirecting to login...");
        next({ name: "login" });
      }
    } catch (error) {
      console.error("Auth check failed:", error);
      next({ name: "login" });
    }
  } else {
    next();
  }
});
```

Ova funkcija provjerava svaku rutu koju korisnik pokuša posjetiti. Ako ruta zahtijeva autentifikaciju (`requiresAuth`), funkcija šalje zahtjev backendu za provjeru korisničkog statusa koristeći `axios`. Ako je korisnik autentificiran, dozvoljava mu se pristup ruti, inače se preusmjerava na stranicu za prijavu.

Jedna od važnih funkcionalnosti Vue Router-a je mogućnost prosljeđivanja podataka između ruta putem `props`. U ovoj aplikaciji, ruta za chat koristi `props` za prijenos korisničkog emaila, omogućujući personalizaciju iskustva.

```
{
  path: "/chat/:userEmail",
  component: chatPage,
  name: "chatPage",
  props: true,
},
```

Ova konfiguracija omogućava da se korisnički email prenese kao parametar u ruti, čime se olakšava rad s podacima unutar komponente 'chatPage'.

Prilikom provjere autentifikacije mogu se dogoditi greške, poput neuspješnog zahtjeva prema backendu. U takvim situacijama, aplikacija je konfigurirana da korisnika preusmjeri na stranicu za prijavu.

```
} catch (error) {
    console.error("Auth check failed:", error);
    next({ name: "login" });
  }
```

Ovaj pristup osigurava da aplikacija ostane sigurna i korisniku ne dopušta pristup zaštićenim sadržajima bez prethodne provjere.

Router konfiguracija u ovoj aplikaciji pruža robusno rješenje za navigaciju i upravljanje korisničkim pristupom. Kroz implementaciju `beforeEach` funkcije i korištenje `props`, aplikacija osigurava sigurno i personalizirano iskustvo za svakog korisnika, dok istovremeno zadržava fleksibilnost u upravljanju različitim dijelovima aplikacije.

5.2. UPRAVLJANJE STANJEM APLIKACIJE

Upravljanje stanjem aplikacije ključni je aspekt razvoja složenih web aplikacija jer omogućuje učinkovitu komunikaciju i razmjenu podataka između različitih komponenti. U ovoj aplikaciji

za fitness, upravljanje stanjem implementirano je korištenjem 'Pinia Store', modernog alata za upravljanje stanjem u okviru Vue.js. U ovom dijelu opisat će se kako je Pinia Store korišten za upravljanje stanjem aplikacije te kako ovaj pristup doprinosi poboljšanju korisničkog iskustva i održivosti koda.

5.2.1. IMPLEMENTACIJA PINIA STORE-A U APLIKACIJI

U ovoj aplikaciji za fitness, Pinia Store korišten je za upravljanje s različitim aspektima stanja. Svaka datoteka u folderu 'stores' predstavlja specifičan dio aplikacije koji zahtijeva zasebno upravljanje podacima:

1. chatLogsCollectionStore.js

```
export const useChatLogsCollectionStore = defineStore(
  "chatLogsCollectionStore",
  {
    actions: {
      async getUsersChatLogs(recipient) {
        try {
          const response = await axios.get(
            `${config.BACKEND_URL}/chat-logs/${recipient}`,
            {
              withCredentials: true,
            }
          );
          return response.data.data.chatLog.messages;
        } catch (error) {
          console.error(error.response.data);
        }
      },
    },
  }
);
```

Koristi `defineStore` iz Pinia za definiranje `store-a` s imenom `chatLogsCollectionStore`. `getUserChatLogs(recipient)` asinkrona funkcija koja koristi `axios` za slanje GET zahtjeva na backend API kako bi dobila chat logove za određenog primatelja. URL za API poziv je definiran u `config.BACKEND_URL`. Ova funkcija vraća poruke iz chat loga.

2. friendsStore.js

```
export const useFriendsStore = defineStore("friendsStore", {
  state: () => ({
    sentRequests: [],
    acceptedRequests: [],
    pendingRequests: [],
  }),
  actions: {
    async fetchRequests() {
      try {
        const [pendingResponse, sentResponse, acceptedResponse] =
          await Promise.all([
            axios.get(`${config.BACKEND_URL}/gym-bros/requests`, {
              withCredentials: true,
            }),
            axios.get(`${config.BACKEND_URL}/gym-bros/requests/sent`, {
              withCredentials: true,
            }),
            axios.get(`${config.BACKEND_URL}/gym-bros`, {
              withCredentials: true,
            }),
          ]);

        this.pendingRequests = pendingResponse.data.data || [];
        this.sentRequests = sentResponse.data.data || [];
        this.acceptedRequests = acceptedResponse.data.data || [];
      } catch (error) {
        console.error("Error fetching requests:", error);
      }
    },
    async sendFriendRequest(email) {    },
    async cancelRequest(email) {    },
    async acceptRequest(email) {    },
    async denyRequest(email) {  },
  getters: {
    isRequestPending: (state) => (email) => state.sentRequests.includes(email),
    isRequestSent: (state) => (email) => state.sentRequests.includes(email),
    isFriend: (state) => (email) => state.acceptedRequests.includes(email),
  },
});
```

`stateRequests` lista e-mailova korisnika kojima je poslan zahtjev. `acceptedRequests` lista e-mailova korisnika koji su prihvatili zahtjev. `pendingRequests` lista e-mailova korisnika koji su poslali zahtjev i čekaju odgovor. `fetchRequests()` asinkrona funkcija koja koristi `axios` za slanje GET zahtjeva na backend API kako bi dobila podatke o čekanju, poslanim i prihvaćenim zahtjevima. `Promisle.all` se koristi za paralelno slanje tri zahtjeva. `sendFriendRequest(email)` asinkrona funkcija koja šalje POST zahtjev za slanje zahtjeva na backend API i zatim osvježava podatke o zahtjevima. `cancelRequest(email)` asinkrona funkcija koja šalje POST zahtjev za otkazivanje zahtjeva i osvježava podatke o zahtjevima. `accpetRequests(email)` asinkrona funkcija koja šalje POST zahtjev i osvježava podatke o zahtjevima. `denyRequest(email)` asinkrona funkcija koja šalje POST zahtjev za odbijanje zahtjeva. Getteri `isRequestpending(email)` provjerava je li zahtjev u fazi čekanja. `isRequestSent(email)` provjerava je li zahtjev za prijateljstvo poslan. `isFriend(email)` provjerava je li korisnik prijatelj.

3. recipesAPIStore.js

```
export const useRecipesAPIStore = defineStore("recipesAPIStore", {
 state: () => ({}),
 getters: {},
 actions: {
  async fetchRecipeData() {   },
  async saveRecipe(recipe) {
  },
  async fetchUsersRecipes() {   },
  async deleteRecipe(recipeId) {    });
```

`fetchRecipeData()` asinkrona funkcija koja koristi `fetch` za dohvaćanje podataka o receptima s vanjskom API-ja. `saveRecipe(recipe)` asinkrona funkcija koja koristi `axios` za slanje POST zahtjeva za spremanje recepta na backend. Funkcija vraća odgovor od servera. `fetchUsersRecipes()` asinkrona funkcija koja koristi `axios` za dohvaćanje svih recepata korisnika s backend servera. `deleteRecipe(recipeID)` asinkrona funkcija koja koristi `axios` za slanje DELETE zahtjeva za brisanje recepta s backend servera.

4. recommendedWorkoutsCollectionStore.js

```
export const useRecommendedWorkoutsCollectionStore = defineStore(
 "recommendedWorkoutsCollectionStore",
```

```
 {
   state: () => ({}),
   getters: {},
   actions: {
    async fetchRecommendedWorkoutsByName(workoutPlan) {
     try {
      const response = await axios.get(
       `${config.BACKEND_URL}/recommendedWorkouts/${workoutPlan}`,
       { withCredentials: true }
      );

      return response.data.data.recommendedWorkouts;
     } catch (error) {
      console.error(
       "Error fetching recommended workouts:",
       error.response?.data || error.message
      );
      throw error;
     }
    },
   },
 }
);
```

`fetchRecommendedWorkoutsByName(workoutPlan)` asinkrona funkcija koja koristi `axios` za dohvaćanje preporučenih workout planova sa servera. Funkcija uzima ime wokout plana kao parametar i vraća podatke o preporučenim workoutima.

5. userDiaryCollectionStore.js

```
export const useUserDiaryCollectionStore = defineStore(
 "userDiaryCollectionStore",
 {
   state: () => ({}),
   getters: {},
   actions: {
    async getUserDiary() {    },
    async saveDiaryEntry(content, date) {    },
    async deleteDiaryEntry(diaryId) {
     try {
      const token = this.getToken();
```

```
        const response = await axios.delete(
          `${BACKEND_URL}/diary/${diaryId}`,
          {
            headers: {
              Authorization: `Bearer ${token}`,
            },
            withCredentials: true,
          }
        );
        return response;
      } catch (error) {
        this.handleError(error);
      }
    },
    handleError(error, customMessage = "An error occurred") {
      console.error(customMessage, error.response?.data || error.message);
    },
  },
 }
);
```

`getUserDiary()` asinkrona funkcija koja dohvaća dnevnik korisnika s poslužitelja. `saveDiaryEntry(content, date)` asinkrona funkcija koja sprema novu stavku u dnevnik na poslužitelj. `deleteDiaryEntry(diaryId)` asinkrona funkcija koja briše stavku iz dnevnika na poslužitelju.

6. usersCollectionStore.js

```
export const useUsersCollectionStore = defineStore("usersCollectionStore", {
 state: () => ({
  searchResults: [],
  searchLoading: false,
  userEmail: "",
 }),
 getters: {
  getUserEmail: (state) => state.userEmail,
 },
 actions: {
  async updateUserProfilePicture(formData) {  },
  async updateUserData(formData) {  },
  async fetchUserData(email, password) {  },
```

```
async logoutUser() {    },
async registerUser(firstName, lastName, email, password) {
  try {
    const response = await axios.post(`${config.BACKEND_URL}/auth/signup`, {
      firstName,
      lastName,
      email,
      password,
    });
    return response;
  } catch (error) {
    console.error("Error registering user:", error.response?.data);
  }
},
async getUserProfile() {
  try {
    const response = await axios.get(`${config.BACKEND_URL}/user`, {
      withCredentials: true,
    });
    this.userEmail = response.data.data.user.email;
    webSocketManager.connect(this.userEmail);
    return response;
  } catch (error) {
    console.error("Error fetching user profile:", error);
  }
},
async searchUsers(query) {
  this.searchLoading = true;
  try {
    const response = await axios.get(
      `${config.BACKEND_URL}/users/search/${query}`,
      {
        withCredentials: true,
      }
    );
    this.searchResults = response.data.data.users;
  } catch (error) {
    console.error("Error searching users:", error.response?.data);
  } finally {
    this.searchLoading = false;
```

```
    }
  },
  async fetchGymBrosData(email) {});
```

State-ovi `searchResults` pohranjuje rezultate pretraživanja korisnika. `searchLoading` boolean koji označava je li pretraživanje u tijeku. `userEmail` pohranjuje email korisnika. Getter `getUserEmail` vraća email korisnika iz state-a. `updateUserProfilePicture(formData)` asinkrona funkcija koja ažurira profilnu sliku korisnika. `updateUserData(formData)` asinkrona funkcija koja ažurira podatke korisnika. `fetchUserData(email, password)` asinkrona funkcija koja prijavljuje korisnika i pohranjuje JWT token u `localStorage.` `logoutUser()` asinkrona funkcija koja odjavljuje korisnika i zatvara WebSocket vezu. `registerUser(firstName, lastName, email, password)` asinkrona funkcija koja registrira novog korisnika. `getUserProfile()` asinkrona funkcija koja dohvaća profil korisnika i inicijalizira WebSocket vezu. `searchUsers(query)` asinkrona funkcija koja pretražuje korisnike prema zadanom upitu. `fetchGymBrosData(email)` asinkrona funkcija koja dohvaća podatke o korisniku na temelju email adrese.

6. weightCollectionStore.js

```
export const useWeightCollectionStore = defineStore("weightCollectionStore", {
 state: () => ({}),
 getters: {},
 actions: {
  async recordWeight(weight) {
   try {
    const token = this.getToken();
    const response = await axios.post(
     `${BACKEND_URL}/weight`,
     { weight },
     { withCredentials: true }
    );
    return response;
   } catch (error) {
    this.handleError(error, "Error saving weight");
   }
  },
  async deleteWeight(date) {
   try {
```

```
      const token = this.getToken();
      const response = await axios.delete(`${BACKEND_URL}/weight/${date}`, {
        withCredentials: true,
      });
      return response;
    } catch (error) {
      this.handleError(error, "Error deleting weight");
    }
  },
  async updateChart() {});
```

`recordWeight(weight)` asinkrona funkcija koja zapisuje novu težinu u bazu podataka. `deleteWeight(date)` asinkrona funkcija koja briše težinu na temelju datuma. `updateChart()` asinkrona funkcija koja dohvaća sve zabilježene težine za ažuriranje grafikona.

7. workoutPlansCollectionStore.js

```
export const useWorkoutPlansCollectionStore = defineStore(
 "workoutPlansCollectionStore",
 {
  state: () => ({
    userWorkouts: [],
  }),
  getters: {
    getAllUserWorkouts: (state) => state.userWorkouts,
    getUserWorkoutPlanData: (state) => (id) => {
      return Array.isArray(state.userWorkouts)
        ? state.userWorkouts.find((plan) => plan._id.toString() === id)
        : null;
    },
  },
  actions: {
    async saveNewUserWorkoutPlan(formData) {     },
    async fetchUserWorkouts() {
      const response = await axios.get(`${config.BACKEND_URL}/workout-plan`, {
        withCredentials: true,
      });
      this.userWorkouts = response.data.data.workoutPlans;
    },
    async deletePlan(planId) {
      try {
```

```
        const response = await axios.delete(
          `${config.BACKEND_URL}/workout-plan/${planId}`,
          {
            withCredentials: true,
          }
        );
        return response.data;
      } catch (error) {
        console.error("Error deleting plan:", error);
        throw new Error("Failed to delete plan");
      }
    },
  },
}
);
```

State `userWorkouts` polje koje pohranjuje sve korisničke planove vježbi.

Getteri `getAllUserWorkouts` vraća sve planove vježbi korisnika, `getUserWorkoutPlanData(id)` vraća plan vježbi za određeni ID ako postoji. `saveNewUserWorkoutPlan(formData)` asinkrona funkcija koja sprema novi plan vježbi korisnika. `fetchUserWorkouts()` asinkrona funkcija koja dohvaća sve planove vježbi korisnika iz API-ja i pohranjuje ih u `userWorkouts`. `deletePlan(planId)` asinkrona funkcija koja briše plan vježbi na temelju ID-a.

5.2.2. INTERAKCIJA KOMPONENTI PUTEM PINIA STORE-A

Pinia Store omogućuje komponentama unutar aplikacije da međusobno komuniciraju i razmjenjuju podatke na učinkovit način. Kada komponenta ažurira stanje u Pinia Store-u, te promjene automatski se reflektiraju u svim komponentama koje koriste to stanje. Na primjer:

- Sinkronizacija stanja: Kada korisnik promijeni podatke na svom profilu, te promjene automatski se ažuriraju u svim dijelovima aplikacije koji prikazuju podatke o korisniku, uključujući navigaciju i profilnu stranicu.
- Automatsko ažuriranje UI-a: Kada korisnik doda novi recept ili personalizirani plan vježbanja, te promjene odmah su vidljive na odgovarajućim stranicama aplikacije, zahvaljujući stanju u Pinia Store-u.

5.2.3. PREDNOSTI I IZAZOVI KORIŠTENJA PINIA STORE-A

Korištenje Pinia Store-a donosi brojne prednosti u razvoju ove aplikacije, uključujući:

- Jednostavno upravljanje složenim stanjima: Pinia Store pojednostavljuje upravljanje složenim stanjima u aplikaciji, omogućujući centralizaciju podataka i bolju organizaciju koda.
- Poboljšane performanse: Pinia nudi bolje performanse u odnosu na starije alate poput Vuex-a, zahvaljujući optimiziranom načinu rada s reaktivnim stanjem.
- Jednostavna integracija s Vue 3: Pinia je izgrađen s obzirom na Vue3 i njegov Composition API, što omogućuje jednostavnu i prirodnu integraciju u aplikaciju.

Međutim, bilo je i izazova, uključujući potrebu za dodatnim učenjem novih koncepata i metoda specifičnih za Pinia, kao i prilagodbu nekih postojećih dijelova koda kako bi se potpuno iskoristile sve mogućnosti koje ovaj alat nudi.

5.3. KONFIGURACIJA APLIKACIJE I GLOBALNE POSTAVKE

U ovom poglavlju bit će opisane ključne konfiguracijske datoteke i globalne postavke koje su korištene u razvoju klijentskog dijela web aplikacije za fitness. Ova konfiguracija omogućava aplikaciji da ispravno komunicira s backend serverom, upravlja stanjem i olakšava komunikaciju između komponenti.

5.3.1. KONFIGURACIJA APLIKACIJSKIH POSTAVKI

Datoteka 'config.json' sadrži osnovne konfiguracijske postavke za aplikaciju, uključujući URL-ove za backend, API za recepte i WebSocket veze. Ova datoteka omogućuje centralizirano upravljanje važnim postavkama aplikacije, što olakšava održavanje i prilagodbu.

```
{
  "BACKEND_URL": "https://ready-set-gym-backend-77y4.onrender.com",
  "RECIPES_API_URL": "https://www.themealdb.com/api/json/v1/1/random.php",
  "WEBSOCKET_URL": "wss://ready-set-gym-backend-77y4.onrender.com/socket?userEmail="
}
```

- BACKEND_URL: Ova stavka definira osnovni URL za backend server aplikacije. Korištenje varijable umjesto direktnog unosa URL-a u kodu omogućuje lakše mijenjanje backend servera bez potrebe za izmjenama u cijeloj aplikaciji.

- RECIPES_API_URL: Ova stavka sadrži URL za API koji se koristi za dohvaćanje nasumičnih recepata. Centralizacija ovog URL-a olakšava promjenu izvora recepata bez potrebe za izmjenom koda u više komponenti.
- WEBSOCKET_URL: Definira osnovni URL za WebSocket veze koje se koriste za real-time komunikaciju, poput chata između korisnika. Centralizacija ove postavke omogućuje jednostavno održavanje i prilagodbu WebSocket veze.

5.3.2. INICIJALIZACIJA APLIKACIJE

Datoteka 'main.js' predstavlja ulaznu točku za vašu Vue aplikaciju. Ovdje se kreira instanca aplikacije te se integriraju ključni alati i servisi.

```
import { createApp } from "vue";
import { createPinia } from "pinia";
import App from "./App.vue";
import router from "./router";
import { initializeApp } from "firebase/app";
import { getStorage } from "firebase/storage";


const firebaseConfig = {
  apiKey: "AIza[redacted]",
  authDomain: "fitnessapp-448c7.firebaseapp.com",
  projectId: "fitnessapp-448c7",
  storageBucket: "fitnessapp-448c7.appspot.com",
  messagingSenderId: "759431144786",
  appId: "1:759431144786:web:31187370d92cba0b4a468b",
};

const firebaseApp = initializeApp(firebaseConfig);
const storage = getStorage(firebaseApp);

const pinia = createPinia();
const app = createApp(App);

app.provide("firebaseStorage", storage);
```

```
app.use(router);
app.use(pinia);
app.mount("#app");
```

Ovdje se inicijalizira Firebase aplikacija s konfiguracijom koja omogućuje integraciju s Firebse Storage servisom. Ova integracija omogućuje pohranu multimedijskih sadržaja poput profilnih slika i drugih datoteka.

Kreiranje instance Pinia za upravljanje stanjem aplikacije omogućuje centralizirano upravljanje podacima koji su dostupni različitim komponentama. Ovo olakšava održavanje i skaliranje aplikacije.

Integracija Vue Routera omogućuje navigaciju unutar aplikacije, definiranje ruta i upravljanje autentifikacijom putem `beforeEach` hooka. Ovaj dio koda osigurava da su rute zaštićene i da samo autentificirani korisnici mogu pristupiti određenim dijelovima aplikacije.

5.3.3. KOMUNIKACIJA IZMEĐU KOMPONENTI POMOĆU EVENT BUS-A

Event bus je obrazac koji se koristi za komunikaciju između komponenata koje nisu u direktnoj vezi, kao što su komponente roditelj-dijete. U aplikaciji 'eventBus.js' datoteka koristi 'mitt' za upravljanje događajima.

```
import mitt from "mitt";

const eventBus = mitt();

export default eventBus;
```

'mitt' je lagani event emitter koji se koristi za slanje i primanje događaja između različitih dijelova aplikacije. Na primjer, možete emitirati događaj u jednoj komponenti i slušati ga u drugoj, bez potrebe za prop-drillingom. Ovo značajno pojednostavljuje komunikaciju između komponenti, posebno u većim aplikacijama.

Event bus se može koristiti za komunikaciju između komponenti kao što su modalni prozori, obavijesti ili prijenos podataka između različitih stranica unutar aplikacije.

6. ZAKLJUČAK

U ovom završnom radu prikazana je izrada i implementacija web aplikacije koja korisnicima omogućuje praćenje njihovog zdravlja putem niza funkcionalnosti, uključujući BMI kalkulator, vođenje dnevnika, praćenje promjena tjelesne težine, prikaz recepata te prikaz planova vježbanja kao i kreiranje vlastitih. Proces razvoja obuhvatio je detaljnu fazu izrade prototipa, razvoj korisničkog sučelja i funkcionalnosti, te testiranje i optimizaciju aplikacije.

Korištenjem modernih tehnologija kao što su Vue 3 Composition API, Pinia Store, Firebase i Bootstrap, razvijena je responzivna i intuitivna aplikacija koja odgovara potrebama korisnika. Implementacija upravljanja stanjem aplikacije putem Pinia Store-a omogućila je centralizaciju podataka i konzistentno korisničko iskustvo, dok su provedena testiranja osigurala stabilnost i pouzdanost sustava.

Tijekom razvoja, identificirani su i prevladani izazovi, poput integracije različitih alata i optimizacije performansi. Primjenom odgovarajućih metodologija i tehnologija, osigurana je funkcionalnost i kvaliteta korisničkog iskustva, što je bilo ključno za uspjeh aplikacije.

Ovaj rad predstavlja značajan doprinos u razvoju korisnički orijentirane web aplikacije, demonstrira mogućnost implementacije složenih funkcionalnosti u modernom razvojnom okruženju te pruža temelj za daljnja istraživanja i unapređenja.

Zaključno, ovaj završni rad uspješno demonstrira povezanost teorije i prakse u razvoju moderne web aplikacije te pruža korisno iskustvo u rješavanju složenih tehničkih problema i implementaciji naprednih tehnologija.

7. LITERATURA

[1] **Benyon, D.** (2013). *Designing Interactive Systems: A Comprehensive Guide to HCI, UX and Interaction Design*. 3rd Edition. Pearson.

[2] **Smith, J., & White, M.** (2022). *Fullstack Vue 3: The Complete Guide to Vue.js* (2nd ed.). Fullstack.io. Dostupno na: https://dokumen.pub/fullstack-vue-3-the-complete-guide-to-vuejs-2nbsped.html

[3] **Vuex Organization.** (2023). *Pinia Store Documentation*. Dostupno na: https://pinia.vuejs.org/

[4] **Google.** (2023). *Firebase Documentation*. Dostupno na: https://firebase.google.com/docs/

[5] **Bootstrap.** (2023). *Bootstrap 5 Documentation*. Dostupno na: https://getbootstrap.com/docs/5.0/getting-started/introduction/

[6] **CircleCI.** (2019, October 30). Unit testing vs. integration testing. *CircleCI*. https://circleci.com/blog/unit-testing-vs-integration-testing/

[7] **Axios.** (2023). *Axios HTTP Client Documentation*. Dostupno na: https://axios-http.com/docs/intro

[8] **Vue.js Documentation.** (n.d.). *Introduction to Vue 3*. Dostupno na : https://devdocs.io/vue~3/guide/introduction

8. POPIS SLIKA