Nuxt.js fitnes aplikacija

Bilić, David

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni diplomski studij Računarstvo

NUXT.JS FITNESS APLIKACIJA

Diplomski rad

David Bilić

Osijek, 2021.

SADRŽAJ:

1. UVOD

Cilj ovog rada je izraditi web aplikaciju koja koristi bazu podataka ispunjenu detaljima o raznim vrstama namirnica, a glavna namjera aplikacije je svojim korisnicima pružiti podatke o idealnoj dnevnoj količini unesenih kalorija i raspodjeli osnovnih hranjivih sastojaka (proteini, ugljikohidrati, masti) na osnovu unesenih podataka o njihovom tijelu i željenom fitness cilju, a zatim im omogućiti izradu ispravnog plana prehrane i kroz vrijeme izgraditi jednostavniju analitiku koja korisnicima omogućuje vođenje evidencije o tome koliko su se zapravo držali preporuka aplikacije.

Najprije je potrebno ukratko pojasniti teorijsku podlogu na temelju koje će se obavljati izračuni u aplikaciji, a zatim prijeći na samu izradu aplikacije. U današnje vrijeme, jedan od glavnih uzroka smanjene kvalitete života stanovništva, umora, slabosti pa čak i nemogućnosti ostvarivanja željenog fitness cilja predstavlja neupućenost u izradu ispravnog prehrambenog plana. Jedan od ciljeva ovog rada je također dokazati korisnicima koliko je to zapravo jednostavno i isplativo.

Web aplikacija koja predstavlja rješenje ovog rada će biti izrađena pomoću Nuxt.js razvojnog okruženja, te programski implementirana pomoću programskog jezika Javascript. Aplikacija će koristiti Google-ovu Backend-as-a-Service platformu Firbase. Za pohranu podataka će biti korištena Firebase-ova Cloud Firestore baza podataka, dok će se za izradu sustava autentifikacije i autorizacije koristiti Firebase Authentication usluga. Za izradu ove aplikacije je korišteno znanje iz više predmeta programerske skupine predmeta, a dodatno je bilo potrebno proučiti Google-ovu platformu Firebase u svrhu izrade plana funkcioniranja i korištenja aplikacije.

U idućem poglavlju je obavljen kratki pregled aplikacija koje se bave sličnom problematikom. U poglavlju 3 je dana kratka teorijska podloga o pravilnoj prehrani, energetskoj vrijednosti namirnica i korištenim formulama za izradu izračuna. U poglavlju 4 slijedi detaljnije upoznavanje sa korištenim tehologijama. Poglavlje 5 će prikazati zamišljeni model baze podataka s obzirom da je Firebase Firestore noSql baza podataka posebne strukture koja koristi dokumente i kolekcije. Poglavlje 6 prikazuje programiranje aplikacije u razvojnom okruženju Nuxt.js pomoću programskog jezika Javascript, a posljednje poglavlje prikazuje testiranje ispravnog rada apliakcije i usporedbu rezultata za različite unesene podatke.

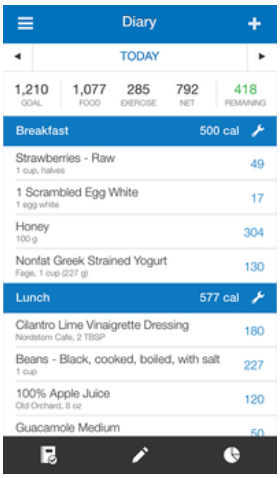
1.1. Zadatak diplomskog rada

Potrebno je razraditi model web aplikacije s bazom podataka koja omogućava registriranje i prijavu korisnika, unos podataka o tijelu (visina, težina, dob, spol, fizička aktivnost, tip tijela) i željenog fitness cilja korisnika na osnovu kojega će korisniku, na osnovu unesenih podataka, biti generirani idealan dnevni unos kalorija uz idealnu raspodjelu hranjivih sastojaka (proteina, ugljikohidrata i masti) koje bi trebao unositi da ostvari željeni fitness cilj. Aplikacija sadrži detalje određenog broja namirnica za koje korisnik može odabirati količine i povezivati ih u obroke. Korisnik obroke mora povezivati i pomoću njih stvarati jelovnike. Stvorene jelovnike korisnik mora razvrstavati po datumima kada ih planira konzumirati. Aplikacija treba generirati upozorenja u slučaju stvaranja jelovnika koji uzrokuju dnevna kalorijska odstupanja, a pošto korisnik ima mogućnost osvježavanja podataka o tijelu, uvid u jednostavnu analitiku koja u prikazuje graf izmjene njegove mase kroz vrijeme na temelju o osvježavanja podataka, graf unesenih kalorija kroz vrijeme na temelju razvrstavanja jelovnika po danima, te graf njegove fizičke aktivnosti. Aplikaciju treba programski ostvariti, ispitati na odgovarajućem skupu ulaznih podataka i prikladno analizirati.

2. SLIČNI PRISTUPI I RJEŠENJA

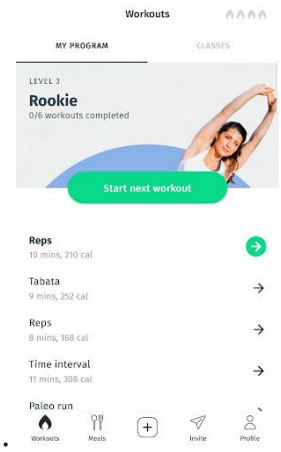
U ovom poglavlju će biti razmotrene druge aplikacije koje se bave ovim područjem rada, uz značajke koje nude svojim korisnicima.

Najpoznatija aplikacija koja se bavi ovim područjem rada je *Calorie Counter - MyFitnessPal*. Ovo je mobilna aplikacija podržana na iOS i Android operacijskim sustavima. Prema [1], ovo je besplatna aplikacija koja nudi kupovinu proširenja, a omogućuje svojim korisnicima zapis obroka i fizičke aktivnosti, postavljanje fitness ciljeva, te praćenje napretka. Slika 2.1 prikazuje izgled navedene aplikacije.



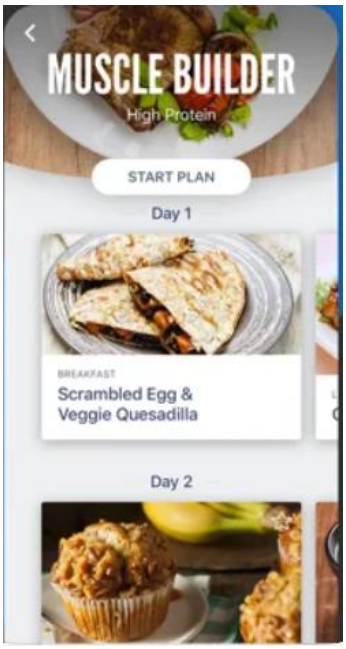
Slika 2.1 *Calorie Counter – MyFitnessPal*

Aplikacija *8fit Workouts & Meal planner* je mobilna aplikacija koja je također podržana na iOS i Android platformama, a prema [2] svojim korisnicima nudi različite efikasne planove tjelovježbe, unaprijed izrađene planove prehrane, te smjernice koje će ih upućivati da postignu željene fitness ciljeve. Slika 2.2 prikazuje korisničko sučelje ove aplikacije



Slika 2.2 *8fit Workouts & Meal planner*

U ovom području rada se nalazi i aplikacija pod nazivom *Fitness Buddy*. Prema [3] ova aplikacija svojim korisnicima služi kao virtualni osobni trener i nutricionist u jednom, sa stotinama različitih tjelovježbi i personaliziranim planovima prehrane i receptima. Sve tjelovježbe nude jasne instrukcije i prikaze, što ovu aplikaciju čini idealnom za početnike.



Slika 2.2 *Fitness Buddy*

3. OPIS PROBLEMA

U ovom poglavlju bit će pojašnjena teorijska podloga o energetskoj vrijednosti hrane, računanja energetske vrijednost hrane pomoću hranjivih sastojaka, te računanja indeksa tjelesne mase i bazalne stope metabolizma.

Jedan od glavnih izvora smanjene kvalitete života, skraćene životne dobi i raznih poremećaja u tijelu, ali i nemogućnosti postizanja željenog fitness cilja predstavlja nepravilna prehrana. Većina ljudi ne mari za količinu energije i količinu određenih hranjivih sastojaka koje unosi u svoje tijelo zbog manjka neupućenosti u sustav računanja kalorija i stvaranje plana pravilne prehrane za svoje potrebe. Nekada je odabir zdravih namirnica za prehranu bio lak, lista opcija je bila relativno kratka, a ljudi su na tržnicama mogli kupovati samo svježe, cjelovite namirnice. Prema [4, str.5] cjelovita hrana obuhvaća voće, povrće, meso, plodove mora, mlijeko i orašaste plodove. No u današnje vrijeme, većina hrane koju osoba može pronaći na polici je prerađena hrana. Tu je i ultra-prerađena hrana u koju prema [4, str. 5-6] ulaze ulja, masti, rafinirani škrob, te razni šećeri. Kada se sve to spoji, postaje sve teže pronaći hranu koja svojom energetskom vrijednošću odgovara energiji koju određena osoba potroši u jednom danu.

Prema [5, str. 1], točna procjena energetske vrijednosti hrane je nužna za nošenje s problemima pravilne prehrane, pothranjenosti i pretilosti. Postavlja se pitanje, što označava pojam energije? Prema [6], razgradnjom hrane se oslobađa određena količina energije. U tijelu postoje tri organske hranjive tvari koje mogu biti korištene kao izvor energije:

- Proteini
- Ugljikohidrati
- Masti

Prema [4, str 7.], preostale hranjive tvari su: vitamini, minerali i voda, ali oni ne služe kao izvor energije. Proteini, ugljikohidrati i masti se još nazivaju i makronutrijentima jer ih tijelo zahtijeva u velikim količinama, stoga se ovaj rad bavi računanjem idealne količine i raspodjele makronutijenata. Energija koju otpuštaju proteini, ugljikohidrati i masti se može mjeriti u kalorijama (cal) – tako malenim mjerama za energiju, da ih jedna jabuka pruža desetke tisuća. Stoga je u svrhu olakšanog izračunavanja uvedena veličina 1000 puta veća od jedne kalorije, kilokalorija ($kcal$) . Prema [5, uvod], kilokalorija predstavlja količinu energije koja je potrebna da bi se kilogram vode ugrijao za 1℃.

Iduće pitanje koje se postavlja je kako odrediti energetsku vrijednost određene namirnice? Prema [4, str. 8] $1g$ proteina i ugljikohidrata iznosi 4 $kcal$, dok $1g$ masti iznosi 9 $kcal$. Ovo su podaci koji će u radu biti korišteni za sve kalkulacije. Kao što je vidljivo, masti imaju veću energetsku gustoću nego proteini i ugljikohidrati, što znači da će obrok koji sadržava više masti rezultirati lakšim povećanjem težine.

Niz zdravih tjelesnih težina je identificiran pomoću zajedničke mjere visine i težine osobe – indeksa tjelesnog mase, čija je skraćenica BMI (engl. *Body mass indeks*). Prema [7] postoje sljedeće BMI kategorije:

- Pothranjenost $=< 18.5$
- Normalna težina $=< 18.5 - 24.9$
- Povećana težina $=< 25 - 29.9$
- Pretilost $>= 30$

Formula za izračunavanje indeksa tjelesne mase je:

$$BMI = \frac{m\,[kg]}{(h[m])^2} \qquad (3-1)$$

Druga važna mjera koja je potrebna za izračunavanje idealnog dnevnog unosa kalorija korisnika je stopa bazalnog metabolizma, čija je skraćenica BMR (engl. *Basal metabolic rate*). Prema [8], BMR predstavlja količinu energije koja je potrebna za održavanje osnovnih životnih funkcija organizma, odnostno minimalna količina enrgije koju bi osoba trebala unijeti u svoj organizam da bi on normalno funkcionirao s obzirom na svoj spol, visinu, težinu i dob.

Postoji više formula za izračunavanje osnovnog BMR-a, a biti će korištene sljedeće formule:

$$BMR(muškarci) = (10 * m[kg]) + (6.25 * h[m]) - (5 * dob) + 5 \qquad (3-2)$$

$$BMR(žene) = (10 * m[kg]) + (6.25 * h[m]) - (5 * dob) - 161 \qquad (3-3)$$

Potom će se osnovni BMR osobe ovisno o fizičkoj aktivnosti množiti sa faktorom fizičke aktivnosti koji će iznositi:

- 1.2 – fizička aktivnost niti jednom ili 1 tjedno
- 1.375 - fizička aktivnost 2 do 3 puta tjedno
- 1.55 - fizička aktivnost 4 do 5 puta tjedno
- 1.725 - fizička aktivnost više od 5 puta tjedno

Da bi osoba na zdrav način osigurala svoj željeni fitness cilj, vrši se posljednja računska operacija da bi se dobila finalna vrijednost dnevnog preporučenog unosa kalorija:

- Nadodati 350 kcal ako osoba želi povećati mišićnu masu
- Oduzeti 350 kilokalorija ako osoba želi izgubiti na težini

Ukoliko osoba ima za cilj zadržati težinu, nema potrebe za posljednjom transformacijom.

3.1. Idejno rješenje aplikacije

Prije početka izrade aplikacije je uvijek potrebno osmisliti kako će ona funkcionirati, te kako će izgledati tok korisnikova kretanja kroz nju.

S obzirom da su svi podaci u aplikaciji vezani za korisnika koji ju koristi, da bi ju korisnik uopće mogao koristiti, on mora posjedovati korisnički račun. Ako korisnik pokuša pristupiti bilo kojoj komponenti stranice unutar aplikacije bez da se prethodno prijavio, mora postojati posrednički dio programa (engl. middleware) koji će ga prilikom pristupanja bilo kojoj stranici preusmjeriti na stranicu za prijavu/registraciju.

Korisnik će za prijavu u aplikaciju koristiti svoju adresu elektroničke pošte i zaporku, a za autentifikaciju će biti korištenova Google-ova Firebase Authentication usluga. Ako korisnik ne posjeduje korisnički račun, on će vršiti registraciju u dva koraka. U prvom koraku će unositi svoje osobne podatke. Osobni podaci korisnika uključuju:

- Ime
- Prezime
- Adresu elektroničke pošte
- Lozinku

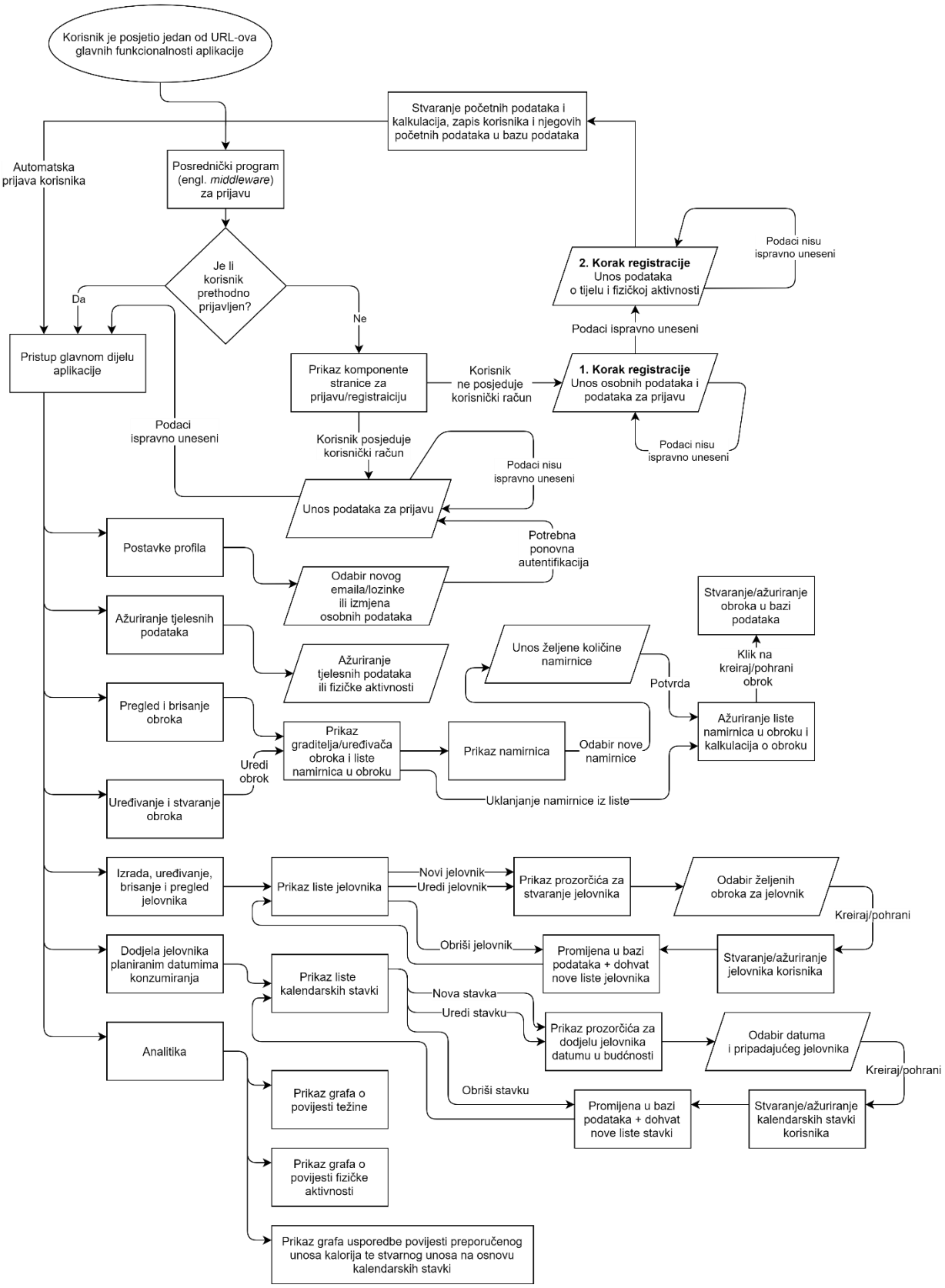
Zatim će na drugom koraku unositi potrebne podatke o svojem tijelu. Podaci o tijelu uključuju:

- Visinu [cm]
- Težinu [kg]
- Dob
- Spol
- Tjednu fizičku aktivnost
- Željeni cilj (mršavljenje, rast mišićne mase, održavanje težine)

Pri čemu će spol i tjedna fizička aktivnost biti izražene enumeracijama. Navedeni podaci će predstavljati početni korak pri izradi analitike i izračunavanju potrebnog dnevnog kalorijskog unosa i idealne preraspodijele hranjivih sastojaka. Nakon izrade komponente koja će prestavljati stranicu za prijavu i registraciju korisnika, potrebno je izraditi sljedeće komponente stranica:

- Postavke profila – korisnik ovdje može uređivati svoje osobne podatke, te mijenjati adresu elektroničke pošte za prijavu i lozinku uz potrebnu ponovljenu autentifikaciju
- Podaci o tijelu – Ova stranica sadrži formu jednaku onoj koja se nalazi na drugom koraku registracije, a služi da bi korisnik mogao ažurirati podakte o svom tijelu. Svi ažurirani podaci o tijelu će pomoći pri generiranju grafova na analitici.
- Pregled i brisanje obroka – S obzirom da komponenta stranice za izradu i uređivanje obroka sadrži dosta kontrola, radi poboljšanja korisničkog iskustva i lakšeg korištenja aplikacije će postojati i ova stranica koja će izlistavati izrađene obroke
- Izrada i uređivanje obroka – Ova stranica će sadržavati sve namirnice koje korisnik može dodati u obrok uz listu svih 'vaganih' namirnica koje predstavljaju obrok. Pod terminom 'vagana' mislimo na namirnicu kojoj je dodjeljena količina s obzirom na to da će se u bazi podataka svi podaci o namirnicama odnositi na količinu od jednog grama. Ova Vue komponenta ujedno predstavlja stranicu za uređivanje i za izradu novog obroka, ovisno o tome je li na stranici za pregled obroka odabrana opcija uređivanja određenog obroka.
- Izrada, uređivanje, pregled i brisanje jelovnika – Nakon stvaranja obroka, korisnik će na ovoj stranici stvorene obroke dodavati u jelovnike. Svaki jelovnik bi se trebao odnositi na sve obroke koje želi unijeti u jednom danu.
- Izrada, uređivanje, pregled i brisanje dnevnih jelovnika – Nakon stvaranja jelovnika, korisnik na ovoj stranici stvorene jelovnike može dodjeljivati raznim datumima kada ih planira konzumirati. Treba paziti da jednom kada se datumu dodjeli jelovnik, taj datum više nije dostupan za odabir dok ga korisnik ne oslobodi od određenog jelovnika. Ova stranica će također pomoći pri izradi analitike jer će omogućiti prikaz korisnikova unosa kalorija po datumima.
- Analitika – Ovo je posljednja komponenta koja će koristiti povijest unosa podataka o tijelu korisnika, te kalorije iz dnevnih jelovnika.

Slika 3.1 prikazuje dijagram toka zamišljenih kretanja korisnika kroz aplikaciju



Slika 3.1 *Dijagram toka korisničkog kretanja kroz aplikaciju*

4. KORIŠTENE TEHNOLOGIJE

Nakon što je osmišljena logike aplikacije, preostalo je na temelju navedenih zahtjeva iz prethodnog poglavlja odabrati tehonologije koje će biti korištene prilikom njezine konkretne realizacije. U ovom poglavlju su opisane sve odabrane tehnologije.

4.1. Osnove Firebase platforme

Kao što je već spomenuto, da bi korisnik pristupio bilo kojem dijelu aplikacije, on mora posjedovati korisnički račun i biti prijavljen. Razlog tomu je što se svi podaci aplikacije grade isključivo oko prijavljenog korisnika i njegovih podataka. Također, druga važna stavka koju je potrebno odabrati je baza podataka za pohranu svih prethodno navedenih korisničkih podataka.

Google je upravo za potrebe ovakvih aplikacija stvorio svoju Backend-as-a-Service platformu pod nazivom Firebase. Ona svojim korisnicima znatno olakšava osmišljavanje, te razvoj web i mobilnih aplikacija, a njezine brojne usluge pružaju svojim korisnicima mogućnost da ne moraju brinuti o upravljanju i programiranju poslužitelja, te se mogu fokusirati na strukturiranje baze podataka.

Neke od vrlo korisnih i važnih usluga koje Firebase platforma pruža su:

- *Google Analytics*
- *Firebase Authentication*
- *Realtime Database*
- *Cloud Firestore*
- *Storage*, za pohranu slika i određenih dokumenata
- *Crashlytics*
- *Hosting*
- *Machine Learning*

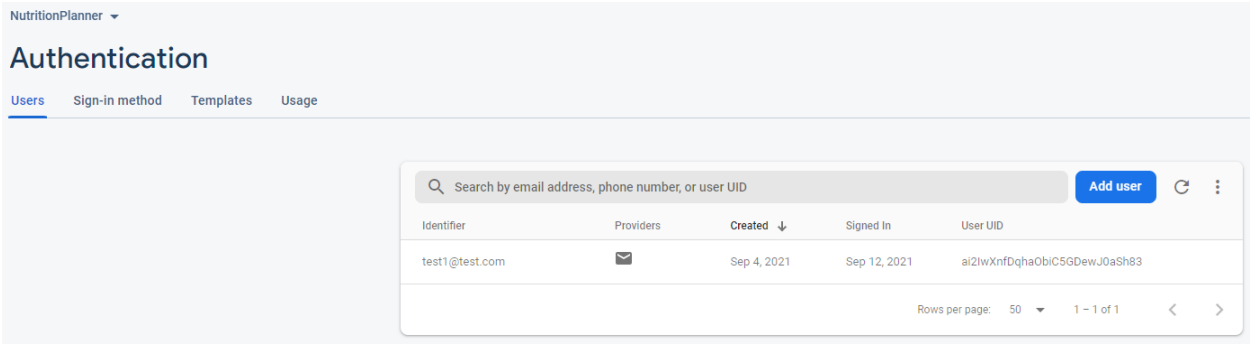
Za potrebe ove aplikacije bit će korištene Firebase Authentication usluga za sustav autentifikacije korisnika, te Cloud Firestore baza podataka za pohranu namirnica i svih korisničkih podataka.

4.1.1. Firebase Authentication

U današnje vrijeme, svaka ozbiljnija aplikacija treba imati ugrađen sustav autentifikacije i autorizacije korisnika. Poznavanje korisničkog identiteta pruža razne mogućnosti poput pohrane njegovih podataka, pohrane podataka o njegovu ponašanju ili izrade personaliziranog izgleda

aplikacije. Prema [9] Firebase Authentication usluga podržava autentifikaciju korisnika putem zaporke, broja mobilnog telefona pa čak i popularnih društvenih mreža poput Facebook-a, Google-a, Twitter-a i drugih.

Da bi se korisnik prijavio u aplikaciju, prvo od njega treba zatražiti podatke za prijavu. To mogu biti kombinacija adrese elektroničke pošte i zaporke ili OAuth token pružen od strane federativnog stvaratelja identiteta (Facebook, Google, Twitter..). Nakon slanja podataka za prijavu, Google-ove usluge sa strane poslužitelja će obaviti verifikaciju podataka i vratiti klijentu odgovor o uspješnosti prijave.



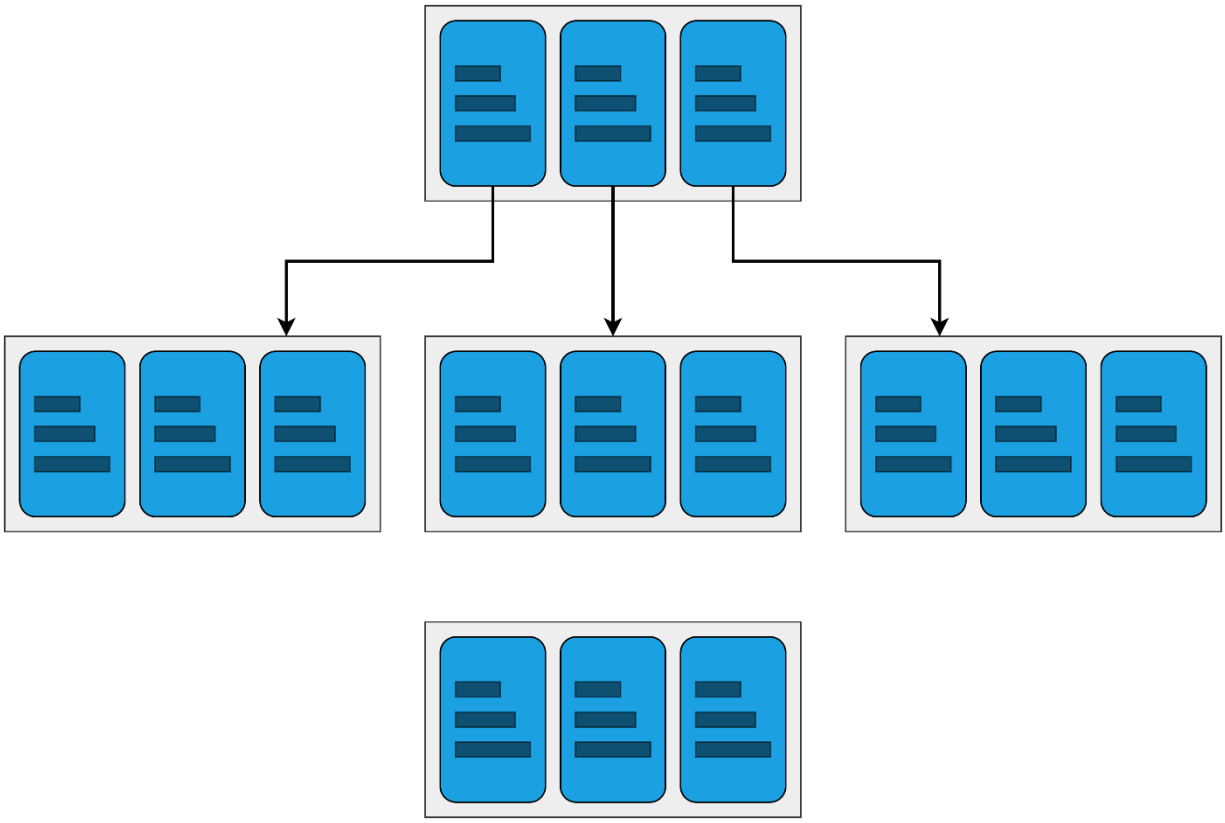
Slika 4.1 *Korisničko sučelje Firebase Authentication usluge*

4.1.2. Cloud Firestore

Prema [10], Cloud Firestore predstavlja NoSql bazu podataka pohranjenu u oblaku računala, a sadrži posebnu strukturu. Svi podaci se pohranjuju u dokumente u obliku 'ključ: vrijednost'. Spomenuti dokumenti se pohranjuju unutar kolekcija, koje predstavljaju kontejnere za dokumente koje možemo koristiti da bi korisnici organizirali svoje podatke i slali upite.

Dokumenti podržavaju brojne tipove podataka, od jednostavnih stringova i brojeva, do kompleksnih, ugniježđenih objekata. Bitno je naglasiti da iako se dokumenti grupiraju u kolekcije, svaki dokument uz svoje podatke može sadržavati i podkolekcije, što je idealno za ovu aplikaciju. Također je moguće zaštititi podatke unutar Cloud Firestore baze podataka ograničavanjem Cloud Firestore sigurnosnih pravila pomoću Firebase Authentication Usluge.

Slika 4.2 prikazuje izgled strukture Cloud Firestore baze podataka.



Slika 4.2 *Cloud Firestore struktura kolekcija, dokumenata i podkolekcija*

4.2. Osnove Vue.js razvojnog okvira

Za izradu web aplikacije koristiti će se razvojni okvir (engl. *framework*) za Vue.js pod nazivom Nuxt.js. Stoga je prvo potrebno shvatiti osnove razvojnog okvira Vue.js.

Prema [11], Vue.js je progresivni razvojni okvir koji kao i ostali razvojni okviri za izgradnju web aplikacija, služi za njihovu olakšanu izradu, a koristi programski jezik Javascript. On se primarno koristi za izgradnju SPA (engl. *Single Page Applications*).

4.2.1. Što je SPA?

Prema [12], aplikacija s jednom stranicom (engl. SPA – *Single Page Applications*) predstavlja web aplikaciju koja se sastoji od samo jednog HTML dokumenta koji služi kao ovojnica svim komponentama aplikacije koje predstavljaju ostale stranice, čija se interakcija s krajnjim korisnikom provodi pomoću programskog jezika Javascript, HTML-a i CSS-a.

Većina programskog koda SPA aplikacije se programira na Frontendu i izvodi u web poslužitelju, nasuprot tradicionalnim web aplikacijama koje se oslanjaju na interakcije s web poslužiteljem i ponovno učitavanje stranica kad god korisnik navigira na novu stranicu unutar web aplikacije.

Prema [13], ovakve aplikacije obavljaju interakciju s korisnikom na način da dinamički mijenjaju dijelove jedne stranice, umjesto da se na svaki korisnikov klik sa servera učitava potpuno nova stranica.

4.2.2. Vue instanca

Prema [14], svaka Vue aplikacija se pokreće kreiranjem instance Vue aplikacije pomoću Vue konstruktora. Prilikom kreiranja instance Vue aplikacije, korisnik konstruktoru predaje objekt koji predstavlja opcije korijenske Vue komponente, koja će predstavljati roditeljsku komponentu svim ostalim komponentama. Aplikaciju je zatim potrebno vezati (engl. *mount*) na određeni HTML element unutar DOM-a. Vue korijenska kompnenta predstavlja ulaznu točku za iscrtavanje cijele hijerarhije komponenata unutar spomenutog HTML elementa.

Kada se instanca Vue aplikacije (a samim time i korijenske Vue komponente) stvori, Vue sva svojstva objekta koje je korisnik definirao kao povratnu vrijednost *data()* funkcije dodaje u svoj sustav reaktivnosti. Jednom kada se vrijednost nekog od svojstava promijeni, HTML element na koji je povezana Vue aplikacija će reagirati i ponovno se iscrtati da bi prikazao najnovije vrijednosti.

Uz *data()* funkciju, postoje i druge opcije koje korisnik može nadodati na korijensku Vue komponentu. Najčešće korištene opcije su:

- Metode (engl. *methods*)
- Računska svojstva (engl. *computed properties*)
- Svojstva koje komponenta može primiti iz roditeljskih komponenti, ukoliko se ne radi o korijenskoj komponenti (engl. *props*)
- Metode životnog ciklusa (*beforeCreate, created, beforeMount, mounted*…)

Slika 4.3 prikazuje primjer instanciranja Vue aplikacije pomoću predanih opcija korijenske Vue komponente.

```
<div id="app">Korijenska komponenta je vezana za ovaj element</div>
```

```
const app = new Vue({
  // Vue component options object
  el: '#app',
  data() {
    return {
      firstName: 'John',
      lastName: 'Doe',
    }
  },
  computed: {
    fullName() {
      return this.firstName + ' ' + this.lastName
    },
  },
  methods: {
    logFullname() {
      console.log(this.fullName)
    },
  },
})
```

Slika 4.3 *Primjer stvaranja Vue aplikacije i vezanja iste na HTML predložak*

4.2.3. Značajke Vue aplikacija

Vue.js je sustav koji omogućuje svojim korisnicima deklarativno ispisivanje podataka unutar DOM-a koristeći jednostavnu sintaksu. Da bi korisnik ispisao podatke Vue.js aplikacije unutar HTML elemenata u obliku teksta, on treba koristiti sintaksu dvostrukih vitičastih zagrada ('{{}}'). Unutar dvostrukih vitičastih zagrada korisnik također može pisati bilo kakve Javascript izraze.

Potrebno je stvoriti novi objekt Vue.js aplikacije i vezati ga za HTML element unutar kojega će ona biti pokrenuta. Slika 4.4 prikazuje primjer stvaranja Vue aplikacije, te ispis teksta unutar HTML elementa koji će sadržavati vrijednost varijable *message* ukoliko ona ne predstavlja lažnu vrijednost, a u suprotnom će se ispisati zamjenska poruka. Svaki put kada se promijeni vrijednost varijabli *message* , to će uzrokovati ponovno iscrtavanje HTML elementa za koji je aplikacija vezana, da bi se na korisničkom sučelju uvijek prikazala najnovija izračunata vrijednost Javascript izraza zapisanog unutar dvostrukih vitičastih zagrada. Na ovaj način je maksimalno umanjena potreba korisnika da direktno pomoću Javascripta dohvaća reference na HTML elemente i osvježava ih najnovijim podacima.

```
<div id="app">
  {{ message ? message : 'Message does not exist' }}
</div>
```

```
const app = new Vue({
  el: '#app',
  data() {
    return {
      message: 'Hello World',
    }
  },
})
```

Slika 4.4 *Interpolacija teksta*

Vue.js također omogućuje proširivanje HTML-a pomoću posebnih HTML atributa nazvanih Vue direktivama. Sve Vue direktive imaju prefiks 'v-' da bi korisnicima bilo jasno da se radi o posebnim atributima koje im je pružio Vue, a ti atributi primijenjuju posebno reaktivno ponašanje na iscrtani DOM. Također, sve Vue direktive omogućuju pisanje Javascript izraza direktno unutar HTML predloška. Najpoznatije vue direktive su:

- **v-bind** – Sintaksa dvostrukih vitičastih zagrada se ne može koristiti za povezivanje s HTML atributima. Da bi se HTML atribut povezao s podatkom iz Vue aplikacije ili da bi se njihova vrijednost odredila Javascript izrazom, potrebno je koristiti v-bind direktivu. Skraćenica za ovu direktivu je znak ':'. Slika 4.5 prikazuje korištenje v-bind direktive.

```
<div id="app">
  <p :title="title">Hover over me to see my newest title!</p>
</div>
```

```
const app = new Vue({
  el: '#app',
  data() {
    return {
      title: 'This page was loaded on ' + new Date().toLocaleString(),
    }
  },
})
```

Slika 4.5 *Korištenje v-bind direktive*

- **v-model** – vrlo moćna Vue direktiva koja omogućuje dvosmjerno vezanje podataka (engl. *Two way data binding*). Ona doslovno veže *'value'* atribute HTML elemenata za unos

podataka (<input/>, <select/>, <textarea/> i slični elementi) sa varijablama koje je korisnik deklarirao unutar svoje Vue aplikacije. Bilo koja izmjena podatka uzrokovana promjenom od strane programera putem Javascripta ili korisnika (primjerice kada pritisne tipku za unos teksta), vezana varijabla i *'value'* atribut HTML elementa će se uvijek međusobno osvježavati.

```
<div id="app">
  <input v-model="inputText" type="text" />
</div>
```

```
const app = new Vue({
  el: '#app',
  data() {
    return {
      inputText: '',
    }
  },
})
```

Slika 4.6 *Korištenje v-model direktive*

- **v-for** – Koristi se da bi se prikazala lista stavki iz polja podataka ili svojstva nekog objekta. Ova direktiva zahtjeva posebnu sintaksu, gdje korisnik odabire alias za jednu stavku iz liste. Korištenje ove direktive prikazano je slikom 4.7.

```
<div id="app">
  <p v-for="(item, index) in items" :key="index">
    {{ item.title }}
  </p>
</div>
```

```
const app = new Vue({
  el: '#app',
  data() {
    return {
      items: [
        { title: 'title 1' },
        { title: 'title 2' },
        { title: 'title 3' },
        { title: 'title 4' },
      ],
    }
  },
})
```

Slika 4.7 *Korištenje v-for direktive*

- **v-if** – Koristi se za uvjetno iscrtavanje HTML elemenata ili Vue komponenti. Blok će biti stvoren i prikazan samo ukoliko izraz predan ovoj direktivi odgovara istini. Ovu direktivu

je također moguće koristiti u kombinacijama sa v-else ili v-else-if direktivama. Korištenje uvjetnog iscrtavanja prikazano je slikom 4.8.

```
<div id="app">
  <p v-if="rendered">rendered</p>
  <p v-else>rendered is falsy</p>
</div>
```

```
const app = new Vue({
  el: '#app',
  data() {
    return {
      rendered: true,
    }
  },
})
```

Slika 4.8 *Korištenje v-if direktive*

- **v-on** – Ova direktiva omogućuje osluškivanje DOM događaja i izvođenje Javascript koda kada do njih dođe. Ovoj direktivi korisnik može predati funkciju rukovatelja događaja koja će odmah kao prvi argument primiti objekt nativnog DOM događaja. Skraćenica za ovu direktivu je znak '@'. Osluškivanje događaja nad gumbom prikazano je slikom 4.9.

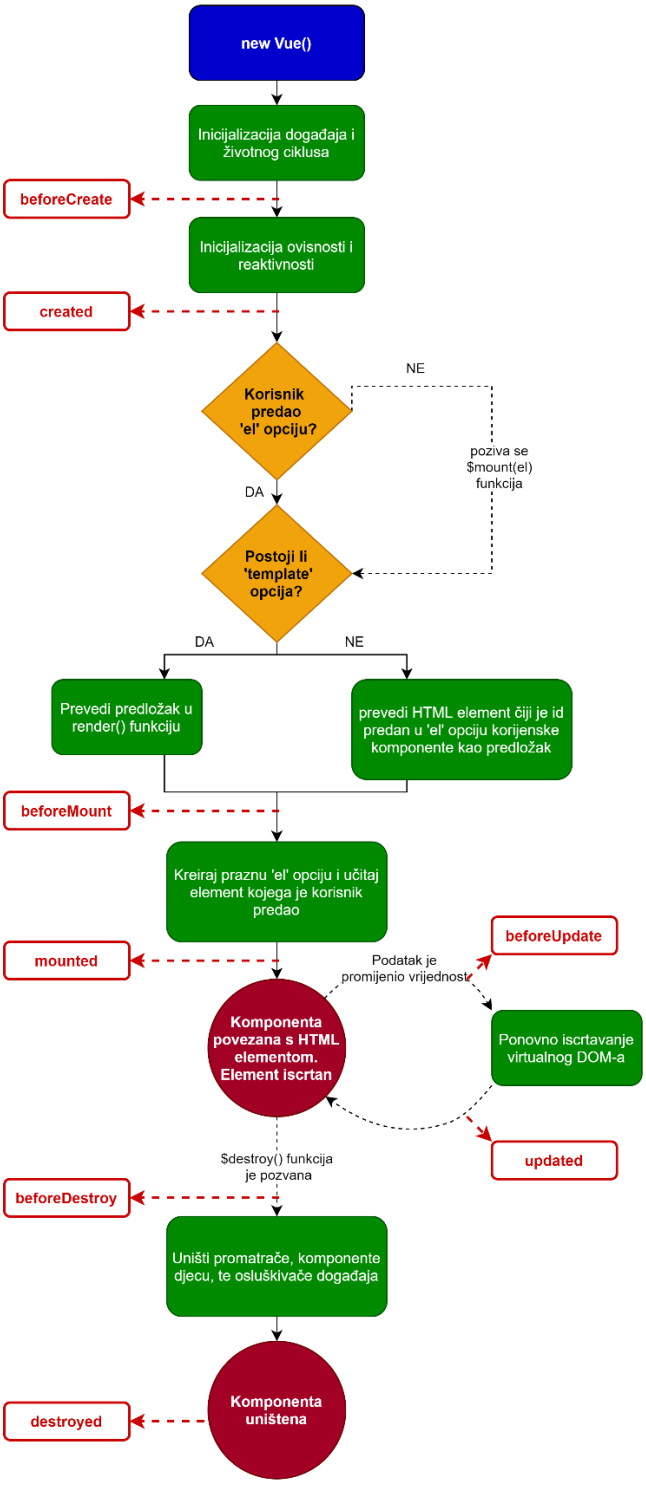
```
<div id="app">
  <button @click="incrementCounter">{{ counter }}</button>
</div>
```

```
const app = new Vue({
  el: '#app',
  data() {
    return {
      counter: 0,
    }
  },
  methods: {
    incrementCounter(event) {
      this.counter++
      console.log('Event object', event)
    },
  },
})
```

Slika 4.9 *Korištenje v-on direktive*

4.2.4. Životni ciklus Vue aplikacije

Danas većina modernih aplikacija ima nekakav životni ciklus. Životni ciklus aplikacije programeru znatno olakšava pronalazak pogrešaka, te odabir načina kojim redoslijedom će se izvoditi određeni dijelovi koda bitni za rad komponenata. Slika 4.10 prikazuje životni ciklus Vue aplikacije prema [14].



Slika 4.10 *Životni ciklus Vue aplikacije*

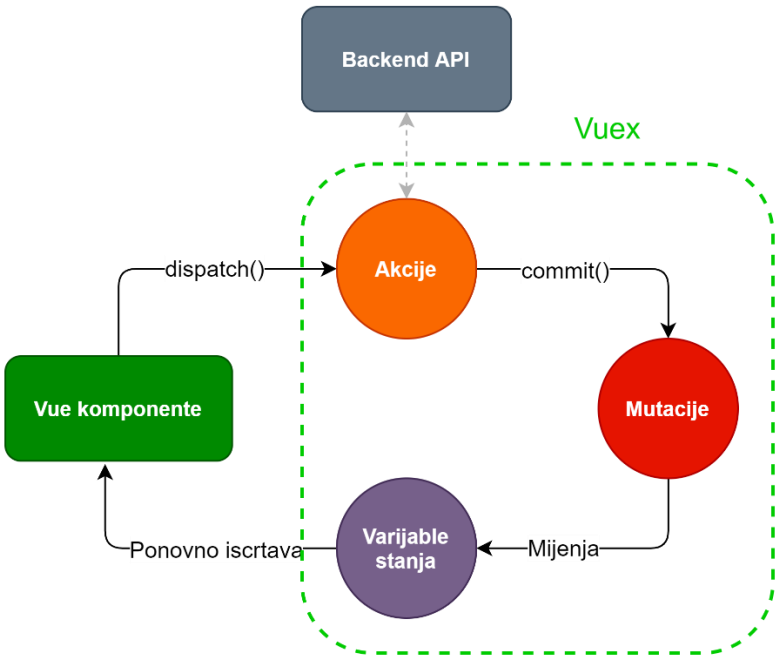
4.2.5. Vuex

Aplikacije vrlo često rastu u kompleksnosti zbog toga što su u njima višestruki dijelovi stanja aplikacije raspršeni po mnogim komponentama, a u isto vrijeme postoje razna međudjelovanja između tih dijelova stanja. Kako bi riješio ovaj problem, Vue je stvorio biblioteku za upravljanje varijablama stanja aplikacije pod nazivom *Vuex*.

Prema [15], Vuex je globalni *Singleton* koji omogućuje programerima da na jednom mjestu drže sve globalne varijable aplikacije koje trebaju biti dostupne na više mjesta, a one predstavljaju stanje aplikacije (engl. *application state*) kojemu sve komponente aplikacije imaju pristup i mogu ga izmijenjivati po unaprijed određenim pravilima.

Jedna datoteka Vuex store-a bi se trebala sastojati od barem tri objekta:

- Stanje (engl. *state*) – sadrži varijable koje su dostupne svim komponentama Vue aplikacije
- Mutacije (engl. *mutations*) – sadrži funkcije, gdje bi idealno svaka funkcija trebala postavljati samo jednu varijablu stanja na vrijednost koja joj je predana kao drugi argument, gdje je prvi argument funkcije mutacije uvijek cijeli objekt stanja. Mutacije ne smiju izvršavati asinkrone operacije, one moraju biti sinkrone funkcije.
- Akcije (engl. *actions*) – sadrži funkcije slične mutacijama, ali razlika je u tome što umjesto da one mijenjaju varijable stanja, one se koriste da pozivaju mutacije pomoću funkcije *commit()*. Akcije mogu sadržavati asinkrone operacije i često tomu i služe. Akcije pružaju sučelje *Vuex store-a* prema komponentama koje ih pozivaju pomoću funkcije *dispatch()*.



Slika 4.11 *Način funkcioniranja Vuex store-a*

4.3. Nuxt.js

Prema [16], Nuxt.js je razvojni okvir za Vue.js koji programeru pruža odlične značajke razvoja poput iscrtavanja na strani servera (engl. *Server Side Rendering*, skraćenica *SSR*), automatski generirane rute za Vue router, poboljšan način upravljanja meta oznakama i SEO (engl. Search engine optimization) poboljšanja.

Da bi poboljšao SEO, Nuxt.js koristi iscrtavanje na strani server (SSR). Prilikom inicijalnog posjeta web stranice, SSR pravi AJAX pozive u svojim posebim metodama životnog ciklusa koje se mogu izvoditi na serveru i povlači početne podatke za aplikaciju. Prije nego se životni ciklus Nuxt aplikacije na strani servera završi, Nuxt iscrtava Vue.js komponente u obliku HTML stringova na strani servera. Zatim te stringove šalje natrag klijentu, odnosno web pretraživaču koji će znati kako prikazati taj HTML string u obliku web stranice popunjene podacima. Prema [17], ova značajka je od velike važnosti jer omogućuje vrlo brzo čitanje DOM elemenata sa Google-ovim SEO parserom. Google-ov SEO parser analizira sve DOM elemente ogromnom brzinom čim se učita DOM web stranice. S druge strane, tipične SPA aplikacije koje su građene s razvojnim okvirima poput Vue.js-a, React-a i Angulara dohvaćaju podatke s backend servera tek nakon učitavanja DOM-a, te zbog toga Google-ov SEO parser ne može analizirati sve DOM elemente jer oni još uvijek nisu prikazani dok odgovarajući podaci ne stignu sa servera.

Nuxt.js ima vrlo sličnu arhitekturu kao i Vue.js, postoje dvije glavne razlike:

- Vue router
- Glavna App.vue komponenta

Prilikom korištenja Vue.js razvojnog okvira, programer sam mora popunjavati Vue Router i vezati koja komponenta će se prikazivati na kojoj ruti, dok Nuxt.js sam generira Vue Router objekt baziran na direktoriju *pages* i strukturi direktorija i Vue komponenti unutar njega.

Nuxt.js podržava još jednu značajku koja je vezana za strukturu projekta, a to je mogućnost definiranja predložaka omotača (engl. *layout*). Vue.js aplikacije imaju glavnu App.vue datoteku, koja je omotač za korijensku Vue komponentu i sve ostale kompnente aplikacije. Nuxt.js omogućuje korištenje više takvih omotača, gdje svaka komponenta stranice može definirati koji će biti njezin omotač. Primjerice, ako postoji komponenta stranice za prijavu i registraciju, nema potrebe da se iznad nje nalazi navigacijska traka i da joj se sa strane nalazi navigacijska ladica. Stoga se za ovakvu stranicu može stvoriti poseban *layout*.

5. MODEL BAZE PODATAKA

Prije same izrade aplikacije, potrebno je isplanirati i pojasniti korišteni model podataka unutar aplikacije. Baza podataka će se sastojati od 2 glavne kolekcije:

- Korisnici (engl. *Users*)
- Namirnice (engl. *Groceries*)

Kolekcija 'Namirnice' (engl. *Groceries*) će sadržavati podatke o svim namirnicama koje će biti ponuđene korisniku kada bude htio izgraditi svoje obroke. Ovo je kolekcija koju dijele svi korisnici, te ju svi koriste isključivo za čitanje. Unutar ove kolekcije će se nalaziti po jedan dokument za svaki tip namirnice, da bi namirnice bile logički grupirane. Postoji pet tipova namirnica, odnosno pet dokumenata unutar kolekcije 'Namirnice':

- Voće (engl. *Fruit*)
- Povrće (engl. *Vegetable*)
- Meso (engl. *Meat*)
- Plodovi mora (engl. *Seafood*)
- Mliječni proizvodi (engl. *Dairy products*)

U svrhu predstavljanja jedne namirnice bit će korištena klasa *Grocery*. Svaka namirnica posjeduje jedinstveni ID, naziv, količinu kalorija po gramu, te količine hranjivih sastojaka po gramu (proteina, ugljikohidrata i masti). Svaki dokument jednog tipa namirnica će sadržavati polje svih namirnica koje mu pripadaju.

Sljedeća klasa koja je potrebna za stvaranje obroka se zove *WeightedGrocery*. Kada korisnik stvara svoj obrok, on dodaje namirnicu u njega klikom na gumb s oznakom '+' koji će se nalaziti pokraj svake namirnice. Zatim mu se treba otvoriti prozorčić u kojemu će unijeti količinu željene namirnice u obrok. Količina će biti izražena u gramima. Jednom kada korisnik unese ispravnu količinu i potvrdi svoj unos, stvara se objekt klase *WeightedGrocery*. Naziv *WeightedGrocery* polazi od toga da je korisnik sada odabrao namirnicu čiji su podaci u bazi podatka izraženi u količinama po jednom gramu, odabrao željenu količinu te namirnice, te ju dodao u svoj obrok. Svaki objekt *WeightedGrocery* klase se sastoji od objekta namirnice, količine namirnice u gramima te objekta *nutrients* koji sadrži izračunatu ukupnu količinu kalorija i ostalih hranjivih sastojaka s obzirom da ovo više nije namirnica od jednog grama.

Prije nastavka je potrebno pojasniti klasu koja će predstavljati dokument jednog korisnika u bazi podataka. Svaki korisnik će biti predstavljen klasom *User* i imati će svoj jedinstveni ID koji odgovara identifikacijskom broju kojega će generirati Firebase Authentication usluga prilikom njegove registracije. Isti taj ID će biti korišten kao ID dokumenta u kojega se korisnik pohranjuje. Objekt korisnika osim jedinstvenog ID-ja sadrži datum njegova stvaranja, te dva dodatna objekta koji će sadržavati njegove:

- Osobne podatke (objekt *personalInfo)*
- Tjelesne podatke i kalkulacije (objekt *bodyInfo)*

U potpoglavlju 3.2, Slika 3.1 prikazuje dijagram idejnog rješenja aplikacije. Taj dijagram prikazuje zamišljenu registraciju u dva koraka. Prvi korak predstavlja unos osobnih podataka korisnika, a drugi unos tjelesnih podataka. Upravo iz tog razloga objekt User sadrži navedena dva objekta *personalInfo*, te *bodyInfo.* Objekt *personalInfo* će sadržavati ime, prezime, te adresu elektroničke pošte. Objekt bodyInfo je nešto kompliciraniji, a sadržavati će korisnikovu visinu, težinu, dob, spol, količinu fizičke aktivnosti, željeni fitness cilj, te 2 dodatna objekta koji se izračunavaju prilikom korisnikove registracije, te svaki put kada korisnik na svom profilu izmijeni podatke o svojem tijelu. Navedena dva objekta predstavljaju:

- Izračune (objekt *calculations*) – ovaj objekt na osnovu ispravno unesenih podataka o tijelu sadržava korisnikov indeks tjelesne mase (BMI, engl. *Body mass index)* i stopu bazalnog metabolizma (BMR, engl. *Basal metabolic rate*). Korištene su formule iz poglavlja 3
- Idealnu raspodjelu hranjivih sastojaka (objekt *distribution*) – Ovaj objekt sadrži idealnu dnevnu raspodjelu hranjivih sastojaka koje bi korisnik trebao unijeti da bi dosegao izračunatu stopu bazalnog metabolizma (BMR) sadržanu u prethodnom pojašnjenom objektu.

Posljednje svojstvo koje objekt bodyInfo sadržava će predstavljati datum njegova stvaranja, a ovo svojstvo je dodano u svrhu kreiranja analitike, jer će se objekt *bodyInfo* koristiti svaki put kada korisnik osvježi svoje podatke na komponenti koja predstavlja stranicu korisničkog profila.

S obzirom da se svi podaci osim namirnica vežu za svoje vlasnike, svaki dokument korisnika će sadržavati 4 kolekcije:

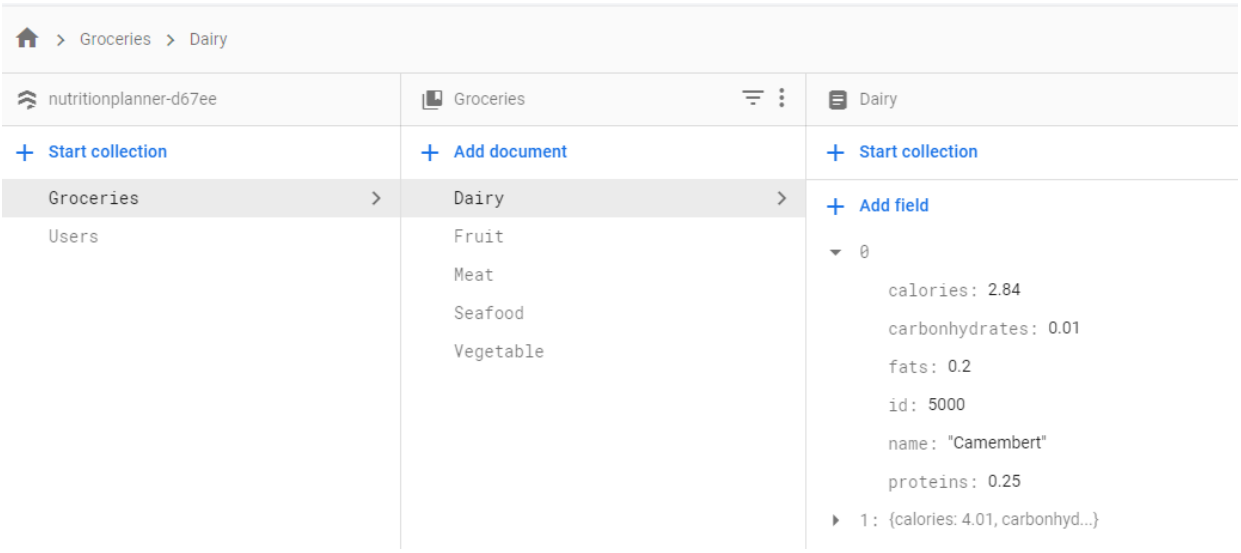
- Obroci (engl. *Meals*)
- Jelovnici (engl. *Menus*)
- Kalendar (engl. *Calendar*)
- Tjelesna povijest (engl. *Body history)*

Svaki dokument kolekcije *Meals* će sadržavati objekt klase *Meal* koji predstavlja jedan korisnikov obrok. Svaki obrok će se sastojati od jedinstvenog ID-ja, imena, datuma kreiranja, polja objekata klase *WeightedGrocery*, te objekta nutrients, koji će imati svrhu kao i u klasi *WeightedGrocery*. On sadrži ukupan broj kalorija, te hranjivih sastojaka cijelog obroka u gramima.

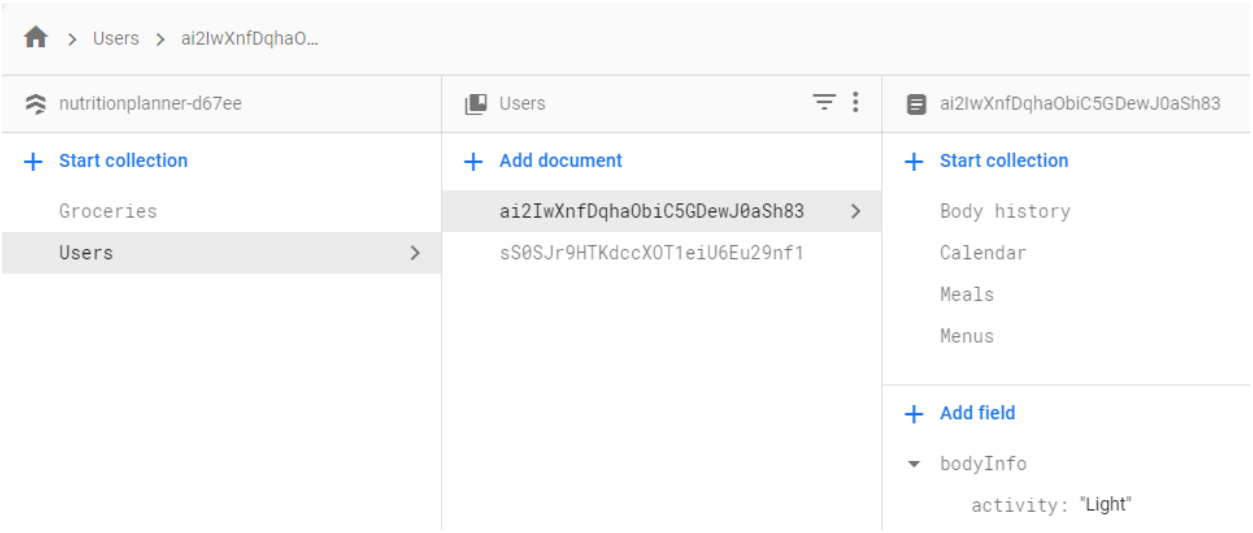
Kolekcija *Menus* će sadržavati dokumente koji predstavljaju jelovnika, a jelovnici su objekti klase *Menu.* Svaki jelovnik će sadržavati jedinstveni id, naziv, datum kreiranja, te polje *meals,* koje predstavlja polje jedinstvenih ID-jeva svih obroka koje je korisnik dodao u ovaj jelovnik.

Kolekcija *Calendar* će sadržavati dokumente koji predstavljaju kalendarske stavke, pri čemu jedna kalendarska stavka predstavlja objekt koji će sadržavati jedinstveni ID, ID jelovnika kalendarske stavke, te ponoć datuma na koji se spomenuti jelovnik planira konzumirati, izražen u milisekundama od 01.01.1970.. Na ovaj način će se izbjeći neočekivana ponašanja aplikacije jer biti onemogućeno stvoriti više kalendarskih stavki koje pripadaju istom datumu. U svrhu stvaranja ovakvih objekata je stvorena jednostavna klasa *CalendarItem*.

Kolekcija *Body history* će sadržavati povijest korsnikovih *bodyHistory* objekata, počevši od trenutka registracije, a zatim svaki put kada on ažurira svoj profil stvarati će se novi zapis. Bitno je napomenuti da će se pri svakom stvaranju zapisa o povijesti korisničkog tijela, novom dokumentu dodijeliti jedinstveni ID predstavljen milisekundama koje su prošle od 01.01.1970. do ponoći trenutnog dana. Na taj način se broj zapisa po danu ograničava na maksimalno jedan, a u slučaju da korisnik pohrani neispravan podatak, može ga ažurirati tako što će samo unijeti ispravan podatak i opet pohraniti podatke o svojem tijelu.

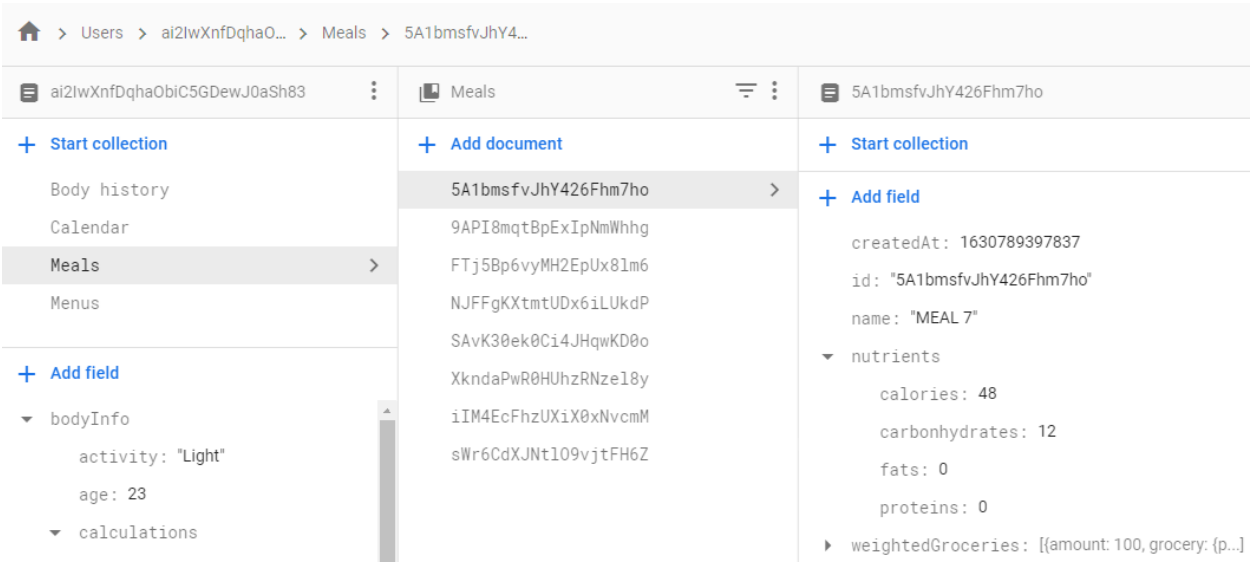


Slika 5.1 *Kolekcija Groceries uz dokumente koji predstavljaju različite vrst3 namirnica*

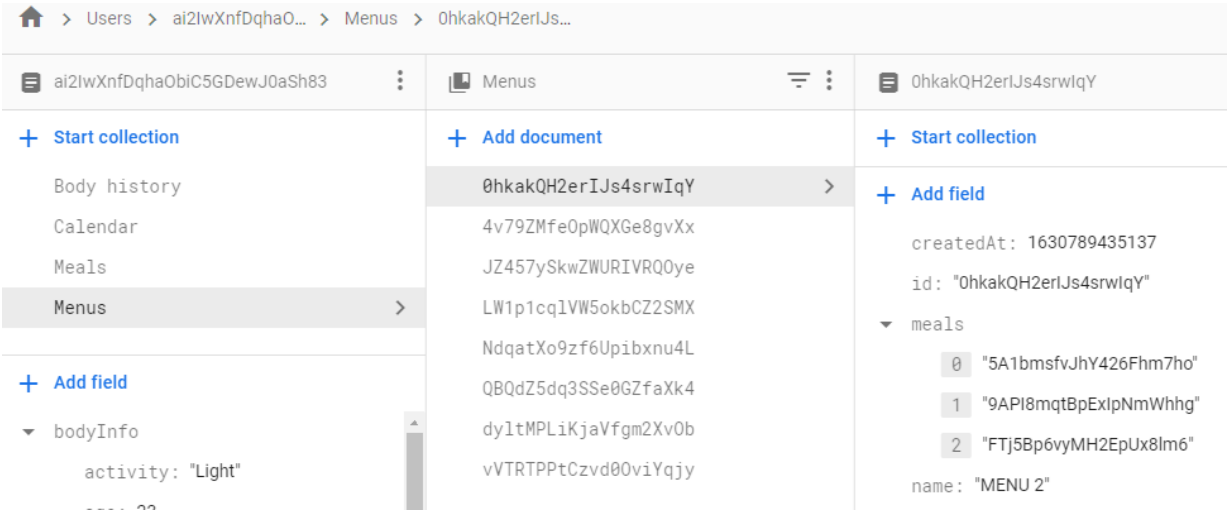


Slika 5.2 *Kolekcija Users uz popis korisnikovih podkolekcija*

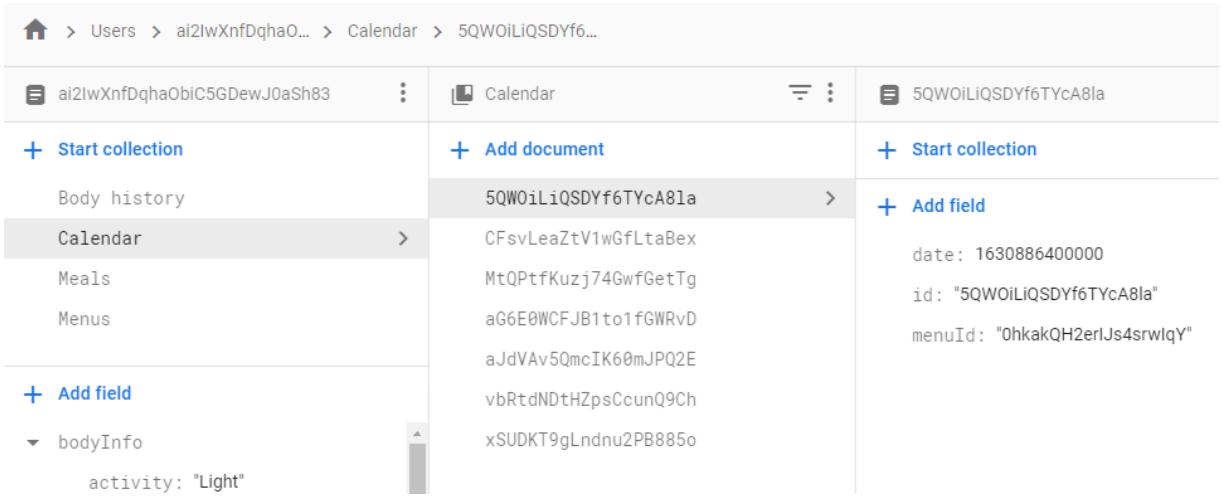
Slika 5.3 *Izgled objekta jednog korisnika*



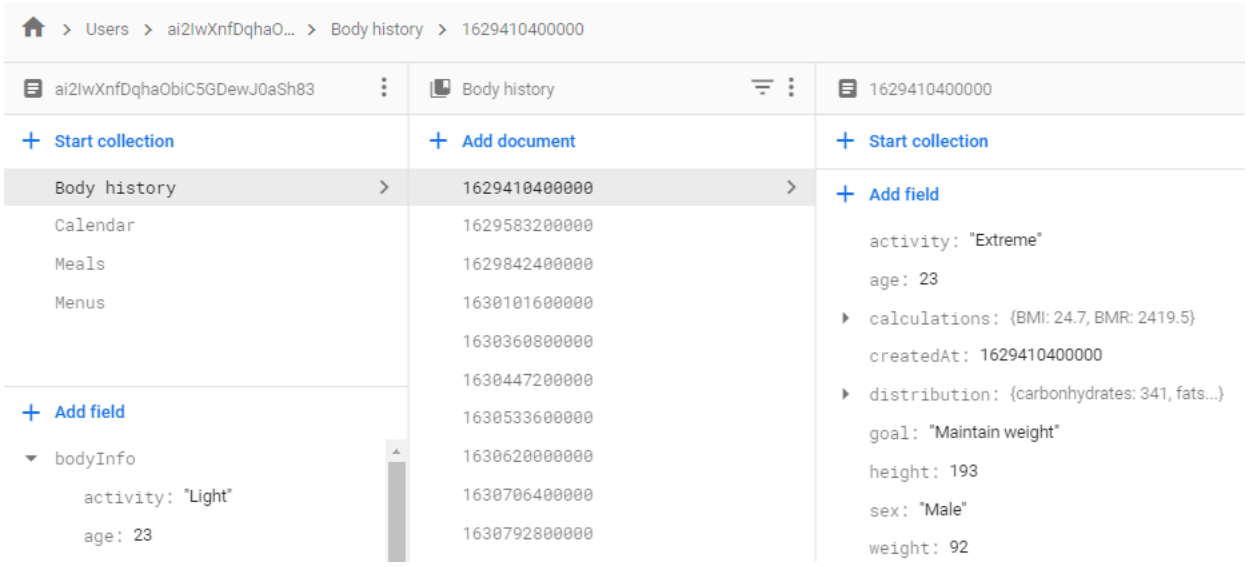
Slika 5.4 *Izgled podkolekcije Meals i jednog objekta korisničkog obroka klase Meal*



Slika 5.5 *Izgled podkolekcije Menus i jednog objekta korisničkog jelovnika klase Menu*



Slika 5.6 *Izgled podkolekcije Calendar i jedne kalendarske stavke klase CalendarItem*



Slika 5.7 *Izgled podkolekcije Body history i objekta klase BodyInfo*

6. IZRADA APLIKACIJE

Nakon pojašnjenja problema, planiranja rješenja aplikacije, donošenja odluke o korištenim platformama za implementaciju projekta, te osmišljavanja modela podataka, može se započeti s implementiranjem konkretnog programskog rješenja. Ovo poglavlje će za svaku komponentu koja predstavlja jednu stranicu sadržavati jedno poglavlje, a započet će s postavljanjem projekta.

6.1. Postavljanje projekta

Jedna od glavnih stavki koje su potrebne za uspješnu izradu projekta i mogućnost njegova daljnjeg održavanja je njegovo pravilno postavljanje. Stoga je postavljanje projekta prva stvar koja će biti odrađena prije izrade prve komponente stranice.

Nuxt.js sve konfiguracijske podatke sadrži u datoteci nuxt.config.js. Za početak je u ovoj datoteci potrebno registrirati sve module i proširenja koji će se koristiti uz njihove konfiguracije. Prvi i osnovni modul koji će biti korišten u ovoj aplikaciji je *Vuetify.js*. Stoga se u nuxt.config.js dodaje paleta osnovnih boja, ikonice i željeni font. Slika 6.1 prikazuje Vuetify opcije podešene u nuxt.config.js datoteci.

```
vuetify: {
  treeShake: true,
  defaultAssets: {
    font: {
      family: 'Poppins',
    },
    icons: 'mdi',
  },
  theme: {
    themes: {
      light: {
        primary: '#110F48',
        secondary: '#E63956',
        tertiary: '#D8F4FF',
        accent: '#F8F8FA',
        error: '#FF5252',
        info: '#2196F3',
        success: '#4CAF50',
        warning: '#FFC107',
      },
    },
  },
},
```

Slika 6.1 *Vuetify konfiguracija*

Zatim je potrebno instalirati npm paket *@nuxtjs/toast* da bi na olakšan način korisniku prikazivali poruke o uspjehu, neuspjehu, te informacijske poruke. Odabran je oblik poruke, trajanje, te mogućnost zadržavanja prelaskom miša.

```
toast: {
  keepOnHover: true,
  duration: 1500,
  theme: 'bubble',
},
```

Slika 6.2 *@nuxtjs/toast konfiguracija*

Najvažniji npm paket kojega je potrebno instalirati je *@nuxtjs/firebase*. Nakon instaliranja ovog modula, potrebno je stvoriti njegov konfiguracijski objekt, tako da bi znao na koju Firebase aplikaciju će se vezati ovo programsko rješenje. Tu je i drugi objekt pod nazivom *servisi* (engl. *services*). U njemu je potrebno dodati sve Firebase usluge koje će se korisiti u ovoj aplikaciji. Za svaki uslugu postoje predefinirane opcije i one će se koristiti za Cloud Firestore uslugu, a za Firebase Authentication uslugu će se koristiti posebna konfiguracija koja omogućuje okidanje akcije iz *Vuex store-a* svaki put kada se stanje prijavljenog korisnika promijeni.

```
'@nuxtjs/firebase',
{
  config: {
    apiKey: [redacted],
    authDomain: [redacted],
    projectId: [redacted],
    storageBucket: [redacted],
    messagingSenderId: [redacted],
    appId: [redacted],
  },
  services: {
    auth: {
      persistence: 'local', // default
      initialize: {
        onAuthStateChangedAction: 'onAuthStateChangedAction',
        subscribeManually: false,
      },
    },
    firestore: true,
  },
```

Slika 6.3 *@nuxtjs/firebase konfiguracija*

Posljednje vanjsko proširenje koje je potrebno dodati u projekt se zove apexcharts.js, u svrhu stvaranja grafova na analitici. Ovo proširenje nije potrebno dodatno konfigurirati.

Zatim je u aplikaciju potrebno dodati *Vuex store*. Prvi razlog tomu je taj što će gotovo svaka komponenta koja predstavlja stranicu čitati puno korisničkih podataka, a drugi je taj jer je prilikom dodavanja @nuxt/firebase konfiguracije dodana Vuex akcija koja će se pozivati svaki puta kada se korisnik prijavi ili odjavi. To je vidljivo na slici 6.2. Da bi se dodao *Vuex store* unutar Nuxt.js aplikacije, jedino što je potrebno napraviti je unutar projektnog direktorija *store* dodati datoteku *index.js*. Zatim je u njoj potrebno definirati objekt stanja, akcije i mutacije.

Slika 6.4. prikazuje inicijalni objekt stanja, a on sadržava:

- Podatke o navigacijskoj traci i navigacijskoj ladici.
- Podatke o trenutno prijavljenom korisniku
- Sve postojeće namirnice
- Korisnikove obroke, jelovnike, kalendarske stavke i povijest tjelesnih podataka

```
export const state = () => ({
  navigationDrawer: false,
  navbarTitle: strings.DRAWER_LABELS.DEFAULT,
  user: null,
  meals: [],
  menus: [],
  calendarItems: [],
  bodyHistory: [],
  groceries: {
    [enums.GROCERY_TYPE.FRUIT]: [],
    [enums.GROCERY_TYPE.VEGETABLE]: [],
    [enums.GROCERY_TYPE.MEAT]: [],
    [enums.GROCERY_TYPE.DAIRY]: [],
    [enums.GROCERY_TYPE.SEAFOOD]: [],
  },
})
```

Slika 6.4 *Globalno stanje aplikacije*

Za svako navedeno svojstvo objekta stanja treba definirati odgovarajuću mutaciju koja će ga postavljati, te odgovarajuću akciju koja će pozivati mutaciju. Slika 6.5 prikazuje mutacije za postavljanje *user* varijable stanja (*SET_USER*) i njezino uklanjanje (*CLEAR_USER*). Slika 6.6. prikazuje akciju *$fetchUser* za dohvaćanje podataka iz *kolekcije* Users o prijavljenom korisniku i akciju *$clearUser* za uklanjanje podatka o prijavljenom korisniku.

```
SET_USER(state, user) {
  state.user = Object.assign( target: {}, user)
},
CLEAR_USER(state) {
  state.user = null
},
```

Slika 6.5 *Mutacije za postavljanje i uklanjanje stanja prijavljenog korisnika*

```
async $fetchUser({ commit }) {
  try {
    const document = await this.$fire.firestore
      .collection(enums.COLLECTIONS.USERS) CollectionReference<DocumentData
      .doc(this.$fire.auth.currentUser.uid) DocumentReference<DocumentDa
      .withConverter(UserConverter) DocumentReference<User>
      .get()

    if (!document.exists)
      throw new Error(strings.TOASTS.ERROR.USER_DOCUMENT_UNDEFINED)
    commit('SET_USER', document.data())
  } catch (e) {
    console.log(e.message)
  }
},
$clearUser({ commit }) {
  commit('CLEAR_USER')
},
```

Slika 6.6 *Akcije za dohvaćanje podataka o prijavljenom korisniku i njegovu uklanjanju.*

Za primjer će biti prikazane akcije i mutacije za dohvaćanje korisnikovih obroka i njihovo uklanjanje. Važno je napomenuti da su akcije i mutacije za postavljanje i uklanjanje jelovnika, kalendarskih stavki i povijesti tjelesnih podataka gotovo identične onima za manipuliranje obrocima. Slika 6.7. prikazuje mutacije za manipuliranje obrocima, a slika 6.8. prikazuje akcije za manipuliranje obrocima.

```
SET_MEALS(state, meals) {
  state.meals.splice( start: 0, state.meals.length, ...meals)
},
CLEAR_MEALS(state) {
  state.meals.splice( start: 0, state.meals.length)
},
```

Slika 6.7 *Mutacije za manipulaciju korisničkim obrocima*

```
async $fetchMeals({ commit }) {
  try {
    const collection = await this.$fire.firestore
      .collection(enums.COLLECTIONS.USERS) CollectionReference<DocumentData>
      .doc(this.$fire.auth.currentUser.uid) DocumentReference<DocumentData>
      .collection(enums.COLLECTIONS.MEALS) CollectionReference<DocumentData>
      .get()

    commit('CLEAR_MEALS')
    const meals = []
    collection.forEach( callback: (document :QueryDocumentSnapshot<DocumentData> ) => meals.push(document.data()))
    commit('SET_MEALS', meals)
  } catch (e) {
    console.log(e.message)
  }
},
$clearMeals({ commit }) {
  commit('CLEAR_MEALS')
},
```

Slika 6.8 *Akcije za manipulaciju korisničkim obrocima*

Glavni razlog ovakvog načina definiranja mutacija i akcija za manipuliranjem svakom varijablom stanja je taj da bi se svakom prijavom korisnika u Vuex store trebali odmah povući svi njegovi podaci kojima bi zatim sve komponente mogle pristupati bez ikakve potrebe za time da svaka komponenta stranice uvijek povlači sve podatke koji joj trebaju. Isto tako, kada se korisnik odjavi iz aplikacije, svi podaci bi se trebali očistiti za dolazak novog korisnika.

Upravo zbog toga je definirana posebna akcija pod nazivom *onAuthStateChanged()* koja se poziva svaki puta kada se neki korisnik prijavi ili odjavi iz Firebase Authentication usluge. Slika 6.9. prikazuje akciju *onAuthStateChanged()*.

```
async onAuthStateChangedAction({ dispatch }, { authUser }) {
  if (!authUser) {
    await Promise.all( values: [
      dispatch('$clearUser'),
      dispatch('$clearMeals'),
      dispatch('$clearMenus'),
      dispatch('$clearCalendarItems'),
      dispatch('$clearBodyHistory'),
    ])

    this.$router.replace( location: '/auth')
  } else {
    await Promise.all( values: [
      dispatch('$fetchUser'),
      dispatch('$fetchGroceries'),
      dispatch('$fetchMeals'),
      dispatch('$fetchMenus'),
      dispatch('$fetchCalendarItems'),
      dispatch('$fetchBodyHistory'),
    ])
  }
},
```

Slika 6.9 *Akcija onAuthStateChanged()*

Za kraj je potrebno definirati dvije posredničke funkcije (engl. Middleware) . Prva se zove *authRedirect()* i ona će se postaviti na roditeljsku komponentu svim komponentama koje predstavljaju stranice (Vue za ovakvu komponentu koristi naziv *layout*), osim stranice za prijavu. Ova funkcija služi da bi se korisnika preusmjerilo na stranicu za prijavu ukoliko on pokuša pristupiti nekoj stranici, a da se prethodno nije prijavio. Ova funkcija će se pozvati prije instanciranja svake komponente stranice osim stranice za prijavu.

```
export default function ({ app, redirect }) {
  if (!app.$fire.auth.currentUser) return redirect('/auth')
}
```

Slika 6.10 *authRedirect() middleware*

Druga posrednička funkcija se zove *indexRedirect(),* a postavlja se na stranicu za prijavu da bi se korisnika preusmjerilo na pregled njegovih obroka ukoliko pokuša pristupiti stranici za prijavu a već je prijavljen.

```
export default function ({ app, redirect }) {
  if (app.$fire.auth.currentUser) return redirect('/')
}
```

Slika 6.11 *indexRedirect() middleware*

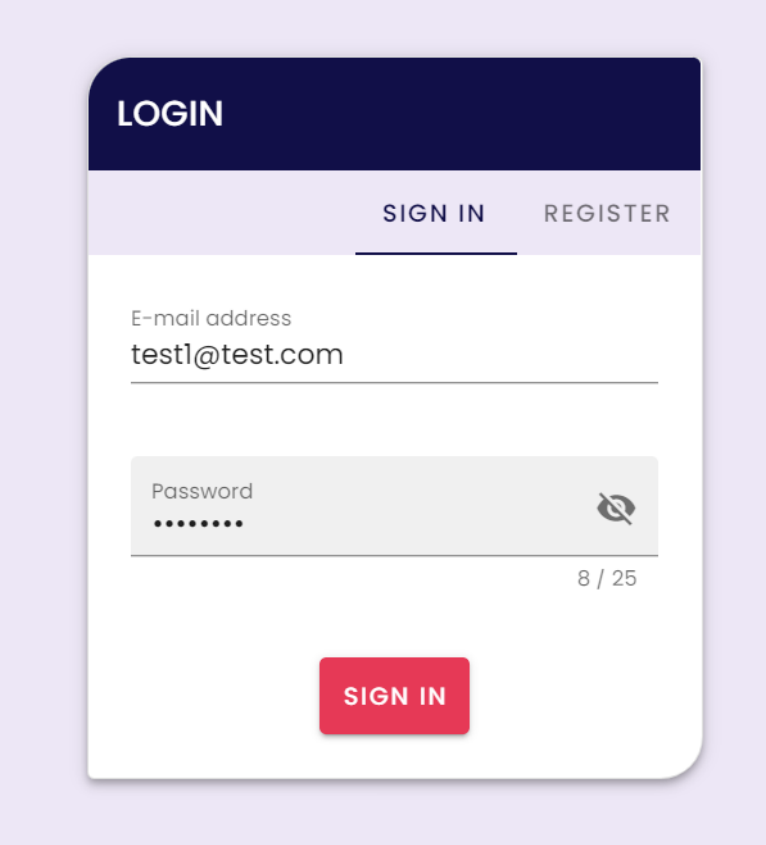
Ovime je postavljanje projekta završeno.

6.2. Prijava i registracija korisnika

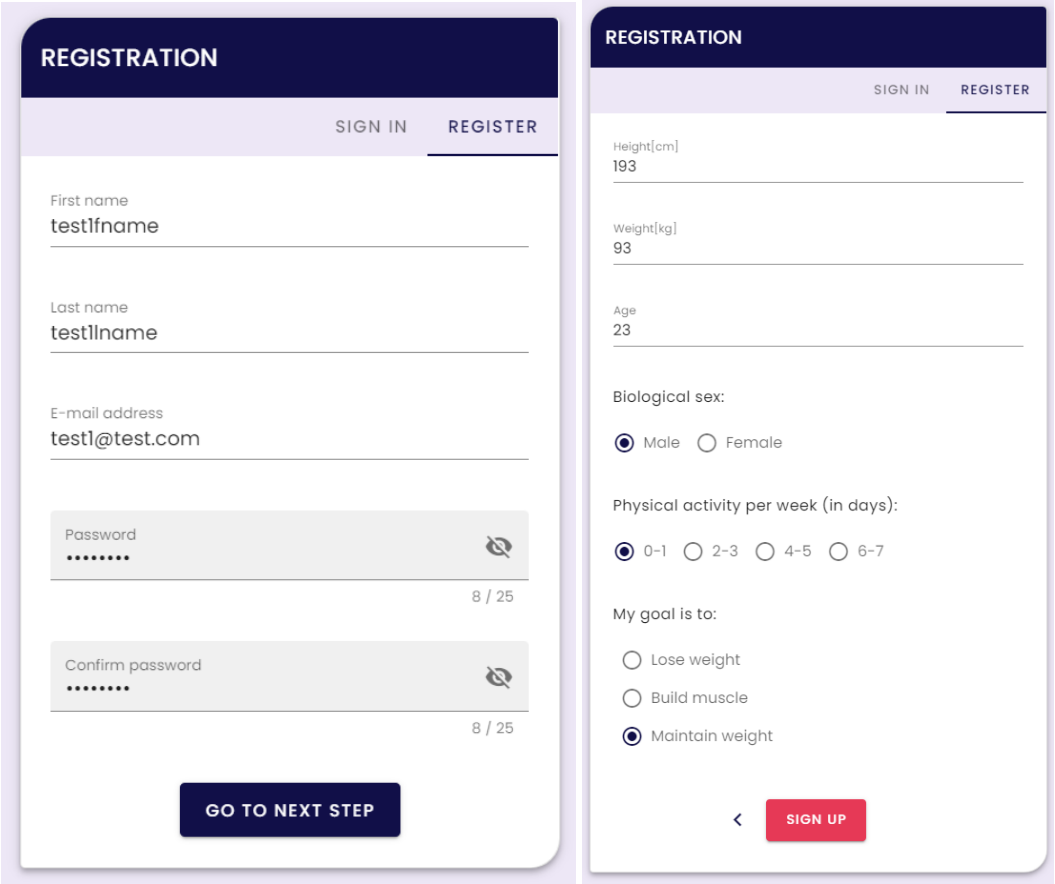
Kada korisnik prvi put pristupi aplikaciji, bit će preusmjeren na stranicu za prijavu i registraciju. Stvoren je *form* objekt koji sadrži dva svojstva, *login* objekt i *register* objekt. *Login* objekt sadrži sva polja koja će se pomoću Vue-ove v-model direktive vezati na HTML elemente za unos podataka za prijavu korisnika, a register objekt sadrži dva dodatna objekta: *personalInfo* i *bodyInfo*. Objekt *personalInfo* sadrži sva polja koja predstavljaju korisnički unos za prvi korak registracije, dok *bodyInfo* sadrži korisnički unos drugog koraka registracije. Slika 6.12 prikazuje cijeli *form* objekt. Slika 6.13 prikazuje formu za prijavu u aplikaciju, dok slika 6.14 prikazuje oba koraka registracije korisnika..

```
form: {
  login: {
    email: '',
    password: '',
  },
  register: {
    personalInfo: {
      firstName: '',
      lastName: '',
      email: '',
      password: '',
      confirmPassword: '',
    },
    bodyInfo: {
      height: null,
      weight: null,
      age: null,
      sex: enums.BIOLOGIAL_SEX.MALE,
      activity: enums.ACTIVITY_FACTOR.LIGHT,
      goal: enums.GOALS.MAINTAIN,
    },
  },
},
```

Slika 6.12 *Objekt korisničkog unosa za prijavu ili registraciju*



Slika 6.13 *Forma za prijavu u aplikaciju*



Slika 6.14 *Forma za registraciju*

Kada korisnik popuni podatke za prijavu i klikne na gumb *Sign in*, pozvat će se metoda *attemptLogin()*. Ova metoda poziva metodu Firebase Authentication usluge pod nazivom *signInWithEmailAndPassword()*. Ukoliko metoda Firebase Authentication usluge ne baci iznimku, prikazuje se Toast poruka o uspješnosti prijave i korisnika se preusmjerava na pregled njegovih obroka. Slika 6.15 prikazuje metodu *attemptLogin()*

```
async attemptLogin() {
  this.waiting.submitting = true
  try {
    const { email, password } = this.form.login

    await this.$fire.auth.signInWithEmailAndPassword(email, password)

    this.$showSuccessToast(strings.TOASTS.LOGIN.SUCCESS)
    this.$router.replace( location: '/')
  } catch (e) {
    console.log(e.message)
    this.$showErrorToast(strings.TOASTS.LOGIN.FAIL)
  }
  this.waiting.submitting = false
},
```

Slika 6.15 *Metoda za pokušaj prijave korisnika u sustav*

U slučaju registracije, jednom kada korisnik ispravno unese sve podatke za oba koraka registracije, pritisak na gumb *Sign up* će pozvati metodu *attemptRegister()*. Ova metoda će stvoriti objekt klase *User*, koja će odmah u konstruktoru obaviti kalkulacije indeksa tjelesne mase, stope bazalnog metabolizma, te preporučene raspodjele makronutijenata. Nakon toga će se pomoću metode Firebase Authentication usluge *createUserWithEmailAndPassword* stvoriti novi korisnik. Zatim će se u kolekciju *Users* zapisati novi objekt korisnika, a odmah će se u njegovoj podkolekciji *Body history* stvoriti i prvi zapis povijesti tjelesnih podataka korisnika.

```
calculateBMI() {
  this.calculations.BMI = +(
    this.weight / Math.pow( x: this.height / 100,  y: 2)
  ).toFixed( fractionDigits: 2)
}
```

Slika 6.16 *Metoda za računanje indeksa tjelesne mase korisnika*

```
calculateBMR() {
  const calculationsPerSex = {
    [enums.BIOLOGIAL_SEX.MALE]: () =>
      10 * this.weight + 6.25 * this.height - 5 * this.age + 5,
    [enums.BIOLOGIAL_SEX.FEMALE]: () =>
      10 * this.weight + 6.25 * this.height - 5 * this.age - 161,
  }
  this.calculations.BMR = calculationsPerSex[this.sex]()
  const factorsPerActivity = {
    [enums.ACTIVITY_FACTOR.LIGHT]: 1.2,
    [enums.ACTIVITY_FACTOR.MODERATE]: 1.375,
    [enums.ACTIVITY_FACTOR.FREQUENT]: 1.55,
    [enums.ACTIVITY_FACTOR.EXTREME]: 1.725,
  }
  this.calculations.BMR *= factorsPerActivity[this.activity]
  const actionsPerGoal = {
    [enums.GOALS.MAINTAIN]: () => {},
    [enums.GOALS.LOSE_WEIGHT]: () => (this.calculations.BMR -= 350),
    [enums.GOALS.BUILD_MUSCLE]: () => (this.calculations.BMR += 350),
  }
  actionsPerGoal[this.goal]()
}
```

Slika 6.17 *Metoda za računanje stope bazalnog metabolizma korisnika*

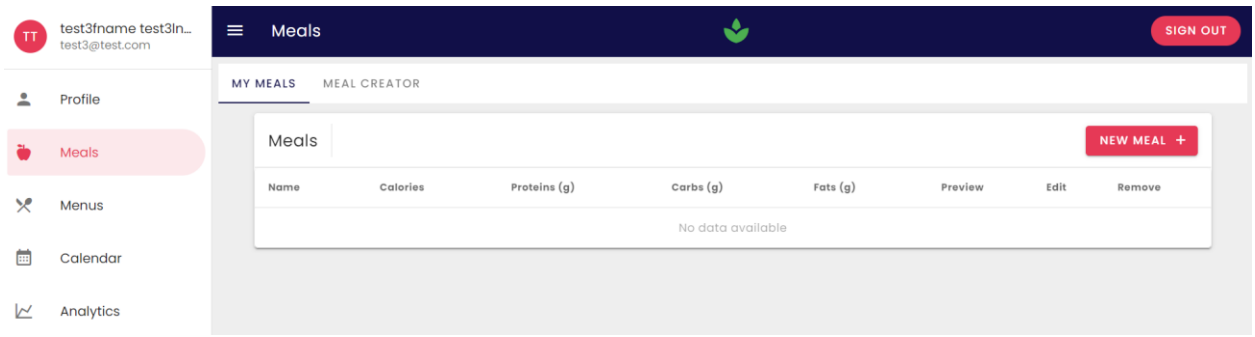
```
calculateNutrientDistribution() {
  const fatCalories = this.calculations.BMR * 0.21
  const fatGrams = Math.round( x: fatCalories / 9)

  const carbsCalories = ((this.calculations.BMR - fatCalories) / 6.3) * 4.5
  const carbsGrams = Math.round( x: carbsCalories / 4)

  const proteinGrams = Math.round(
    x: (this.calculations.BMR - fatCalories - carbsCalories) / 4
  )

  this.distribution = new NutrientDistribution(
    proteinGrams,
    carbsGrams,
    fatGrams
  )
}
```

Slika 6.18 *Metoda za računanje idealne raspodjele makronutrijenata unutar BMR-a korisnika*

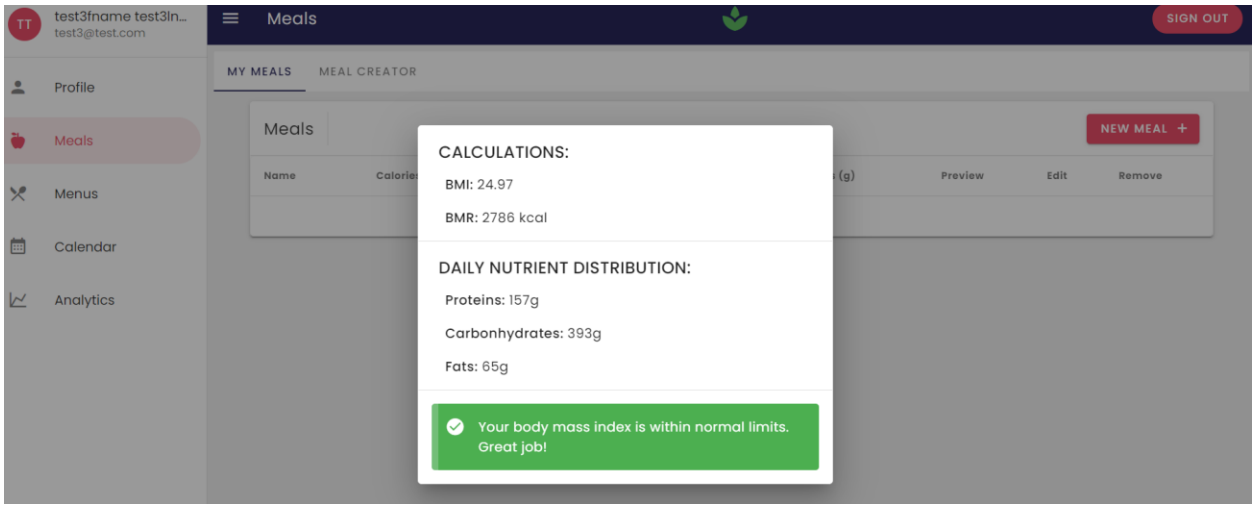
Nakon uspješne prijave ili registracije, korisnik će ostati prijavljen i biti će preusmjeren na pregled svojih obroka. Treba se prisjetiti da je metoda iz Vuex store-a *onAuthStateChangedAction* iz baze podataka povukla sve korisničke podatke i oni su spremni za korištenje u cijeloj aplikaciji.

6.3. Obroci

Jednom kada se korisnik prijavio, na navigacijskoj traci može vidjeti zeleni gumb u obliku lista. Kada klikne na njega, vidjet će izračune za svoje tijelo koji su za njega prvi put obavljeni u koraku registracije prilikom stvaranja objekta korisnika. Prva komponenta stranice koju korisnik vidi je stranica za izlistavanje svih korisnikovih obroka. S obzirom da još uvijek nije stvorio niti jedan obrok, on treba pritisnuti gumb *New meal*, koji će ga odvesti na susjednu komponentu stranice koja služi za stvaranje/uređivanje obroka. Slika 6.19. prikazuje stranicu za izlistavanje obroka u kojoj trenutno nema podataka i gumb za prikaz korisnikovih kalkukacija. Slika 6.20. prikazuje modal koji ispisuje korisnikove kalkulacije i poruku o njegovoj težini. Slika 6.21 prikazuje svojstvo za izračunavanje BMI poruke. Slika 6.22. prikazuje stranicu stvaranje i uređivanje obroka na koju će korisnik biti odveden klikom na *New meal.*



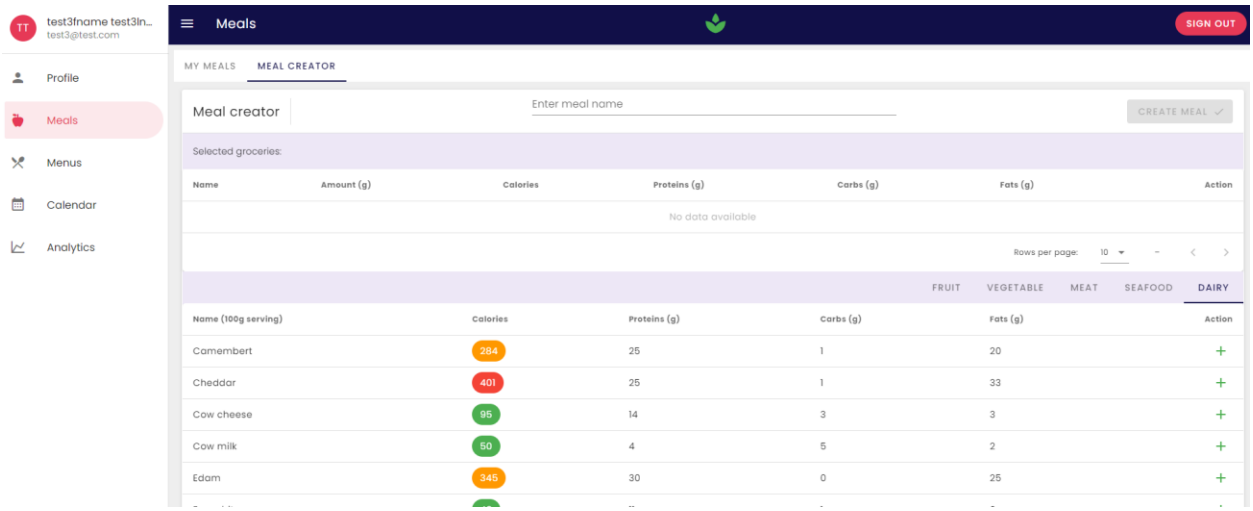
Slika 6.19 *Prazna lista korisnikovih obroka*



Slika 6.20 *Korisničke kalkulacije i BMI poruka korisnika*

```
$computedBMIAlert() {
  if (!this.$userLoggedIn) return { message: '', type: 'info' }
  const { BMI } = this.user.bodyInfo.calculations
  let message = ''
  let type = ''

  if (BMI < 18.5 && BMI > 16.5) {
    message =
      'Increase your daily calorie intake based on your BMR. You are mildly malnourished.'
    type = 'info'
  } else if (BMI <= 16.5 && BMI > 15) {
    message =
      'Increase your daily calorie intake based on your BMR. You are mildly malnourished.'
    type = 'warning'
  } else if (BMI <= 15) {
    message =
      'Your BMI is below all limits, you are severely malnourished! Urgently increase your daily calorie intake!'
    type = 'error'
  } else if (BMI > 25 && BMI < 30) {
    message =
      'You have an elevated body mass index. Reduce your daily calorie intake.'
    type = 'info'
  } else if (BMI >= 30 && BMI < 35) {
    message =
      'You are within the limits of normal obesity. Significantly reduce your daily calorie intake.'
    type = 'warning'
  } else if (BMI >= 35 && BMI < 40) {
    message =
      'You are within the limits of normal obesity. Significantly reduce your daily calorie intake.'
    type = 'error'
  } else if (BMI >= 40) {
    message =
      'Your body mass index is outside all limits and is too dangerous for your health!'
    type = 'error'
  } else {
    message = 'Your body mass index is within normal limits. Great job!'
    type = 'success'
  }
  return { message, type }
},
```

Slika 6.21 *Računanje BMI poruke*



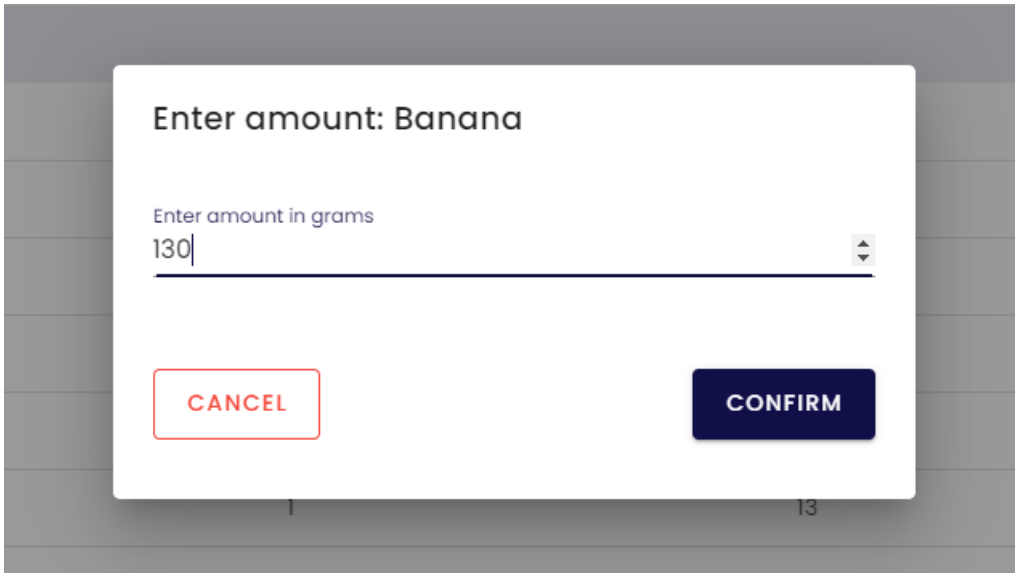
Slika 6.22 *Kreator/uređivač obroka*

Roditeljska komponenta stranica za izlistavanje obroka i za kreiranje/uređivanje obroka sadrži referencu *selectedMeal*, čija je inicijalna vrijednost null. Vrijednost reference mijenja stranica koja izlistava obroke kada korisnik odabere opciju za uređivanje obroka, a čita ju stranica za stvaranje/uređivanje obroka. Unutar stranice za kreiranje/uređivanje obroka, vrijednost varijable *selectedMeal* služi da bi se odredilo radi li se o kreiranju novog obroka, ili uređivanju postojećeg. Također postoji objekt *meal* koji sadržava uneseni naziv obroka, te polje *weightedGroceries* koje predstavlja polje svih namirnica s težinom koje su dodane u obrok. U *created()* metodi životnog ciklusa stranice za kreiranje/uređivanje obroka se provjerava je li vrijednost *selectedMeal* objekta postavljena, ako je onda se poziva metoda *loadMeal() koja će meal* objekt popuniti nazivom i svim namirnicama *iz selectedMeal* objekta. Svaki put kada korisnik napusti stranicu za kreiranje/uređivanje obroka roditeljska komponenta će referencu *selectedMeal* resetirati na *null* vrijednost.

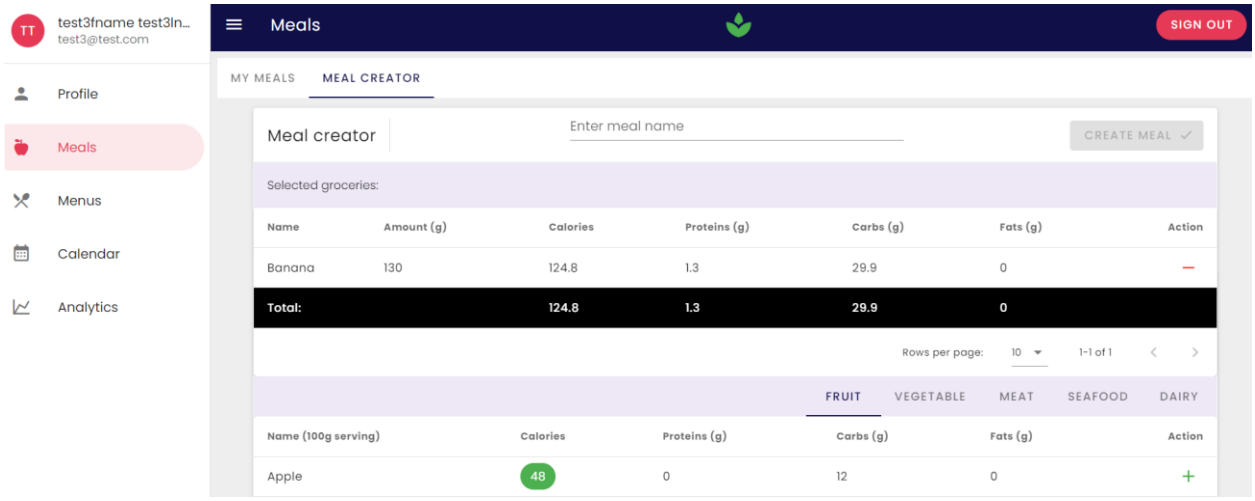
```
created() {
  this.selectedMeal && this.loadMeal()
},
methods: {
  loadMeal() {
    this.meal.name = this.selectedMeal.name
    this.$replaceItems(
      this.meal.weightedGroceries,
      this.selectedMeal.weightedGroceries
    )
  },
```

Slika 6.23 *Pokušaj učitavanja naziva obroka i svih njegovih namirnica prilikom stvaranja stranice za kreiranje/uređivanje obroka*

S obzirom da još uvijek nije stvoren niti jedan obrok, korisnik bi trebao stvoriti svoj prvi obrok. Klik na zeleni gumb s oznakom '+' pokraj jedne od namirnica će otvoriti prozorčić za unos količine željene namirnice. Kada korisnik odabere željenu veličinu, pritisak na *confirm* će pozvati metodu *confirmWeighing()*. Ova metoda će prvo provjeriti postoji li već u obroku namirnica čija se količina sada unijela. Ako postoji, stvori novu objekt klase *WeightedGrocery* gdje će količina biti zbroj prethodne i trenutno odabrane količine, a ako ne postoji, onda neće doći do nikakvog zbrajanja, već će se jednostavno stvoriti novi objekt klase WeightedGrocery. Zatim se nova namirnica s težinom nadodaje u polje *meal.weightedGroceries.* Slika 6.24 prikazuje prozorčić za unos količine namirnice, slika 6.25. prikazuje dodanu 'vaganu' namirnicu u objekt *meal,* a slika 6.26 prikazuje metodu *confirmWeighing().*



Slika 6.24 *Dodavanje nove namirnice u obrok*



Slika 6.25 *Objekt WeightedGrocery klase nadodan u obrok*

```
confirmWeighing() {
  this.waiting.submitting = true
  const index = this.meal.weightedGroceries.findIndex(
    (weightedGrocery) =>
      this.selectedGrocery.id === weightedGrocery.grocery.id
  )
  if (index >= 0) {
    const existingGrocery = this.meal.weightedGroceries[index]
    this.meal.weightedGroceries.splice(
      index,
      deleteCount: 1,
      new WeightedGrocery(
        grocery: { ...this.selectedGrocery },
        amountGrams: this.form.amount + existingGrocery.amount
      )
    )
  } else {
    this.meal.weightedGroceries.push(
      new WeightedGrocery( grocery: { ...this.selectedGrocery }, this.form.amount)
    )
  }
  this.closeWeighingDialog()
  this.waiting.submitting = false
},
```

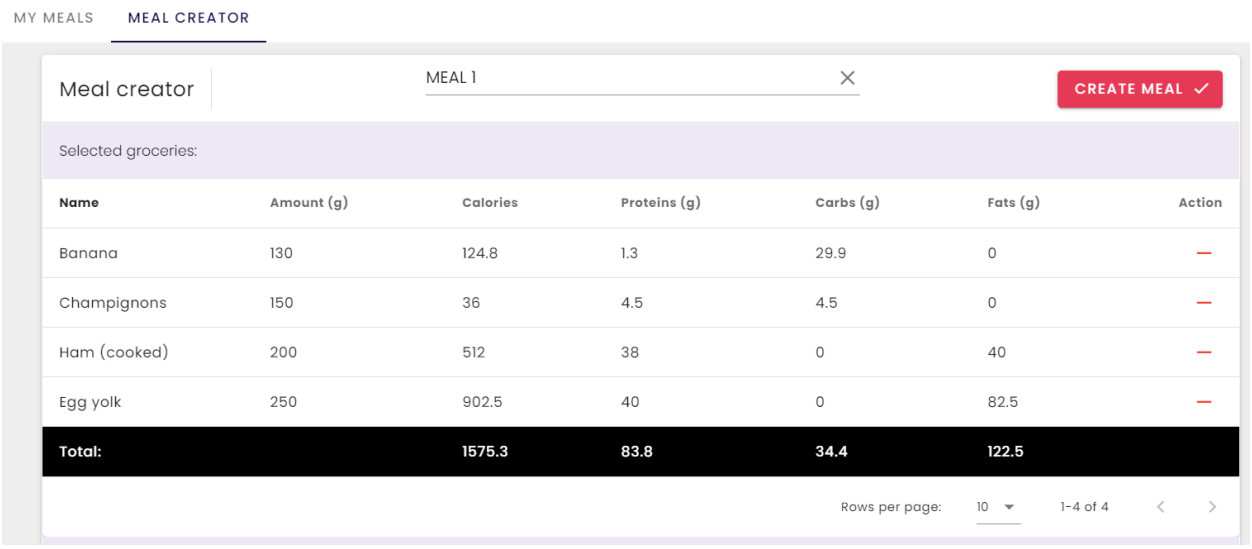
Slika 6.26 *Metoda confirmWeighing() za dodavanje namirnice u obrok*

Na dnu tablice dodanih namirnica u obrok se nalaze ukupne vrijednosti kalorija i makronutrijenata obroka. Za to je korišteno Vue računsko svojstvo (engl. *computed property*) pod nazivom *sums.* Slika 6.27 prikazuje spomenuto svojstvo.

```
computed: {
  sums() {
    return this.meal.weightedGroceries.reduce(function (
      accumulator,
      weightedGrocery
    ) {
      // loop over each item in the array
      accumulator.proteins = +(
        (accumulator.proteins || 0) + weightedGrocery.nutrients.proteins
      ).toFixed(2)
      accumulator.carbonhydrates = +(
        (accumulator.carbonhydrates || 0) +
        weightedGrocery.nutrients.carbonhydrates
      ).toFixed(2)
      accumulator.fats = +(
        (accumulator.fats || 0) + weightedGrocery.nutrients.fats
      ).toFixed(2)
      accumulator.calories = +(
        (accumulator.calories || 0) + weightedGrocery.nutrients.calories
      ).toFixed(2)

      return accumulator
    },
    {})
  },
```

Slika 6.27 *Računsko svojstvo sums*

Nakon dodavanja namirnice u obrok, korisnik naravno ima mogućnost i uklanjati ju iz obroka. Sada će biti dodano više namirnica u obrok i odabrati će se ime obroka.



Slika 6.28 *Računsko svojstvo sums*

S obzirom da obrok nije prazan, te da je odabrano ime koje odgovara definiranim pravilima za imenovanje obroka, korisnik ima mogućnost kliknuti na gumb *Create Meal*. Sam tekst gumba govori da se ne radi o uređivanju obroka već stvaranju novoga. Klik na taj gumb će pozvati metodu *saveMeal(). Ova metoda se grana na 2 slučaja:*

- Ako *selectedMeal* sadrži lažnu vrijednost, radi se o kreiranju obroka. U kolekciji *Users* pod dokumentom trenutnog korisnika se ulazi u podkolekciju *Meals*, te se tamo stvara novi dokument koji će predstavljati novi obrok.
- U suprotnom se radi o uređivanju obroka. Potrebno je dohvatiti id odabranog obroka koji se uređuje, te u podkolekciji *Meals* urediti dokument čiji id odgovara id-ju odabranog obroka.

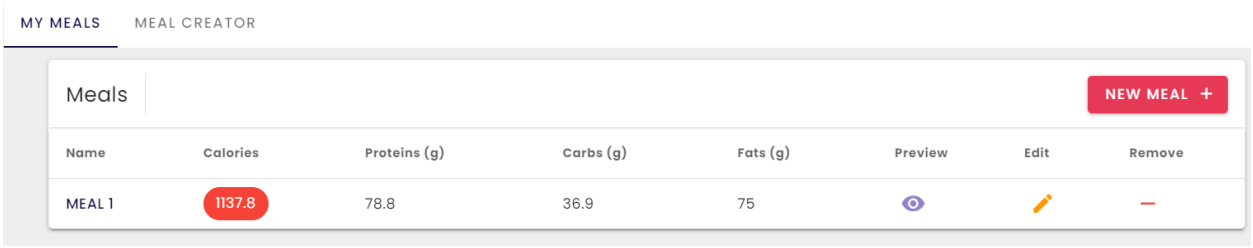
Zatim se preko Vuex akcije osvježavaju svi korisnikovi obroci da bi korisnik vidio svoje najnovije podatke. Nakon toga se korisnika prebacuje na stranicu koja izlistava sve njegove obroke, što će rezultirati resetiranjem reference *selectedMeal* na *null*.

Slika 6.29 prikazuje metodu za pohranu novostvorenog ili uređenog obroka

```
// Creates or edits an item depending if selected item is null or not
async saveMeal() {
  this.waiting.submitting = true
  try {
    if (!this.selectedMeal) {
      // CREATION MODE
      const docRef = await this.$fire.firestore
        .collection(enums.COLLECTIONS.USERS)
        .doc(this.$fire.auth.currentUser.uid)
        .collection(enums.COLLECTIONS.MEALS)
        .doc()
      const meal = new Meal(docRef.id, this.meal.name, weightedGroceries: [
        ...this.meal.weightedGroceries,
      ])
      await docRef.withConverter(MealConverter).set(meal)
      this.$showSuccessToast(strings.TOASTS.MEAL.CREATION.SUCCESS)
    } else {
      // EDIT MODE
      const { id } = this.selectedMeal
      const meal = new Meal(id, this.meal.name, weightedGroceries: [
        ...this.meal.weightedGroceries,
      ])
      await this.$fire.firestore
        .collection(enums.COLLECTIONS.USERS)
        .doc(this.$fire.auth.currentUser.uid)
        .collection(enums.COLLECTIONS.MEALS)
        .doc(id)
        .withConverter(MealConverter)
        .set(meal)
      this.$showSuccessToast(strings.TOASTS.MEAL.EDIT.SUCCESS)
    }
    await this.fetchMeals()
    this.$router.push('/meals')
  } catch (e) {
    console.log(e)
    this.$showErrorToast(strings.TOASTS.MEAL.CREATION.FAIL)
  }
```

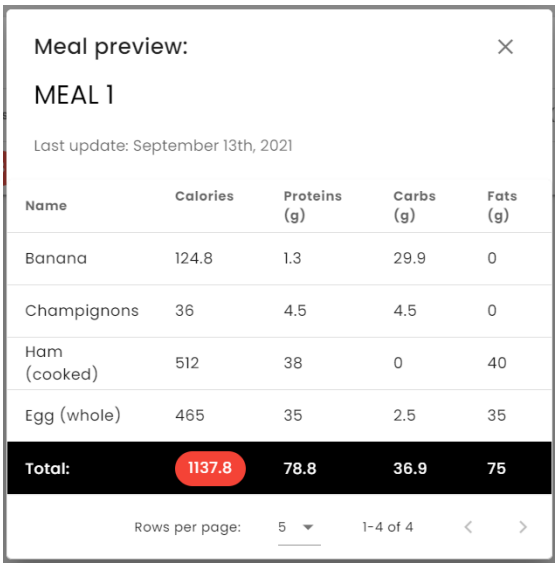
Slika 6.29 *Stvaranje/uređivanje obroka*

Zatim korisnik na listi svojih obroka može vidjeti svoj novi obrok. Crvena boja kod ukupnog broja kalorija označava da ovaj obrok sadrži previše kalorija, uz pretpostavku da će se dnevni jelovnika korisnika sastojati od barem 3 obroka.



Slika 6.30 *Stvaranje/uređivanje obroka*

Korisnik može otvoriti pregled obroka, poslati obrok na uređivanje što će uzrokovati postavljanje *selectedItem* reference na njega, ili ga obrisati iz baze podataka. Slika 6.31 prikazuje prozorčić za pregled obroka, a slika 6.32 prikazuje programski kod za brisanje obroka iz baze podataka. Treba pripaziti da brisanje obroka mora rezultirati njegovim uklanjanjem iz svih jelovnika u koje je dodan, te brisanje jelovnika i dnevnih jelovnika u kojima je on jedini obrok. Za to se brine metoda *cleanUpMealInDatabase()*.

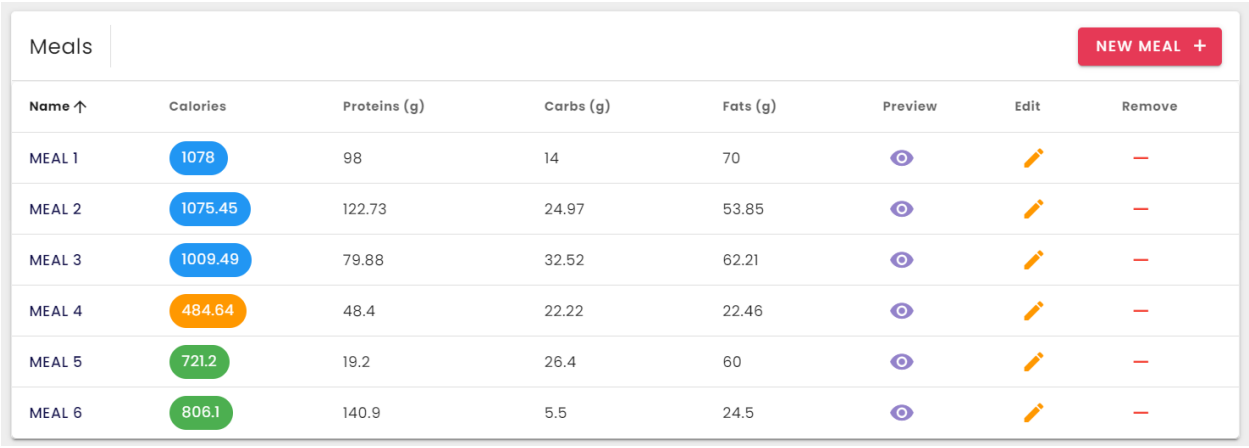


Slika 6.31 *Pregled obroka*

```
async removeItemConfirm() {
  if (!this.selectedItem.toRemove) return
  this.waiting.submitting = true
  try {
    await this.cleanUpMealInDatabase(this.selectedItem.toRemove)

    await this.$fire.firestore
      .collection(enums.COLLECTIONS.USERS)
      .doc(this.$fire.auth.currentUser.uid)
      .collection(enums.COLLECTIONS.MEALS)
      .doc(this.selectedItem.toRemove.id)
      .delete()

    this.$showInfoToast(strings.TOASTS.MEAL.DELETION.SUCCESS)
  } catch (e) {
    console.log(e.message)
    this.$showErrorToast(strings.TOASTS.MEAL.DELETION.FAIL)
  }
  this.closeRemoveDialog()
  this.waiting.submitting = false
  await this.fetchMeals()
},
```

Slika 6.32 *Uklanjanje obroka*

Prije prelaska na stvaranje jelovnika je stvoreno više obroka. Slika 6.33 prikazuje 6 stvorenih obroka.



Slika 6.33 *Obroci koji će se vezati u jelovnike*

Boje kalorija unutar aplikacije predstavljaju sljedeće:

- Narančasta – broj kalorija je prenizak u odnosu na BMR korisnika.
- Plava – broj kalorija prihvatljiv, malo niži ili malo viši od očekivanog
- Zelena – broj kalorija je gotovo idealan i odstupanja su mala.
- Crvena – broj kalorija je previsok u odnosu na BMR korisnika.

Kod jelovnika se u obzir uzima BMR, a kod obroka se pretpostavlja da će korisnik jesti barem tri obroka dnevno, stoga se uzima vrijednost BMR/3.

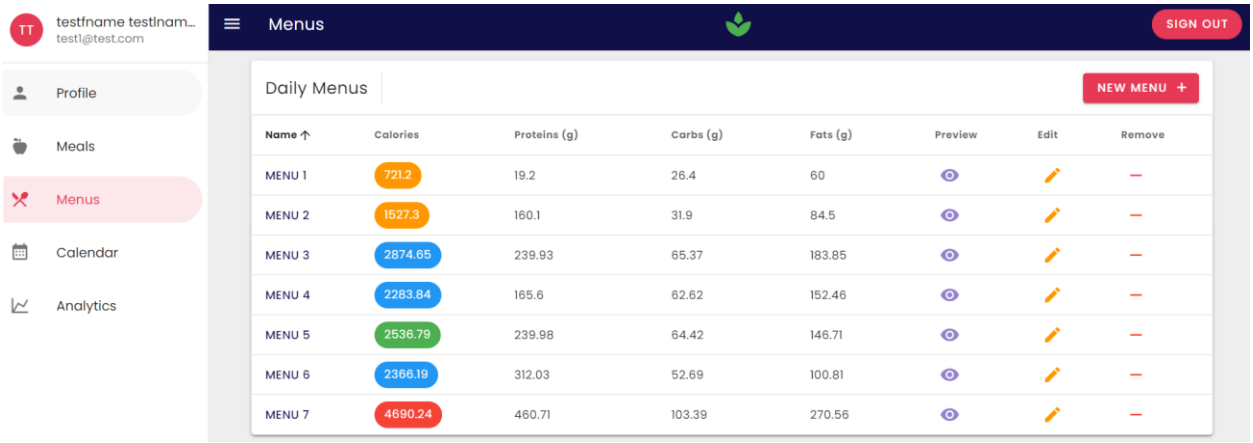
6.4. Jelovnici

Upravljanje jelovnicima je odrađeno unutar jedne stranice, za razliku od upravljanja obrocima. Klik ga gumb *New menu* će otvoriti prozorčić za stvaranje novog jelovnika ukoliko vrijednost reference selectedItem lažna, a u suprotnom će selectedItem biti jedan od jelovnika koji se trenutno uređuje što znači da je trenutno otvoren mod za uređivanje.

U podacima komponente stranice upravljanja jelovnicima postoji objekt *menu*, koji će u sebi sadržavati naziv i polje id-jeva svih obroka koji su dodanu u jelovnik. Obroci se u jelovnik dodaju putem <select/> HTML elementa, gdje svaka opcija sadržava id jednog obroka.

Za potrebe pojašnjavanja jelovnika je unaprijed stvoreno nekoliko jelovnika. Slika 6.34 prikazuje neke stvorene jelovnike. S obzirom da je u poglavlju 5 prikazano da će svaki jelovnik sadržavati polje id-jeva obroka, treba naći način da se prikažu konkretni podaci svih obroka. Zbog toga je

stvoreno računsko svojstvo *$extendedMenus* koje predstavlja polje svih jelovnika, gdje će se u svaki jelovnik mapirati cijeli objekti obroka koji mu pripadaju umjesto da se samo koristi njihov id. Ovo će biti jako korisno za generiranje poruka pohvale ili upozorenja korisnicima, kao i prikazivanja konkrentih kalkulacija njihovih jelovnika. Slike 6.35. prikazuje *$extendedMenus* polje.

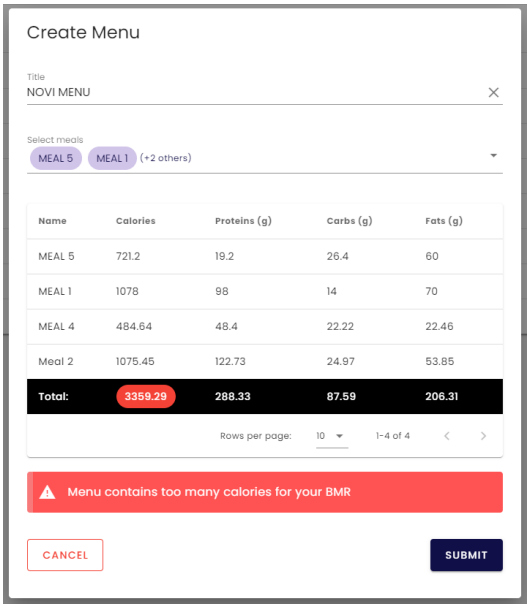


Slika 6.34 *Izgled popisa jelovnika*

```
$extendedMenus() {
  if (!this.menus || !this.meals) return []
  return this.menus.map((menu) => {
    const { id, name, createdAt, meals } = menu
    const menuMeals = this.meals.filter((meal) => meals.includes(meal.id))
    const menuTotals = this.$genTotalsForMenu(menuMeals)
    return {
      id,
      name,
      createdAt,
      meals: [...menuMeals],
      nutrients: {
        calories: menuTotals.calories,
        proteins: menuTotals.proteins,
        carbonhydrates: menuTotals.carbonhydrates,
        fats: menuTotals.fats,
      },
    }
  })
},
```

Slika 6.35 *Računsko svojstvo polja jelovnika u koje su mapirani konkretni objekti obroka koji im pripadaju*

Slika 6.36. prikazuje prozorčić za stvaranje novog jelovnika. Treba obratiti pažnju da se prilikom svakog dodavanja/uklanjanja obroka u/iz jelovnika ponovno računa poruka pohvale ili upozorenja o broju kalorija koju jelovnik sada sadrži.



Slika 6.36 *Izgled prozorčića za stvaranje novog jelovnika*

Slika 6.37 prikazuje računsko svojstvo za izračunavanje poruke pohvale/upozorenja korisnika koja mu se prikazuje na dnu prozorčića za stvaranje/uređivanje obroka.

```
$computedBMRAlert() {
  if (
    !this.$userLoggedIn ||
    !this.selectedItemSums ||
    !this.selectedItemSums.calories
  )
    return null
  let type = ''
  let message = ''
  const { calories } = this.selectedItemSums
  const { BMR } = this.user.bodyInfo.calculations
  if (calories > BMR + 300) {
    type = 'error'
    message = 'Menu contains too many calories for your BMR'
  } else if (calories < BMR - 450) {
    type = 'warning'
    message = 'Menu contains too few calories for your BMR'
  } else if (calories > BMR + 100 || calories < BMR - 250) {
    type = 'info'
    message = 'Menu is somewhat respecting your BMR'
  } else {
    type = 'success'
    message = 'Menu respects for your BMR'
  }

  return { type, message }
},
```

Slika 6.37 *Generiranje poruke prilikom stvaranja/uređivanja/pregleda jelovnika*

Prilikom stvaranja jelovnika opet postoji dva načina rada:

- Stvaranje novog jelovnika u bazi podataka ukoliko je *selectedItem* lažna vrijednost
- Uređivanje jelovnika ukoliko *selectedItem* nije lažna vrijednost

Klik na crveni gumb s oznakom '-' i ovdje otvara prozorčić za potvrdu brisanja. A u slučaju potvrde je potrebno prvo obrisati sve kalendarske stavke vezane za jelovnik koji se briše, a zatim obrisati jelovnik. Slika 6.38 prikazuje logiku brisanja jelovnika.

```
getCalendarItemIdsForMenu(menu) {
  return this.calendarItems
    .filter((calendarItem) => calendarItem.menuId === menu.id)
    .map((item) => item.id)
},

async removeItemConfirm() {
  if (!this.selectedItem) return
  this.waiting.submitting = true
  try {
    const calendarIdsToRemove = this.getCalendarItemIdsForMenu(
      this.selectedItem
    )
    if (calendarIdsToRemove.length) {
      const promises = []
      calendarIdsToRemove.forEach((calendarId) => {
        promises.push(
          this.$fire.firestore
            .collection(enums.COLLECTIONS.USERS)
            .doc(this.$fire.auth.currentUser.uid)
            .collection(enums.COLLECTIONS.CALENDAR)
            .doc(calendarId)
            .delete()
        )
      })
      await Promise.all(promises)
      await this.fetchCalendarItems()
    }
    await this.$fire.firestore
      .collection(enums.COLLECTIONS.USERS)
      .doc(this.$fire.auth.currentUser.uid)
      .collection(enums.COLLECTIONS.MENUS)
      .doc(this.selectedItem.id)
      .delete()
    this.$showInfoToast(strings.TOASTS.MENU.DELETION.SUCCESS)
    await this.fetchMenus()
  } catch (e) {
    console.log(e.message)
    this.$showErrorToast(strings.TOASTS.MENU.DELETION.FAIL)
  }
  this.closeRemoveDialog()
  this.waiting.submitting = false
},
```

Slika 6.38 *Uklanjanje jelovnika i pripadajućih kalendarskih stavki iz baze podataka*

6.5. Kalendarske stavke ili dnevni jelovnici

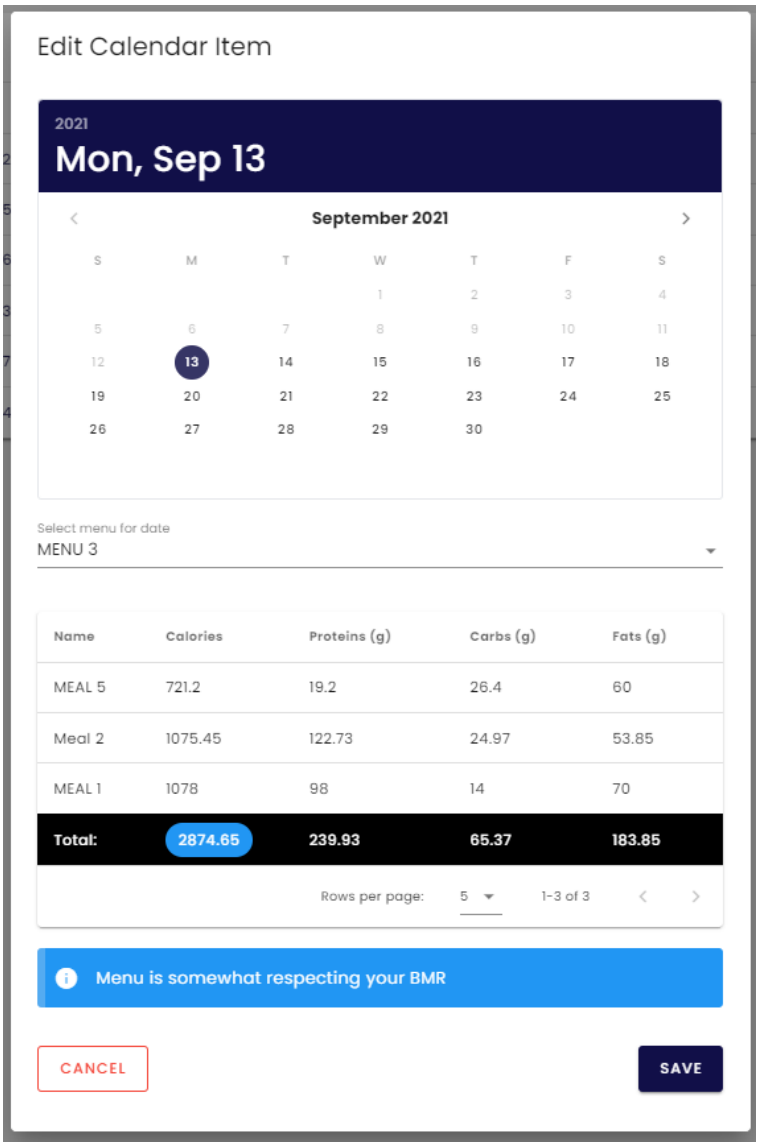
Nakon što su jelovnici stvoreni, potrebno je stvoriti plan konzumiranja tih jelovnika, odnosno korisnik treba datumima u budućnosti dodijeliti jelovnike koje planira konzumirati. Na taj način će mu se generirati i analitika koja će prikazivati kojim danom je unosio koliko kalorija.

Slika 6.39 *Stvorene kalendarske stavke*

Klik na *New calendar item* će otvoriti prozorčić koji korisniku omogućuje odabir datuma, te odabir stvorenog jelovnika. Isto tako, klik na simbol olovkice će poslati kalendarsku stavku na uređivanje. U podacima stranice je stvoren form objekt, koji će sadržavati datum u obliku broja milisekundi proteklih od 01.01.1970, te id jelovnika koji je pridružen datumu. Treba pripaziti da korisnik ne smije jednom datumu dodijeliti više od jednog jelovnika. Stoga će *form.date* objekt dodatno uvijek biti ponoć odabranog dana. Tu je i *allowedDates* računsko svojstvo, koje vraća istinu za svaki datum koji se smije koristiti, te laž za svaki datum koji se ne smije koristiti. Također, korisniku nije omogućeno stvarati kalendarske stavke u prošlosti, odnosno odabirati datume prije današnjeg dana. Slika 6.40 prikazuje programski kod računskog svojstva *allowedDates,* slika 6.41 prozorčić za stvaranje/uređivanje kalendarske stavke.

```
allowedDates(v) {
  const selectedItemDate = this.form.date
    ? new Date(this.form.date).getTime()
    : null
  const dateForValue = new Date(v).getTime()
  return (
    !this.calendarItems.find((item) => {
      return item.date === dateForValue
    }) || selectedItemDate === dateForValue
  )
},
```

Slika 6.40 *Računsko allowedDates*



Slika 6.41 *Uređivanje/stvaranje kalendarske stavke*

Zatim preostaje prikazati programski kod za stvaranje nove kalendarske stavke u bazi podataka, te uređivanje postojeće stavke. Korisnik naravno može i obrisati kalendarsku stavku klikom na gumb s oznakom '-'. Slika 6.42 prikazuje programski kod za brisanje kalendarske stavke. Slika 6.43 prikazuje programski kod za stvaranja/uređivanje kalendarske stavke.

```
async removeItemConfirm() {
  if (!this.selectedItem) return
  this.waiting.submitting = true
  try {
    await this.$fire.firestore
      .collection(enums.COLLECTIONS.USERS)
      .doc(this.$fire.auth.currentUser.uid)
      .collection(enums.COLLECTIONS.CALENDAR)
      .doc(this.selectedItem.id)
      .delete()

    this.$showInfoToast(strings.TOASTS.CALENDAR_ITEM.DELETION.SUCCESS)
    await this.fetchCalendarItems()
  } catch (e) {
    console.log(e.message)
    this.$showErrorToast(strings.TOASTS.CALENDAR_ITEM.DELETION.FAIL)
  }
  this.closeRemoveDialog()
  this.waiting.submitting = false
},
```

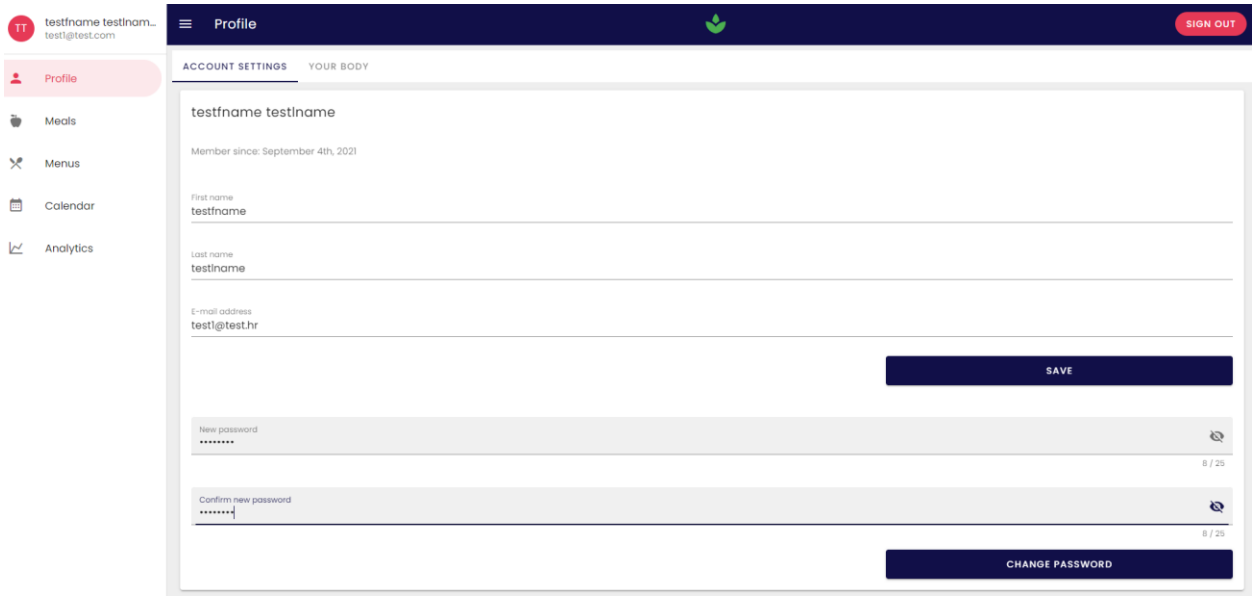
Slika 6.42 *Uklanjanje kalendarske stavke*

```
async submit() {
  this.waiting.submitting = true
  try {
    if (!this.selectedItem) {
      // CREATION MODE
      const dateMs = new Date(this.form.date).getTime()
      const { id: menuId } = this.form.menu
      const docRef = await this.$fire.firestore
        .collection(enums.COLLECTIONS.USERS)
        .doc(this.$fire.auth.currentUser.uid)
        .collection(enums.COLLECTIONS.CALENDAR)
        .doc()

      const newCalendarItem = new CalendarItem(docRef.id, dateMs, menuId)
      await docRef
        .withConverter(CalendarItemsConverter)
        .set(newCalendarItem)

      this.$showSuccessToast(strings.TOASTS.CALENDAR_ITEM.CREATION.SUCCESS)
    } else {
      // EDIT MODE
      const { id } = this.selectedItem
      const { id: menuId } = this.form.menu
      const dateMs = new Date(this.form.date).getTime()
      const calendarItem = new CalendarItem(id, dateMs, menuId)
      await this.$fire.firestore
        .collection(enums.COLLECTIONS.USERS)
        .doc(this.$fire.auth.currentUser.uid)
        .collection(enums.COLLECTIONS.CALENDAR)
        .doc(id)
        .withConverter(CalendarItemsConverter)
        .set(calendarItem)

      this.$showSuccessToast(strings.TOASTS.CALENDAR_ITEM.EDIT.SUCCESS)
    }
    await this.fetchCalendarItems()
  } catch (e) {
    console.log(e)
    this.$showErrorToast(strings.TOASTS.CALENDAR_ITEM.CREATION.FAIL)
  }
```

Slika 6.43 *Stvaranje/uređivanje kalendarske stavke*

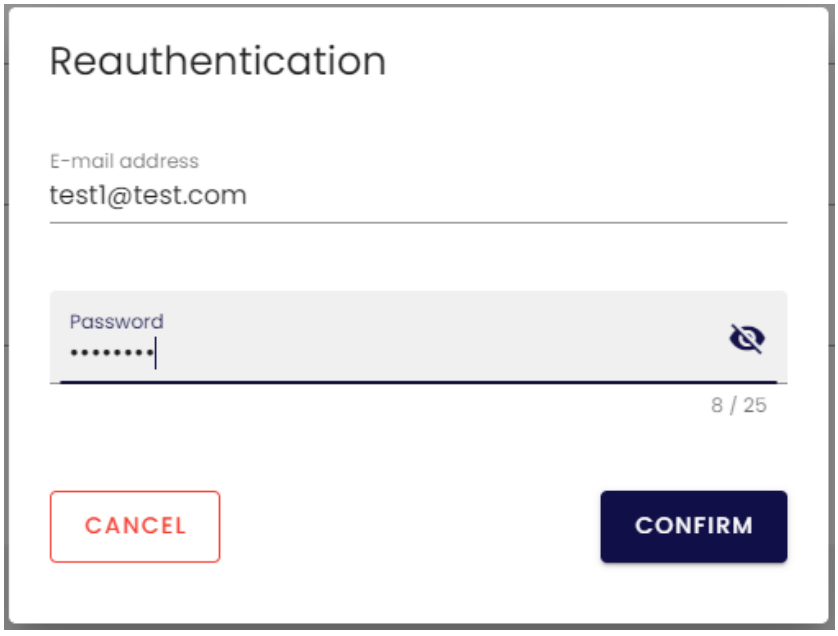
6.6. Korisnički profil i povijest tjelesnih podataka

Korisnik na svojem profilu može uređivati svoje osobne podatke i podatke za prijavu. Slika 6.44 prikazuje stranicu za izmjenu osobnih podataka i lozinke.



Slika 6.44 *Korisnički profil*

Kada korisnik želi pohraniti izmjene, od njega se zahtjeva ponovna autentifikacija kako bi Firebase Authentication usluga prihvatila promijenu podataka za prijavu. Kada korisnik odabere pohranu osobnih podataka ili lozinke, otvara mu se prozorčić za ponovnu autentifikaciju. Slika 6.45 prikazuje prozorčić za ponovnu autentifikaciju.



Slika 6.45 *Prozorčić za ponovnu autentifikaciju*

Klik na *Confirm* će pozvati metodu *reauthenticateAndProceed()* koja će ovisno o gumbu koji je pritisnut *(Save ili Change password)* ili ažurirati korisnikov email za prijavu, te ažurirati dokument korisnika, ili ažurirati lozinku za prijavu. Slika 6.46 prikazuje metodu *reauthenticateAndProceed()*

```
async reauthenticateAndProceed() {
  this.waiting.submitting = true
  const { email, password } = this.reauthenticationForm
  try {
    const credential =
      await this.$fireModule.auth.EmailAuthProvider.credential(
        email, password)
    await this.$fire.auth.currentUser.reauthenticateWithCredential(
      credential)
    if (this.reauthenticationPurpose === enums.REAUTHENTICATION_PURPOSE.EMAIL) {
      await this.submitPersonalData()
    } else {
      await this.submitNewPassword()
    }
  } catch (e) {
    this.$toast.error(strings.TOASTS.USER.UPDATE.ACCOUNT.FAIL)
    console.log(e.message)
  }
  this.closeReathenticationDialog()
  this.waiting.submitting = false
},
```

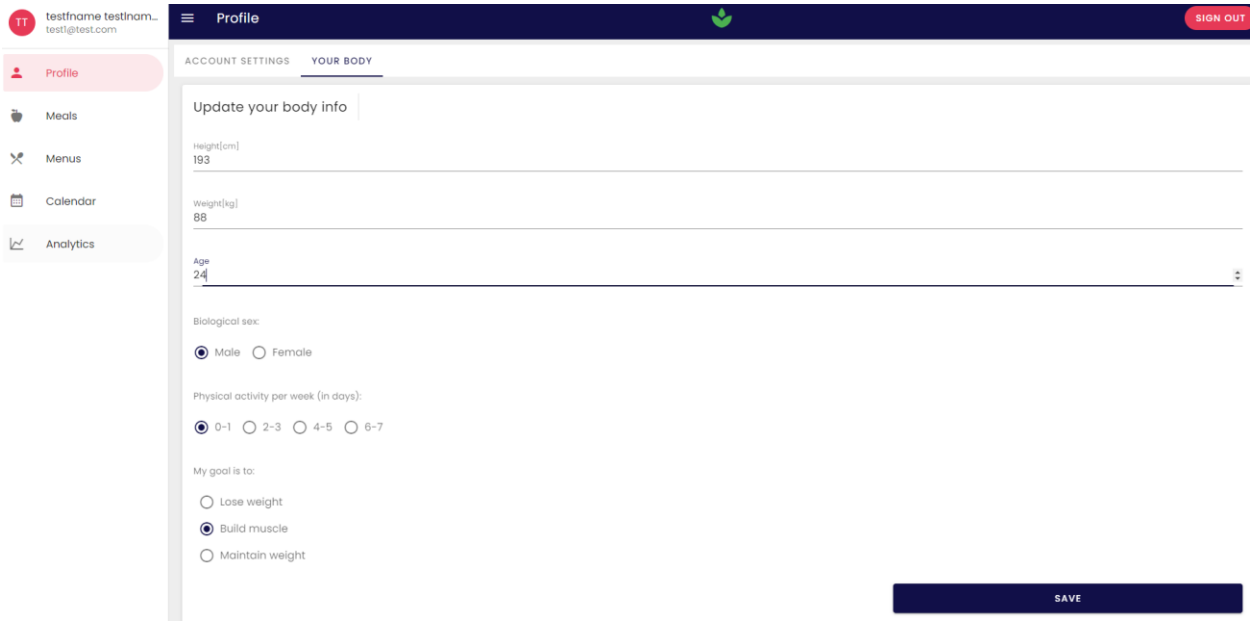
Slika 6.46 *Metoda za ponovnu autentifikaciju i pohranu promjena korisničkog profila*

```
async submitPersonalData() {
  if (!this.$userLoggedIn) return
  this.loading.personalData = true
  try {
    await this.$fire.auth.currentUser.updateEmail(
      this.personalInfoForm.email
    )

    const { id } = this.user
    const updatedUser = this.genUserObject()

    await this.$fire.firestore
      .collection(enums.COLLECTIONS.USERS)
      .doc(id)
      .withConverter(UserConverter)
      .set(updatedUser)

    this.$showSuccessToast(strings.TOASTS.USER.UPDATE.ACCOUNT.SUCCESS)
    await this.$fire.auth.signOut()
  } catch (e) {
    throw new Error(e.message)
  }
  this.loading.personalData = false
  this.resetPersonalDataValidation()
},
```

```
async submitNewPassword() {
  if (!this.$userLoggedIn) return

  this.loading.newPassword = true
  try {
    await this.$fire.auth.currentUser.updatePassword(
      this.newPasswordForm.password
    )
    this.$showSuccessToast(strings.TOASTS.USER.UPDATE.ACCOUNT.SUCCESS)
    await this.$fire.auth.signOut()
  } catch (e) {
    throw new Error(e.message)
  }
  this.loading.newPassword = false
  this.resetPasswordFormValidation()
},
```

Slika 6.47 *Metode za pohranu osobnih podataka i izmjenu lozinke*

Klik na *Your Body* prozorčić korisnika vodi na istu formu kakvu je popunio na drugom koraku registracije. Ovdje se od njega očekuje da unese nove podatke o svojem tijelu i svojoj fizičkoj aktivnosti. Slika 6.48. prikazuje stranicu za ažuriranje tjelesnih podataka.



Slika 6.48 *Stranica za ažuriranje tjelesnih podataka*

Kada korisnik pohrani svoje najnovije tjelesne podatke stvara se novi objekt *BodyInfo* klase i izvode se nove korisničke kalkulacije, a u *Body history* podkolekciju njegova dokumenta u bazi podataka se upisuje dokument čiji će id biti ponoć dana kada je izrađena izmjena, izražena u milisekundama proteklim od 01.01.1970.. Na ovaj način će korisnik moći u bazu spremiti samo jedan dokument dnevno, a ako više puta u jednom danu pohrani nove podatke, u bazi će se nalaziti samo najnoviji objekt *BodyInfo* stvoren tog dana. Slika 6.49 prikazuje metodu za pohranu najnovijih tjelesnih podataka korisnika.

```
async saveBodyInfo() {
  if (!this.$userLoggedIn) return
  this.waiting.submitting = true
  try {
    const { id, personalInfo, createdAt } = this.user
    const { height, weight, age, sex, activity, goal } = this.form

    const midnightDate = new Date()
    midnightDate.setHours( hours: 0,  min: 0,  sec: 0,  ms: 0)
    // Only save the newest body info for one day
    const bodyInfoDocId = midnightDate.getTime().toString()
    const newBodyInfo = new BodyInfo(
      height,
      weight,
      age,
      sex,
      activity,
      goal,
      midnightDate.getTime()
    )
    const updatedUser = new User(
      id,
      personalInfo: { ...personalInfo },
      newBodyInfo,
      createdAt
    )
    await this.$fire.firestore
      .collection(enums.COLLECTIONS.USERS)
      .doc(id)
      .collection(enums.COLLECTIONS.BODY_HISTORY)
      .doc(bodyInfoDocId)
      .withConverter(BodyInfoConverter)
      .set(newBodyInfo)

    await this.$fire.firestore
      .collection(enums.COLLECTIONS.USERS)
      .doc(id)
      .withConverter(UserConverter)
      .set(updatedUser)
    await this.fetchUser()
    await this.fetchBodyHistory()
    this.$showSuccessToast(strings.TOASTS.USER.UPDATE.BODY.SUCCESS)
  } catch (e) {
    console.log(e.message)
    this.$showErrorToast(strings.TOASTS.USER.UPDATE.BODY.FAIL)
  }
  this.waiting.submitting = false
  this.resetFormValidation()
},
```

Slika 6.49 *Ažuriranje tjelesnih podataka i stvaranje novog dokumenta unutar Body history kolekcije korisnika*

6.7. Analitika

S obzirom na to da svaka kalendarska stavka sadržava datum i id pripadajućeg jelovnika, stvoreno je $extendedCalendarItems polje koje će umjesto id-ja jelovnika sadržavati apsolutno sve njegove podatke i ukupne kalkulacije. Slika 6.50 prikazuje ovo računsko svojstvo.

```
$extendedCalendarItems() {
  if (!this.menus || !this.menus.length || !this.meals) return []
  return this.calendarItems.map((calendarItem) => {
    const { id, date, menuId } = calendarItem
    const menu = this.menus.find((menu) => menu.id === menuId)
    const meals = menu ? menu.meals : []

    const menuMeals = this.meals.filter((meal) => meals.includes(meal.id))
    const menuTotals = this.$genTotalsForMenu(menuMeals)
    return {
      id,
      date,
      menu: {
        id: menu.id,
        name: menu.name,
        createdAt: menu.createdAt,
        meals: [...menuMeals],
        nutrients: {
          calories: menuTotals.calories,
          proteins: menuTotals.proteins,
          carbonhydrates: menuTotals.carbonhydrates,
          fats: menuTotals.fats,
        },
      },
    }
  })
},
```

Slika 6.50 *$extendedCalendarItems računsko svojstvo*

Za stvaranje analitike korištena je biblioteka apexCharts.js. Stvoreno je pet grafova:

- BMI kroz vrijeme
- Fizička aktivnost kroz vrijeme
- Usporedba unesenih kalorija i BMR-a kroz vrijeme
- Usporedba unesenih proteina i preporučenih unosa proteina kroz vrijeme
- Usporedba unesenih ugljikohidrata i preporučenih unosa ugljikohidrata kroz vrijeme
- Usporedba unesenih masti i preporučenih unosa masti kroz vrijeme

Slika 6.51. prikazuje primjer grafa usporedbe unesenih kalorija i BMR-a kroz vrijeme

```
calorieIntakeSeries() {
  return [
    {
      name: 'BMR',
      data: this.bodyHistory.map((historyItem) => [
        historyItem.createdAt,
        historyItem.calculations.BMR,
      ]),
    },
    {
      name: 'Calendar-calories',
      data: this.$extendedCalendarItems
        .map((calendarItem) => [
          calendarItem.date,
          calendarItem.menu.nutrients.calories,
        ])
        .sort( compareFn: (a, b) => (a[0] > b[0] ? 1 : -1)),
    },
  ]
},
```

Slika 6.51 *Podaci za graf usporedbe BMR-a i korisničkog unosa kalorija kroz vrijeme*

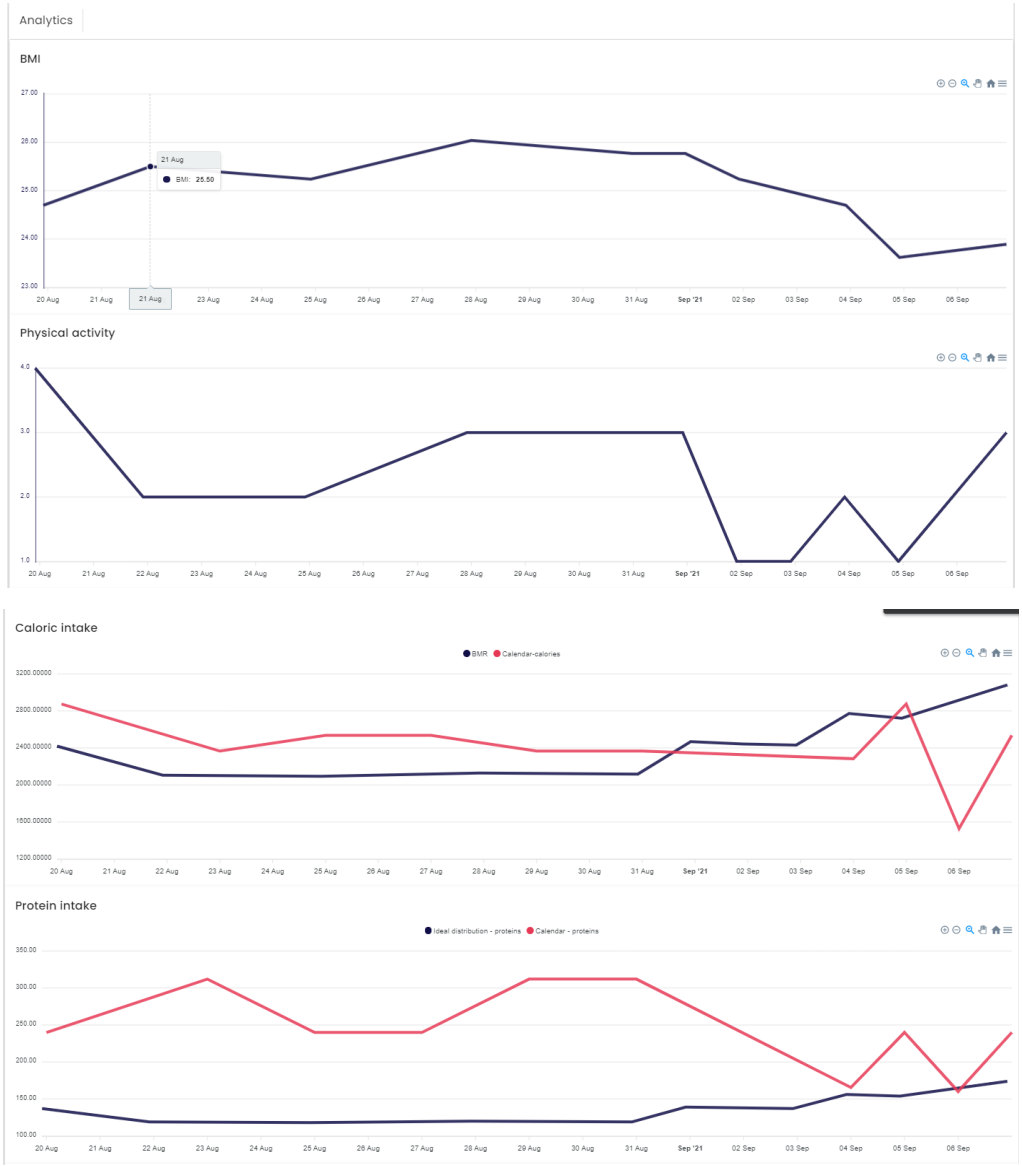
```
doubleDateTimeOptions: {
  xaxis: {
    type: 'datetime',
  },
  legend: {
    position: 'top',
  },
  colors: ['#110F48', '#E63956'],
},
```

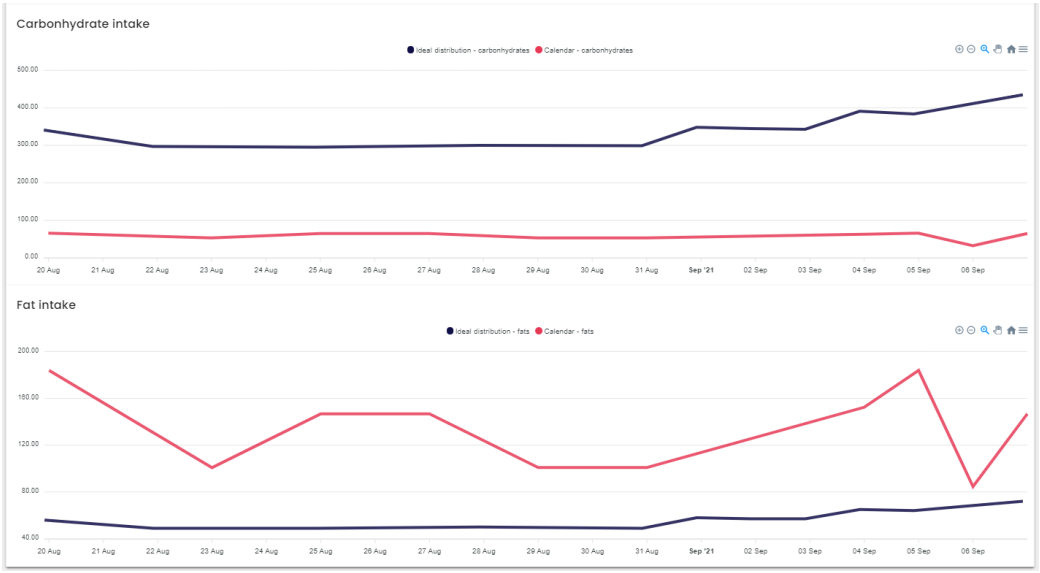
Slika 6.52 *Opcije grafa usporedbe BMR-a i korisničkog unosa kalorija kroz vrijeme*

```
<v-card class="rounded-0">
  <v-card-title>Caloric intake</v-card-title>
  <div class="chart-wrapper">
    <apexchart
      type="line"
      :options="doubleDateTimeOptions"
      :series="calorieIntakeSeries"
      height="400"
    />
  </div>
</v-card>
```

Slika 6.53 *Graf usporedbe BMR-a i korisničkog unosa kalorija kroz vrijeme unutar HTML predloška*

Na kraju je preostalo prikazati primjer izgleda analitike za određene podatke koji su stvoreni tokom vremena. Slika 6.54 prikazuje primjer izgleda analitike korisnika.





Slika 6.54 *Izgled analitike korisnika*

7. ZAKLJUČAK

Cilj ovog diplomskog rada je bio izrada web aplikacije pomoću Nuxt.js razvojnog okvira koja svojim korisnicima pruža pomoć pri izradi pravilnog plana prehrane. Neovisno o tome radi li se o postizanju nekakvog fitness cilja, ili o očuvanju zdravlja pojedinca. Prema [18], pretilost u svijetu se skoro utrostručila od 1975., a 2016., više od 1 900 000 000 odraslih osoba je patilo od prekomjerne težine. Većina ljudske populacije živi u zemljama u kojima prekomjerna težina ubije više ljudi nego glad. Cilj ovog rada je također prikazati koliko je jednostavno voditi računa o svojem zdravlju putem unošenja odgovarajuće količine energije u svoje tijelo.

Osnovni zadaci ove aplikacije su bili omogućiti korisniku uvid u osnovne kalkulacije vezane za njegove tjelesne podatke, te mu pomoći izračunati kako da raspodjeli makronutrijente unutar dnevne količine kilokalorija koje će unijeti, a da ne osjeća glad, slabost i potrebu za unošenjem veće količine energije od potrebne. Korisnik ima ponuđen velik broj namirnica koje može promatrati i povezivati u svoje obroke. Zatim ima mogućnost vezati sve obroke koje je stvorio u jelovnike da bi na različite načine dosegao dnevni cilj energetske vrijednosti i imao raznoliku prehranu svaki dan. Aplikacija mu pri stvaranju obroka pomaže, upozorava ga ukoliko previše odstupa od kalkulacija ili ga ohrabruje ako ih se drži. Zatim u svrhu stvaranja analitike o svojem tijelu može stvarati plan prehrane tako što će svoje jelovnike raspoređivati po danima i imati prikazanu listu svih obroka poredanih po danima konzumiranja. Korisnik bi uvijek aplikaciju trebao osvježavati svojim najnovijim tjelesnim podacima, da bi uvijek imao najtočnije kalkulacije koje mu odgovaraju i pomažu pri daljnjem postizanju rezultata. Nakon svega, tu je i analitika koja korisniku može prikazati vremenske grafove njegova držanja postavljenih ciljeva.

Zamišljeno idejno programsko rješenje se pokazalo ispravnim, te je programsko rješenje uspješno ostvareno. Ispitivanje funkcionalnosti je pokazalo da aplikacija može izvršiti sve postavljene zahtjeve, a daljnjom razradom teorijske podloge i nadodavanjem nekih novih funkcionalnosti bi mogla postati još korisnija i preciznija

LITERATURA

[1] Google play, Calorie Counter-MyFitnessPal, [online], dostupno na: https://play.google.com/store/apps/details?id=com.myfitnesspal.android&hl=en&gl=US, [14.09.2021.]

[2] 8fit, The 8git way, [online], dostupno na: https://8fit.com/about/, [14.09.2021.]

[3] Have a personal trainer in your pocket, dostupno na: https://www.azumio.com/apps/fitness-buddy/overview, [14.09.2021.]

[4] S.R.Rolfes, K. Pinnam, E. Whitney, Understanding Normal & Clinical Nutrition, 12th Edition, USA, 2019.

[5] A. L. Merrill, B. K. Watt, Energy value of foods… basis and derivation, Agricultural Research Service, Agriculture Handbook No.74, February 1973.

[6] K. Žuna, Hrana koja daje energiju i snagu, hrana za snagu, izdržljivost i koncentraciju [online], dostupno na: https://www.krenizdravo.hr/budi-fit/fitness-prehrana/hrana-koja-daje-energiju-i-snagu-hrana-za-snagu-izdrzljivost-i-koncentraciju, [08.09.2021.]

[7] National Heart, Lung and Blood institute [online], dostupno na: https://www.nhlbi.nih.gov/health/educational/lose_wt/BMI/bmicalc.htm, [08.09.2021.]

[8] Internetska nutricionistička enciklopedija [online], dostupno na: https://definicijahrane.hr/, [08.09.2021.]

[9] Firebase Authentication [online], dostupno na: https://firebase.google.com/docs/auth, [08.09.2021.]

[10] Cloud Firestore [online], dostupno na: https://firebase.google.com/docs/firestore, [08.09.2021.]

[11] Vue.js [online], dostupno na: https://vuejs.org/v2/guide/, [08.09.2021.]

[12] G. Fink, I. Flatow, Introducing Single Page Applications, 2014.

[13] K. Lawson, What is a Single Page Application and Why Do People Like Them so Much?, Best Practices [online], dostupno na: https://www.bloomreach.com/en/blog/2018/07/what-is-a-single-page-application.html, [08.09.2021.]

[14] Vue instance [online], dostupno na: https://vuejs.org/v2/guide/instance.html, [08.09.2021.]

[15] Vuex [online], dostupno na: https://vuex.vuejs.org/, [08.09.2021.]

[16] Nuxt.js [online], dostupno na: https://nuxtjs.org/docs/2.x/concepts/views, [08.09.2021.]

[17] D. Svjetličić, Nuxt.js over Vue.js: when should you use it and why, 2019., [online], dostupno na: https://www.bornfight.com/blog/nuxt-js-over-vue-js-when-should-you-use-it-and-why/, [08.09.2021.]

[18] World health organization - Obesity and overweight, dostupno na: https://www.who.int/news-room/fact-sheets/detail/obesity-and-overweight, [08.09.2021.]

SAŽETAK

Cilj diplomskog rada je razrada modela web aplikacije izrađene pomoću razvojnog okvira Nuxt.js, koja svojim korisnicima pruža pomoć pri izradi pravilnog plana prehrane na osnovu njihovih trenutnih tjelesnih podataka. Aplikacija je napisana programskim jezikom Javascript unutar razvojnog okruženja WebStorm. Za ključne funkcionalnosti aplikacije je korištena Google-ova platforma Firebase, odnosno njezine dvije usluge – Firebase Authentication usluga za izradu korisničkih računa, te Cloud Firestore baza podataka.

Korisniku je omogućen uvid u njegove tjelesne kalkulacije i preporučeni su mu dnevni unos kalorija i raspodjela makronutrijenata unutar istog raspona. Korisnik na raspolaganju ima brojne namirnice koje svojevoljno veže u obroke, a pomoću svojih stvorenih obroka može stvarati svoje jelovnike, pri čemu jedan jelovnik predstavlja sve obroke koje planira konzumirati u jednom danu. Aplikacija mu pomaže tako što mu prikazuje upozorenja ovisno o tome koliko jelovnici koje je kreirao odstupaju od preporučenih kalkulacija koje su izrađene za njega.

Korisnik uvijek treba ažurirati aplikaciju svojim najnovijim tjelesnim podacima da bi aplikacija uuvijek sadržavala najtočnije kalkulacije. Svaki puta kada korisnik ažurira podatke o svojem tijelu, ti podaci se pohranjuju u svrhu stvaranja analitike. Korisnik može generirati još jedan dio analitike tako što će svoje jelovnike vezati za datume kada ih planira konzumirati. Na taj način može imati uvid koliko se držao onoga što mu je aplikacija preporučila, te koliki je napredak uspio napraviti.

Ključne riječi: aplikacija, makronutrijenti, Nuxt.js, pravilna prehrana, Vue.js, web,.

ABSTRACT

NUXT.JS FITNESS APPLICATION

The aim of the thesis is to develop a model of a web application created using the framework Nuxt.js, which provides its users with assistance in creating a proper diet plan based on their current body data._The application is written in the Javascript programming language within the WebStorm development environment. For the key functionalities of the application, Google's Firebase platform was used, ie its two services - Firebase Authentication service for creating user accounts, and Cloud Firestore database.

The user is given an insight into his body calculations and is given a recommendation about his daily calorie intake and distribution of macronutrients within the same range. The user has at his disposal a number of groceries that he voluntarily ties into meals, and with the help of his created meals he can create his own menus, where one menu represents all the meals he plans to consume in one day. The app helps him by showing him alerts depending on how much the menus he has created deviate from the recommended calculations made for him.
The user should always update the app with their latest body data so that the app always contains the most accurate calculations. Each time a user updates their body data, that data is stored for the purpose of creating analytics. The user can generate another piece of analytics by linking their menus to the dates when they plan to consume them. In this way, he can have an insight into how much he adhered to what the app recommended to him, and how much progress he managed to make.

Keywords: application, macronutrients, Nuxt.js, proper nutrition, Vue.js, web.

ŽIVOTOPIS

David Bilić je rođen 09.03.1998. u Slavonskom Brodu, Hrvatska. Stanuje u Slavonskom brodu s boravištem u Osijeku. Godine 2003. upisuje OŠ Bogoslav Šulek u Slavonskom Brodu. Nakon odličnog uspjeha u osnovnoj školi, 2012. godine upisuje Tehničku školu Slavonski Brod, gdje je sudjelovao na državnom natjecanju iz elektrotehnike. 2016. godine, nakon izvrsnog uspjeha ostvaruje izravan upis na preddiplomski sveučilišni studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijeku. Nakon završenog preddiplomskog svučilišnog studija, upisuje diplomski sveučilišni studij računarstva, blok Programskog Injženjerstva. Od programskih jezika poznaje strukture C-a, C++-a, C-a, Jave, Kotlina i Javascripta. Izvrsno poznaje engleski jezik. Od vrlina navodi želju za usavršavanjem.

_____

David Bilić

PRILOZI

Prilog 1: Završni rad u docx i pdf formatu

Prilog 2: Projekt web aplikacije izrađen u Nuxt.js razvojnom okruženju