iOS Aplikacija za vođenje dnevnika prehrane

Šutalo, Domagoj

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:826187

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-07-14**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

MOBILNA iOS APLIKACIJA ZA VOĐENJE DNEVNIKA PREHRANE

Diplomski rad

Domagoj Šutalo

Osijek, 2023.

SADRŽAJ

1. UVOD 1

2. PREGLED PODRUČJA TEME 3

2.1. Postojeća rješenja 3

2.1.1. MyFitnessPal 3

2.1.2. Cronometer 4

2.1.3. Lose it! 5

2.1.4. Noom 7

2.1.5. Lifesum 7

3. TEHNOLOGIJE I ALATI POTREBNI ZA IZRADU APLIKACIJE 9

3.1. Xcode 9

3.2. Swift i SwiftUI 10

3.3. Firebase 12

3.4. AVFoundation i CodeScanner 12

3.5. Open Food Facts API 13

4. IDEJNO RJEŠENJE I MODEL MOBILNE APLIKACIJE 14

4.1. Dijagram tijeka aplikacije 14

4.2. Programska arhitektura aplikacije 17

4.3. Arhitektura baze podataka 18

5. PROGRAMSKO RJEŠENJE MOBILNE APLIKACIJE 20

5.1. Postavljanje korisnika 20

5.2. Skeniranje linijskog koda proizvoda 23

5.3. Dohvaćanje proizvoda preko API poziva 25

5.4. Baza podataka 32

5.5. Postavljanje cilja korisnika 41

6. NAČIN RADA APLIKACIJE 44

6.1. Korištenje aplikacije 44

7. REZULTATI PROVEDENOG ANKETNOG ISTRAŽIVANJA 57

8. ZAKLJUČAK 61

LITERATURA 62

ŽIVOTOPIS 65

SAŽETAK 66

ABSTRACT 67

PRILOZI 68

1. UVOD

Globalizacija i urbanizacija, stres, ubrzani, a s druge strane sjedilački način života doprinijeli su razvoju jednog od većih problema u svijetu, problemu pretilosti. Situacija je takva da je pretilost postala veći problem od problema pothranjenosti. O tome govori i činjenica da je Svjetska zdravstvena organizacija pretilost proglasila kao najveći, globalni, kronični zdravstveni problem kod odraslih osoba. Pretilost je dobila veću pažnju za vrijeme COVID i post-COVID krize jer su osobe koje su oboljele od COVID-19 bolesti imale veći rizik od hospitalizacije i smrti [1]. Također, COVID je indirektno utjecao na povećanje broja pretilih osoba u svijetu zbog mjera koje su donesene radi borbe protiv tog virusa. Zatvarajući ljude u domove i samim tim smanjenje fizičkih aktivnosti. Pretilost nastaje nakupljanjem masti što je posljedice većeg unosa energije od one koju tijelo troši. Prema [2] danas se u svijetu više od pola milijardi ljudi smatra pretilima. Statistika nije štedjela ni Republiku Hrvatsku, gdje gotovo dvije trećine odraslih (65%) ima prekomjernu tjelesnu masu. Od tog broja, 42% je prehranjeno, dok je preostalih 23% pretilo. Više od dvije trećine pacijenata s kardiovaskularnim bolestima su pretili, a zanimljivo je da se čak 14 do 20% svih smrtnih slučajeva uzrokovanih karcinomom može povezati s pretilošću. Pretilost je još povezana s nastankom bolesti endokrinog, lokomotornog, gastro-intestinalnog sustava i drugih. Da bi se ova kronična bolest uspješno liječila, ili još bolje, spriječila, ključno je ispraviti energetsku neuravnoteženost. Radi se o energetskoj neuravnoteženosti gdje je potrošnja energije manja od energije unesene u tijelo. Poznajući svoju ukupnu dnevnu potrošnju energije (*eng. Total Daily Energy Expenditure*), osoba može kroz manji unos kalorija i u kombinaciji s fizičkom aktivnošću, smanjiti postotak masti u tijelu i izaći iz zone pretilosti.

Radom je potrebno izraditi iOS aplikaciju koja omogućava korisniku vođenje dnevnika prehrane gdje korisnik unosi svoje ciljeve, a aplikacija mu predlaže dnevni unos kalorija s postotkom makronutrijenata.

Aplikacija bi trebala biti pisana u Swiftu, te se pri izradi treba koristi SwiftUI okvir. Informacije o korisniku se trebaju spremati u bazu podataka, dok arhitektura aplikacija treba biti postavljena prema MVVM (Model-View-ViewModel) obrascu.

U radu je potrebno ukazati na korisnost ovakve aplikacije te na osnovi istraživanja tržišta prikazati mogući broj korisnika te ciljanu skupinu.

Nakon uvoda, u drugom poglavlju istražena su već postojeća rješenja, dok su u trećem poglavlju opisane tehnologije i alati potrebni za izradu aplikacije. U četvrtom poglavlju prolazi se kroz idejno rješenje i model mobilne aplikacije, gdje će biti prikazan dijagram tijeka

aplikacije i programska arhitektura aplikacije. U petom poglavlju opisano je programsko rješenje mobilne aplikacije, dok je u šestom poglavlju prikazan način rada aplikacije. U sedmom poglavlju provedeno je istraživanje o potrebi za ovakvom aplikacijom. Zaključak rada dan je u zadnjem poglavlju u kojemu se također navodi moguća nadogradnja aplikacije dane radom.

2. PREGLED PODRUČJA TEME

Prema [3], u razdoblju od prosinca 2017. godine do veljače 2018. godine, provedeno je istraživanje kojemu je cilj bio istražiti opće znanje ispitanika o pretilosti, saznati životne navike ispitanika, kao i njihov indeks tjelesne mase (ITM) koji je služio kao podatak o pretilosti ispitanika. U istraživanju je sudjelovalo 678 ispitanika u dobi od 18 do 30 godina, te su rješavanju ankete mogli pristupiti stanovnici s cijelog područja Republike Hrvatske. Prema rezultatima istraživanja, sveukupan broj pretilih osoba (ITM iznad 30) iznosi 12%. Na uzorku od ukupnog broja ispitanika, ovaj postotak predstavlja 78 osoba, što je značajan broj osoba s obzirom na ukupan broj ispitanika. Također, podaci dobiveni ovim istraživanjem ukazuju na problem u drugom ekstremu, a to je pothranjenost, gdje se čak 20,7% ispitanika izjasnilo da njihov ITM iznosi ispod 20. Istraživanjem se zaključilo da su Hrvati upoznati s problemom pretilosti, no usprkos tome, brojke ukazuju na sve veći broj osoba s prekomjernom težinom. Iako se gotovo trećina ispitanika (31,3%) bavi tjelesnom aktivnošću barem 2 do 3 puta tjedno, trend nezdravog prehrambenog obrasca i dalje je prisutan u Republici Hrvatskoj. Prema [4], istraživanje provedeno u 2019. godini također ukazuje na trend nezdrave prehrane u Republici Hrvatskoj, gdje gotovo dvije trećine Hrvata ima prekomjernu tjelesnu masu (ITM iznad 25). Upravo taj problem ukazuje na potrebu za rješenjima poput aplikacije za vođenje dnevnika prehrane.

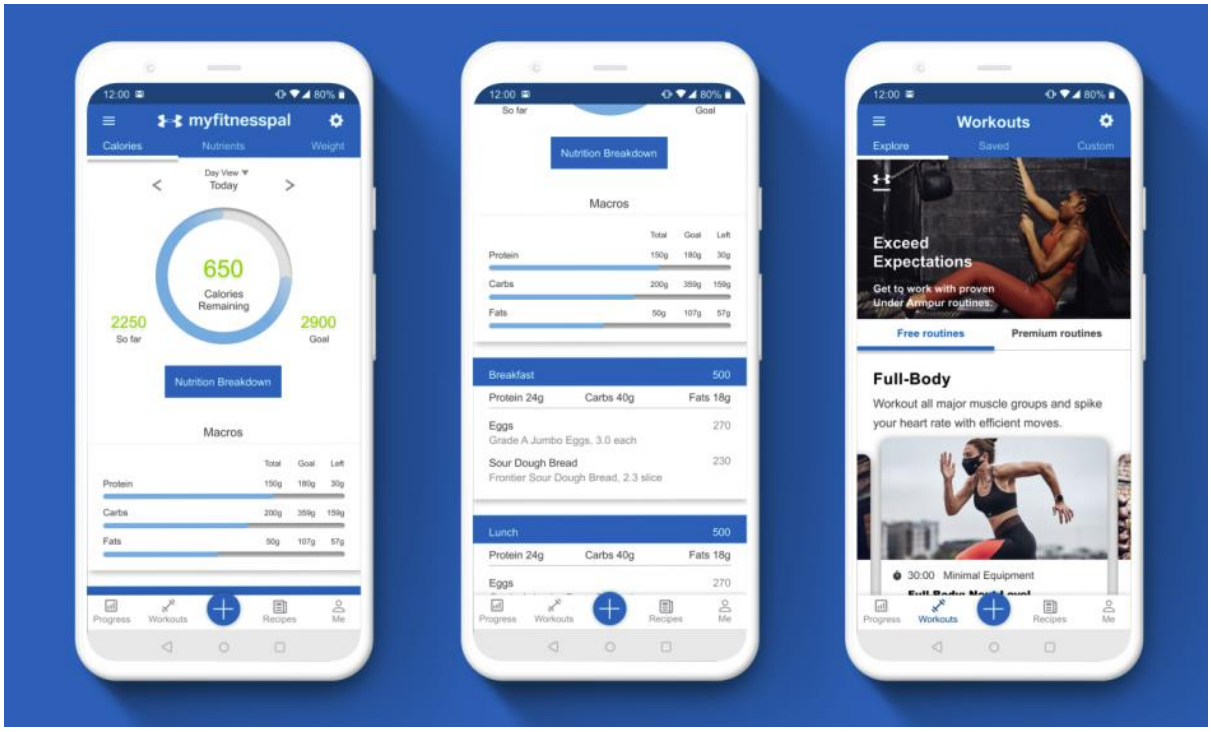
2.1. Postojeća rješenja

U današnjem digitalnom dobu, briga o zdravoj prehrani i tjelesnoj aktivnosti postaje lakša nego ikad zahvaljujući raznim aplikacijama dostupnim na pametnim telefonima. Pet najpoznatijih aplikacija su MyFitnessPal, Cronometer, Lose it!, Noom i Lifesum, koje pružaju korisnicima alate za praćenje dnevnika prehrane. Navedene aplikacije biti će opisane u nastavku.

2.1.1. MyFitnessPal

Prema [5], MyFitnessPal predstavlja jednu od najpoznatijih aplikacija za vođenje dnevnika prehrane, koja od 2005. godine korisnicima omogućuje praćenje dnevnog unosa kalorija, vode, tjelesne težine i obavljenih vježbi tijekom treninga. Aplikacija također može biti prilagođena od strane korisnika prema njihovim potrebama, omogućujući im postavljanje vlastitih ciljeva. Ciljevi se, u tom slučaju, dijele na gubitak, dobivanje ili održavanje tjelesne težine. S obzirom da je jedna od najpoznatijih aplikacija, ona također posjeduje jednu od najvećih baza podataka o prehrambenim proizvodima. Iako je aplikacija besplatna za korištenje, neke od

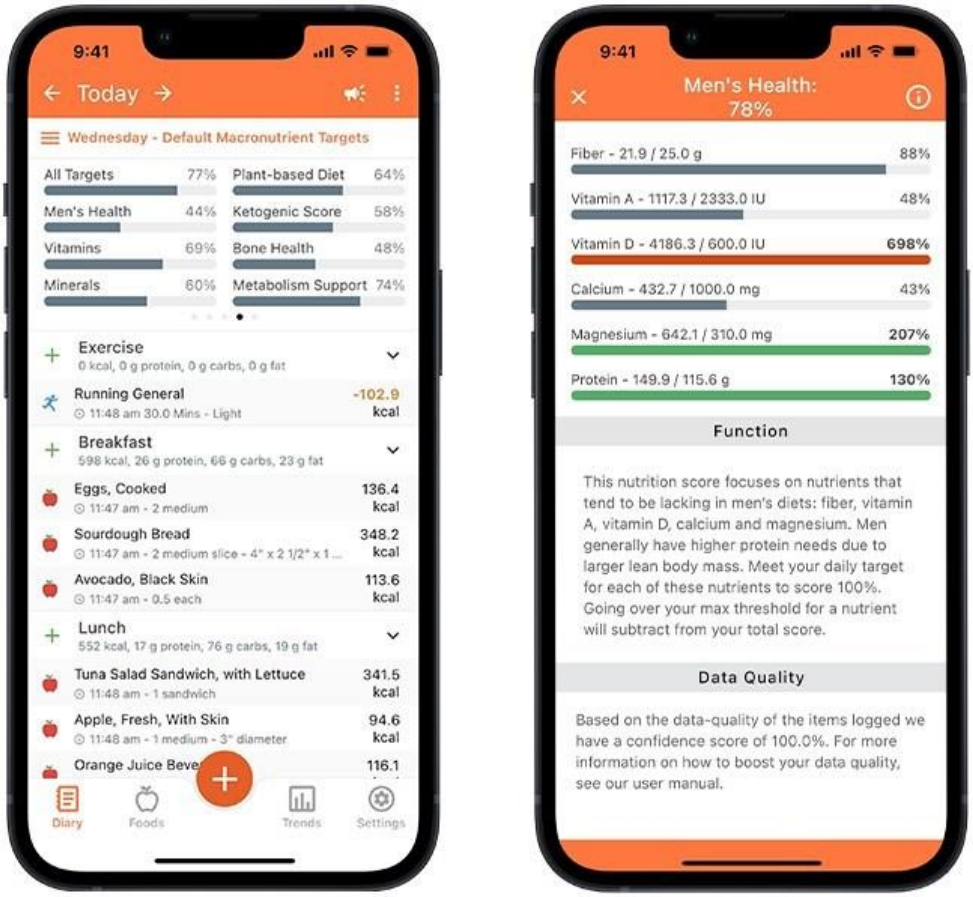
funkcionalnosti dostupne su samo korisnicima s pretplatom. Također, besplatna verzija aplikacije sadrži reklame. Na slici 2.1. prikazan je izgled MyFitnessPal aplikacije.



Slika 2.1. Prikaz MyFitnessPal aplikacije [6]

2.1.2. Cronometer

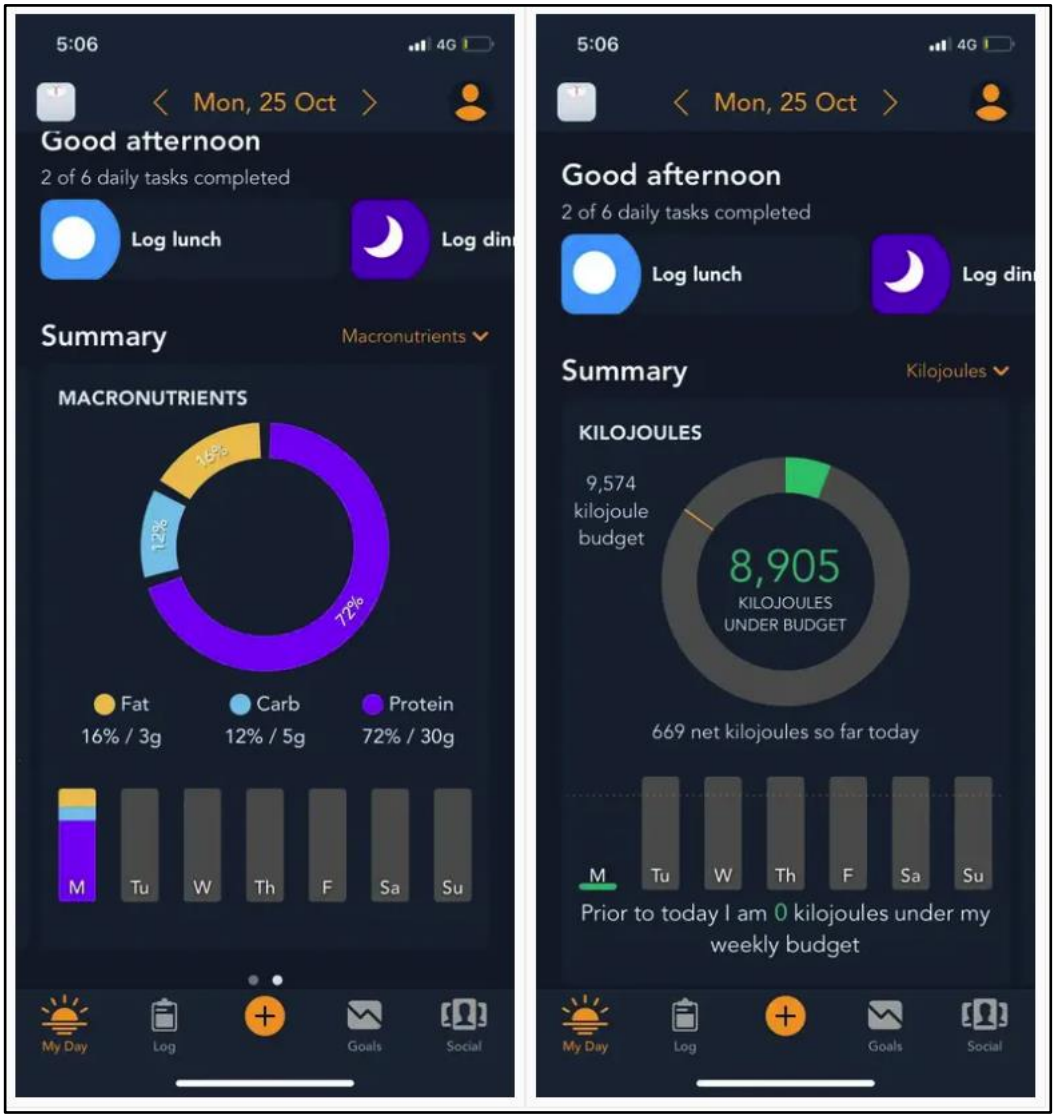
Prema [7], praćenje svakodnevnog unosa kalorija te pregled makronutrijenata i mikronutrijenata omogućen je putem Cronometer aplikacije. Na početku korištenja aplikacije, ciljevi mogu biti postavljeni od strane korisnika, kao i upisivanje obavljenih vježbi tijekom treninga i unos vode. Kao i za MyFitnessPal, omogućeno je besplatno preuzimanje Cronometer aplikacije na pametne telefone, iako aplikacija također sadrži i funkcionalnosti koje su dostupne samo korisnicima s pretplatom. Aplikacija također može biti percipirana kao kompleksna za nove korisnike koji se prvi put susreću s praćenjem prehrane. Na slici 2.2. prikazan je izgled Cronometer aplikacije.



Slika 2.2. Prikaz Cronometer aplikacije [8]

2.1.3. Lose it!

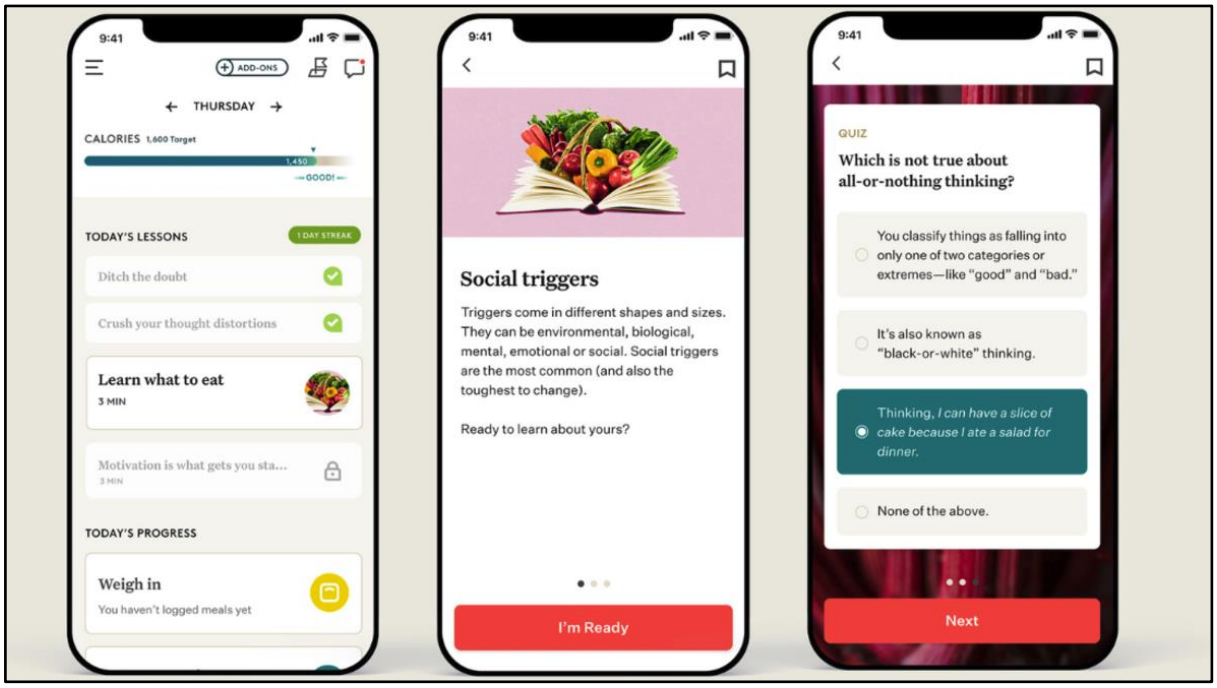
Prema [9], Lose it! predstavlja aplikaciju za praćenje dnevnika prehrane koja je prvenstveno bazirana na gubitku kilaže. Putem ove aplikacije omogućeno je praćenje unosa hrane, tjelesnih aktivnosti i postavljanje zdravstvenih ciljeva. Na početku korištenja aplikacije korisniku je prikazan obrazac koji je korišten kako bi aplikacije bolje razumjela korisnike potrebe. Nakon ispunjavanja obrasca, korisnik je u mogućnosti odabrati različite planove mršavljenja. Korisničko sučelje je jednostavno i jasno, čak i za početnike, ali postavljanje profila korisnika može biti dugotrajno i zamorno zbog previše polja za ispunjavanje. Na slici 2.3. prikazan je izgled Lose it! aplikacije.



Slika 2.3. Prikaz Loose it! aplikacije [9]

2.1.4. Noom

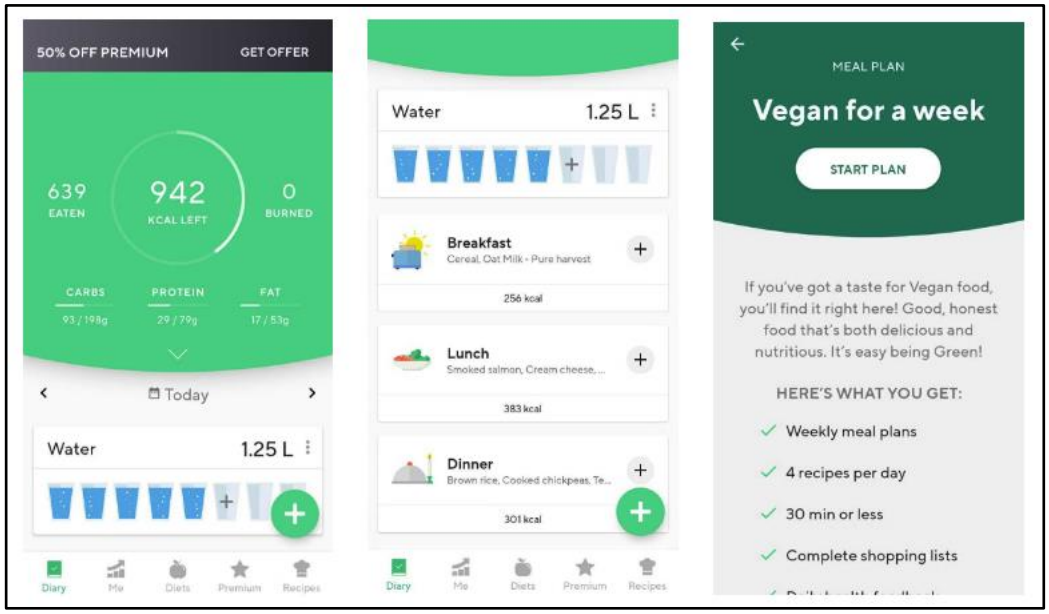
Prema [10], Noom predstavlja mobilnu aplikaciju za zdravlje i mršavljenje koja se ističe svojim jedinstvenim pristupom. Alati za praćenje prehrane i vježbanja pruženi su ovom aplikacijom koja se također temelji i na psihološkim i edukativnim principima. Personalizirani planovi prehrane i vježbanja te svakodnevne lekcije o zdravlju i motivaciji ponuđeni su korisnicima putem Noom aplikacije. Aplikacija nudi besplatno probno razdoblje od samo dva tjedna nakon čega korisnici moraju platiti pretplatu kako bi nastavili s korištenjem aplikacije. Na slici 2.4. prikazan je izgled Noom aplikacije.



Slika 2.4. Prikaz Noom aplikacije [10]

2.1.5. Lifesum

Prema [11], Lifesum predstavlja mobilnu aplikaciju za praćenje dnevnika prehrane. Korisnicima je omogućeno praćenje unosa hrane, odrađenih vježbi u danu kao i unos vode. Personalizirani planovi i recepti pružaju se putem aplikacije, a informacije o kalorijama i nutritivnim vrijednostima hrane također su dostupne. Iako se nekim značajkama aplikacije može pristupiti besplatno, dosta značajki i funkcionalnosti se plaća, a besplatna verzija također u sebi sadrži brojne reklame. Na slici 2.5. prikazan je izgled Lifesum aplikacije.



Slika 2.5. Prikaz Lifesum aplikacije [11]

3. TEHNOLOGIJE I ALATI POTREBNI ZA IZRADU APLIKACIJE

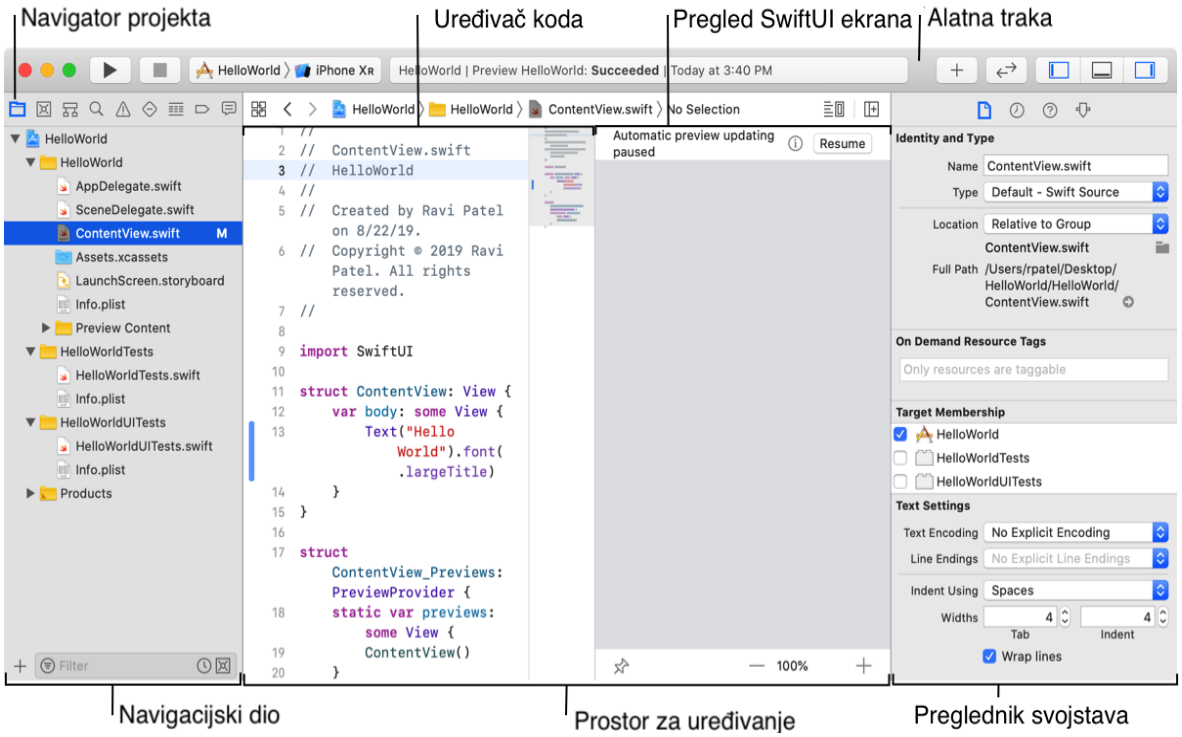
Za izradu mobilne aplikacije, koja je rezultat ovog diplomskog rada, korištene su različite tehnologije i alati, počevši od okruženja za izradu mobilne aplikacije, pa sve do baze podataka potrebne za pohranu i dohvaćanje spremljenih podataka. U nastavku ovog poglavlja, svaka od ovih tehnologija i alata detaljno je opisana.

3.1. Xcode

Prema [12] Xcode predstavlja Apple integrirano razvojno okruženje (*eng. Integrated Development Environment*) te se koristi za izradu aplikacija za Apple uređaje uključujući: iPad, iPhone, Apple pametni sat, Apple TV i Mac. Xcode pruža širok spektar alata za upravljanje svim fazama razvojnog procesa, uključujući izradu aplikacije, testiranje, optimizaciju i postavljanje aplikacije na Apple trgovinu aplikacijama (*eng. App Store*). Xcode je korišten isključivo za razvoj aplikacija za Apple uređaje i dostupan je samo za korisnike koji koriste MacOS operativni sustav.

U Xcode okruženju se mogu koristiti dva različita sučelja za izradu aplikacija: Storyboard i SwiftUI. Storyboard predstavljeno je kao sučelje koje olakšava organizaciju i upravljanje različitim zaslonima ili prikazima (*eng. views*) u aplikaciji. Pomoću njega programeri mogu definirati različite prijelaze između zaslona, povezati ih s akcijama te pridružiti kontrolere prikaza kako bi upravljali logikom aplikacije. SwiftUI s druge strane predstavlja okvir koji koristi deklarativni pristup programiranju, te je on relativno novi alat u Xcode-u koji omogućuje brzo i jednostavno stvaranje korisničkog sučelja.[13]

Najčešće korišten programski jezik za pisanje koda u Xcode okruženju je Swift koji predstavlja Apple-ov vlastiti jezik za razvoj aplikacija. Xcode također podržava pisanje koda u Objective-C programskom jeziku, koji predstavlja stariji jezik koji se koristio za izradu aplikacija do 2014. godine. Od 2014. godine Objective-C je zamijenjen Swift jezikom, o kojemu će više biti pisano u sljedećem dijelu ovog poglavlja. Na slici 3.1. prikazano je XCode sučelje s njegovim glavnim dijelovima s lijeva na desno: navigacijski dio, prostor za uređivanje koda, prostor za brzi pregled SwiftUI ekrana bez potrebe za pokretanjem aplikacije, preglednik svojstava te alatna traka na vrhu sučelja.



Slika 3.1. Prikaz Xcode sučelja [13]

3.2. Swift i SwiftUI

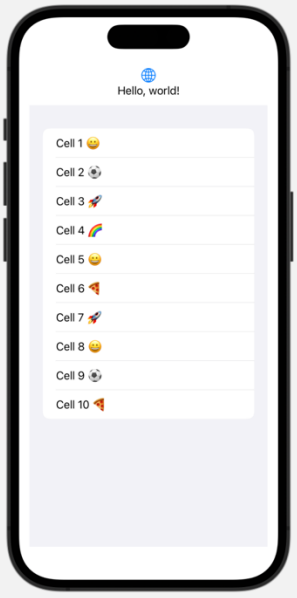
Kao što je i ranije navedeno Swift predstavlja programski jezik za pisanje programa za Apple uređaje koji je predstavljen 2014. godine na WWDC-u (*eng. Worldwide Developers Conference*). Iako je već postojao Objective-C jezik koji se koristio za izradu aplikacija za Apple uređaje, njegova sintaksa je starija i datira iz 1984. godine, prije nego što su se pojavili skriptni jezici koji su popularizirali jednostavniju i elegantnu sintaksu. Objective-C bio je pionir u mnogim aspektima objektno orijentiranog programiranja i samim time omogućivao veliku prilagodljivost u promjeni ponašanja programa tijekom izvršavanja. To je rezultiralo manjim brojem otkrivenih grešaka tijekom razvoja. Umjesto toga, greške su često postajale vidljive tek kada bi korisnici krenuli koristiti program, što bi rezultiralo rušenjem programa [14, str. 15]. Swift je također stvoren kako bi postao zamjena Objective-C jezika, budući da treba biti sigurniji jer njegova sintaksa potiče korisnika na pisanje čistog i konzistentnog koda. Swift je nudio i brže performanse o čemu govore njegove brojke - duplo brži od Objective-C i osam puta brži od jednog od najpoznatijih programskih jezika, Pythona [15]. Na slici 3.2. prikazan je jednostavan Swift program.

```swift
import UIKit

var optionalArray: [String]? = nil

optionalArray = ["Dark Necessities",
                 "Can't stop",
                 "Otherside",
                 "Scar tissue",
                 "Snow"]

if let optionalArray {
    print(optionalArray)
} else {
    print("OptionalArray is nil")
}

```

Slika 3.2. Prikaz jednostavnog Swift programa

Ranije spomenuti SwiftUI, koji je predstavljen 2019. godine na WWDC-u (*eng. Worldwide Developers Conference*), predstavlja Apple deklarativni okvir koji poboljšava način na koji programeri stvaraju korisnička sučelja za svoje aplikacije. Također pruža deklarativni pristup koji omogućuje programerima da definiraju željeno korisničko sučelje navodeći kako bi trebalo izgledati i ponašati se. On se razlikuje od prethodnih Apple okvira korisničkog sučelja kao što je UIKit i AppKit koji su zahtijevali imperativan stil programiranja. Pomoću SwiftUI okvira programeri mogu izraditi korisnička sučelja za različite Apple platforme kao što su: iOS, iPadOS, watchOS, tvOS i macOS [16]. Na slici 3.3. prikazana je jednostavna aplikacija napravljena korištenjem SwiftUI okvira.

```swift
//  Created by Domagoj Šutalo on 14.06.2023..
//

import SwiftUI

struct ContentView: View {

    let emojis = ["😀", "🚀", "🌈", "🍕", "⚽"]

    var body: some View {
        VStack {
            Image(systemName: "globe")
                .imageScale(.large)
                .foregroundColor(.accentColor)
            Text("Hello, world!")
            List(1...10, id: \.self) { number in
                Text("Cell \(number) \(emojis.randomElement() ?? "")")
            }
        }
        .padding()
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}

```



Slika 3.3. Prikaz jednostavne aplikacije stvorene pomoću SwiftUI okvira

3.3. Firebase

Prema [17], Firebase je prvobitno bio pružatelj usluge dopisivanja preko interneta (*eng. online chat*) za različite web stranice putem API-ja (*eng. Application Programming Interface*) pod nazivom Envolve. S vremenom je postao sve popularniji jer su ga programeri počeli koristiti za razmjenu podataka aplikacija, poput stanja igre, u stvarnom vremenu. To je rezultiralo odvajanjem Firebasea od Envolvea. Danas je on dio Googlea i predstavlja skup alata za razvoj, izgradnju i poboljšanje aplikacija. Neki od Firebase alata predstavljaju:

- *Crashlytics* koji pomaže programerima pronaći i popraviti problem koji rezultira prisilnim prekidom aplikacije (*eng. crash*)
- Dopisivanje preko oblaka (*eng. cloud messaging*)
- Autentifikacija koja omogućuje laku izradu korisničkog računa
- Baza podataka u stvarnom vremenu (*eng. Realtime Database*) koja omogućuje spremanje podataka na oblak i čitanje tih istih podataka

Budući da je za potrebe ove aplikacije bilo potrebno izraditi korisnika, u aplikaciji je korišten alat autentifikacije. Također, kako bi korisnik mogao unositi proizvode u svoj dnevnik prehrane, bilo je potrebno koristiti alat za bazu podataka u stvarnom vremenu koji omogućuje spremanje i dohvaćanje unesenih proizvoda.

3.4. AVFoundation i CodeScanner

Prema [18] AVFoundation predstavlja Apple okvir (*eng. framework*) koji se koristi za rad s multimedijima na Apple uređajima i operativnim sustavima macOS, iOS, watchOS i tvOS. Ovaj okvir integrira nekoliko ključnih tehnoloških rješenja koja zajedno omogućuju raznovrsne funkcionalnosti za upravljanje, pregled, snimanje, reprodukciju i obradu audiovizualnih medija. Radi toga, AVFoundation se može koristiti za reprodukciju video sadržaja i snimanje zvuka. Također, omogućuje obradu video zapisa tijekom snimanja i reprodukcije u stvarnom vremenu, te pruža brojne druge funkcije za upravljanje s multimedijom. On postaje ključan za razvoj aplikacija na Apple platformi, te omogućuje programerima pristup različitim hardverskim dijelovima uređaja poput kamere, mikrofona, zvučnika i drugih audio-vizualnih komponenti. Zbog svojih mogućnosti za obradu multimedije, AVFoundation se koristi u CodeScanner okviru koji prema [19] predstavlja okvir koji olakšava implementaciju sustava za skeniranje linijskog koda, QR kodova i sličnih.

3.5. Open Food Facts API

Open Food Facts predstavlja otvorenu bazu podataka o hrani koja je javno dostupna. Ona sadrži informacije o mnogim vrstama prehrambenih proizvoda te se o svakom proizvodu mogu naći njegovi detaljniji podaci kao što su njegovi sastojci, nutritivna vrijednost, alergeni i ekološke karakteristike proizvoda. Osnovna svrha ove baze podataka je pružiti transparentne informacije o hrani njenim potrošačima, a samim time i bolji izbor hrane. Budući da je baza podataka otvorenog tipa, to znači da je baza dostupna svima i omogućava svakom korisniku dodavanje novih proizvoda s pratećim podacima. Upravo na toj ideji je ova baza podataka i zaživjela: korisnici koji nesebično pridonose inicijativi ove baze podataka [20].

Korištenje API poziva za pretragu proizvoda po imenu ili linijskom kodu detaljnije je objašnjeno u sljedećem poglavlju.

4. IDEJNO RJEŠENJE I MODEL MOBILNE APLIKACIJE

Velik broj ljudi danas nije zadovoljno izgledom svoga tijela, vode borbu s pretilošću ili smanjenom tjelesnom masom te nisu sigurni odakle početi. Iako je uvođenje nekakve vrste tjelovježbe u svakodnevni život iznimno bitno, također je bitno stvoriti plan uravnotežene prehrane. Upravo to predstavlja cilj ove aplikacije - omogućiti korisnicima unos njihovih ciljeva i prepustiti aplikaciji stvaranje plana prehrane.

U ovom poglavlju prikazan je dijagram tijeka aplikacije, predstavljena je programska arhitektura aplikacije, kao i arhitektura baze podataka.

4.1. Dijagram tijeka aplikacije

Dijagram tijeka aplikacije izrađen je s namjerom da olakša razumijevanje aplikacije i njenih funkcionalnosti. Kao što se može iščitati sa slike 4.1., korisnik prilikom ulaska u aplikaciju dolazi do zaslona za registraciju, odnosno do zaslona za prijavu ako je registracija već obavljena. Korisnikova adresa elektroničke pošte i zaporka pohranjuju se u bazu podataka prilikom registracije. U slučaju uspješne registracije ili prijave, korisnik je preusmjeren na glavni zaslon aplikacije. Za registraciju potreban je unos valjane adrese elektroničke pošte i izrada zaporke koja mora biti dulja od šest znakova.

Kao što je vidljivo na slici 4.1., nakon korisnikove prijave u aplikaciju, ako u lokalnoj bazi podataka postoje već spremljeni ciljevi korisnika, tada se prikazuje zaslon dnevnika prehrane za odabrani datum.

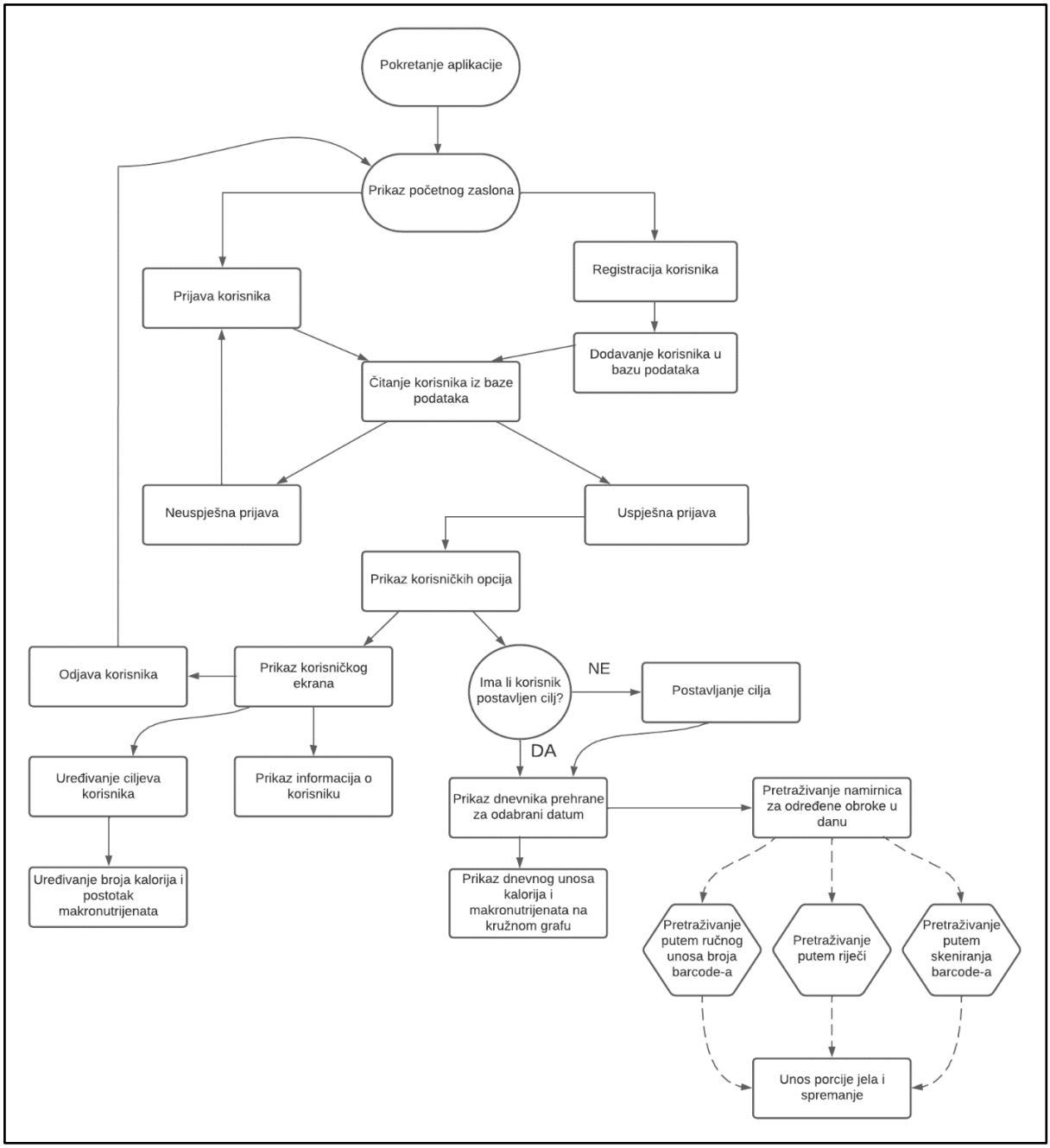
Ukoliko postoje prethodno spremljeni ciljevi korisnika u lokalnoj bazi podataka, nakon prijave korisnika u aplikaciju, prikazuje se zaslon dnevnika prehrane za odabrani datum. U slučaju da u lokalnoj bazi podataka nema spremljenih podataka o ciljevima korisnika tada se prikazuje zaslon za ciljeve. Na tom zaslonu korisnik upisuje osnovne informacije o sebi i njegove ciljeve - gubitak, dobitak ili održavanje tjelesne težine. Tek nakon unošenja podataka, korištenje aplikacije može biti nastavljeno. Nakon spremanja korisnikovih ciljeva, prikazuje se već spomenuti zaslon dnevnika prehrane za odabrani datum. Na tome zaslonu korisniku se prikazuju četiri obroka: doručak, ručak, večera i užina. Za svaki od obroka, korisniku je omogućeno dodavanje namirnica koje je koristio za svako jelo, što ga vodi na sljedeći zaslon.

Sljedeći zaslon predstavlja takozvani zaslon pretraživanja, gdje je korisnik u mogućnosti pretražiti namirnice na tri načina:

- Putem pretraživanja namirnice po imenu
- Putem pretraživanja namirnice ručnim unosom broja linijskog koda s proizvoda

- Putem pretraživanja namirnice pomoću skeniranja linijskog koda s proizvoda

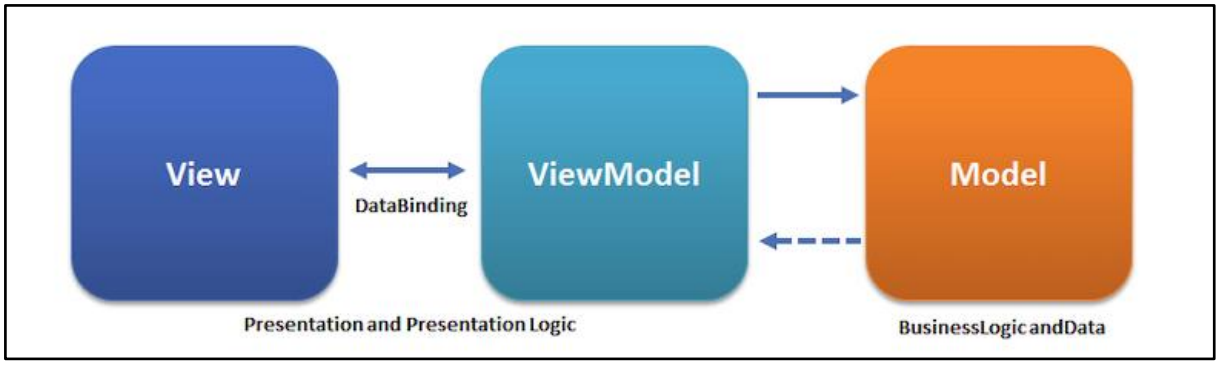
Koji god od ova tri načina korisnik odabere, na sljedećem zaslonu dobiva mogućnost unošenja porcije, odnosno gramažu namirnice koju je koristio. Nakon što se unese porcija, podaci o namirnici i njezinoj količini pohranjuju se u bazu podataka. Prilikom povratka na zaslon dnevnika prehrane, prethodno spremljeni podaci o namirnicama se dohvaćaju i prikazuju na zaslonu. Dohvaća se također broj kalorija tih namirnica, broj kalorija za određeni obrok i ukupan broj kalorija u danu. Kako bi korisnik imao bolji uvid u kalorije i makronutrijente koje je unio u danu, na ovome zaslonu može kliknuti na dugme koje će mu prikazati novi zaslon s kružnim grafom. Na zaslonu dnevnika prehrane korisnik također može pristupiti zaslonu koji prikazuje opcije za korisnika. Korisnik ovdje može vidjeti općenite informacije o sebi, urediti svoje ciljeve, broj kalorija, te postotak makronutrijenata koje treba unijeti u danu. Na kraju korisnik također na tome zaslonu ima opciju i odjave, koja ga vraća na početni zaslon za prijavu u sustav. Dijagram aplikacije prikazan je na slici 4.1.



Slika 4.1. Prikaz dijagrama aplikacije

4.2. Programska arhitektura aplikacije

Programska arhitektura aplikacije temelji se na MVVM (*eng. Model - View - ViewModel*) obrascu kojeg se može prevesti kao Model - Pogled - Pogledni model. Razlog za korištenje ove arhitekture, bilo je uvođenje strukture u projekt što omogućuje lako snalaženje u kodu. Ova arhitektura također omogućuje lakše testiranje aplikacije, za razliku od koda koji u jednom dokumentu (*eng. file*) sadrži sve klase, funkcije i poslovnu logiku. Osnovna ideja MVVM-a je jasno razdvajanje odgovornosti između različitih komponenti. Model treba sadržavati izvor podataka, te je on zadužen za upravljanje logike tih podataka. Pogled, kako mu ime sugerira, služi za prikaz korisničkog sučelja i potpuno je neovisan o poslovnoj logici. On nije svjestan postojanja modela, kao što ni model nije svjestan postojanja pogleda. Budući da model i pogled nisu svjesni postojanja jedno drugoga, tu je potreban pogledni model koji služi kao posrednik između modela i pogleda. On implementira i izlaže javna svojstva i naredbe koje pogled koristi putem vezivanja podataka. Ukoliko dođe do promjene nekakvog stanja u modelu, pogledni model će obavijestiti pogled da se ažurira [21, str. 220]. Slika 4.2 prikazuje međusobni odnos elemenata MVVM arhitekture.

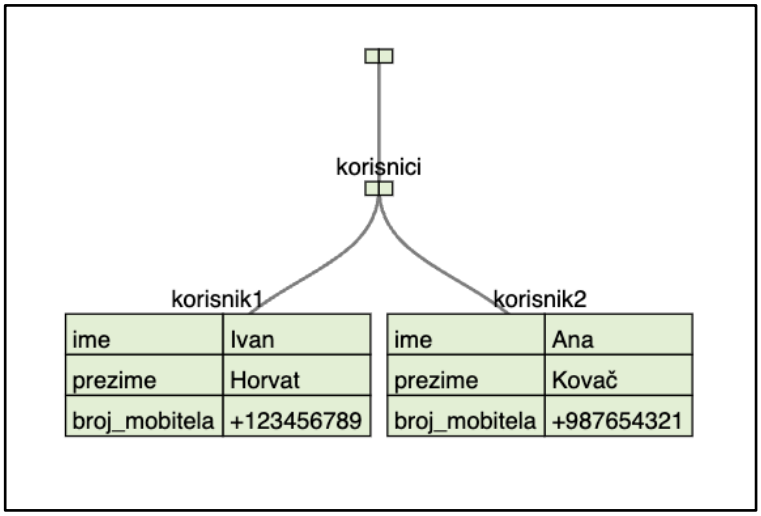


Slika 4.2. Prikaz MVVM arhitekture [22]

4.3. Arhitektura baze podataka

Kao što je već istraženo u prethodnom poglavlju, Firebase predstavlja razvojni alat koji olakšava izradu mobilnih i web aplikacija. Također su spomenute neke od komponenti Firebasea, a ovdje će više biti pisano o njegovoj bazi podataka u stvarnom vremenu. Firebase baza podataka u stvarnom vremenu (*eng. Realtime database*) predstavlja alat koji omogućuje aplikaciji sinkronizaciju i pohranjivanje podataka u stvarnom vremenu. Ova tehnologija ima ključnu ulogu u stvaranju aplikacije koje reagiraju na trenutne promjene i omogućuje korisnicima da dijele podatke u stvarnom vremenu. Važno je napomenuti kako ova baza podataka ne izgleda kao standardna SQL (*eng. Standard Query Language*) baza podataka, gdje se podaci spremaju u tablice, već je ona tipa NoSQL. S obzirom na mnogo tipova NoSQL baza podataka, u ovom slučaju se koristi hijerarhijska struktura baza podataka gdje su podaci organizirani u obliku JSON stabla, pri čemu svaki čvor u bazi podataka predstavlja jedan JSON objekt. Na slici 4.3. prikazan je primjer jednostavnih podataka u JSON formatu, dok su na slici 4.4. ti podaci prikazani u obliku stabla.

Slika 4.3. Prikaz podataka u JSON formatu



Slika 4.4. Prikaz JSON podataka u obliku stabla

Za potrebe korištenja aplikacije ono što je potrebno spremati je slijedeće:

- korisnikov jedinstveni identifikacijski ključ
- datum pod kojim su spremljeni obroci korisnika
- proizvodi koje je korisnik unosio

S time na umu, ovako bi izgledala putanja do jednog proizvoda koji je spremljen u bazu podataka: **/korisnici/korisnikovKljuč/datumi/29-08-2023/ključObroka/proizvodi**

Prikaz stvarne baze podataka ove aplikacije prikazan je na slici 4.5.

Slika 4.5. Prikaz Firebase baze podataka

5. PROGRAMSKO RJEŠENJE MOBILNE APLIKACIJE

U ovom poglavlju prikazano je programsko rješenje mobilne aplikacije, odnosno prikaz najbitnijih dijelova koda. S obzirom da je aplikacija stvorena da bude personalizirana, a ne generička, u prvom dijelu ovog poglavlja biti će prikazano stvaranje i postavljanje korisnika. Kako bi se omogućilo lakše pronalaženje nekog proizvoda, korisniku se nudi nekoliko ponuđenih opcija za pronalazak proizvoda, uključujući i skeniranje linijskog koda (*eng. barcode*). Njegova implementacija biti će objašnjena u drugom dijelu ovog poglavlja. Za omogućavanje pretraživanja proizvoda i iščitavanja kalorija i makronutrijenta iz njih, bilo je neophodno pronaći bazu podataka koja sadrži širok raspon dostupnih proizvoda i istovremeno nudi pouzdane informacije. Pristupanje tim podacima i proizvodima objašnjeno je u trećem dijelu ovog poglavlja, dok je spremanje istih podataka opisano u četvrtom dijelu poglavlja. U svrhu definiranja potrebne dnevne količine kalorija i pravilne raspodjele makronutrijenata, korisnik će imati mogućnost postavljanja svojih ciljeva, koji će ovisiti o nekoliko parametara. O njima će biti pisano u zadnjem dijelu ovog poglavlja.

5.1. Postavljanje korisnika

Prilikom planiranja izrade aplikacije javila se potreba za stvaranjem personalizirane aplikacije umjesto generičke. Upravo zbog personalizacije postoji mehanizam stvaranja korisnika. U slučaju nemogućnosti stvaranja korisnika, aplikacija bi se mogla koristiti od strane više korisnika odjednom. Tu bi se stvorio problem, s obzirom da se ne bi moglo raspoznati koje proizvode je unio koji korisnik. Za rješavanje tog problema i omogućivanja izrade personalizirane aplikacije, korišten je Firebase, o kojemu je više bilo pisano u prethodnom poglavlju. Firebase omogućuje jednostavno postavljanje korisnika pri čemu je za registraciju i prijavu korisnika potrebna samo adresa elektroničke pošte i zaporka. Kako bi se korisnik uspješno registrirao potrebno je imati valjanu adresu elektroničke pošte i zaporku koja sadrži najmanje šest simbola pri čemu korisnik, naravno, ne smije ostaviti ova dva polja za ispunu praznima. Ukoliko korisnik ne ispuni neke od ovih uvjeta, biti će mu prikazana greška u kojoj je rečeno što nije ispravno odrađeno.

Firebase mehanizam za ovjeru autentičnosti u sebi sadrži provjeru valjanosti adrese elektroničke pošte i zaporke, kao i provjeru jesu li navedena polja popunjena. Iz toga razloga potrebno je samo provjeriti postoji li rezultat registracije i prijave, odnosno jesu li one uspješne ili je prilikom unosa nastala greška. U slučaju nastanka greške, ona se sprema u objekt

errorAlert koji će tu grešku prikazati na ekranu prilikom pokušaja registracije ili prijave. U suprotnom će doći do uspješne registracije ili prijave korisnika. Ove funkcionalnosti prikazane su na slici 5.1. koja sadrži pogledni model *(eng. view model)* koji je zadužen za sve funkcionalnosti prilikom registracije ili prijave korisnika.

```
class AuthViewModel: ObservableObject {
    @Published var authModel: AuthModel = .init()
    let auth = Auth.auth()
    @Published var errorAlert = ErrorAlert()

    var isSignedIn: Bool {
        return auth.currentUser != nil
    }

    func signIn() {
        auth.signIn(withEmail: authModel.email, password: authModel.password) { [weak self] (result, error) in
            guard result != nil, error == nil else {
                self?.errorAlert.showAlert = true
                self?.errorAlert.errorMessage = error?.localizedDescription ?? "Error occured"
                return
            }

            DispatchQueue.main.async {
                self?.authModel.signedIn = true
            }
        }
    }

    func signUp() {
        auth.createUser(withEmail: authModel.email, password: authModel.password) { [weak self] (result, error) in
            guard result != nil, error == nil else {
                self?.errorAlert.showAlert = true
                self?.errorAlert.errorMessage = error?.localizedDescription ?? "Error occured"
                return
            }

            DispatchQueue.main.async {
                self?.authModel.signedIn = true
            }
        }
    }
```

Slika 5.1. Prikaz AuthViewModel klase

Na slici 5.2. je prikazano korištenje ove klase prilikom prikaza polja za unos adrese elektroničke pošte i zaporke. Slika 5.2. sadrži pozive funkcija koje izrađuju sve što je prikazano na ekranu: polja za unos adrese elektroničke pošte, zaporke i dugmad za prijavu/registraciju korisnika. Na slici 5.3. te funkcije su prikazane u cijelosti.

```
struct SignView: View {
    @EnvironmentObject var viewModel: AuthViewModel
    @State private var type: SignType

    init(type: SignType) {
        self.type = type
    }

    var body: some View {
        VStack {
            getTextField(text: "Email address", saveIn: $viewModel.authModel.email)
            getTextField(text: "Password", saveIn: $viewModel.authModel.password, isSecure: true)

            Button {
                switch type {
                case .signIn:
                    viewModel.signIn()
                case .signUp:
                    viewModel.signUp()
                }
            } label: {
                self.getButtonLabel()
            }
            .padding()

            self.getNavigationLink()
        }
        .padding()
        .navigationTitle(type == .signIn ? "Sign in" : "Create account")
        .alert(isPresented: $viewModel.errorAlert.showAlert) {
            CustomAlert.showErrorAlert(message: viewModel.errorAlert.errorMessage)
        }
        Spacer()
    }
```

Slika 5.2. Prikaz korištenja AuthViewModel-a prilikom izrade SignView prikaza (*eng. view*)

```
    @ViewBuilder
    private func getTextField(text: String, saveIn parameter: Binding<String>, isSecure: Bool? = false) -> some View {
        if let isSecure,
           isSecure {
            SecureField(text, text: parameter).authTextFieldModifier()
        } else {
            TextField(text, text: parameter).authTextFieldModifier()
        }
    }

    @ViewBuilder
    private func getButtonLabel() -> some View {
        switch self.type {
        case .signIn:
            Text("Sign in").authButtonModifier()
        case .signUp:
            Text("Create account").authButtonModifier()
        }

    }

    @ViewBuilder
    private func getNavigationLink() -> some View {
        if type == .signIn {
            NavigationLink("Create account", destination: SignView(type: .signUp))
                .padding()
        }
    }
}
```

Slika 5.3. Prikaz ViewBuilder funkcija koje stvaraju SignView prikaz

Firebase konzola za autentifikaciju u sebi pohranjuje i prikazuje korisnike koji su uspješno registrirani. Imajući to na umu, nakon uspješne registracije korisnika, to postaje vidljivo ne samo u aplikaciji, već i u konzoli. U konzoli se također prikazuje adresa elektroničke pošte svih korisnika, kao i datum kreiranja računa, datum zadnje prijave i korisnikov jedinstveni ključ za identifikaciju. Firebase konzola prikazana je na slici 5.4.

Slika 5.4. Prikaz Firebase konzole za autentifikaciju

5.2. Skeniranje linijskog koda proizvoda

Budući da je većinu namirnica danas moguće kupiti u trgovinama, svaka ta namirnica mora na sebi sadržavati linijski kod. Pomoću njega skener na blagajni prepoznaje o kojemu se proizvodu radi i koliko košta. Isto je tako pomoću AVFoundation i CodeScanner okvira, o kojima je više bilo pisano u prethodnom poglavlju, moguće napraviti skener na Apple uređaju. On će biti u mogućnosti prepoznati linijski kod nekog proizvoda i dohvatiti podatke o tom proizvodu. Podaci koji su ovdje potrebni su broj kalorija koje sadrži pojedini proizvod kao i njegovi makronutrijenti - ugljikohidrati, proteini i masti. Kada korisnik klikne na dugme za skeniranja linijskog koda u aplikaciji, ona prvo od korisnika zahtijeva mogućnost pristupa kameri. Ukoliko korisnik dopusti da se kamera koristi za potrebe skeniranja koda, tada se provjerava postoji li uopće kamera na uređaju, te ako postoji, poziva se CodeScanner pogled. On otvara kameru i prikazuje sadržaj kamere na zaslonu. Na slici 5.5. vidljiv je prikaz SearchView pogleda na kojemu se nalazi dugme za pristup skeneru proizvoda.

```
struct SearchView: View {
    @StateObject var viewModel: SearchViewModel = .init()
    @Environment(\.dismiss) var dismiss
    private let date: Date
    private let diaryKey: MealKey
    @State var barcode = ""

    init(date: Date, diaryKey: MealKey) {
        self.date = date
        self.diaryKey = diaryKey
    }

    var body: some View {
        VStack(alignment: .leading) {
            HStack {
                createTextField()
                Button {
                    viewModel.isShowingScanner = true
                } label: {
                    Image(systemName: "barcode.viewfinder")
                        .font(.system(size: 25))
                }
                .padding(.trailing)

                Button {
                    viewModel.isShowingManualBarcodeInput = true
                } label: {
                    Image(systemName: "square.and.pencil")
                        .font(.system(size: 25))
                }
                .padding(.trailing)
            }
```

Slika 5.5. Prikaz SearchView pogleda

```
import CodeScanner

class SearchViewModel: ObservableObject {
    private let repository: FoodRepository

    @Published var isShowingScanner: Bool = false
    @Published var isShowingManualBarcodeInput: Bool = false
    @Published var searchedFood: String = ""
    @Published var products = [Product]()
    @Published var isLoading: Bool = false
    @Published var isShowingFoodView: Bool = false
    @Published var listHeader: SearchType = .recents
    @Published var errorAlert = ErrorAlert()

    var productToAdd: Product?

    private var cancellables: Set<AnyCancellable> = .init()

    init(){
        repository = FoodRepositoryImplementation()
    }
```

Slika 5.6. Prikaz SearchViewModel klase

SearchView pogled u sebi sadrži dugme za otvaranje CodeScanner pogleda, s ikonicom imena *"barcode.viewfinder"*. Kao što je vidljivo na slici 5.5., kada se dugme pritisne, *viewModel.isShowingScanner* vrijednost se postavlja na istinitu (*eng. true*). U tom slučaju *viewModel* predstavlja objekt klase SearchViewModel koja je prikazana na slici 5.6. Ova varijabla je bitna jer na taj način aplikacija prikazuje CodeScanner prozor (*eng. sheet*) ukoliko je njena vrijednost istinita. CodeScanner prozor vidljiv je na slici 5.7.

```
.sheet(isPresented: $viewModel.isShowingScanner) {
    let discoverySession = AVCaptureDevice.DiscoverySession(deviceTypes: [.builtInWideAngleCamera],
                                                            mediaType: .video,
                                                            position: .unspecified)

    if let mainCameraDevice = discoverySession.devices.first(where: { $0.position == .back }) {
        VStack {
            let videoCaptureDevice = mainCameraDevice
            CodeScannerView(codeTypes: [.ean13],
                            simulatedData: "4014400929058",
                            videoCaptureDevice: videoCaptureDevice,
                            completion: viewModel.handleScan)
        }
        .loadingOverlay(isLoading: $viewModel.isLoading)
    } else {
        Text("Main camera not found.")
    }
}
```

Slika 5.7. Prikaz CodeScannerView prozora

5.3. Dohvaćanje proizvoda preko API poziva

U prijašnjem poglavlju spomenuta je Open Food Facts online baza podataka, gdje je spomenuto kako je ona nastala, što sadrži i koja je njena svrha. Kao nastavak na prošlo poglavlje ovdje je prikazano njeno korištenje preko API poziva.

U ovoj aplikaciji ova baza podataka bila je korištena prilikom korisnikove pretrage proizvoda koje unosi u dnevnik prehrane. Korisnik može pretraživati proizvode na tri načina:

- Pretraga putem ključnih riječi u imenu proizvoda
- Pretraga putem skeniranja linijskog koda
- Pretraga putem upisa broja linijskog koda

Pretraživanje proizvoda započinje u SearchView pogledu, pri čemu se poziva njegov pogledni model, SearchViewModel. On se koristi za definiranje svih funkcionalnosti SearchView-a. Na slici 5.8. može se vidjeti ViewBuilder funkcija, koja poziva svoj

pogledni model i metodu *searchFood*. Na slici 5.9. prikazana je *searchFood* metoda iz SearchViewModel klase.

```
struct SearchView: View {

    @ViewBuilder
    private func createTextField() -> some View {
        HStack {
            Image(systemName: "magnifyingglass")
            TextField(
                "Search for food",
                text: $viewModel.searchedFood,
                onCommit: viewModel.searchFood
            )

        }
        .disableAutocorrection(true)
        .padding()
        .frame(maxWidth: .infinity, maxHeight: 35, alignment: .center)
        .overlay(
                RoundedRectangle(cornerRadius: 8)
                    .stroke(Color(.black), lineWidth: 1)
            )
        .keyboardType(.webSearch)
        .padding()
    }
}
```

Slika 5.8. Prikaz SearchView pogleda

```
class SearchViewModel: ObservableObject {

    private var cancellables: Set<AnyCancellable> = .init()

    init(){
        repository = FoodRepositoryImplementation()
    }

    func onAppear() {
        listHeader = .recents
        ProductsManager.getSavedProducts(key: .productsKey) { [weak self] products, _, error in
            guard error == nil else {
                self?.errorAlert.showAlert = true
                self?.errorAlert.errorMessage = error?.localizedDescription ?? "Search error"
                return
            }
            self?.products = products
        }
    }

    func searchFood() {
        isLoading = true
        listHeader = .searched
        repository
            .getFoods(for: searchedFood)
            .map { $0.products.filter{ $0.doesExist() }}
            .receive(on: RunLoop.main)
            .sink(
                receiveCompletion: { [weak self] completion in
                    self?.handleCompletion(completion: completion)
                    self?.isLoading = false
                },
                receiveValue: { [weak self] foods in
                    self?.products = foods
                }
            )
            .store(in: &cancellables)
    }
```

Slika 5.9. Prikaz SearchViewModel klase i *searchFood* metode

Na slici 5.9. također je vidljivo kreiranje *repository* objekta, na kojemu se poziva metoda *getFoods(for: searchedFood).* Metoda *getFoods* kao rezultat vraća listu proizvoda koja je potrebna kada korisnik pretražuje proizvod po imenu. Korisnik tada iz liste proizvoda odabire onaj koji mu najviše odgovara. Na slici 5.10. vidljive su *handleScan*, *getFood* i *handleCompletion* metode koje se koriste prilikom korisnikovog skeniranja linijskog koda s nekog proizvoda ili pri ručnom unosu broja linijskog koda. Poziv *handleScan* metode vidljiv je na slici 5.7.

```
class SearchViewModel: ObservableObject {
    func handleScan(result: Result<ScanResult, ScanError>?) {
        isLoading = true
        if let result {
            switch result {
            case .success(let scanResult):
                self.getFood(for: scanResult.string)
            case .failure(let scanError):
                errorAlert.showAlert = true
                errorAlert.errorMessage = "Scanning failed: \(scanError.localizedDescription)"

            }
        }
    }

    func getFood(for filter: String) {
            repository.getFood(for: filter)
                .receive(on: DispatchQueue.main)
                .sink(receiveCompletion: handleCompletion, receiveValue: { [weak self] fetchedFood in
                    self?.productToAdd = fetchedFood.product
                    self?.isLoading = false
                    self?.isShowingScanner = false
                    self?.isShowingFoodView = true

                })
                .store(in: &self.cancellables)
    }

    private func handleCompletion(completion: Subscribers.Completion<AppError>) {
        switch completion {
        case .finished:
            print("Fetching food completed")
        case .failure(let error):
            errorAlert.showAlert = true
            errorAlert.errorMessage = "Fetching food failed with error: \(error.localizedDescription)"
        }
    }
}
```

Slika 5.10. Prikaz metoda korištenih prilikom unosa ili skeniranja broja linijskog koda

Repository klasa, zajedno s RestManager klasom definira metode za dohvaćanje podataka koje se u JSON obliku nalaze na određenim URL adresama. Open Food Facts baza podataka se poziva pomoću tih URL adresa na sljedeći način:

- Ako se radi o nizu slova (*eng. String*), tada se poziva slijedeća URL adresa: *"https://world.openfoodfacts.org/cgi/search.pl?search_terms=\(safeFilter)&search_simple=1&action=process&json=1"*. U tom slučaju, *\(safeFilter)* predstavlja niz slova koje je korisnik unio kako bi pronašao proizvod koji ga zanima.
- Ako se radi o broju linijskog koda, tada se poziva slijedeća URL adresa: "*https://world.openfoodfacts.org/api/v0/product//(barcode).json"*. U ovom slučaju, *\(barcode)* predstavlja broj linijskog koda kojeg je korisnik skenirao pomoću kamere ili ručno upisao.

Repository protokol i njegova implementacija prikazane su na slici 5.11., dok je RestManager klasa, zajedno s *parseFromURL* metodom, prikazana su na slici 5.12.

```
protocol FoodRepository {
    func getFoods(for filter: String) -> AnyPublisher<Foods, AppError>
    func getFood(for filter: String) -> AnyPublisher<Food, AppError>
}

class FoodRepositoryImplementation: FoodRepository {
    func getFoods(for filter: String) -> AnyPublisher<Foods, AppError> {
        return RestManager.parseFromURL(for: filter)
    }

    func getFood(for filter: String) -> AnyPublisher<Food, AppError> {
        return RestManager.parseFromURL(for: filter)
    }
}
```

Slika 5.11. Prikaz Repository protokola i njegove implementacije

```
class RestManager {

    static func parseFromURL<T: Decodable>(for filter: String) -> AnyPublisher<T, AppError> {
        let safeFilter = filter.toURLSafe()
        var urlString = ""
        if let barcode = Int(filter) {
            urlString = "https://world.openfoodfacts.org/api/v0/product/\(barcode).json"
        } else {
            urlString =
                "https://world.openfoodfacts.org/cgi/search
                .pl?search_terms=\(safeFilter)&search_simple=1&action=process&json=1&page_size=25"
        }

        guard let url = URL(string: urlString) else {
            return Fail(error: .decodingError(
                NSError(domain: "RestManager", code: 0, userInfo: [NSLocalizedDescriptionKey: "Invalid URL"]
                    ))).eraseToAnyPublisher()
        }

        return SerializationManager.parseFromURL(for: url)
            .eraseToAnyPublisher()
    }
}
```

Slika 5.12. Prikaz RestManager klase i metode *parseFromURL*

Na slici 5.12. može se vidjeti da metoda *parseFromURL* kao rezultat vraća pozivanje istoimene metode iz klase SerializationManager. Ta metoda dohvaća sve JSON objekte dobivene pozivanjem predanog URL-a. Nakon toga ih dekodira iz JSON objekata u model koji mu se predaje. Ukoliko se dogodila greška prilikom dohvaćanja JSON objekata i njegovih dekodiranja, tada ova metoda za rezultat vraća *AppError* enumeraciju. Njom se definira greška nastala prilikom dohvaćanja podataka iz baze. Model u ovom slučaju, kao što je definirano u FoodRepository protokolu (Slika 5.11.), može biti tipa Food ili Foods. Na slici 5.13. prikazana je SerializationManager klasa koja u sebi ima definiranu *parseFromURL* metodu. Na slici 5.14. prikazana je AppError enumeracija.

```swift
class SerializationManager {

    static func parseFromURL<T: Decodable>(for url: URL) -> AnyPublisher<T, AppError> {
        return URLSession.shared.dataTaskPublisher(for: url)
            .mapError { AppError.networkError($0) }
            .map { $0.data }
            .decode(type: T.self, decoder: JSONDecoder())
            .mapError { AppError.decodingError($0) }
            .eraseToAnyPublisher()
    }
}
```

Slika 5.13. Prikaz SerializationManager klase i parseFromURL metode

```swift
enum AppError: Error {
    case generic
    case decodingError(Error)
    case networkError(Error)
}

extension AppError: LocalizedError {
    var errorDescription: String? {
        switch self {

        case .generic:
            return "An unknown error occured"
        case .decodingError(let error):
            return error.localizedDescription
        case .networkError(let error):
            return error.localizedDescription
        }
    }
}
```

Slika 5.14. Prikaz AppError enumeracije

Modeli Food i Foods složeni su od više manjih struktura, stoga su na slici 5.15., slici 5.16. i slici 5.17. prikazani modeli i strukture od kojih su sastavljeni.

```
struct Food: Codable, Hashable {
    var product: Product

   init() {
       self.product = Product()
    }
}

struct Foods: Codable, Hashable {
    var products: [Product]

   init() {
       self.products = [Product]()
    }
}
```

Slika 5.15. Prikaz modela Food i Foods

```
struct Product: Codable, Hashable {
    var id: String
    var productName: String?
    var nutriments: Nutriments

    private enum CodingKeys : String, CodingKey {
        case productName = "product_name", id = "_id"
        case nutriments, servingSize
    }
    var servingSize: String?

    init() {
        self.id = "0000000000000"
        self.productName = "Unknown"
        self.nutriments = Nutriments()
    }
}
```

Slika 5.16. Prikaz Product strukture

```
struct Nutriments: Codable, Hashable {

    var carbohydrates: Double?
    var proteins: Double?
    var fat: Double?

    private enum CodingKeys : String, CodingKey {
        case carbohydrates, proteins, fat
    }

    init() {
        self.carbohydrates = 0.0
        self.proteins = 0.0
        self.fat = 0.0
    }
```

Slika 5.17. Prikaz Nutriments strukture

5.4. Baza podataka

Nakon što bi korisnik odradio API poziv, dohvatio proizvode, odabrao koji i koliko tog proizvoda želi unijeti u svoj dnevnik prehrane, poziva se mehanizam spremanja odabranog proizvoda u bazu podataka. Kao što je već ranije spomenuto, za potrebe spremanja i dohvaćanja spremljenih proizvoda, korištena je Firebase baza podataka u stvarnom vremenu. Važno je napomenuti da se ti proizvodi dohvaćaju iz baze podataka svaki put kada je prikazan zaslon dnevnika prehrane ili zaslon za pretraživanje proizvoda. Na zaslonu dnevnika prehrane korisnik može vidjeti sve proizvode koje je unosio za određeni obrok u danu. Prilikom ulaska na zaslon za pretraživanje proizvoda, korisnik može vidjeti listu proizvoda koji su bili prethodno pretraživani. Budući da se proizvodi dohvaćaju prilikom ulaska na zaslon dnevnika prehrane i na zaslon za pretraživanje proizvoda, to dohvaćanje proizvoda postavljeno je u metode *onAppear* na oba zaslona.

Za potrebe spremanja i dohvaćanja spremljenih proizvoda, koristi se klasa ProductsManager, koja u sebi posjeduje metode za obavljanje spomenutih funkcionalnosti. Metoda za spremanje proizvoda naziva se *setData*, dok se metoda za dohvaćanje proizvoda naziva *getSavedProducts.* Na slici 5.18. prikazana je klasa SearchViewModel i metoda *onAppear,* dok je na slici 5.19. vidljiv prikaz DiaryViewModel klase i metode *onAppear*. Na slici 5.18. i slici 5.19. također je vidljivo pozivanje *getSavedProducts* metode.

```
class SearchViewModel: ObservableObject {
    private let repository: FoodRepository

    @Published var isShowingScanner: Bool = false
    @Published var isShowingManualBarcodeInput: Bool = false
    @Published var searchedFood: String = ""
    @Published var products = [Product]()
    @Published var isLoading: Bool = false
    @Published var isShowingFoodView: Bool = false
    @Published var listHeader: SearchType = .recents
    @Published var errorAlert = ErrorAlert()

    var productToAdd: Product?

    private var cancellables: Set<AnyCancellable> = .init()

    init(){
        repository = FoodRepositoryImplementation()
    }

    func onAppear() {
        listHeader = .recents
        ProductsManager.getSavedProducts(key: .productsKey) { [weak self] products, _, error in
            guard error == nil else {
                self?.errorAlert.showAlert = true
                self?.errorAlert.errorMessage = error?.localizedDescription ?? "Search error"
                return
            }
            self?.products = products
        }
    }
```

Slika 5.18. Prikaz klase SearchViewModel i metode *onAppear*

```
class DiaryViewModel: ObservableObject {
    let repository: FoodRepository
    private var cancellables = Set<AnyCancellable>()

    @Published var errorAlert = ErrorAlert()
    @Published var diaryModel = DiaryModel()
    @Published var isShowingScanner: Bool = false
    @Published var isShowingSearch: Bool = false
    @Published var isLoading: Bool = true

    var tappedProduct: Product?

    init() {
        self.repository = FoodRepositoryImplementation()
    }

    func onAppear(forDate date: Date) {
        loadProducts(forDate: date) { [weak self] in
            self?.getTotalDailyCalories()
            self?.isLoading = false
        }
    }

    private func loadProducts(forDate date: Date, completion: @escaping (() -> Void)){
        var remainingOperations = MealKey.allCases.count
        for key in MealKey.allCases {
            if key != .productsKey {

                ProductsManager.getSavedProducts(for: date, key: key) { [weak self] products, mealType, error in
                    guard error == nil else {
                        self?.errorAlert.showAlert = true
                        self?.errorAlert.errorMessage = error?.localizedDescription ?? "Error occured"
                        return
                    }
                    var meal = self?.getMeal(forType: key.getMealType())

                    if let mealType { meal = self?.getMeal(forType: mealType) }
                    meal?.products = products
                    meal?.calculateCalories()
```

Slika 5.19. Prikaz klase DiaryViewModel i metode *onAppear*

Na slici 5.18. i slici 5.19. osjenčano je mjesto na kojemu se dohvaćani proizvodi postavljaju na proizvode koji će kasnije biti prikazani na zaslonima. Kod zaslona za pretraživanje, ti proizvodi se spremaju u lokalnu varijablu *products*, s obzirom da taj zaslon prikazuje samo proizvode koji su nedavno pretraživani, odnosno pokazuje samo jednu listu. Na zaslonu dnevnika prehrane ti proizvodi spremaju u objekt *meal*, koji u sebi sadrži varijablu *products*. Objekt meal u ovom slučaju predstavlja svaki obrok (doručak, ručak, večera, užina), pri čemu se taj obrok ažurira s novim proizvodima na poziv metode *onAppear*. Ovisno koliko ima obroka toliko se i lista prikazuje, u ovom slučaju to su četiri liste za svaki obrok.

```
enum DatabaseError: Error {
    case invalidUserID (String)
    case missingKey (String)
    case convertingError (String)
}

extension DatabaseError: LocalizedError {
    var errorDescription: String? {
        switch self {
        case .invalidUserID(let message):
            return message
        case .missingKey(let message):
            return message
        case .convertingError(let message):
            return message
        }
    }
}

private enum Constants {
    static let data = "food data"
    static let users = "users"
    static let productsKey = "products"
}

class ProductsManager {
    static let database = Database.database().reference()
    static let userId = AuthService.shared.getUserId()
    static let productsRef = database.child(Constants.users)
```

Slika 5.20. Prikaz DatabaseError enumeracije i klase ProductsManager

Za lakše razumijevanje daljnjeg koda, na slici 5.20. prikazana je enumeracija DatabaseError, koja služi za jednostavno objašnjenje greške, ukoliko aplikacija uopće dođe do te greške. Primjeri greški do kojih u ovom slučaju može doći su:

- *invalidUserID* - ako korisnikov jedinstveni ključ za identifikaciju nije pronađen
- *missingKey* - ako ključ za spremanje u bazu podataka nije pronađen
- *convertingError* - ako konvertiranje podataka iz JSON oblika u lokalni model ili obrnuto nije moguće

U nastavku će biti prikazana metoda *getSavedProducts* čije je pozivanje u klasama SearchViewModel i DiaryViewModel vidljivo na slici 5.18. i slici 5.19. Budući da bi ova metoda sama po sebi bila dugačka, u njoj se također pozivaju druge, manje metode, radi smanjenja složenosti svake od njih. Na slici 5.21. prikazana je metoda *getSavedProducts*, dok se na slici 5.22. prikazuje metoda *fetchProducts*.

```
    static func getSavedProducts(for date: Date? = nil,
                                 key: MealKey,
                                 completion: @escaping ([Product], MealType?, Error?) -> Void) {
        var selectedDate = Self.getCurrentDate()
        if let date {
            selectedDate = Self.getCurrentDate(from: date)
        }

        guard let userId else {
            let error = DatabaseError.invalidUserID("Error while fetching user ID")
            completion([], nil, error)
            return
        }

        fetchProducts(for: selectedDate, key: key, userId: userId, completion: completion)
    }
}
```

Slika 5.21. Prikaz metode *getSavedProducts*

```
private static func fetchProducts(for selectedDate: String,
                                  key: MealKey,
                                  userId: String,
                                  completion: @escaping ([Product], MealType?, Error?) -> Void) {

        let dates = productsRef.child(userId).child(Constants.data)

        dates.observeSingleEvent(of: .value) { datesSnap in
            handleFetchingProducts(datesSnap: datesSnap, key: key, selectedDate: selectedDate, completion:
                completion)
        }
    }
```

Slika 5.22. Prikaz metode *fetchProducts* koju poziva metoda *getSavedProducts*

Metoda *getSavedProducts* kao argumente prima:

- date - datum za koji korisnik želi dobiti spremljene proizvode
- key - ključ pod kojim je spremljen taj proizvod. U ovom slučaju taj ključ može biti jedan od četiri obroka (doručak, ručak, večera, užina) ili ključ pod kojim su spremljeni nedavno pretraživani proizvodi
- completion - funkcija u kojoj se spremljeni proizvodi postavljaju u varijablu proizvoda u klasama DiaryViewModel i SearchViewModel - vidljivo na slici 5.18. i slici 5.19.

Metoda također provjerava postoji li korisnikov jedinstveni ključ za identifikaciju - *userId.* U toj metodi se također poziva metoda *fetchProducts.* Ona dohvaća bazu podataka, točnije putanju na kojoj su prikazani svi datumi pod kojima je spremljen barem jedan proizvod. Nakon toga metoda *handleFetchingProducts* u sebi prolazi kroz svaki datum, dok ne dođe do onog datuma kojeg je korisnik postavio u aplikaciji i do onog ključa pod kojim je spremljen određeni proizvod. Ukoliko su proizvodi pronađeni pod određenim ključem ili pod određenim datumom zajedno s određenim ključem, tada se proizvodi šalju putem *completion* funkcije u

DiaryViewModel ili SearchViewModel klase. U slučaju nastanka greške, ona se također šalje putem *completion* funkcije u navedene klase, a zatim se prikazuje na zaslonu.
Na slici 5.23. vidljiv je prikaz metode *handleFetchingProducts*, dok se na slici 5.24. prikazuju metode *updateSearchedProducts* i *decodeProducts*.

```
    private static func handleFetchingProducts(datesSnap: DataSnapshot,
                                               key: MealKey, selectedDate: String,
                                               completion: @escaping ([Product], MealType?, Error?) -> Void) {
        var searchedProducts = [Product]()
        for dateSnap in datesSnap.children {
            guard let dateSnap = dateSnap as? DataSnapshot else {
                continue
            }

            if let productData = dateSnap.childSnapshot(forPath: key.rawValue).value as? [String: Any] {
                do {
                    let fetchedProducts = try decodeProducts(from: productData)

                    if key != .productsKey, dateSnap.key == selectedDate {
                        completion(fetchedProducts, key.getMealType(), nil)
                        return
                    } else if key == .productsKey {
                        updateSearchedProducts(fetchedProducts, &searchedProducts)
                        completion(searchedProducts, nil, nil)
                    } else {
                        completion([], nil, nil)
                    }

                } catch {
                    completion([], nil, error)
                }
            } else {
                if key != .productsKey {
                    completion([], nil, nil)
                }
            }
        }
    }
```

Slika 5.23. Prikaz metode *handleFetchingProducts* koju poziva metoda *fetchProducts*

```
private static func updateSearchedProducts(_ fetchedProducts: [Product], _ searchedProducts: inout [Product]) {
    for fetchedProduct in fetchedProducts {
        if let index = searchedProducts.firstIndex(where: { $0.id == fetchedProduct.id }) {
            searchedProducts[index] = fetchedProduct
        } else {
            searchedProducts.append(fetchedProduct)
        }
    }
}

private static func decodeProducts(from productData: [String: Any]) throws -> [Product] {
    guard let jsonString = productData[Constants.productsKey] as? String else {
        throw DatabaseError.missingKey("Missing key '\(Constants.productsKey)' in product data dictionary.")
    }

    guard let jsonData = jsonString.data(using: .utf8) else {
        throw DatabaseError.convertingError("Failed to convert JSON string to data")
    }

    return try JSONSerializer.decode([Product].self, from: jsonData)
}
```

Slika 5.24. Prikaz metoda *updateSearchedProducts* i *decodeProducts* koje poziva metoda *handleFetchingProducts*

Do sada je bilo prikazano samo dohvaćanje podataka iz baze, no kako bi se ti podaci mogli dohvatiti potrebno ih je prvo spremiti u bazu. To spremanje u bazu, kao što je već spomenuto

na početku ovo dijela, se odvija nakon što korisnik odabere proizvod koji želi spremiti u dnevnik prehrane. Zaslon na kojemu korisnik odabere proizvod naziva se FoodView, a kada korisnik klikne na dugme za spremanje proizvoda, na tom zaslonu se poziva metoda *updateProduct* ili *updateDiaryProduct*. Poziv metoda *updateProduct* i *updateDiaryProduct* prikazan je na slici 5.25.

```
struct FoodView: View {
    var body: some View {
                ToolbarItem(placement: .navigationBarTrailing) {
                    Button {
                        switch key {
                        case .breakfastKey, .lunchKey, .dinnerKey, .snacksKey:
                            ProductsManager.updateDiaryProduct(for: date,product: product, key: key, at: index) {
                                error in
                                guard error == nil else {
                                    self.errorAlert.showAlert = true
                                    self.errorAlert.errorMessage = error?.localizedDescription ?? "FoodView error"
                                    dismiss()
                                    return
                                }
                                dismiss()
                            }

                        case .productsKey:
                            ProductsManager.updateProduct(for: date, product: product, key: key) { error in
                                guard error == nil else {
                                    self.errorAlert.showAlert = true
                                    self.errorAlert.errorMessage = error?.localizedDescription ?? "FoodView error"
                                    dismiss()
                                    return
                                }
                                dismiss()
                            }
                            if let diaryKey {
                                ProductsManager.updateProduct(for: date, product: product, key: diaryKey) { error in
                                    guard error == nil else {
                                        self.errorAlert.showAlert = true
                                        self.errorAlert.errorMessage = error?.localizedDescription ?? "FoodView error"
                                        dismiss()
                                        return
                                    }
                                    dismiss()
                                }
                            }
                        }
```

Slika 5.25. Poziv metoda *updateProduct* i *updateDiaryProduct* u klasi FoodView

Metoda *updateProduct* poziva se kada korisnik sprema proizvod na zaslonu za pretraživanje proizvoda, dok se metoda *updateDiaryProduct* poziva prilikom promjene veličine porcije proizvoda na zaslonu dnevnika prehrane. Primjer promjene veličine porcije može biti promjena porcije zobenih pahuljica iz 100 u 200 grama. Na slici 5.26. prikazana je metoda *updateProduct*, dok je na slici 5.27. prikazana metoda *updateDiaryProduct*.

```
static func updateProduct(for date: Date? = nil,
                          product: Product,
                          key: MealKey,
                          completion: ((Error?) -> Void)? = nil) {

    getSavedProducts(for: date,key: key) { products, _, error in
        guard error == nil else { completion?(error); return }
        var updatedProducts = products

        if let index = updatedProducts.firstIndex(where: { $0.id == product.id }) {
            if key == .productsKey {
                updatedProducts[index] = product
            } else {
                updatedProducts.append(product)
            }
        }
        else {
            updatedProducts.append(product)
        }

        setData(for: date, products: updatedProducts, key: key)
        completion?(nil)
    }
}
```

Slika 5.26. Prikaz metode updateProduct

```
static func updateDiaryProduct(for date: Date? = nil,
                               product: Product,
                               key: MealKey,
                               at index: Int?,
                               completion: ((Error?) -> Void)? = nil) {
    guard let index else { return }

    getSavedProducts(for: date,key: key) { products, mealType, error in
        guard error == nil else {
            completion?(error)
            return
        }
        var updatedProducts = products
        if !products.isEmpty {
            if updatedProducts[index].id == product.id {
                updatedProducts[index] = product
            }
        }

        setData(for: date, products: updatedProducts, key: key)
        completion?(nil)
    }
}
```

Slika 5.27. Prikaz metode updateDiaryProduct

Na slici 5.26. i slici 5.27. može se vidjeti kako ove metode u sebi pozivaju već spomenutu metodu *getSavedProducts*. Ta metoda, kao rezultat, vraća spremljene proizvode koji se sada

ažuriraju, dodavanjem novog proizvoda ili promjenom već spremljenog proizvoda. Metode *updateProduct* i *updateDiaryProduct* također u sebi pozivaju *setData* metodu, koja te proizvode sprema u bazu podataka. Prikaz *setData* metode vidljiv je na slici 5.28.

```
static private func setData(for date: Date?,
                            products: [Product],
                            key: MealKey,
                            completion: ((Error?) -> Void)? = nil) {
    var selectedDate = ProductsManager.getCurrentDate()
    if let date = date {
        selectedDate = ProductsManager.getCurrentDate(from: date)
    }

    guard let userId else {
        let error = DatabaseError.invalidUserID("Error while fetching user ID")
        completion?(error)
        return
    }

    do {
        let encodedProducts = try JSONSerializer.encode(products)
        guard let jsonString = String(data: encodedProducts, encoding: .utf8) else {
            let error = DatabaseError.convertingError("Error: Failed to convert encoded products to JSON string")
            completion?(error)
            return
        }

        let productData: [String: Any] = [Constants.productsKey: jsonString]

        let path = productsRef.child(userId).child(Constants.data).child(selectedDate).child(key.rawValue)
        path.setValue(productData) { error, _ in
            if let error = error {
                completion?(error)
            }
        }

    } catch {
        completion?(error)
    }
}
```

Slika 5.28. Prikaz metode *setData*

Kao i u metodi *getSavedProducts,* u metodi *setData* se također provjerava postoji li korisnikov jedinstveni ključ za identifikaciju, budući da će se pod njim spremati proizvodi. Metoda također za argument prima određeni datum i ključ, pod kojim će proizvod biti spremljen. U metodi *getSavedProducts,* proizvodi se pokušavaju konvertirati iz JSON oblika u lokalni model proizvoda, dok se u ovoj metodi vrši konverzija iz lokalnog modela proizvoda u JSON oblik. Također kao kod *getSavedProducts* metode, u slučaju nastanka greške, ona se prikazuje na zaslonu s kojega je metoda pozvana.

5.5. Postavljanje cilja korisnika

Kao što je već navedeno u uvodu ovog poglavlja, korisniku će se prilikom korištenja dnevnika prehrane omogućiti postavljanje vlastitih ciljeva, bili oni dobivanje, gubitak ili održavanje kilaže. Kako bi korisniku bio prikazan broj dnevnih kalorija i raspodjela makronutrijenata, prvo mora popuniti nekoliko polja. Ta polja su:

- Dob
- Spol
- Težina
- Visina
- Razina aktivnosti
- Tjedni cilj

Prva četiri parametra su razumljiva sama po sebi, dok razina aktivnosti predstavlja koliko je korisnik aktivan u svakodnevnom životu. Korisniku je za to polje ponuđeno nekoliko odgovora:

- Neaktivan/sjedeći posao
- Lagani treninzi, 1-2 puta tjedno
- Srednje aktivan, 3-5 treninga tjedno
- Aktivan, 6-7 treninga tjedno
- Jako aktivan, 2 treninga tjedno

Tjedni cilj predstavlja što je korisnikov cilj u konačnici i koliko intenzivno želi napredovati. Za to polje je također ponuđeno nekoliko odgovora:

- Održavanje trenutne kilaže
- Gubitak 0,25kg tjelesne mase po tjednu
- Gubitak 0,5kg tjelesne mase po tjednu
- Dobitak 0,25kg tjelesne mase po tjednu
- Dobitak 0,5kg tjelesne mase po tjednu

Na slici 5.29. vidljiv je prikaz polja s odabirom pri čemu svaka enumeracija (enum) prikazuje jedno polje. Na slici 5.30. i slici 5.31. može se vidjeti model za ovaj pogled koji je nazvan Athlete.

```
enum Gender: String, CaseIterable, Codable {
    case man = "Man"
    case woman = "Woman"
}

enum ActivityLevel: String, CaseIterable, Codable {
    case sedentary = "Little or no excercise, office job"
    case lightlyActive = "Light excercise (1-2 days/week)"
    case moderatelyActive = "Moderate excercise (3-5 days/week)"
    case highlyActive = "Intense excercise (6-7 days/week)"
    case veryActive = "Very active (2x excercises/day)"
}

enum WeeklyGoal: String, CaseIterable, Codable {
    case maintain = "Maintain current weight"
    case lose25 = "Lose 0.25 kg per week"
    case lose5 = "Lose 0.5kg per week"
    case gain25 = "Gain 0.25kg per week"
    case gain5 = "Gain 0.5kg per week"
}
```

Slika 5.29. Prikaz polja s odabirom

```
struct Athlete {
    var basicInfo: BasicInfo
    var additionalInfo: AdditionalInfo

    init() {
        self.basicInfo = BasicInfo()
        self.additionalInfo = AdditionalInfo()
    }
```

Slika 5.30. Prikaz modela Athlete

Kao što je vidljivo na slici 5.30., model Athlete se sastoji od BasicInfo i AdditionalInfo varijabli, koje zapravo predstavljaju strukture od kojih je građena Athlete struktura. Na slici 5.31. prikazane su BasicInfo i AdditionalInfo strukture.

```
struct BasicInfo: Codable {
    var gender: Gender
    var age: String
    var weight: String
    var feet: String
    var inches: String
    var height: String

    init() {
        self.gender = .man
        self.age = ""
        self.weight = ""
        self.feet = ""
        self.inches = ""
        self.height = ""
    }
}

struct AdditionalInfo: Codable {
    var activityLevel: ActivityLevel
    var heightMesurment: HeightMesurment
    var weightMesurment: WeightMesurment
    var weeklyGoal: WeeklyGoal

    init() {
        self.activityLevel = .sedentary
        self.heightMesurment = .cm
        self.weightMesurment = .kg
        self.weeklyGoal = .maintain
    }
}
```

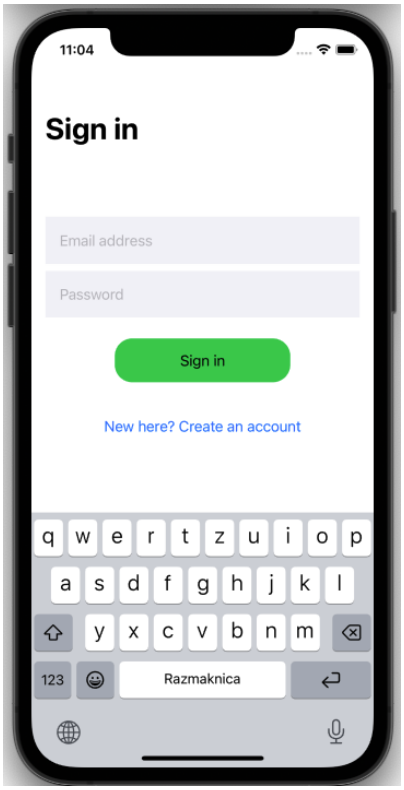
Slika 5.31. Prikaz BasicInfo i AdditionalInfo struktura

6. NAČIN RADA APLIKACIJE

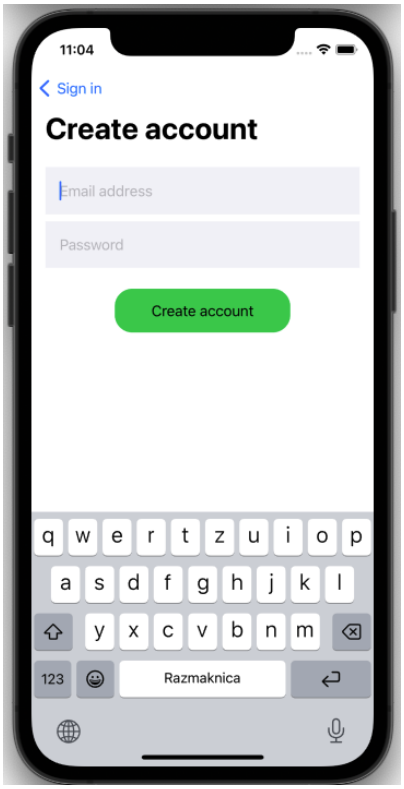
U ovome poglavlju detaljno je opisano korištenje mobilne aplikacije, prikazujući vizualno sve zaslone aplikacije, od zaslona za registraciju korisnika do pretraživanja, spremanja i prikaza unesenih jela i proizvoda, te na kraju odjave korisnika.

6.1. Korištenje aplikacije

Prilikom ulaska u aplikaciju korisniku se prikazuje početni zaslon koji predstavlja zaslon za prijavu korisnika. U slučaju da korisnik nema izrađen račun za pristup aplikaciji, njega može napraviti klikom na dugme *"New here? Create an account"*. Prilikom klika na to dugme, korisniku se otvara novi zaslon preko kojeg je omogućena izrada računa za prijavu. Na tom zaslonu je potrebno unijeti valjanu adresu elektroničke pošte i postaviti zaporku za izradu računa. Kao što je već ranije spomenuto, u slučaju unosa neispravnog formata adrese elektroničke pošte ili zaporke koja ima manje od šest znakova, korisniku se prikazuje greška. Dok ta greška ne bude ispravljena, korisnik ne može pristupiti aplikaciji. Prikaz zaslona za prijavu korisnika vidljiv je na slici 6.1., dok je na slici 6.2. vidljiv zaslon za registraciju korisnika.



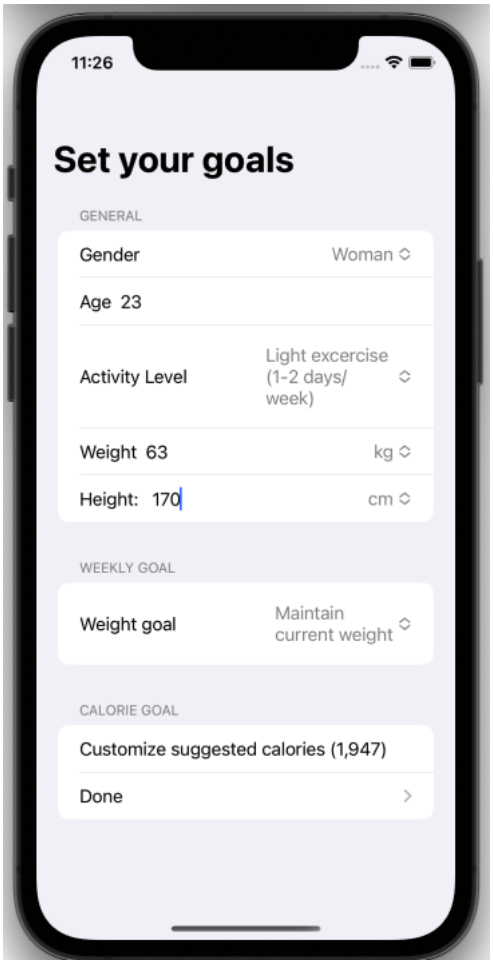
Slika 6.1. Prikaz zaslona za prijavu korisnika



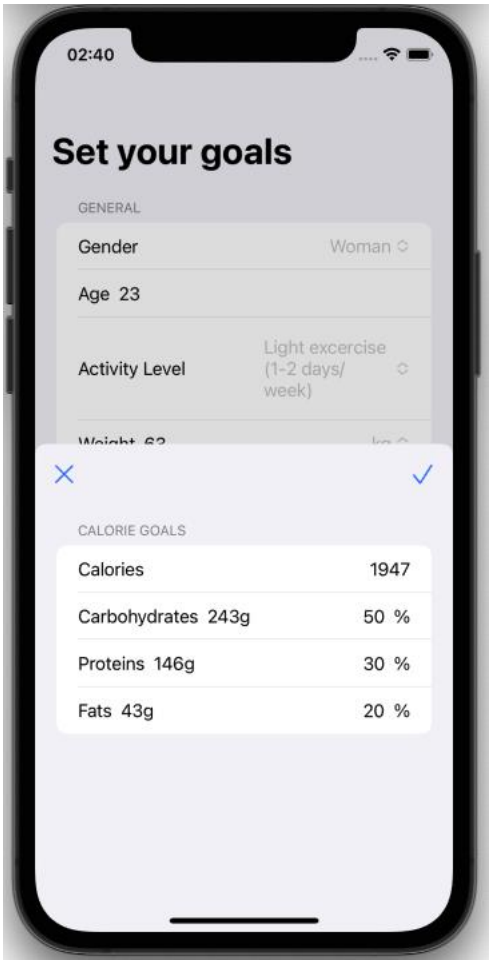
Slika 6.2. Prikaz zaslona za registraciju korisnika

Nakon unosa adrese elektroničke pošte ispravnog formata i ispravne zaporke na zaslonu za prijavu/registraciju korisnika, korisnik se uspješno prijavljuje u sustav. Adresa elektroničke pošte korisnika, zajedno sa zaporkom spremaju se u Firebase bazu podataka. Nakon uspješne prijave u sustav, korisniku se, ovisno o tome jesu li postavljeni ciljevi korisnika, može prikazati jedan od dva zaslona:

- Ukoliko ciljevi korisnika nisu postavljeni: Prikazuje se zaslon za postavljanje ciljeva
- Ukoliko su ciljevi korisnika postavljeni: Prikazuje se zaslon za vođenje dnevnika prehrane



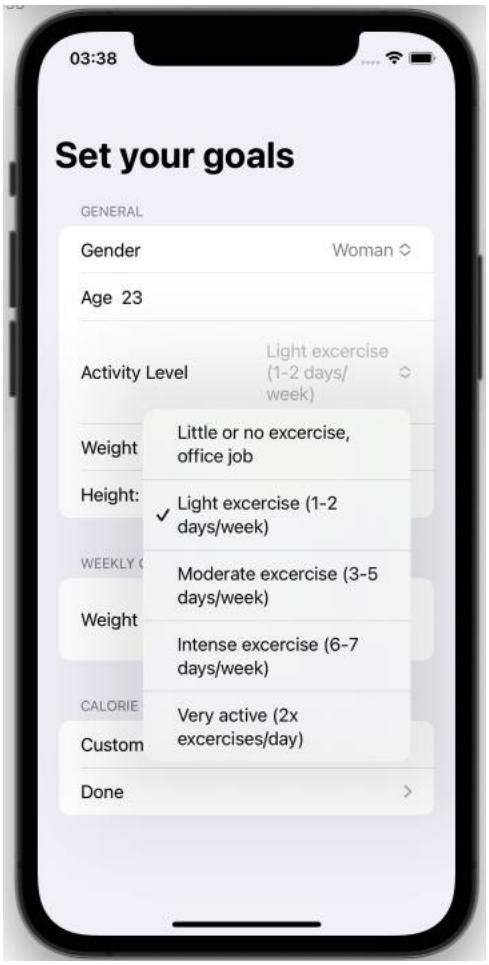
Slika 6.3. Prikaz zaslona za postavljanje ciljeva korisnika



Slika 6.4. Prikaz prozora za korekciju kalorija i makronutrijenata

Na slici 6.3. može se vidjeti zaslon za postavljanje ciljeva korisnika, pri čemu korisnik ispunjava informacije o spolu, godinama, razini aktivnosti, težini i visini. Težinu je moguće izraziti u kilogramima i funtama (*eng. pounds*), a visinu u centimetrima, kao i u stopama i inčima. Kao što je već napisano, postavljeno je pet različitih razina aktivnosti, od korisnika koji je neaktivan pa sve do iznimno aktivan. Kod tjednog cilja (*eng. weekly goal*) moguće je također izabrati pet različitih opcija - od gubitka 0,5kg tjedno do dobitka 0,5kg tjedno. Kada su sva polja unesena korisniku se prikazuje predložen broj kalorija.

Klikom na *"Customize suggested calories"* korisniku se otvara prozor vidljiv na slici 6.4., pri čemu je korisniku omogućeno uređivanje broj kalorija, kao i postotka makronutrijenata koje će unositi u jednome danu. Prilikom završetka unošenja informacija za sva polja, prikazuje se dugme *"Done"* koje korisnik pritišće kada je gotov s postavljanjem ciljeva. Na slici 6.5. vidljiv je prikaz svih razina aktivnosti, dok je na slici 6.6. vidljiv prikaz svih opcija za tjedni cilj.

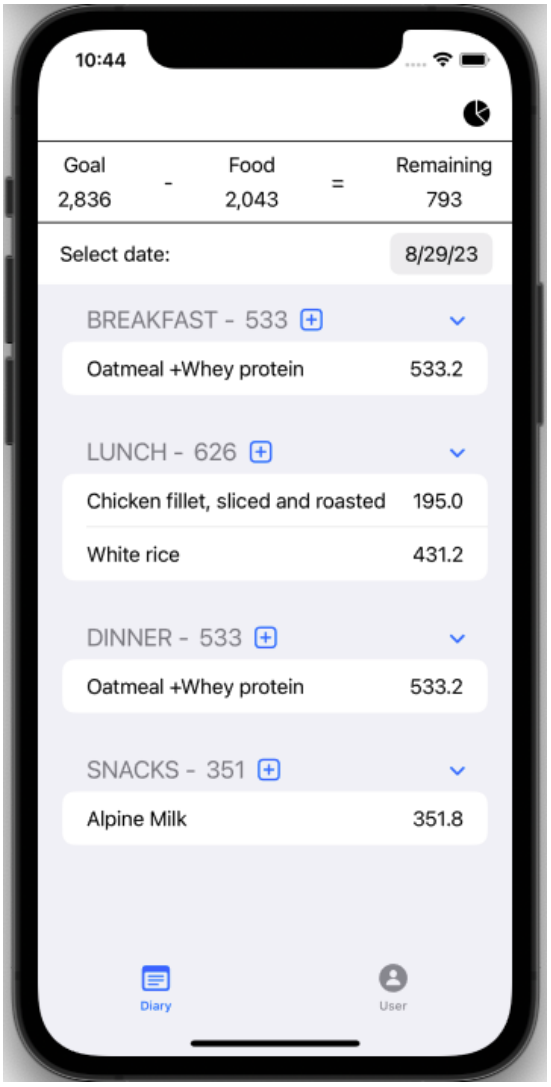


Slika 6.5. Prikaz svih razina aktivnosti

Slika 6.6. Prikaz svih opcija za tjedni cilj

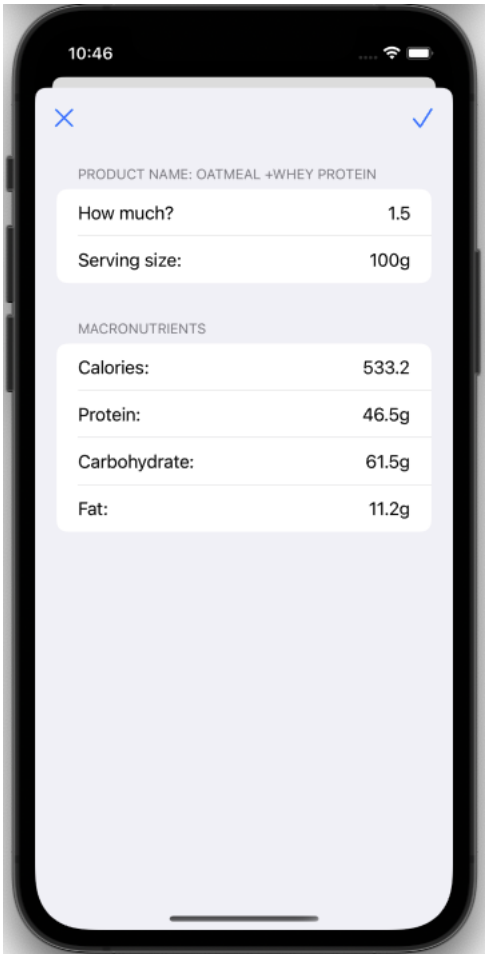
Nakon što je korisnik postavio ciljeve, prikazuje mu se zaslon za vođenje dnevnika prehrane. Ovaj zaslon bi se također prvi prikazao nakon prijave korisnika u aplikaciju, pod uvjetom da su ciljevi korisnika bili prethodno uneseni. Na ovome zaslonu korisnik može pristupiti velikoj većini ostalih zaslona. Veći dio zaslona dnevnika prehrane sastoji se od liste pojedinih obroka u danu, pri čemu je iznad svake liste prikazano njihovo ime - doručak, ručak, večer i užina. Broj desno od imena obroka predstavlja broj kalorija za taj obrok. Proizvodi koji su uneseni za određeni obrok su izlistani ispod imena tog obroka, a od svakog proizvoda vidljivo je njegovo ime i broj kalorija za porciju proizvoda koju je korisnik unio. Na samom desnom rubu zaslona, pored imena obroka, nalazi se strelica usmjerena prema dolje. Pritiskom na tu strelicu se ćelije svih unesenih proizvoda skrivaju za taj obrok. Broj kalorija za taj obrok ostaje još uvijek

vidljiv, a ponovnim pritiskom na spomenutu strelicu se ćelije proizvoda ponovno prikazuju. Na slici 6.7. vidljiv je prikaz zaslona za vođenje dnevnika prehrane.



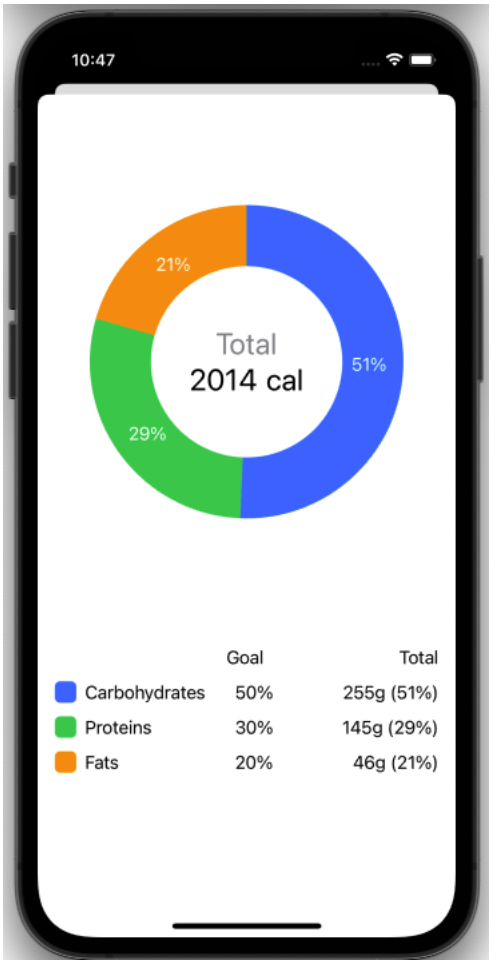
Slika 6.7. Prikaz zaslona za vođenje dnevnika prehrane

Pritiskom na pojedini proizvod otvara se novi prozor koji detaljnije prikazuje proizvod. Na tom prozoru prikazano je ime proizvoda, veličina unesene porcije, broj kalorija i makronutrijenta od kojih se sastoji. Na ovom prozoru korisnik također može urediti broj unesene porcije. Na slici 6.8. vidljiv je prikaz prozora s detaljnim opisom unesenog proizvoda.

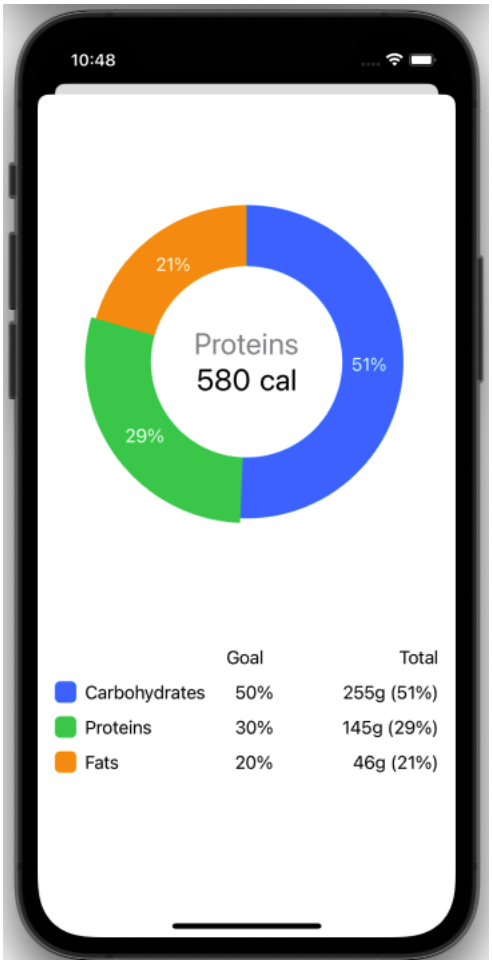


Slika 6.8. Prikaz prozora s detaljnijim opisom unesenog proizvoda

Počevši od vrha zaslona dnevnika prehrane, prvo što korisnik može vidjeti je crni krug s isječcima koji izgleda kao torta. Ta "torta" zapravo predstavlja dugme, koje kada se klikne, otvara novi prozor na kojemu se prikazuje kružni grafikon s postotkom svih makronutrijenata koji su uneseni u cijelome danu. Ulaskom u ovaj prozor, korisniku je prikazan ukupan broj unesenih kalorija u tom danu unutar kružnog grafikona. Klikom na pojedini isječak, broj unutar grafikona se mijenja iz broja ukupnih kalorija u broj kalorija pojedinog makronutrijenta. Ispod kružnog grafikona je također prikazana legenda kako bi korisnik prepoznao koja boja na grafikonu predstavlja koji makronutrijent. Na legendi su također prikazani postotci makronutrijenata koji su postavljeni za cilj u danu i koji postotak je korisnik zapravo unio. Na slici 6.9. vidljiv je prikaz kružnog grafa, dok se na slici 6.10. može vidjeti korisnikov klik na isječak grafa.



Slika 6.9. Prikaz kružnog grafa

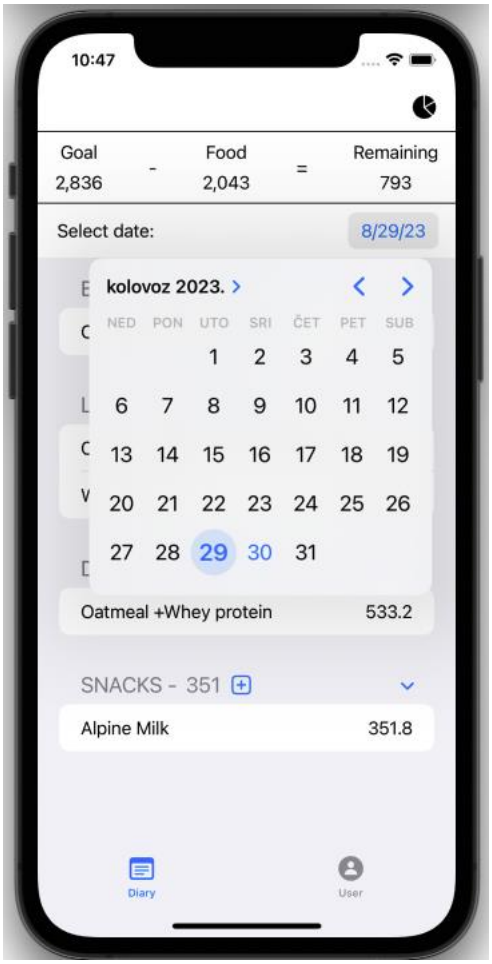


Slika 6.10. Prikaz korisnikovog klika na zeleni isječak grafa

Izlaskom s ovog prozora i povratkom na zaslon dnevnika prehrane, ispod dugmeta za prikaz kružnog grafikona, korisnik može vidjeti traku za praćenje unosa dnevnih kalorija. Traka se proteže cijelom širinom ekrana i u sebi sadrži tri broja koji s lijeva na desno predstavljaju:

- broj kalorija predloženih za unos u jednome danu
- broj kalorija koje je korisnik unio u danu
- ostatak kalorija koje korisnik može unijeti u danu

Ispod trake za praćenje unosa kalorija, može se vidjeti dio zaslona za promjenu datuma. Klikom na današnji datum otvara se kompaktni kalendar na kojemu se može odabrati datum za koji se žele vidjeti uneseni proizvodi. Klikom na drugi datum zaslon se ažurira s proizvodima koji su uneseni taj dan. Broj unesenih kalorija se također mijenja. Na slici 6.11. vidljiv je prikaz kalendara.

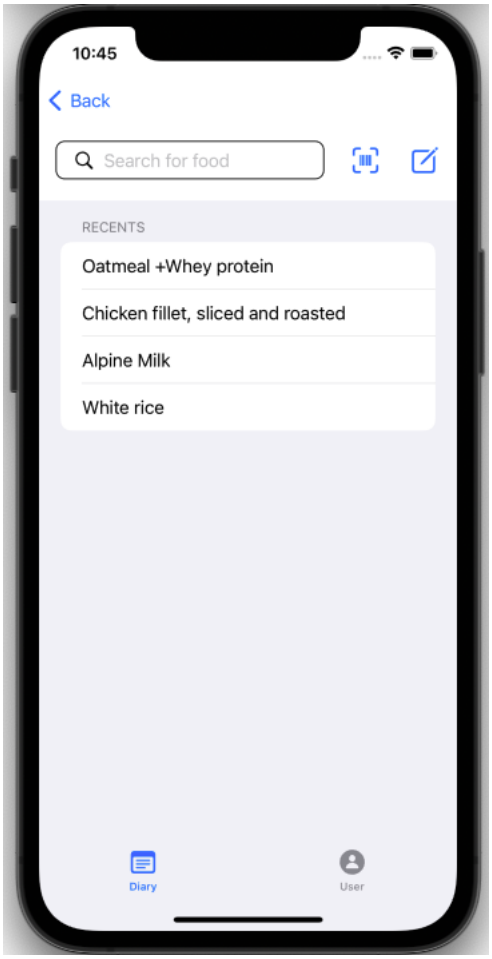


Slika 6.11. Prikaz kalendara

Kao što se može vidjeti na slici 6.7., pored broja kalorija određenog obroka vidljivo je plavo dugme s ikonom znaka plus. Pritiskom na njega, otvara se novi zaslon za pretragu proizvoda. Na zaslonu za pretragu proizvoda, veći dio zaslona rezerviran je za listu proizvoda. U ovom slučaju ti proizvodi predstavljaju nedavno pretraživane i spremljene proizvode, koji mogu biti pretraživani za bilo koji dan, odnosno datum. Budući da ovaj zaslon služi za pretraživanje proizvoda, pri vrhu zaslona korisnik ima tri opcije pretraživanja proizvoda. Krenuvši s lijeva na desno te tri opcije su:

- Pretraživanje proizvoda po imenu - crni pravokutnik s ikonom povećala koji služi kao polje za unos
- Pretraživanje proizvoda putem skeniranja linijskog koda - plavo dugme s ikonom linijskog koda
- Pretraživanje proizvoda putem unosa broja linijskog koda - plavo dugme s ikonom olovke i papira

Na vrhu zaslona s lijeve strane također postoji dugme za povratak nazad na zaslon dnevnika prehrane. Prikaz zaslona za pretraživanje proizvoda sa svim spomenutim komponentama vidljiv je na slici 6.12.

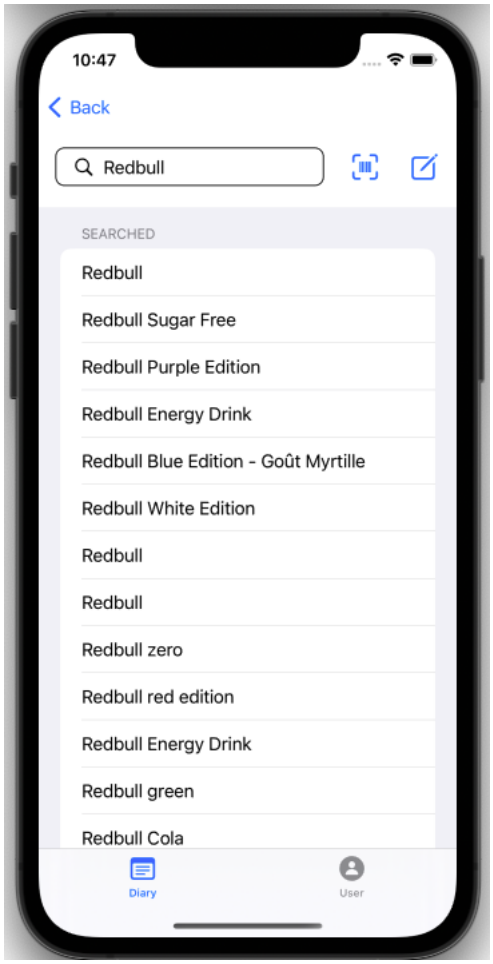


Slika 6.12. Prikaz zaslona za pretraživanje proizvoda

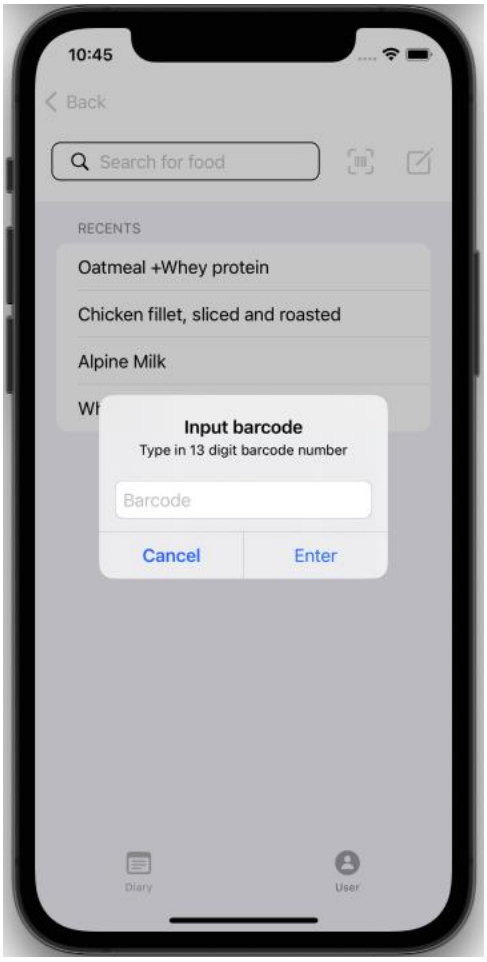
Prilikom unosa imena proizvoda u polje za unos i pritiskom dugmeta na tipkovnici za pretraživanje, lista nedavno pretraživanih proizvoda sada postaje lista proizvoda za unesenu riječ. Tada se također naslov liste mijenja iz *"Recents"* u *"Searched"*. Na slici 6.13. može se vidjeti prikaz liste pretraženog proizvoda.

Na sličan način radi pretraživanje proizvoda putem unosa broja linijskog koda, gdje se pritiskom na dugme otvara prozorčić koji u sebi sadrži polje za unos broja linijskog koda proizvoda. Na slici 6.14. prikazan je prozorčić za unos broja linijskog koda.

Pritiskom na jedan proizvod iz liste pretraživanih proizvoda ili pritiskom *"Enter"* dugmeta nakon unosa valjanog broja linijskog koda u polje za unos, prikazuje se prozor s detaljnim opisom proizvoda, s kojim se korisnik već susreo na zaslonu dnevnika prehrane (Slika 6.8.).



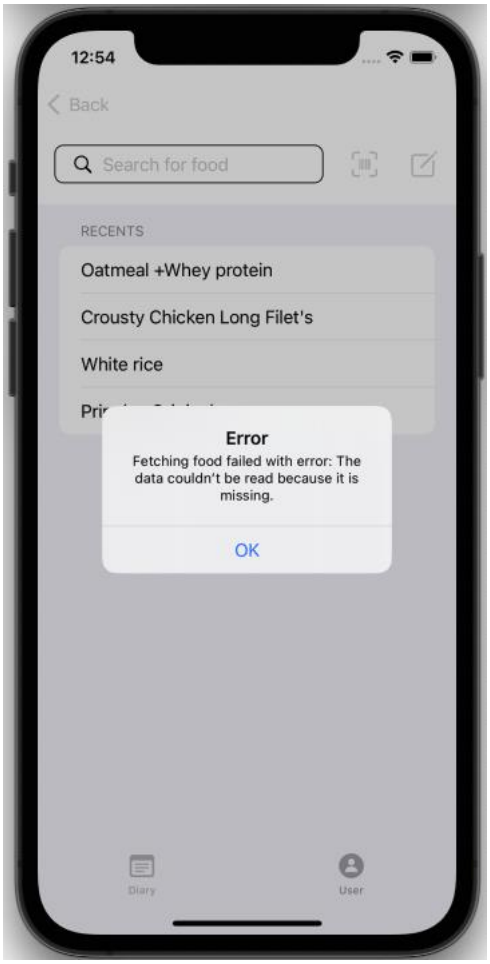
Slika 6.13. Prikaz liste pretraženog proizvoda



Slika 6.14. Prikaz prozorčića za unos broja linijskog koda proizvoda

Prilikom pritiska na dugme za pretraživanje proizvoda putem skeniranja linijskog koda proizvoda, u aplikaciji se otvara prozor koji prikazuje sadržaj zadnje kamere uređaja. Nakon što se uspješno očita linijski kod s proizvoda, otvara se prozor s detaljnim opisom skeniranog proizvoda. Isti prozor otvara se i u slučaju pretraživanja proizvoda putem upisa imena proizvoda ili putem upisa broja linijskog koda.

U slučaju greške prilikom skeniranja linijskog koda proizvoda ili pri unosu krivog broja linijskog koda u polje za unos, aplikacija upozorava korisnika na grešku. Ova greška vidljiva je na slici 6.15.

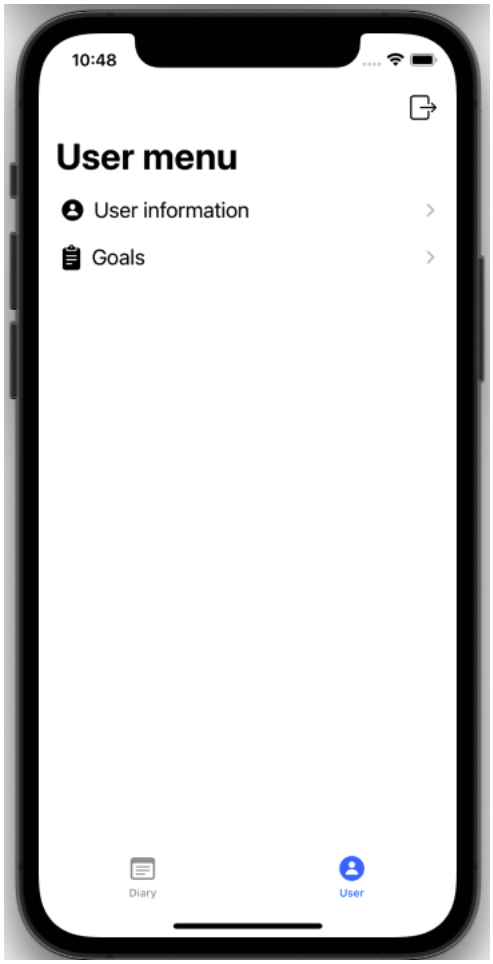


Slika 6.15. Prikaz greške nastale prilikom netočnog očitanja linijskog koda

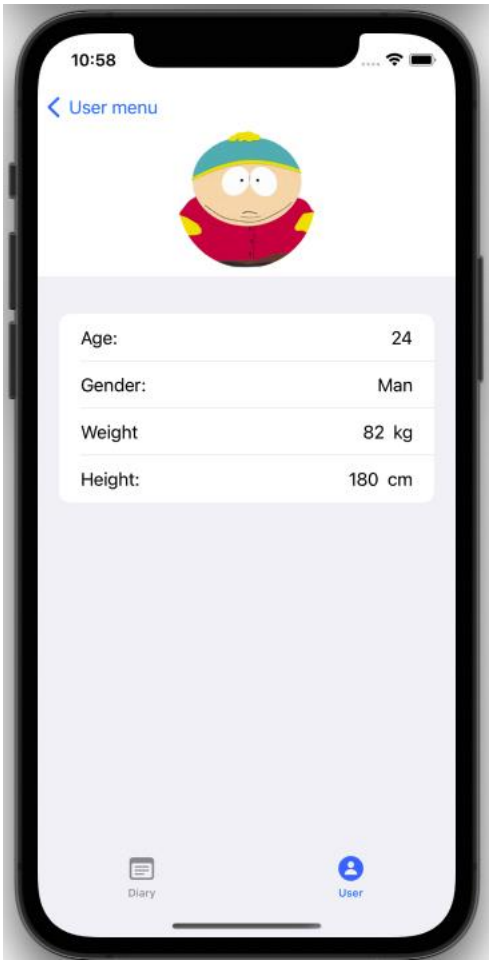
Povratkom na zaslon dnevnika prehrane, na donjem dijelu zaslona korisnik može primijetiti dvije ikone. Lijeva ikona predstavlja zaslon dnevnika prehrane, te budući da je korisnik već na tom zaslonu, ta ikona se prikazuje kao već odabrana - plave je boje. Ukoliko korisnik pritisne drugu ikonu, biti će preusmjeren na drugi zaslon, zaslon korisnika.

Na ovome zaslonu korisnik ima nekoliko opcija. Ispod naslova zaslona korisnik može kliknuti na *"User information"* (*hrv. Korisnikove informacije*), ćeliju koja korisnika vodi na novi zaslon. Na tom zaslonu, kako i samo ime kaže, korisnik može vidjeti informacije o sebi: broj godina, spol, težinu i visinu. Težina i visina, isto kao i na zaslonu za postavljanje ciljeva, mogu biti prikazani u dvije mjerne jedinice: kilogrami i funte za težinu, centimetri te stope i inči za visinu.

Korisnik također iznad tih informacija ima opciju dodavanja slike iz albuma slika. Korisnikov zaslon prikazan je na slici 6.16., dok se zaslon korisnikovih informacija prikazuje na slici 6.17.



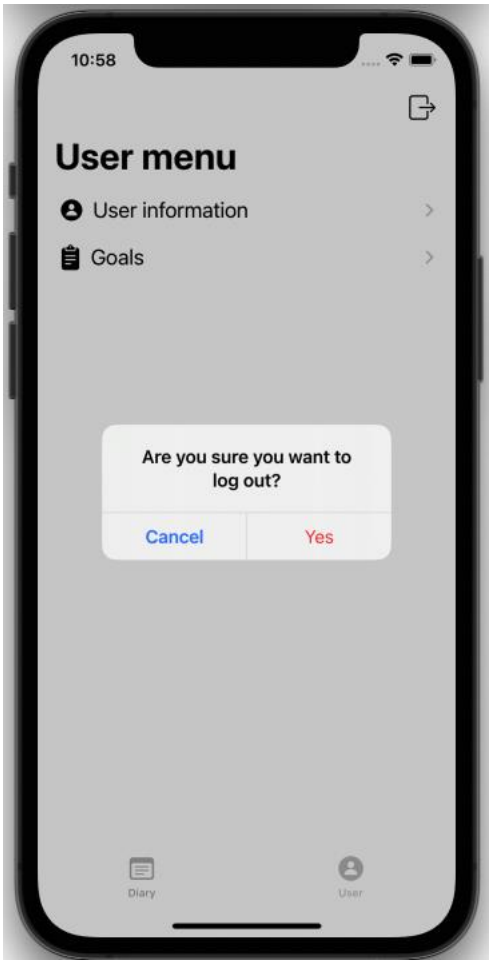
Slika 6.16. Prikaz korisnikovog zaslona



Slika 6.17. Prikaz zaslona korisnikovih informacija

Na korisnikovom zaslonu, ispod opcije za prikaz zaslona korisnikovih informacija, nalazi se i ćelija pritiskom na koju se korisniku prikazuje zaslon za korekciju ciljeva (*eng. goals*). Ovaj zaslon je već viđen na slici 6.3. gdje je korišten za prvobitno postavljanje ciljeva korisnika.

Na kraju, zadnja opcija koju korisnik može odabrati na korisnikovom zaslonu nalazi se u samom desnom vrhu zaslona, prikazana ikonom pravokutnika i strelice. Ova opcija korisniku nudi mogućnost odjave iz aplikacije, pri čemu se odabirom na nju, korisniku otvara prozor s upitom želi li se sigurno odjaviti iz sustava. Odabirom *"Yes"* korisnik se odjavljuje iz sustava te ga aplikacija vraća na zaslon za prijavu koji je vidljiv na slici 6.1. Prikaz prozora za odjavu prikazan je na slici 6.18.



Slika 6.18. Prikaz prozora za odjavu

7. REZULTATI PROVEDENOG ANKETNOG ISTRAŽIVANJA

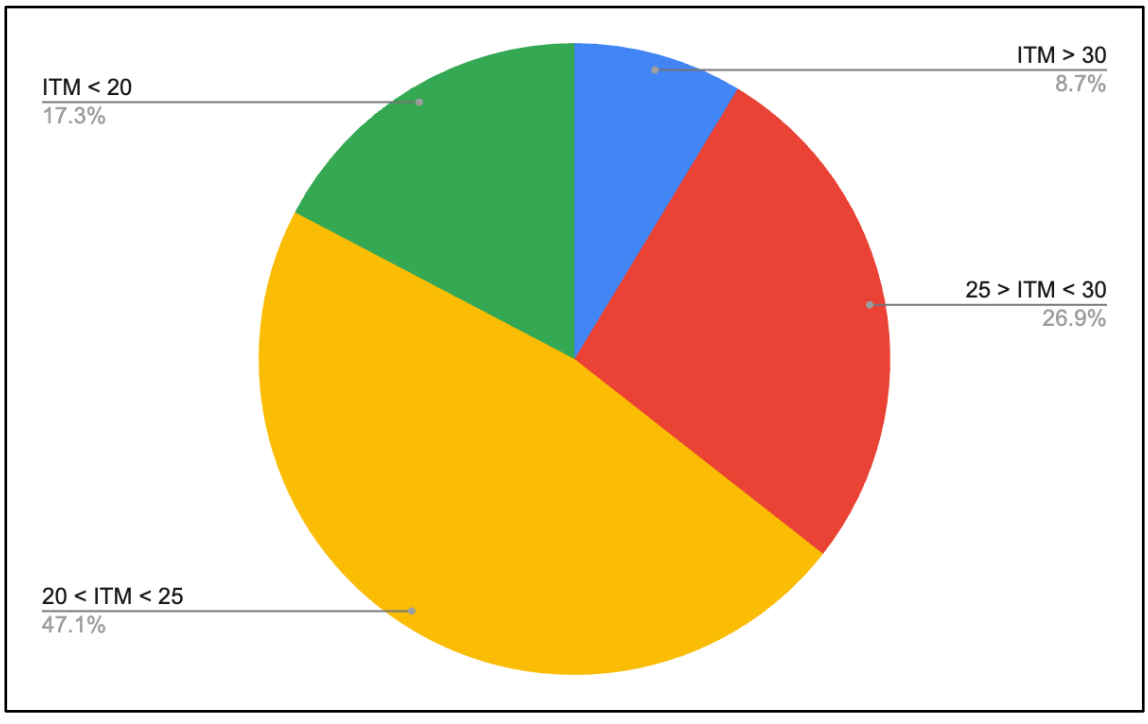
Putem ove aplikacije, cilj je olakšati korisnicima proces praćenja i unosa hrane koju su konzumirali tijekom dana. Time će korisnik imati uvid u broj unesenih kalorija i makronutrijenata u danu, a s time i lakše ostvarivanje svojih ciljeva. Aplikacija je korisna ako za nju postoji ciljana skupina ljudi koji istu namjerava koristiti. Iz tog razloga, za procjenu korisnosti ove aplikacije, provedena je i anketa kao metoda istraživanja. Anketa se sastoji od deset pitanja te je izrađena pomoću Google obrasca (*eng. Google form*). Jedan od ciljeva ankete bio je i istražiti o prehrambenim navikama ispitanika. Tjelesna aktivnost ispitanika također je predmet istraživanja putem ankete, budući da sve više ljudi postaje svjesno problema pretilosti. Isti iskazuju interes za uključivanje tjelovježbe ili sporta u svoju svakodnevnu rutinu, često zanemarujući važnost pravilne prehrane. Anketa isto tako istražuje koliki postotak ljudi je nezadovoljno svojim tjelesnim izgledom te na kraju koliko ljudi uopće vodi evidenciju o broju kalorija i makronutrijenata unesenih u danu. Tablica 7.1. prikazuje spomenutu anketu, prikazujući pitanja i ponuđene odgovore.

Pitanje	**Ponuđen odgovor**
Koliko imate godina?	Unos odgovora
Koji ste spol?	• Muškarac • Žena
Koliko ste visoki? (cm)	Unos odgovora
Koliko iznosi vaša tjelesna težina? (kg)	Unos odgovora
Koliko puta tjedno jedete/naručujete brzu hranu?	• 0 • 1-3 • 4-6 • 7 i više
Jeste li zadovoljni svojim tjelesnim izgledom?	• Da • Ne
Bavite li se sportom/tjelovježbom?	• Da • Ne
Ako se bavite sportom/tjelovježbom, koliko puta tjedno?	• 1 • 2-3 • 4-5

	• 6 i više
Vodite li evidenciju o broju kalorija unesenih u danu?	• Da • Ne
Biste li koristili aplikaciju za vođenje dnevnika prehrane	• Da • Ne

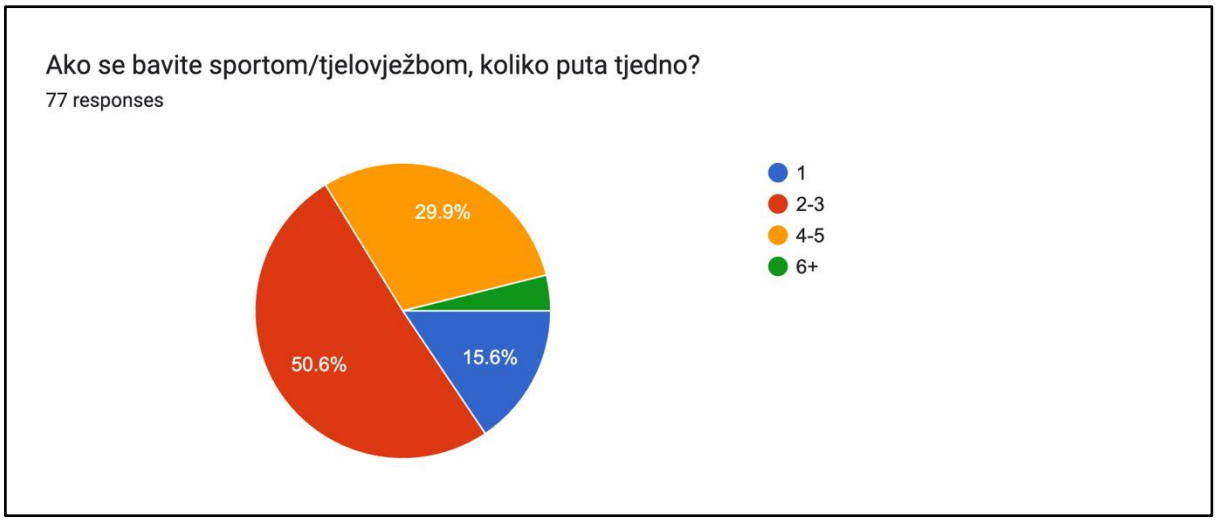
Tablica 7.1. Prikaz ankete za određivanje korisnosti aplikacije

Anketa je provođena u Republici Hrvatskoj u periodu od početka do kraja kolovoza 2023. godine. Njome je prikupljeno 104 odgovora, pri čemu su veći postotak ispitanika bile žene s 62,5% uzorka, dok je muškaraca bilo 37,5%. Najveći broj ispitanih osoba smjestio se unutar dobne skupine od 20 do 30 godina, čineći 80,8% uzorka. Ispitanici stariji od 30 godina činili su 16,4%, dok je udio onih mlađih od 20 godina iznosio 2,8%. Znajući visinu i težinu ispitanika, s lakoćom se može izračunati indeks tjelesne mase svakog ispitanika. Iako nije najpouzdaniji alat za procjenu debljine, budući da također mišićavi pojedinci često imaju veći indeks, on se može koristiti kao indikator za provođenje dodatnih ispitivanja ili poduzimanje određenih mjera. Imajući to na umu, osobe s indeksom većim od 30 bi trebale obratiti posebnu pozornost na svoje zdravlje, dok bi pojedinci s indeksom u intervalu od 25 do 30 i 18,5 do 20 trebali biti na oprezu. Postotak ispitanika koji pripadaju određenom intervalu indeksa tjelesne mase prikazan je na slici 7.1.



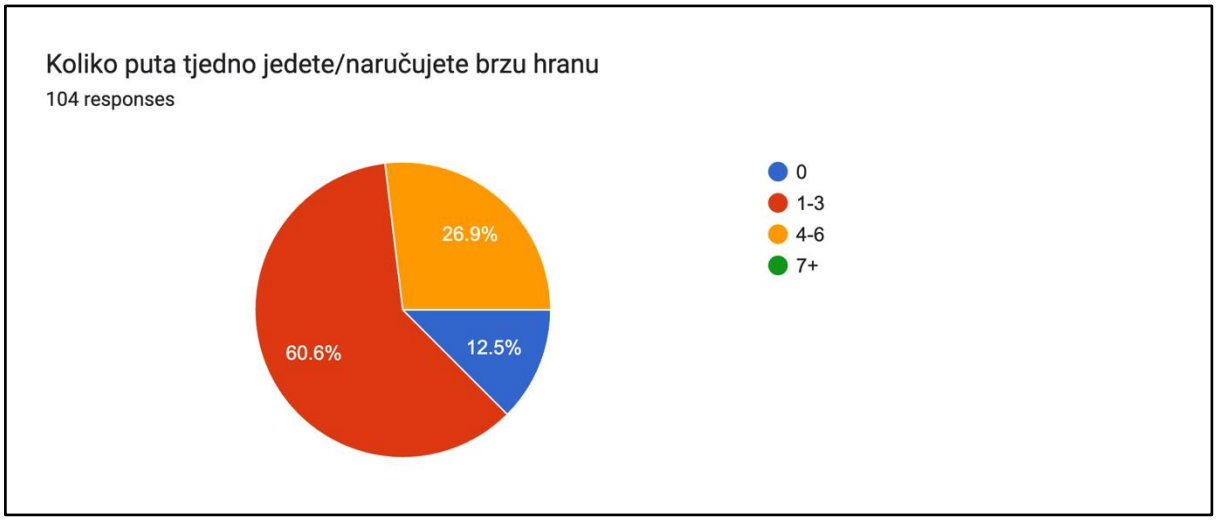
Slika 7.1. Prikaz intervala indeksa tjelesne mase ispitanika

S visokim stupnjem sigurnosti se može zaključiti da su ispitanici upoznati s problemom pretilosti, budući da se sportom ili tjelovježbom bavi njih 74%. Od tog broja, barem 2 do 3 puta tjedno se njime bavi 50%, dok čak 30% provodi 4 ili 5 tjednih treninga. Postotak tjednog broja treninga ispitanika koji se bave sportom ili tjelovježbom prikazano je na slici 7.2.



Slika 7.2. Prikaz postotka tjednog broja treninga ispitanika

Iako je broj ljudi koji se bave sportom ili tjelovježbom više nego zadovoljavajući, više od 50% ispitanika nije zadovoljno svojim tjelesnim izgledom. To možda nije iznenađujuće, s obzirom na to da se trend nezdravog prehrambenog obrasca i dalje nastavlja u Republici Hrvatskoj gdje 60,6% ispitanika barem 1 do 3 puta tjedno konzumira brzu hranu, dok čak 26,9% to čini 4 do 6 puta tjedno. Na slici 7.3. prikazano je koliko često ispitanici konzumiraju brzu hranu tjedno.



Slika 7.3. Prikaz učestalosti konzumiranja brze hrane tjedno

Na kraju, više od tri četvrtine ispitanika (78%) ne vodi evidenciju o unesenom broju kalorija dnevno, dok je četvrtina ispitanika spremna promijeniti navike i započeti vođenje dnevnika prehrane uz pomoć aplikacije, što rezultira s 53% ispitanika koji bi koristili aplikaciju.

8. ZAKLJUČAK

U današnjem modernom načinu života, urbanizacija i sjedilački način života postali su ključni čimbenici u epidemiji pretilosti i nedostatku tjelesne aktivnosti. Dok sve veći broj ljudi postaje svjesno ovih problema i nastoji uključiti tjelesnu aktivnost u svoj svakodnevni život, trend nezdrave prehrane još uvijek ostaje izražen. Poremećaji prehrane također se javljaju kao ozbiljan problem na drugom kraju spektra, gdje pothranjenost postaje jednako zabrinjavajući problem kao i pretilost. Pored svega ovoga, također postoje pojedinci koji ne ulaze ni u jednu od ovih kategorija, ali su unatoč tome nezadovoljni izgledom svoga tijela. S obzirom na navedene izazove i probleme, rješenje koje je prikazano u ovom radu potencijalno je efikasno. U ovom diplomskom radu izrađena je mobilna iOS aplikacija koja omogućuje korisnicima vođenje dnevnika prehrane kako bi bolje razumjeli svoje prehrambene navike, pratili unos kalorija i makronutrijenata, te ostvarili svoje ciljeve u pogledu tjelesne forme i zdravlja.

Za izradu aplikacije korišten je programski jezik Swift i SwiftUI okvir, razvojno okruženje Xcode, a za bazu podataka i autentifikacijske usluge korištena je platforma Firebase. U svrhu skeniranja linijskog koda korišteni su okviri AVFoundation i CodeScanner, dok su informacije o skeniranim i pretraživanim proizvodima dohvaćene putem API poziva prema Open Food Facts bazi podataka. Pri izradi aplikacije korištena je MVVM arhitektura koja omogućuje jasno razdvajanje poslovne logike od korisničkog sučelja. Ona također olakšava testiranje aplikacije i omogućava lakše upravljanje stanjem aplikacije, čime je poboljšana njena pouzdanost i skalabilnost. Potreba za ovakvom aplikacijom provjerena je putem provođenja ankete. Njome je zaključeno da sve više ljudi vidi pretilost kao ozbiljan problem, pri čemu se skoro tri četvrtine ispitanika bavi nekakvom vrstom tjelovježbe. Unatoč tome, broj ispitanika nezadovoljnih svojim tjelesnim izgledom veći je od 50%, ukazujući na to da se trend nezdravog načina prehrane nastavlja. S obzirom na to da dnevni unos kalorija nije praćen od strane 78% ispitanika, aplikacija se pokazala korisnom, budući da bi više od 50% ispitanika sada bilo voljno započeti vođenje dnevnika prehrane.

Budući da se sve aktualne aplikacije često nadograđuju i poboljšavaju, isto tako mjesta za napredak ima i u ovoj aplikaciji. Kao primjer nadogradnje, može se uzeti situacija kada se proizvod skenira, ali nije prisutan u bazi podataka. U tom slučaju korisniku se omogućuje upisivanje informacija o proizvodu s ambalaže proizvoda u aplikaciju, nakon čega se te informacije spremaju u bazu podataka. Također, aplikaciju bi se moglo proširiti dodavanjem funkcionalnosti za unos treninga i vježbi obavljenih u danu.

LITERATURA

[1] D. Štimac, S. Klobučar Majanović, "Hrvatske smjernice za liječenje odraslih osoba s debljinom", Acta Medica Croatica, vol. 76, br. 1, str. 3-18, 2022.

[2] D. Medanić i J. Pucarin-Cvetković, "Pretilost - javnozdravstveni problem i izazov", Acta Medica Croatica, vol. 66, br. 5, str. 347-354, 2012.

[3] D. Maslarda, N. Uršulin-Trstenjak, L. Bressan, " Poremećaj u prehrani – pretilost: prehrambene navike, tjelesna aktivnosti i samoprocjena BMI u Hrvatskoj ", Acta Medica Croatica, vol. 6, br. 1, str. 83-90, 2020.

[4] "Gotovo dvije trećine odraslih osoba u Hrvatskoj ima prekomjernu tjelesnu masu ili debljinu!", Hrvatski zavod za javno zdravstvo, [online], 2021., dostupno na: https://www.hzjz.hr/sluzba-promicanje-zdravlja/gotovo-dvije-trecine-odraslih-osoba-u-hrvatskoj-ima-prekomjernu-tjelesnu-masu-ili-debljinu/, [posjećeno 16.9.2023.]

[5] A. Zlatopolsky, "MyFitnessPal Review: The Best Fitness & Nutrition App?", Showcase, [online], 2023., dostupno na: https://www.si.com/showcase/health/myfitnesspal-review, [posjećeno 14.6.2023.]

[6] A. Choo Tung, "MyFitnessPal redesign", [online], dostupno na: https://chootung.design/myfitnesspal-case-study/, [posjećeno 15.6.2023.]

[7] B. Mayz, "MyFitnessPal Vs Cronometer: The Ultimate Comparison", Fit life fanatics, [online], 2022., dostupno na: https://fitlifefanatics.com/myfitnesspal-vs-cronometer/, [posjećeno 15.6.2023.]

[8] "Cronometer offers premuim feature for free", Newswire, [online], 2022., dostupno na: https://www.prnewswire.com/news-releases/cronometer-offers-premium-feature-for-free-in-june-to-promote-mens-health-month-301556281.html, [posjećeno 15.6.2023.]

[9] "We tried Lose It Calorie Counter: Here's Our Honest Review", GetGymFit, [online], dostupno na: https://getgymfit.com.au/lose-it-app-review/, [posjećeno 15.9.2023.]

[10] C. Kessler, "Noom Review: My experience after using noom for 30 days", Showcase, [online], 2023., dostupno na: https://www.si.com/showcase/health/noom-weight-loss-app-review, [posjećeno 15.9.2023.]

[11] K. Ress, "What Is Lifesum? Is It Better Than MyFitnessPal", MakeUseOf, [online], 2022., dostupno na: https://www.makeuseof.com/lifesum-vs-myfitnesspal/, [posjećeno 15.9.2023.]

[12] "Welcome to Xcode", Apple, [online], 2020., dostupno na: https://help.apple.com/xcode/mac/current/ - /devc8c2a6be1, [posjećeno 14.6.2023.]

[13] "Xcode overview", Apple, [online], 2023., dostupno na: https://developer.apple.com/documentation/xcode, [posjećeno 14.6.2023.]

[14] M. Ward, Swift Programming: The Big Nerd Ranch Guide, Big Nerd Ranch Guides, Atlanta, 2020.

[15] "The good and the Bad of Swift Programming Language", Altexsoft, [online], 2021., dostupno na: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-swift-programming-language/, [posjećeno 14.6.2023.]

[16] "Discover SwiftUI", Swift by Sundell, [online], 2021., dostupno na: https://www.swiftbysundell.com/discover/swiftui/, [posjećeno 14.6.2023.]

[17] "Firebase - Introduction", Geeks for geeks, [online], 2021., dostupno na: https://www.geeksforgeeks.org/firebase-introduction/, [posjećeno 26.6.2023.]

[18] C. Bonik - "AVFoundation Framework", Medium, [online], 2017., dostupno na: https://medium.com/oceanize-geeks/avfoundation-framework-de9a6a8ed453, [posjećeno 19.8.2023.]

[19] N. Fallet, "CodeScanner", Github, [online], 2021., dostupno na: https://github.com/twostraws/CodeScanner, [posjećeno 19.8.2023.]

[20] "Open Food Facts API Documentation", Open Food Facts , [online], dostupno na: https://world.openfoodfacts.org/, [posjećeno 20.8.2023.]

[21] F. Vilmart, G. Scalzo, S. De Simone, Hands-On Design Patterns with Swift, Packt Publishing, 2018.

[22] E. Garcia Gallardo, "What is MVVM Architecture?", BuiltIn, [online], 2023., dostupno na: https://builtin.com/software-engineering-perspectives/mvvm-architecture, [posjećeno 13.8.2023.]

ŽIVOTOPIS

Domagoj Šutalo rođen je 25.3.1999. u Đakovu. Nakon pohađanja Osnovne škole Vladimir Nazor u Đakovu, upisuje jezičnu gimnaziju Antun Gustav Matoš. 2017. godine završava gimnaziju te upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, smjer Računarstvo. Nakon završenog preddiplomskog studija, upisuje diplomski studij Računarstvo, smjer Informacijske i podatkovne znanosti. Od programskih jezika poznaje C, C++, Javu, Swift te najviše uživa u razvoju iOS aplikacija.

SAŽETAK

Sve više ljudi postaje svjesno jednog od većih problema u današnjem načinu života - pretilosti. Mnogi, kao rezultat toga, pokušavaju u svakodnevni život integrirati različite oblike tjelovježbe i sporta, često zanemarujući važnost prehrane. Kao rezultat rada izrađena je iOS mobilna aplikacija koja korisnicima omogućuje vođenje dnevnika prehrane. Aplikacija također omogućuje korisnicima postavljanje osobnih ciljeva, te na osnovu njih predlaže unos određenog broja kalorija i makronutrijenata. Korisnicima je omogućeno pretraživanje proizvoda unosom imena proizvoda ili broja linijskog koda proizvoda, kao i skeniranjem linijskog koda proizvoda. Aplikacija je pisana u XCode okruženju koristeći Swift jezik i SwiftUI okvir. Arhitektura aplikacije postavljena je prema Model-View-ViewModel obrascu, a uz pomoć Firebase platforme omogućena je registracija i prijava korisnika u aplikaciju kao i spremanje i dohvaćanje spremljenih prehrambenih proizvoda. Potreba za ovakvom aplikacijom provjerena je provođenjem ankete gdje bi više od polovice ispitanih koristilo aplikaciju.

Ključne riječi: iOS, dnevnik prehrane, mobilna aplikacija, Swift, MVVM

ABSTRACT

iOS food diary application

An increasing number of people are becoming aware of one of the major issues in today's lifestyle - obesity. Many, as a result, attempt to incorporate various forms of exercise and sports into their daily lives, often neglecting the importance of nutrition. As a result, an iOS mobile application has been developed, allowing users to keep a food diary. The application also enables users to set personal goals and, based on these goals, suggests the intake of a specific number of calories and macronutrients. Users can search for products by entering the product's name or barcode number, as well as by scanning the product's barcode. The application was developed using the Swift language and SwiftUI framework in the XCode environment. The architecture of the application follows the Model-View-ViewModel pattern, while Firebase platform enables the user registration and logging into application, as well as storage and retrieval of saved food products. The need for such an application was verified through a survey, where more than half of the respondents indicated they would use the application.

Key words: iOS, food diary, mobile application, Swift, MVVM

PRILOZI

Prilog 1: Pristup kodu izrađene aplikacije na GitHubu: https://github.com/dsutalo/FitNutrition1