

Mobilna aplikacija za pomoć u pravilnoj prehrani

Markelić, Marin

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Electrical Engineering and Computing / Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva**

Permanent link / Trajna poveznica: <https://um.nsk.hr/um:nbn:hr:168:052234>

Rights / Prava: [In copyright](#)/[Zaštićeno autorskim pravom.](#)

Download date / Datum preuzimanja: **2025-03-20**



Repository / Repozitorij:

[FER Repository - University of Zagreb Faculty of Electrical Engineering and Computing repository](#)



SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 348

**MOBILNA APLIKACIJA ZA POMOĆ U PRAVILNOJ
PREHRANI**

Marin Markelić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 348

**MOBILNA APLIKACIJA ZA POMOĆ U PRAVILNOJ
PREHRANI**

Marin Markelić

Zagreb, lipanj 2024.

DIPLOMSKI ZADATAK br. 348

Pristupnik: **Marin Markelić (0036528109)**

Studij: Računarstvo

Profil: Znanost o mrežama

Mentor: izv. prof. dr. sc. Igor Čavrak

Zadatak: **Mobilna aplikacija za pomoć u pravilnoj prehrani**

Opis zadatka:

Sve veća pažnja usmjerena na očuvanje zdravlja i pravilnu prehranu stvara potrebu za alatima kojim pojedinci mogu jednostavno planirati zdravu prehranu i pratiti unos kalorija. Novije generacije aplikacija, oslanjajući se na velike jezične modele, mogu pružiti dodatnu dimenziju interaktivnosti između aplikacije i korisnika, analizirajući greške i pružajući korisniku interaktivne savjete kako poboljšati prehranu. Cilj diplomskog rada je osmisliti koncept i ostvariti mobilnu aplikaciju za praćenje prehrane uz korištenje velikih jezičnih modela za pružanje savjeta o zdravoj prehrani. Potrebno je proučiti dostupna sučelja i razvojne okvire koji omogućuju automatizirano praćenje ponašanja korisnika i korištenje velikih jezičnih modela. Također je potrebno osmisliti i izvesti aplikaciju koja koristi sučelja za dohvaćanje nutritivnih informacija, interakciju s korisnikom putem velikih jezičnih modela i integraciju s HealthKit razvojnim okvirom za prikupljanje informacija o korisnikovim aktivnostima i dnevnoj potrošnji energije. Naposljetku, nužno je ispitati ostvarenu funkcionalnost aplikacije, performance i korisničko iskustvo.

Rok za predaju rada: 28. lipnja 2024.

Sadržaj

Uvod	1
1. Implementacijski odabiri	2
1.1. Izbor razvojnog okvira korisničkog sučelja.....	2
1.1.1. UIKit.....	2
1.1.2. SwiftUI	2
1.2. Izbor arhitekturnog obrasca	3
1.2.1. Model-View-Presenter.....	3
1.2.2. Model-View-Controller	4
1.2.3. The Composable Architecture	4
2. Arhitektura aplikacije	6
2.1. Pogledi	6
2.2. Predstavljajući.....	6
2.3. Slučajevi upotrebe	7
2.4. Usluge.....	7
2.5. Klijenti	8
3. Dodatni razvojni okviri i web sučelja.....	10
3.1. CalorieNinjas	10
3.2. OpenAI web sučelje.....	11
3.2.1. Asistenti	12
3.2.2. Dretva	16
3.2.3. Pokretanje	16
3.2.4. Poruke	18
3.3. HealthKit	18
3.4. Pohrana podataka.....	19
3.4.1. UserDefaults	19

3.4.2.	Realm.....	19
4.	Implementacija dodatnih razvojnih okvira i web sučelja	20
4.1.	CalorieNinjas	20
4.2.	OpenAI web sučelje.....	20
4.3.	HealthKit	24
4.3.1.	Stvaranje repozitorija i traženje pristupa	26
4.3.2.	Čitanje statičkih podataka.....	28
4.3.3.	Čitanje promjenjivih podataka.....	28
4.4.	Pohrana podataka.....	31
4.5.	Računanje preporučenih kalorija i nutrijenata.....	34
4.5.1.	Računanje preporučenih kalorija	34
4.5.2.	Računanje potrebnih nutrijenata.....	35
5.	Implementacija ekrana.....	38
5.1.	Početni pogled	38
5.2.	Pogled za pretraživanje.....	42
5.3.	Pogled za razgovor	47
5.4.	Pogled korisničkih karakteristika	50
6.	Rezultati i dorađivanje.....	52
6.1.	Mjerenje dnevne potrošnje kalorija	52
6.2.	Inženjerstvo uputa.....	53
6.2.1.	Upiti kod pretraživanja	53
6.2.2.	Upit kod razgovora	58
6.3.	Buduća poboljšanja.....	59
	Zaključak	61
	Literatura	62
	Sažetak.....	65

Summary.....	66
Skraćenice.....	67
Privitak.....	68

Uvod

U današnjem dinamičnom svijetu, održavanje balansirane i zdrave prehrane postalo je izazovno. Mnoge osobe nemaju naviku praćenja dnevnog unosa nutrijenata i kalorija te ne održavaju zdrave prehrambene navike. Problem dodatno pogoršava lak pristup nezdravoj visokokaloričnoj hrani koja je često vrlo ukusna i široko dostupna. Sve više ljudi postaje ovisno o brznoj hrani, nesvjesni njezinog štetnog utjecaja na zdravlje. Posljedica toga je povećanje stope pretilosti i povezanih zdravstvenih problema. Upravljanje zdravom prehranom zahtijeva ne samo svijest o važnosti nutritivne ravnoteže, već i alate koji omogućuju jednostavno i efikasno praćenje prehrambenih navika.

Primarni cilj ovog rada je razviti mobilnu aplikaciju koja korisnicima pomaže pratiti prehrambene navike na jednostavan i učinkovit način. Aplikacija omogućuje praćenje unosa kalorija i nutrijenata te pruža informacije koje pomažu korisnicima u donošenju boljih prehrambenih odluka. Korištenje velikih jezičnih modela omogućava pružanje personaliziranih savjeta temeljenih na analizi prethodno konzumiranih obroka, čime se korisnicima pomaže u donošenju informiranih odluka.

Korisnicima je dostupan virtualni asistent koji pruža dodatne savjete za poboljšanje prehrambenih navika, djelujući kao virtualni nutricionist. Asistent može analizirati podatke o prethodnim obrocima i predložiti zdravije alternative te ponuditi korisne savjete za održavanje uravnotežene prehrane. Osim toga, aplikacija može koristiti podatke iz vanjskih izvora za automatski unos fizičkih karakteristika korisnika te praćenje ukupne dnevne potrošnje kalorija, čime se osigurava cjelovita slika korisnikovog zdravlja i aktivnosti.

Razvoj ove aplikacije uključuje istraživanje i implementaciju različitih tehnologija i pristupa, uključujući integraciju nutritivnih API-ja za dohvaćanje podataka o hrani, korištenje velikih jezičnih modela za generiranje savjeta te integraciju s HealthKit razvojnim okvirom za praćenje fizičkih aktivnosti.

1. Implementacijski odabiri

1.1. Izbor razvojnog okvira korisničkog sučelja

Za razvoj korisničkog sučelja u iOS-u postoje dva razvojna okvira: UIKit i SwiftUI. Razlike između njih su značajne te je ovo najbitniji odabir pri kreiranju nove aplikacije.

1.1.1. UIKit

UIKit je Apple-ov razvojni okvir za korisničko sučelje predstavljen 2008., te se smatra zreloom tehnologijom. Dugi niz godina se koristio kao jedini razvojni okvir korisničkog sučelja za sve Appleove uređaje te je duboko integriran u cijeli Appleov ekosustav.

Koristi hijerarhijski sustav za upravljanje korisničkim sučeljem čiji su najbitniji elementi UIView i UIViewController. UIView je temeljna komponenta koja se iscrtava na ekranu dok UIViewController sadrži UIView-ove te je odgovoran za njih i njihove interakcije. Sadrži *Auto Layout*, sistem za definiranje veza između pogleda koji se brine o responzivnosti [1].

UIKit je razvijen imperativnom programskom paradigmatom. Njom je potrebno eksplicitno kreirati instance klasa i mijenjati im attribute. Time se postiže velika razina kontrole nad elementima koji se prikazuju na ekranu no rezultira velikom količinom repetitivnog koda. Zbog toga je programiranje sporo, ali temeljito.

1.1.2. SwiftUI

SwiftUI je moderni razvojni okvir za korisničko sučelje predstavljen 2019. kojemu je cilj pojednostavniti razvoj aplikacija. Prilagođen je i može se koristiti za sve Apple platforme te se sve više koristi u razvoju aplikacija i počinje zamjenjivati UIKit.

Rađen je po reaktivnim principima programiranja koji koriste vezanje podataka i mehanizme upravljanja koji automatski osvježavaju korisničko sučelje kada se podaci promijene. Podupire dinamični pregled koji omogućava prikaz korisničkog sučelja pored koda koji se osvježava dok programer piše kod te je dizajniran na fleksibilan način koji omogućuje da jedna aplikacija može raditi na više platformi.

SwiftUI koristi deklarativnu sintaksu. Za razliku od imperativne, njom se postiže način programiranja gdje programer opisuje kako želi da korisničko sučelje izgleda i kako se ponaša, a razvojni okvir preuzima odgovornost osvježavanja pogleda. Time se postiže puno brži rad jer se smanjuje broj linija koje su potrebne za istu funkcionalnost i kôd je puno pregledniji jer je lakše odvojiti komponente u zasebne cjeline [2].

Dok je UIKit zreo okvir s bogatim funkcionalnostima, razvoj u njemu je sporiji i složeniji. Iako je SwiftUI-u potrebno još sazrijevanja da bi se moglo potpuno ovisiti njemu kod kompleksnih aplikacija, njegova fleksibilnost, kao i mnoge dodatne mogućnosti čine ga boljim odabirom za ovaj projekt.

1.2. Izbor arhitekturnog obrasca

Nakon razvojnog okvira za korisničko sučelje bitno je odabrati arhitekturni obrazac. On definira kako će aplikacija biti organizirana.

1.2.1. Model-View-Presenter

Model-View-Presenter (MVP) je arhitekturni obrazac za korisničko sučelje koji potiče iz ranih 1990-ih te razdvaja odgovornosti korisničkog sučelja u tri objekta:

- Model (eng. Model) – Definira podatke koji se prikazuju ili nad kojima se vrši obrada
- Pogled (eng. View) – Prikazuje podatke koje mu šalje predstavljач i javlja događaje od interakcije s korisnikom
- Predstavljач (eng. Presenter) – Zadužen za upravljanje pogledom, njegovom logikom i obradom korisničkih interakcija

Kako bi MVP bio potpun, uz njegove elemente se često koriste elementi *Clean* arhitekture. *Clean* arhitektura odvaja odgovornosti poslovne logike aplikacije u niz objekata koji imaju jasnu svrhu. Time se postiže organiziranost aplikacije te se štede resursi jer objekti mogu biti jedinstveni te se jednu instancu može koristiti na više mjesta diljem aplikacije što smanjuje opterećenje [3].

Elementi *Clean* arhitekture:

- Slučaj upotrebe (eng. UseCase) – Komponenta koja objedinjuje funkcionalnosti veće značajke. Zadužena za komunikaciju sa drugim slučajevima upotrebe, klijentima i uslugama
- Klijent (eng. Client) – Zadužen za komunikaciju sa web sučeljima
- Usluga (eng. Service) – Omogućuje komunikaciju sa drugim razvojnim okvirima ili služi za razdvajanje manjih dijelova aplikacije u logičke cjeline

1.2.2. Model-View-Controller

Model-View-Controller (MVC) je arhitekturni obrazac za korisničko sučelje koji je sličan *Model-View-Presenteru* te se slično MVP-u sastoji od tri glavne komponente, model, pogled i upravljač (eng. Controller), gdje je glavna razlika u upravljaču. Iako upravljač ima sličnu zadaću kao predstavljatelj, glavna razlika je u tome da on daje eksplicitne instrukcije pogledu kako da se osvježi [4]. Na ovom pristupu je baziran rad UIKit-a, koji izvorno pruža upravljač za svaki ekran u aplikaciji pomoću klase UIViewController, ali nije pogodan za rad sa SwiftUI-om zbog toga što SwiftUI koristi tehnike vezanja podataka i mehanizme upravljanja kako bi automatski osvježio korisničko sučelje pri promjeni podataka.

1.2.3. The Composable Architecture

The Composable Architecture (TCA) je biblioteka za izgradnju aplikacija koja je Swift adaptacija *Redux* razvojnog okvira koja funkcionira sa postojećim Apple-ovim razvojnim okvirima kao što su SwiftUI, UIKit, Combine i Async/Await.

Potreba za TCA-om se razvila zbog toga što klasični obrasci, među koje spada i MVP, pate od problema kod skaliranja aplikacije. Kako aplikacija raste to je kompliciranije slati podatke među raznim dijelovima aplikacije, spajati funkcionalnosti, raditi testove i brinuti o modularizaciji [5]. TCA koristi *Reducer*, što je u suštini funkcija u kojoj je sadržana cijela poslovna logika neke značajke. On prihvaća događaje i na temelju njih i trenutnog stanja prelazi u iduće stanje.

TCA je obrazac koji ima veoma strmu krivulju učenja. Upravo zbog toga i teške čitljivosti ljudima koji nisu upoznati s njegovim konceptom je u ovom slučaju bolji odabir MVP.

Aplikacija u sklopu ovog rada neće biti kompleksna i imati puno ekrana te značajke TCP-a koje ga čine bolje od MVP-a neće doći do izražaja

2. Arhitektura aplikacije

Korištenjem obrasca *Model-View-Presenter* uz *Clean* arhitekturu dobivamo jasno razdvajanje elemenata korisničkog sučelja i elemenata poslovne logike. Za poslovnu logiku je bitno stvoriti elemente jednom, pri paljenju aplikacije, kako bi sva funkcionalnost bila potpuna, dok se elementi korisničkog sučelja mogu stvarati i brisati ovisno o korisničkim interakcijama sa aplikacijom.

2.1. Pogledi

Aplikacija se sastoji od 4 glavna pogleda. Oni prikazuju informacije na ekranu i dio su aplikacije s kojom korisnik ima interakciju. Svaki pogled ima određenu svrhu te su dizajnirani na intuitivan i lako razumljiv način.

Pogledi aplikacije:

- HomeView – početni pogled
- SearchView – pogled za pretraživanje
- ChatsView – pogled za razgovor
- ProfileView – pogled korisničkih karakteristika

2.2. Predstavljajući

Predstavljajući (eng. Presenters) su elementi zaduženi za svu logiku određenog ekrana. Zajedno sa slučajevima upotrebe čine vezu između korisničkog sučelja i poslovnog sloja. Oni dohvaćaju podatke s poslovnog sloja i pripremaju ih za prikaz na ekranima. Također, obrađuju interakciju s korisnikom primajući akcije od pogleda i na temelju njih ažuriraju podatke ili pozivaju metode slučajeva upotrebe. Postoji ih četiri u aplikaciji, jedan za svaki ekran:

- HomePresenter – predstavljajući početnog pogleda
- SearchPresenter – predstavljajući pogleda za pretraživanje
- ChatsPresenter – predstavljajući pogleda za razgovor
- ProfilePresenter – predstavljajući pogleda korisničkih karakteristika

2.3. Slučajevi upotrebe

Slučajevi upotrebe (eng. UseCases) su elementi zaduženi za agregiranje metoda i podataka od klijenta, usluga i drugih slučajeva upotrebe kako bi se postigla apstrakcija prema predstavljajcima jer je njihov posao brinuti se isključivo o ekranima, a ne i o poslovnoj logici. Svaki ekran ima svoj slučaj upotrebe namijenjen njemu te uz njih postoje još dva sa zasebnim odgovornostima:

- HomeUseCase – Slučaj upotrebe namijenjen početnom pogledu
- SearchUseCase – Slučaj upotrebe namijenjen pogledu za pretraživanje
- ChatsUseCase – Slučaj upotrebe namijenjen pogledu za razgovor
- ProfileUseCase – Slučaj upotrebe namijenjen pogledu korisničkih karakteristika
- NutritionDataUseCase – brine se o slanju i dohvaćanju podataka sa servisa za pohranu te dohvaća dnevni preporučeni unos kalorija i preporučene dnevne vrijednosti za nutrijente.
- HealthKitUseCase – komunicira s HealthKit uslugom i pretvara njezine vrijednosti u strukturu s kojom ostatak aplikacije zna raditi.

2.4. Usluge

Usluge (eng. Services) služe za komunikaciju s drugim razvojnim okvirima, kao što su HealthKit i Realm, te pomažu u organizaciji kôda. One odvajaju specifične funkcionalnosti u zasebne klase, što omogućava ponovnu upotrebu tih funkcionalnosti na više mjesta unutar aplikacije. Na taj način se poboljšava struktura aplikacije, povećava se čitljivost kôda, i olakšava održavanje i testiranje.

Aplikacija sadrži šest servisa:

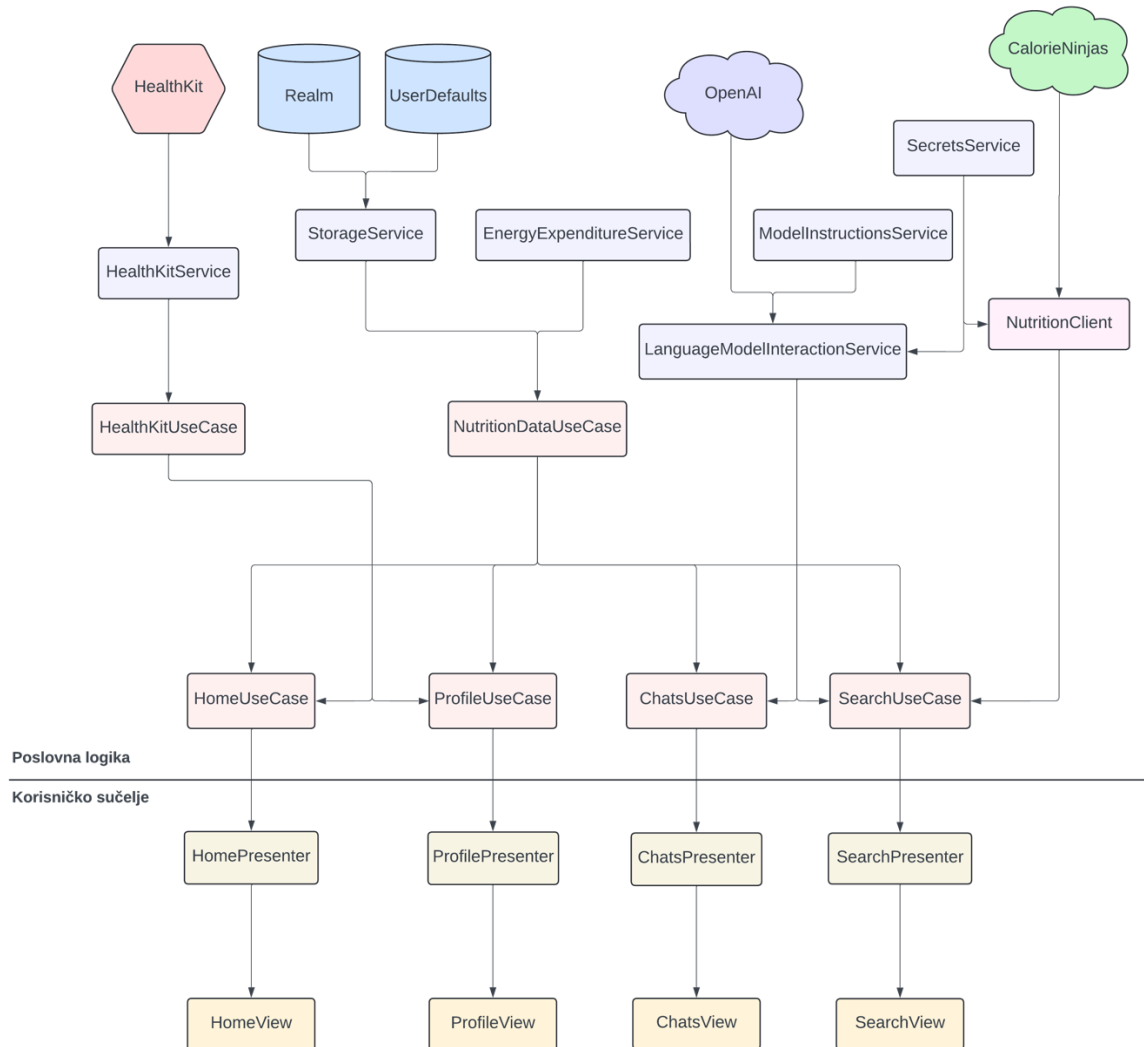
- StorageService – namijenjen spremanju i dohvaćanju podataka iz baza podataka Realm i UserDefaults
- HealthKitService – namijenjen dohvaćanju podataka o zdravlju s razvojnog okvira HealthKit

- `LanguageModelInteractionService` – komunicira s OpenAI web sučeljem pomoću `SwiftOpenAI` paketa kako bi korisniku pružio interakciju s velikim jezičnim modelom
- `EnergyExpenditureService` – za pružene vrijednosti o fizičkim karakteristikama korisnika računa dnevni preporučeni unos kalorija i preporučene dnevne vrijednosti za nutrijente
- `ModelInstructionsService` – zadužen za generiranje upita za veliki jezični model
- `SecretsService` – služi za dohvaćanje ključeva za web sučelja i identifikatora od OpenAI asistenta koji su pohranjeni lokalno

2.5. Klijenti

Klijenti su zaduženi za komunikaciju s vanjskim web sučeljima i obradu HTTP zahtjeva. U aplikaciji je bilo potrebno kreirati samo jednog klijenta za komunikaciju sa web sučeljem za nutriciju. `NutritionClient` komunicira s `CalorieNinjas` web sučeljem i prosljeđuje dohvaćene podatke.

Svi ovi elementi povezani su u strukturu pomoću pravila koje definira da elementi višeg sloja ne smiju ovisiti o elementima nižih slojeva. Također je bitna apstrakcija među elementima kojom se skrivaju svi podaci, osim onih potrebnih, kako bi se smanjila kompleksnost i povećala efikasnosti i čitljivost. Arhitektura aplikacije se može vidjeti na slici (Sl. 2.1)

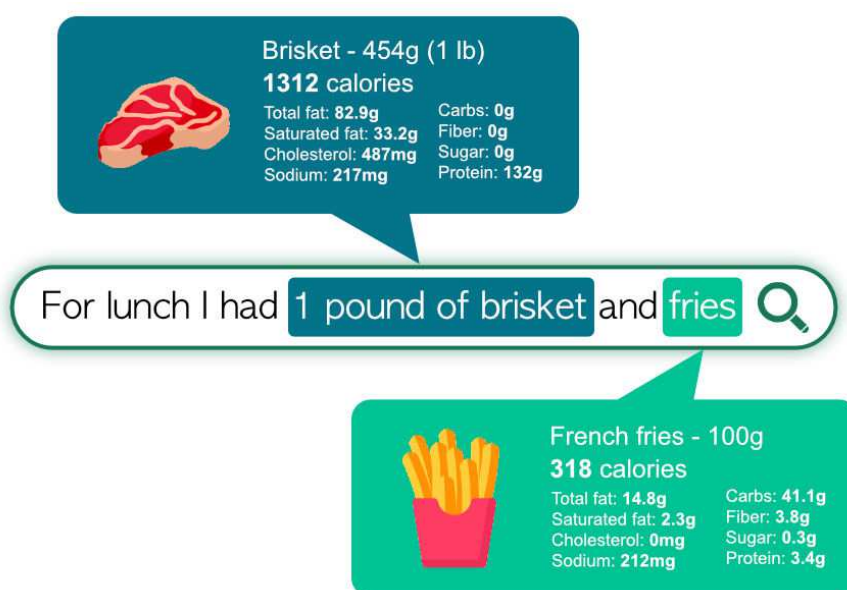


Sl. 2.1 Slika arhitekture aplikacije

3. Dodatni razvojni okviri i web sučelja

3.1. CalorieNinjas

CalorieNinjas je usluga koje pruža web sučelje za detaljne podatke o nutritivnom sadržaju za razne obroke. Sadrži veliku bazu podataka i besplatan plan što ju čini vrijednim resursom za ovakve projekte. Bitna prednost CalorieNinjas usluge je obrada teksta na poslužiteljskoj strani predočena slikom (Sl. 3.1). Nije potrebno formatirati poslani tekst ni na kakav način te će web sučelje vratiti sve namjernice koje je prepoznao u tekstu.



Sl. 3.1 Izvlačenje informacija iz teksta. Preuzeto s [6].

Također, uz nutritivne informacije za obroke u unesenom tekstu nudi i web sučelja za prepoznavanje obroka sa teksta na slikama te recepte koji u opsegu ovog rada nisu bili potrebni.

CalorieNinjas usluga je sposobna traženi obrok razdvojiti po namjernicama od kojih se sastoji te za svaku namjernicu vratiti sljedeće informacije [7]:

- Ime
- Broj kalorija
- Porcija u gramima
- Proteini u gramima
- Masti u gramima

- Zasićene masti u gramima
- Ugljikohidrati u gramima
- Šećer u gramima
- Vlakna u gramima
- Kolesterol u miligramima
- Natrij u miligramima
- Kalij u miligramima

3.2. OpenAI web sučelje

OpenAI sučelje pruža alate za integraciju funkcionalnosti velikih jezičnih modela u aplikacije. To uključuje stvaranje teksta, slika i govora, razumijevanje konteksta, prepoznavanje objekata sa slike i drugo. On je najbolji izbor za web sučelje velikih jezičnih modela jer pruža najviše mogućnosti i veliku fleksibilnost kod generiranja rezultata što je bitno za efikasnost i trošak.

Assistants API nov je način komunikacije s OpenAI web sučeljem. Ovo novo, poboljšano web sučelje za komunikaciju s velikim jezičnim modelima omogućuje izradu asistenta koji predstavlja objekt sa svojim instrukcijama i može baratati modelima, alatima i datotekama kako bi odgovarao na korisničke poruke [8].

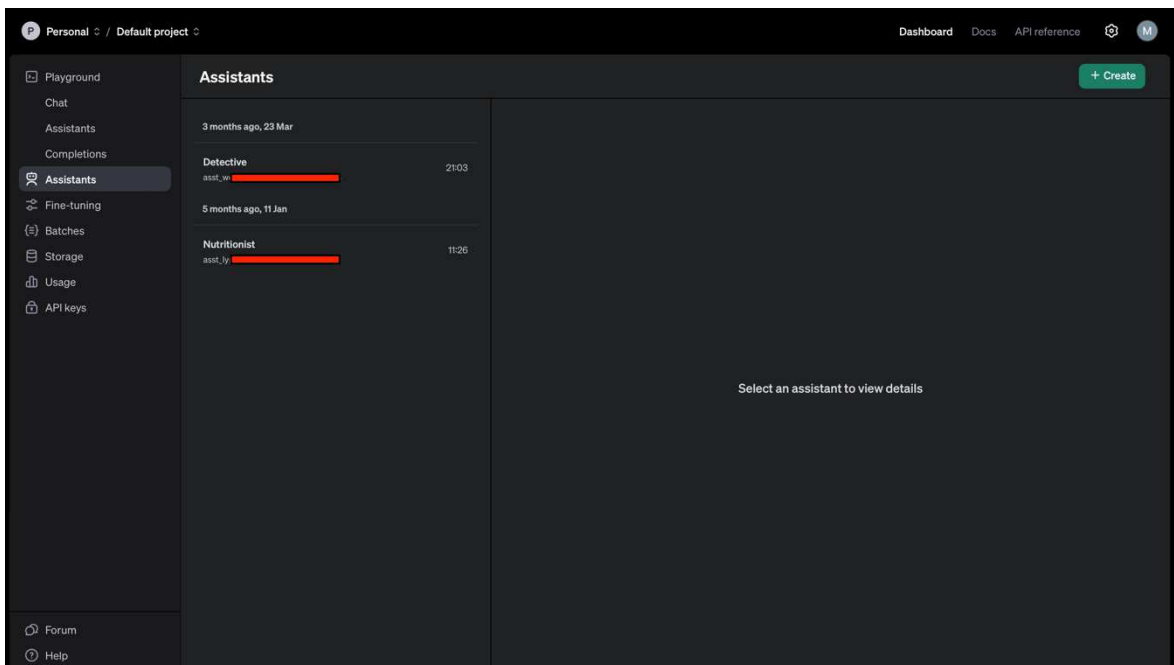
Prije predstavljanja *Assistants API*-ja je implementiranje komunikacije s velikim jezičnim modelom bilo kompleksno. Nije postojao alat koji je pružao zadržavanje konteksta razgovora, te je zbog toga bilo potrebno mijenjati upute sa svakim zahtjevom kako bi se u njemu uključile prijašnje interakcije što je utjecalo na trošak i složenost implementacije. Korištenje različitih zadataka, kao što su generacija, prijevod i sažimanje teksta je zahtijevalo korištenje više krajnjih točaka web sučelja te nije postojao mehanizam za razlikovanje poruka od sustava, modela i korisnika.

Novim pristupom, kojeg donosi *Assistants API*, sažeto je više funkcionalnosti u jednu krajnju točku kako bi komunikacija bila jednostavnija. Dodana je povijest poruka kojoj asistenti imaju pristup te više nije potrebno uključivati stare poruke pri slanju novih. Dodane su tri uloge, *system*, *user* i *assistant*, pomoću kojih se tok razgovora može prirodno pratiti bez oslanjanja na dodatnu logiku.

3.2.1. Asistenti

Prvi i najvažniji koncept *Assistants API*-ja su sami asistenti. Oni su agenti za razgovor koji koriste velike jezične modele kako bi komunicirali s korisnicima na ljudski razumljiv način [9]. Asistent može razgovarati s korisnikom na način da pruža informacije, odgovara na pitanja, pruža preporuke i slično, dok održava kontekst i koherentnost razgovora. Mogu koristiti razne jezične modele sa specifičnim instrukcijama koje utječu na njihovu osobnost i ton razgovora.

Asistent se kreira odlaskom na *Assistants* opciju lijevog izbornika na glavnoj stranici OpenAI web sučelja koje se može vidjeti na slici (Sl. 3.2). Na toj stranici se mogu vidjeti svi kreirani pomoćnici.



Sl. 3.2 *Assistants* opcija u OpenAI *Dashboard-u*

Pritiskom na gumb *Create* pokreće se postupak kreiranja novog asistenta. Na ovom ekranu, prikazan slikom (Sl. 3.4), možemo vidjeti opcije kojima možemo prilagođavati asistenta i identifikator koji je potreban da bi se povezali s njim. Gumb „*Playground*“ vodi na sličan ekran, prikazan na slici (Sl. 3.3), na kojem, uz prilagođavanje parametara modela, možemo razgovarati s njim i tako na jednostavno vidjeti kako promjene parametara utječu na odgovore.

Playground New in Assistants API ↗ Learn more ⚙️

THREAD 🗑️ 📄 Logs 🗄️

Name
Enter a user friendly name
asst_0 [REDACTED]

Instructions
You are a helpful assistant. 🗑️

Model
gpt-4o ⌵

TOOLS

- File search ⌵ ⚙️ + Files
- Code interpreter ⌵ + Files
- Functions** ⌵ + Functions

MODEL CONFIGURATION

Response format
 JSON object ⌵

Temperature 1
[Slider]

Top P 1
[Slider]

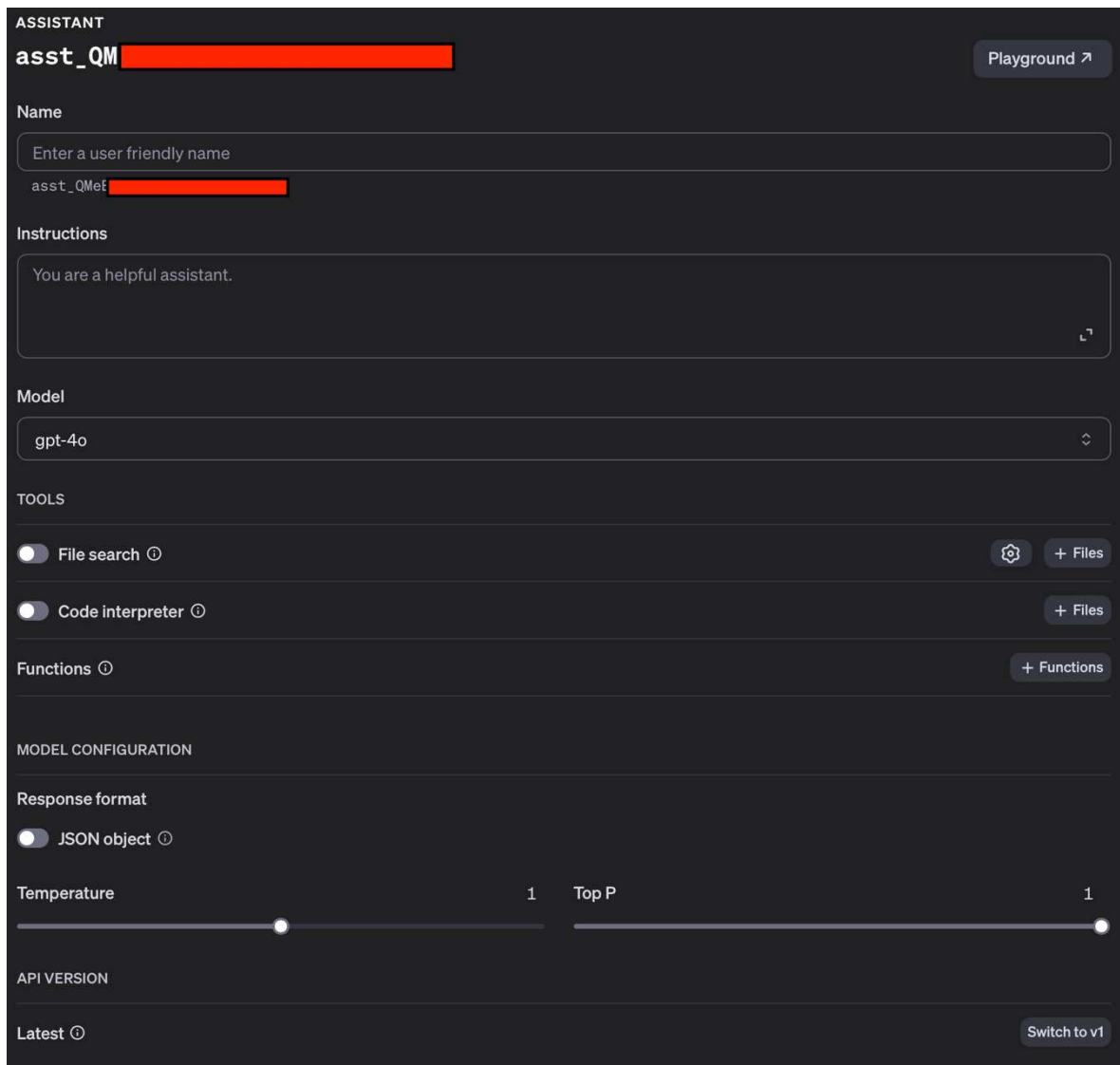
API VERSION
Latest ⌵ Switch to v1

Buttons: 🗑️ Clone Updated 08/06, 17:32

Enter your message... 📎 🗑️ + Run 🏃 ⌵

Playground messages can be viewed by anyone at your organization using the API.

Sl. 3.3 Opcija *Playground*



Sl. 3.4 Opcije prilagodbe asistenta pri kreiranju

U nastavku su opisani svi parametri asistenta čije se vrijednosti mogu prilagođavati.

Ime

Parametar ime (eng. Name) definira ime asistenta

Upute

Ovo je ključan parametar gdje pružamo detaljne upute (eng. Instructions) o tome kako se asistent treba ponašati, odgovarati i odnositi prema korisnicima. Upute mogu uključivati ton asistenta, specifične odgovore za određene upute i bilo kakva pravila o tipovima odgovora koje pruža.

Model

Parametar model definira veliki jezični model kojeg će asistent koristiti. O ovom parametru ovisi inteligencija asistenta, ali i trošak, jer su sposobniji modeli skuplji te je potrebno razmisliti o odabiru na temelju željene funkcionalnosti.

Alati

Postoje dva alata (eng. Tools) koje programer može odabrati. To su *Code interpreter* i *File search*.

Uključivanjem opcije *Code interpreter* asistent dobiva sposobnost pisanja i izvršavanja Python kôda u izoliranom okruženju. Radi toga ovaj alat može obrađivati datoteke sa opširnim podacima i formatiranjem i generirati datoteke s podacima i slike grafova. Također, kôd koji asistent napiše se može izvršavati iterativno, i tako rješavati složene matematičke probleme.

File search daje asistentu znanje izvan velikog jezičnog modela, u obliku informacija koje mu korisnik pruža. Korisnik može učitati svoje dokumente, koje OpenAI automatski parsira i pohranjuje. Asistent pristupa informacijama unutar datoteka kada smatra da je to potrebno iz konteksta poruka.

Funkcije

Asistentu se mogu definirati funkcije (eng. Functions). Funkcije omogućavaju da se za određeni korisnikov upit generira strukturirani rezultat koji je definiran funkcijom za koji bi inače trebalo pisati instrukcije ili regularne izraze što nije garantiralo željene rezultate. Određeni modeli su automatski trenirani za pozivanje funkcija te znaju odrediti kada se treba pozvati, a kada odgovoriti na normalan način.

Format odgovora

Asistent se može ograničiti da odgovara samo u JSON formatu kako bi bilo lakše procesirati odgovore.

Temperatura

Temperatura (eng. Temperature) se u velikim jezičnim modelima odnosi na nasumičnost odgovora te predstavlja vrijednost od 0 do 2. Što se temperatura bliži nuli to će odgovori ista pitanja postati repetitivni i slični dok povećanjem temperature dobivamo kreativnije odgovore.

Top P

Top P je još jedan parametar koji kontrolira nasumičnost modela. Vrijednost ovog parametra se kreće od 0 do 1 i koristi se za kontrolu s kojom će se vjerojatnošću generirati sljedeća riječ na temelju prethodne. Konkretno, *Top P* odabire sljedeću riječ iz najvjerojatnijih opcija koje zajedno čine kumulativnu vjerojatnost *P*. Ovaj pristup omogućava kontrolu nad ravnotežom između kreativnosti i preciznosti generiranog teksta. Korištenjem viših vrijednosti *Top P*, model će koristiti širi spektar mogućnosti, dok će niže vrijednosti ograničiti odabir na najvjerojatnije opcije, smanjujući tako šanse za generiranje manje smislenog ili nesuvislog teksta.

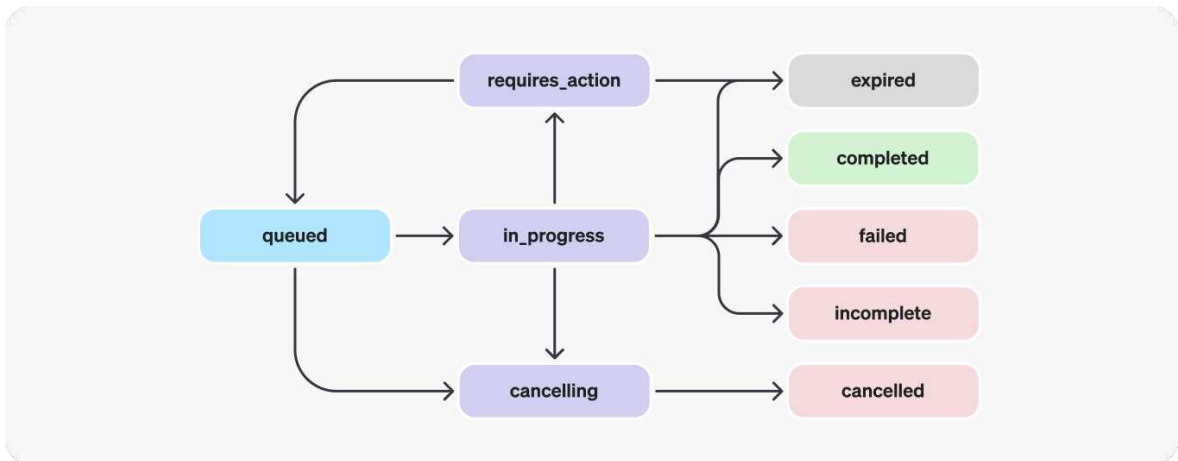
3.2.2. Dretva

Dretva (eng. Thread) se odnosi na razgovor između asistenta i korisnika. Dretva zadržava cijeli kontekst razgovora, uključujući poruke, te ih automatski sažima kako bi stale u kontekst [9]. One omogućavaju asistentu sjećanje prijašnjih interakcija i pružanje odgovora baziranih na prethodnim porukama.

3.2.3. Pokretanje

Pokretanje (eng. Run) predstavlja invokaciju asistenta na dretvi. Pri obavljanju pokretanja asistent koristi svoju konfiguraciju, prethodne poruke s dretve i alate kako bi generirao nove poruke koje se na kraju pokretanja nadodaju na dretvu [9].

Svako pokretanje ima životni ciklus koji može prolaziti kroz više stanja koja su vidljiva na slici (Sl. 3.5).



Sl. 3.5 Dijagram stanja pokretanja. Preuzeto s [9]

- *Queued* – stanje u koje pokretanje ulazi u trenutku kada se kreira ili kada je stanje *requires_action* završeno
- *In_progress* – u ovom stanju asistent koristi model i alate kako bi vršio korake i generirao odgovor. Dok je pokretanje u ovom stanju ne mogu se nadodati nove poruke na dretvu i stvarati nova pokretanja
- *Completed* – ovo stanje signalizira da je pokretanje uspješno završeno te su odgovori i koraci dodani u dretvu. U ovom trenutku se može sigurno nastaviti razgovor sa asistentom
- *Requires_action* – pokretanje ulazi u ovo stanje kada asistent uz pomoć alata za poziv funkcija detektira ime i argumente funkcije koju je potrebno pozvati. Zatim je potrebno pozvati funkciju i proslijediti njene rezultate prije nego asistent može nastaviti s obradom. Ako se odgovor ne pruži unutar 10 minuta stanje prelazi u *expired*
- *Expired* – stanje u koje se prelazi iz stanja *requires_action* ako rezultati pozivanja funkcije nisu predani unutar 10 minuta. Također se u ovo stanje prelazi ako iz bilo kojeg razloga pokretanje traje dugo.
- *Cancelling* – predan je zahtjev za poništavanje pokretanja pomoću *Cancel run* krajnje točke. Poništavanje se ne garantira, a ako uspije se prelazi u stanje *cancelled*
- *Cancelled* – pokretanje je uspješno poništeno
- *Failed* – pokretanje nije bilo uspješno
- *Incomplete* – pokretanje je završilo zbog dosizanja maksimalnog broja tokena

Svako pokretanje rezultira koracima (*Run steps*). Oni predstavljaju detaljnu listu akcija koje je asistent napravio kako bi došao do krajnjeg rezultata te se mogu koristiti kako bi se razumjelo zašto je asistent odgovorio na način na koji je odgovorio.

3.2.4. Poruke

Poruke (eng. Messages) mogu biti kreirane od strane asistenta ili korisnika. Sadrže tekst, sliku ili dokument [9].

3.3. HealthKit

HealthKit je Appleov razvojni okvir namijenjen dijeljenju informacija o zdravlju i fitnessu među aplikacijama na pouzdan i siguran način. Pruža centralni repozitorij za podatke na iOS uređaju putem Health aplikacije što omogućuje korisnicima da gledaju i upravljaju podacima iz više izvora na jednom mjestu. Podaci mogu pristizati s drugih aplikacija, prijenosnih uređaja kao što su pametni satovi i medicinskih uređaja.

Uz dozvolu korisnika, aplikacije komuniciraju s HealthKit repozitorijem čitajući i pišući podatke. Dizajniran je da aplikacije imaju zajednički pristup izgradnji iskustva koje omogućuje da aplikacije ne moraju koristiti sve značajke koje pruža HealthKit nego se fokusirati samo na podskup koji im je bitan [10].

U sklopu ovog rada aplikacija koristi HealthKit samo za čitanje određenih podataka. Podaci se sa HealthKit-a mogu čitati na tri načina:

- Direktnim pozivom metoda – HealthKit pruža metode kojima se može direktno pristupiti karakterističnim podacima, to jest podacima koji se rijetko ili nikada ne mijenjaju
- Upitima – vraća se trenutni snimak traženih podataka
- Dugotrajnim upitima – upiti koji ostaju aktivni u pozadini i šalju obavijesti kada sustav detektira promjene

3.4. Pohrana podataka

Aplikacija koristi pohranu podataka kako bi se pohranili podaci o korisniku i njegovim konzumiranim obrocima. Za pohranu se u aplikaciji koriste dvije različite baze podataka, UserDefaults i Realm. Ovaj sustav pohrane omogućuje aplikaciji da učinkovito upravlja korisničkim podacima i osigurava visoku razinu performansi.

3.4.1. UserDefaults

UserDefaults je jednostavan i efikasan način pohrane podataka na principu ključ-vrijednost za sve Apple platforme te nudi brz i lagan način za trajno spremanje podataka bez potrebe za složene baze podataka. Idealan je za slučajeve kada se trebaju pohraniti jednostavni podaci kao što su korisničke preferencije, postavke i male informacije o stanju aplikacije [11].

3.4.2. Realm

Realm je mobilna baza podataka koja se izvršava na mobilnim uređajima, tabletima ili nosivim uređajima te radi na svim Apple platformama.

Dizajniran je na intuitivan način koji je lagan za učenje i implementaciju i predviđen je radu na mobilnim uređajima [12]. Nudi visoke performanse i brzi pristup podacima zahvaljujući svojoj laganoj arhitekturi. Podržava složene upite i relacije između podataka, omogućujući aplikacijama da skaliraju s velikim količinama podataka bez značajnog utjecaja na performanse. Također, nudi robusne značajke kao što su reaktivnost, automatsko sinkroniziranje podataka i enkripcija, čime se osigurava visoka razina sigurnosti i konzistentnosti podataka.

4. Implementacija dodatnih razvojnih okvira i web sučelja

4.1. CalorieNinjas

Klasa zadužena za komunikaciju sa CalorieNinjas web sučeljem je `NutritionClient`. Dohvaćanje informacija o željenom obroku je jednostavno. Nakon što korisnik unese željeni obrok potrebno je poslati uneseni tekst na adresu web sučelja HTTP metodom GET metodom `getNutritionalInformation()` prikazanom na slici (Sl. 4.1). U toj metodi se kreira HTTP zahtjev, na URL od web sučelja se nadodaje tekst kojeg je korisnik upisao, te se u tijelo zahtjeva stavlja ključ za web sučelje.

```
func getNutritionalInformation(for query: String) async throws -> MealNetworkViewModel {
    guard
        let components = URLComponents(string: apiUrl + query),
        let url = components.url
    else { throw NutritionClientError.basicError }

    var request = URLRequest(url: url)
    request.httpMethod = "GET"
    request.setValue("application/json; charset=utf-8", forHTTPHeaderField: "Content-Type")
    request.setValue(apiKey, forHTTPHeaderField: "X-API-Key")

    let (data, _) = try await URLSession.shared.data(for: request)

    return try JSONDecoder().decode(MealNetworkViewModel.self, from: data)
}
```

Sl. 4.1 Funkcija za dohvaćanje nutritivnih informacija

HTTP zahtjev vraća odgovor u obliku JSON-a koji sadrži listu sastojaka traženog obroka i nutritivne vrijednosti svakog sastojka. Rezultat se zatim pretvara u strukturu `MealNetworkViewModel` i vraća kao rezultat funkcije.

4.2. OpenAI web sučelje

Za komunikaciju aplikacije sa OpenAI web sučeljem se koristi postojeće rješenje `SwiftOpenAI`. `SwiftOpenAI` je paket koji se može dodati u aplikaciju i pruža metode koje omogućuju laganu integraciju sa svim OpenAI sučeljima. U aplikaciji je klasa `LanguageModelInteractionService` zadužena za komunikaciju s OpenAI web sučeljem.

Prvi ekran u aplikaciji u kojoj se koristi ova funkcionalnost je pogled za pretraživanje. Pri traženju obroka će se uz zahtjev za nutritivne informacije poslati zahtjev u kojemu se zahtjeva savjet od modela za obrok koji korisnik želi konzumirati. U zahtjevu će se nadodati osnovne informacije tog obroka i osnovne informacije prošlih obroka tog dana. S ovom funkcionalnošću se odvraća korisnika od loših prehrambenih navika pružajući mu informaciju u tekstualnom formatu kojeg lako shvaća. Pri implementaciji savjeta nije bilo potrebno koristiti *Assistants API* već je bilo dovoljno slati jednu poruku i prikazati odgovor na nju. Za to se koristi metoda `sendSingleMessage()` prikazana slikom (Sl. 4.2).

```
func sendSingleMessage(text: String) async -> String? {
    let parameters = ChatCompletionParameters(
        messages: [.init(role: .user, content: .text(text))],
        model: .gpt35Turbo)
    let chat = try? await service.startChat(parameters: parameters)

    guard let message = chat?.choices.first?.message else { return nil }

    return message.content
}
```

Sl. 4.2 Metoda za dohvaćanje savjeta pri pretraživanju obroka

Drugi ekran sa velikim jezičnim modelom je pogled za razgovor. Svrha pogleda za razgovor je pružiti korisniku mjesto gdje može razgovarati sa asistentom koji mu pruža korisne informacije i navodi ga prema boljim prehrambenim navikama.

Cijela funkcionalnost ekrana je bazirana na *Assistants API*-ju. Pri pokretanju aplikacije ovaj ekran nema poruka te je potrebno upisati nešto kako bi se započeo razgovor. Pri slanju poruke poziva se funkcija `initiateMessageSending()` prikazana slikom (Sl. 4.3).

```
func initiateMessageSending(text: String, conversationId: String?, instructions: String? = nil) async {
    queryStatusSubject.send(.running)
    appendSentMessage(text: text)

    if let conversationId {
        await sendMessage(text: text, conversationId: conversationId, instructions: instructions)
        return
    }

    guard let newThreadId = await createNewThread() else {
        queryStatusSubject.send(.failed)
        return
    }

    await sendMessage(text: text, conversationId: newThreadId, instructions: instructions)
}
```

Sl. 4.3 Metoda za pripremanje slanja poruke

Ova funkcija postavlja status na *running* kako bi se spriječila mogućnost slanja drugih poruka, dodaje privremenu poslanu poruku u razgovor, što poboljšava korisničko iskustvo jer nije potrebno čekati odgovor od web servisa i provjerava postoji li `conversationId` koji označava identifikator dretve. Ako postoji šalje poruku na nju, a ako ne kreira novu dretvu i šalje poruku.

Funkcija `sendMessage()` na slici (Sl. 4.4) kreira novu poruku i dodaje ju na dretvu. Nakon što je poruka dodana na dretvu koristi metodu `retrieveMessages()` koja dohvaća sve poruke na dretvi čime se zamjenjuje privremena poslana poruka sa porukom sa stvarnim podacima i započinje pokretanje koje se nadgleda funkcijom `observeRun()`.

```
private func sendMessage(text: String, conversationId: String, instructions: String? = nil) async {
    let message = MessageParameter(role: .user, content: text)
    let _ = try? await service.createMessage(threadID: conversationId, parameters: message)

    await retrieveMessages(for: conversationId)

    let parameters = RunParameter(assistantID: nutritionAssistantId, instructions: instructions)
    guard let run = try? await service.createRun(threadID: conversationId, parameters: parameters) else {
        queryStatusSubject.send(.failed)
        return
    }

    await observeRun(run: run, threadId: conversationId)
}
```

Sl. 4.4 Metoda za slanje poruke

Pošto aplikacija ne koristi opciju *streaming*, kojom odgovor dolazi u dijelovima s čime se dobiva efekt pisanja, potrebno je periodički provjeravati status pokretanja što radi funkcija `observeRun()` na slici (Sl. 4.5). Nakon što funkcija detektira uspješni status dohvaćaju se poruke i razgovor se osvježava.

```

private func observeRun(run: RunObject, threadId: String) async {
    Task.detached { [weak self] in
        guard let self else { return }

        let counterLimit = 10
        var counter = 0
        var runStatus = RunStatus(from: run.status)
        while runStatus.isStillRunning {
            guard counter <= counterLimit else {
                queryStatusSubject.send(.failed)
                return
            }

            usleep(200000)

            let run = try? await service.retrieveRun(threadID: threadId, runID: run.id)
            guard let runStatusString = run?.status else {
                queryStatusSubject.send(.failed)
                return
            }

            runStatus = RunStatus(from: runStatusString)
            counter += 1
        }

        await retrieveMessages(for: threadId)
        queryStatusSubject.send(.available)
    }
}

```

Sl. 4.5 Metoda za provjeravanje statusa pokretanja

Metoda `retrieveMessages()` na slici (Sl. 4.6) koristi funkcionalnost povijesti dretve kako bi dohvatila zadnjih 20 poruka koje se pretvaraju u objekt `Conversation`. Podaci iz te strukture se nakon učitavanja pokazuju na ekranu u obliku razgovora. Ova funkcija se također koristi pri mijenjanju razgovora gdje se pri pritisku na razgovor šalje identifikator starog razgovora i učitava njegova povijest.

```

func retrieveMessages(for conversationId: String) async {
    guard
        let messages = try? await service.listMessages(
            threadID: conversationId,
            limit: 20,
            order: nil,
            after: nil,
            before: nil,
            runID: nil)
        else { return }

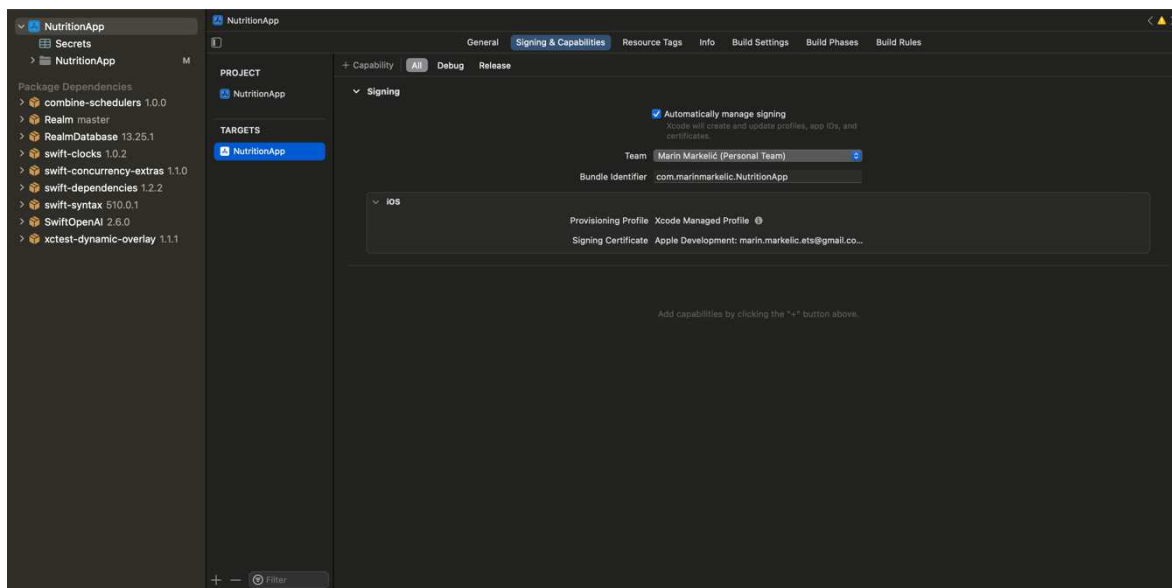
    let conversationMessages = messages.data.map { Message(from: $0) }
    let conversation = Conversation(
        id: conversationId,
        assistantId: nutritionAssistantId,
        messages: conversationMessages)
    conversationSubject.send(conversation)
}

```

Sl. 4.6 Metoda za dohvaćanje povijesti razgovora

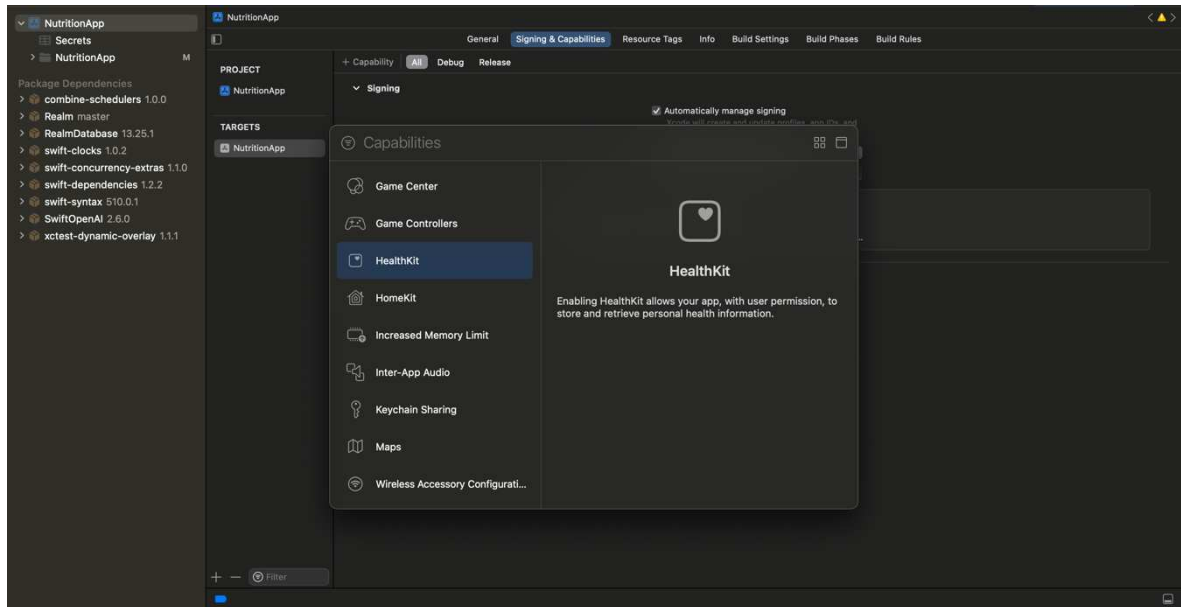
4.3. HealthKit

Prije što možemo početi raditi sa HealthKit-om moramo ga dodati u aplikaciju. U Xcode-u je potrebno odabrati projekt i željeni *target* te pod opcijom *Signing & Capabilities* dodati HealthKit sposobnost. Na slici (Sl. 4.7) možemo vidjeti izbornik na kojemu odabiremo sposobnosti.



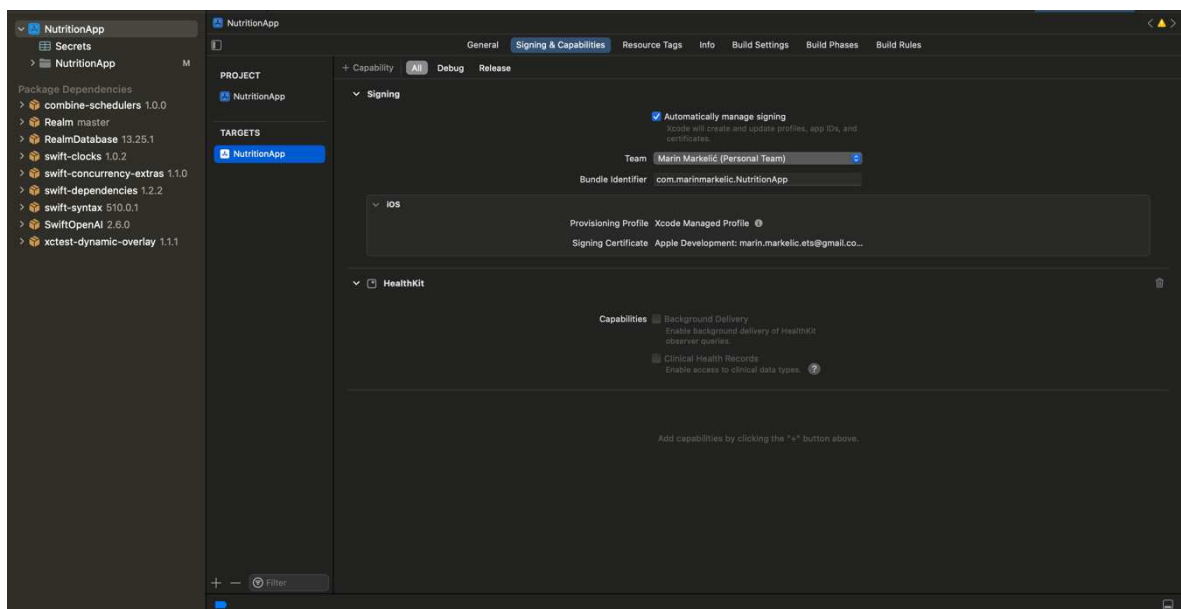
Sl. 4.7 Xcode izbornik za dodavanje sposobnosti

Pritiskom na gumb „+ Capability“ otvara se prozor vidljiv na slici (Sl. 4.8) sa svim dodatnim sposobnostima koje se mogu dodati u aplikaciju. Potrebno je pronaći HealthKit i odabrati ga.



Sl. 4.8 HealthKit u prozoru za dodavanje sposobnosti

Zatim možemo vidjeti da se HealthKit pojavio u prozoru kao na slici (Sl. 4.9). Nakon što smo dodati HealthKit sposobnost možemo početi s implementacijom.



Sl. 4.9 Xcode izbornik za dodavanje sposobnosti nakon dodavanja HealthKit-a

Nakon što je u aplikaciju dodana HealthKit sposobnost možemo koristiti metode istoimenog razvojnog okvira. Bitno je napomenuti da HealthKit nije dostupan na svim platformama i

verzijama operativnih sustava kao naprimjer iPadOS 16 ili sve verzije macOSa. Dostupnost HealthKita se može provjeriti metodom `isHealthDataAvailable()`.

Klasa koja se brine za komunikaciju sa HealthKit razvojnim okruženjem je `HealthKitService`.

4.3.1. Stvaranje repozitorija i traženje pristupa

Najvažniji objekt HealthKit razvojnog okruženja je `HKHealthStore`. To je klasa preko koje možemo pristupati svim podacima kojima HealthKit upravlja. Nakon što smo stvorili instancu `HKHealthStore` bitno je zatražiti dozvolu za sve podatke koje aplikacija čita ili mijenja. U aplikaciji ovog projekta nas zanimaju:

- Spol
- Visina
- Težina
- Datum rođenja
- Dnevna potrošnja kalorija u mirovanju
- Dnevna potrošnja kalorija u aktivnostima

Tipovi koje aplikacija čita su definirani u setu `dataTypes` i prikazani na slici (Sl. 4.10).

```
private let dataTypes: Set = [  
    HKQuantityType(.height),  
    HKQuantityType(.bodyMass),  
    HKQuantityType(.basalEnergyBurned),  
    HKQuantityType(.activeEnergyBurned),  
    HKCharacteristicType(.biologicalSex),  
    HKCharacteristicType(.dateOfBirth),  
]
```

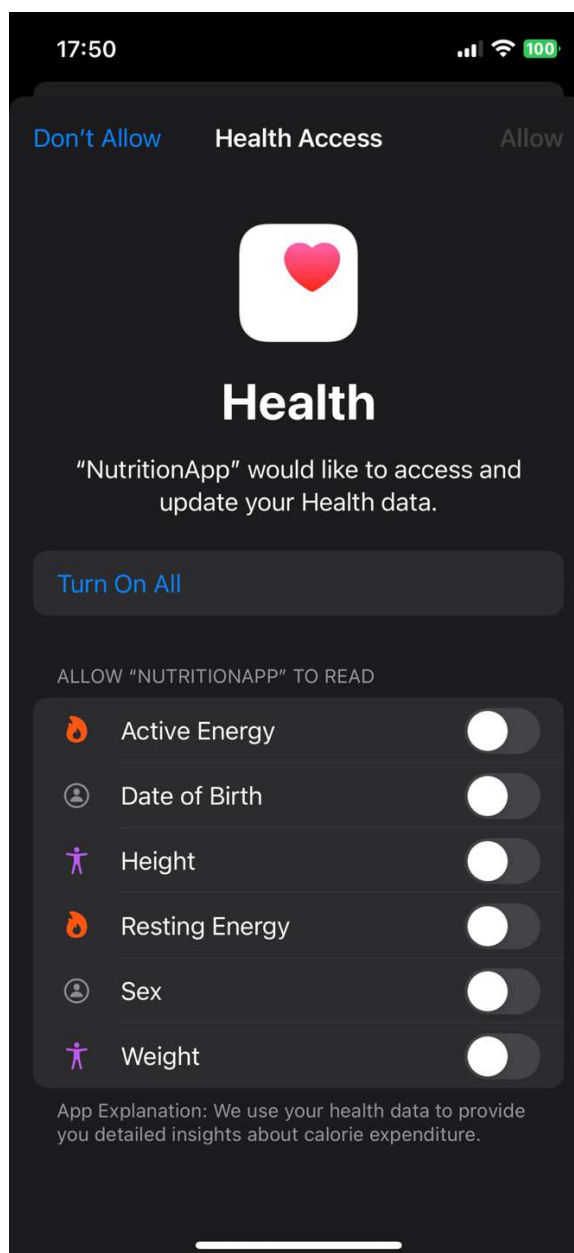
Sl. 4.10 Definiranje tipova koje će aplikacija čitati iz HealthKit-a

Zatim se u inicijalizatoru klase `HealthKitService` poziva funkcija `requestAccess()` vidljiva na slici (Sl. 4.11). Ona prima tri parametra. To su `toShare`, koji je prazan jer ne želimo pisati u repozitorij, `read` u koji predajemo niz tipova podataka za čitanje i anonimnu funkciju koja vraća informaciju o uspješnosti koju ignoriramo.

```
private func requestAccess() {
    if HKHealthStore.isHealthDataAvailable() {
        store.requestAuthorization(toShare: nil, read: dataTypes) { _, _ in }
    }
}
```

Sl. 4.11 Funkcija za traženje pristupa HealthKit podacima

Pri pozivanju ove metode pojavi se dijalog u kojemu korisnik eksplicitno odabire kojim podacima aplikacija može pristupiti (Sl. 4.12). Korisnik može odbiti čitanje svih podataka te aplikacija nastavlja funkcionirati sa smanjenom funkcionalnošću.



Sl. 4.12 Dijalog za odabir informacija koje aplikacija može čitati

4.3.2. Čitanje statičkih podataka

Prvo mjesto korištenja HealthKit-a je na pogledu korisničkih karakteristika. Ovdje se, umjesto da ih korisnik mora ručno upisivati, mogu učitati podaci o spolu, dobi, težini i visini iz HealthKit-a. Ovo su karakteristični podaci koji se učitavaju pozivom metode te je za svaki podatak definirana posebna funkcija. Funkcije za čitanje karakterističnih podataka su vidljive na slici (Sl. 4.13).

```
func fetchSex() -> HKBiologicalSex? {
    try? store.biologicalSex().biologicalSex
}

func fetchDateOfBirth() -> Date? {
    try? store.dateOfBirthComponents().date
}

func fetchHeight() async -> Double? {
    await fetchData(for: HKQuantityType(.height))?.doubleValue(for: .meterUnit(with: .centi))
}

func fetchWeight() async -> Double? {
    await fetchData(for: HKQuantityType(.bodyMass))?.doubleValue(for: .gramUnit(with: .kilo))
}
```

Sl. 4.13 Funkcije za čitanje karakterističnih podataka iz HealthKita

Ovi podaci se agregiraju u HealthKitUseCase-u i pretvaraju u strukturu koju aplikacija prikazuje.

Ako korisnik nije dozvolio pristup nekoj od vrijednosti funkcija za tu vrijednost vraća vrijednost `nil` i korisnik ju mora ručno unijeti.

4.3.3. Čitanje promjenjivih podataka

Drugo mjesto korištenja HealthKit-a je početni pogled. Ovdje se s ostalim podacima o danu prikazuje trenutni iznos potrošenih kalorija. Kako bi iz HealthKit-a dobili taj iznos moramo kombinirati vrijednosti potrošenih kalorija u mirovanju i kalorija potrošenih kroz aktivnosti. Ove dvije vrijednosti nisu karakteristične vrijednosti i mijenjaju se s vremenom pa je potrebno koristiti dugotrajne upite.

Dugotrajni upiti se kreiraju pozivom metode `createCalorieQueries()` pri inicijalizaciji `HealthKitService`-a prikazanu slikom (Sl. 4.14). Ovom metodom kreiramo dva dugotrajna upita, jedan za svaku vrijednost koju čitamo i svakom upitu predajemo anonimnu funkciju koja će se pozvati kada se vrijednost promijeni. U anonimne funkcije je predana funkcija `readBurnedEnergy()`.

```

func createCalorieQueries() {
    let activeEnergyType = HKQuantityType(.activeEnergyBurned)
    let activeEnergyQuery = HKObserverQuery(sampleType: activeEnergyType, predicate: nil) { [weak self] _, _, _ in
        self?.readBurnedEnergy()
    }
    self.activeEnergyQuery = activeEnergyQuery

    let basalEnergyType = HKQuantityType(.basalEnergyBurned)
    let basalEnergyQuery = HKObserverQuery(sampleType: basalEnergyType, predicate: nil) { [weak self] _, _, _ in
        self?.readBurnedEnergy()
    }
    self.basalEnergyQuery = basalEnergyQuery

    store.execute(activeEnergyQuery)
    store.execute(basalEnergyQuery)
}

```

Sl. 4.14 Kreiranje dugotrajnih upita

Funkcija `readBurnedEnergy()` vidljiva na slici (Sl. 4.15) izvršava dva obična upita. U svakom upitu definiramo tip vrijednosti koje želimo čitati i predajemo predikat koji definira da vrijednosti moraju biti od trenutnog dana. Zatim se upiti izvršavaju. Nakon što oba upita završe provjeravamo postoje li obje vrijednosti i ako postoje zbrajamo ih i šaljemo `burntCaloriesSubject`-u. On je instanca klase razvojnog sučelja `Combine`, Appleovog okvira za objavi-pretplati paradigmu. Vrijednost se propagira do početnog pogleda i osvježava svaki put kada funkcija `readBurnedEnergy()` uspije pročitati vrijednosti.

```

func readBurnedEnergy(for date: Date = .now) {
    let dispatchGroup = DispatchGroup()
    let startOfDay = Calendar.current.startOfDay(for: date)
    let endOfDay = Calendar.current.date(byAdding: .day, value: 1, to: startOfDay)!
    let predicate = HKQuery.predicateForSamples(withStart: startOfDay, end: endOfDay, options: .strictStartDate)

    var basalEnergyBurned: Double?
    var activeEnergyBurned: Double?

    dispatchGroup.enter()
    let basalEnergyType = HKQuantityType(.basalEnergyBurned)
    let basalEnergyQuery = HKStatisticsQuery(
        quantityType: basalEnergyType,
        quantitySamplePredicate: predicate,
        options: .cumulativeSum
    ) { (_, result, _) in
        guard let result = result?.sumQuantity() else {
            dispatchGroup.leave()
            return
        }

        basalEnergyBurned = result.doubleValue(for: HKUnit.kilocalorie())
        dispatchGroup.leave()
    }
    store.execute(basalEnergyQuery)

    dispatchGroup.enter()
    let activeEnergyType = HKQuantityType(.activeEnergyBurned)
    let activeEnergyQuery = HKStatisticsQuery(
        quantityType: activeEnergyType,
        quantitySamplePredicate: predicate,
        options: .cumulativeSum
    ) { (_, result, _) in
        guard let result = result?.sumQuantity() else {
            dispatchGroup.leave()
            return
        }

        activeEnergyBurned = result.doubleValue(for: HKUnit.kilocalorie())
        dispatchGroup.leave()
    }
    store.execute(activeEnergyQuery)

    dispatchGroup.notify(queue: .main) { [weak self] in
        guard
            let self,
            let basalEnergyBurned,
            let activeEnergyBurned
        else { return }

        let energy = basalEnergyBurned + activeEnergyBurned
        burntCaloriesSubject.send(energy)
    }
}

```

Sl. 4.15 Čitanje vrijednosti za potrošenu energiju

Također je bitno zaustaviti dugotrajne upite kada ih više ne trebamo koristiti. Ovo se radi prije deinicijalizacije u funkciji `deinit()` prikazanom slikom (Sl. 4.16).

```

deinit {
    guard
        let activeEnergyQuery,
        let basalEnergyQuery
    else { return }

    store.stop(activeEnergyQuery)
    store.stop(basalEnergyQuery)
}

```

Sl. 4.16 Zaustavljanje dugotrajnih upita pri deinicijalizaciji

4.4. Pohrana podataka

Za pohranu je u aplikaciji zadužena klasa `StorageService`. Ona je zadužena za pohranu podataka u `UserDefaults` i `Realm` baze podataka.

Korištenje `UserDefaults`-a je jednostavno te se za spremanje koriste funkcije `setValue()`, koja prima vrijednost i ključ ili `setValuesForKeys()`, koja prima rječnik ključeva i vrijednosti. Za dohvaćanje vrijednosti se koristi metoda `object()` koja prima ključ.

Sve `UserDefaults` metode klase `StorageService` vidljive su na slici (Sl. 4.17).

```

func save(value: Any?, for key: String) {
    UserDefaults.standard.setValue(value, forKey: key)
}

func setValuesForKeys(_ keyedValues: [String : Any]) {
    UserDefaults.standard.setValuesForKeys(keyedValues)
}

func object(for key: String) -> Any? {
    UserDefaults.standard.object(forKey: key)
}

```

Sl. 4.17 Metode za dohvaćanje i spremanje podataka u `UserDefaults`

Realm baza podataka se kreira instanciranjem objekta `Realm` kao na slici (Sl. 4.18). Ovim kôdom će se probati kreirati baza podataka, te ako kreiranje bude neuspješno resetirati konfiguracija baze podataka Realm funkcijom `resetRealm()` na slici (Sl. 4.19) i pokušati ponovo. Ako i resetiranje bude neuspješno aplikacija se neće moći otvoriti.

```
private var realm: Realm {
    guard let realm = try? Realm() else {
        resetRealm()
        return try! Realm()
    }

    return realm
}
```

Sl. 4.18 Kreiranje instance Realm baze podataka

```
private func resetRealm() {
    do {
        let realmURL = Realm.Configuration.defaultConfiguration.fileURL!
        let realmURLs = [
            realmURL,
            realmURL.appendingPathExtension("lock"),
            realmURL.appendingPathExtension("management"),
            realmURL.appendingPathExtension("note"),
        ]

        for url in realmURLs {
            try FileManager.default.removeItem(at: url)
        }
    } catch {
        Swift.print("Error resetting Realm: \(error)")
    }
}
```

Sl. 4.19 Resetiranje konfiguracije Realm baze podataka

Podatkovni modeli su u Realm-u definirani kao klase koje nasljeđuju klasu `Object`. Parametri ovih klasa se automatski spremaju u bazu podataka. Primjer je klasa `MealStorageViewModel` koja sadrži informacije o obroku te listu njegovih sastojaka prikazana na slici (Sl. 4.20).


```

class MealStorageViewModel: Object {

    @Persisted(primaryKey: true) var id: String
    @Persisted var name: String
    @Persisted var date: Date
    @Persisted var items: List<NutritionalItemStorageViewModel>

    override init() {}

    convenience init(from model: MealViewModel) {
        self.init()

        id = model.id
        name = model.name
        date = model.date

        model.items.forEach { item in
            items.append(NutritionalItemStorageViewModel(from: item))
        }
    }
}

```

Sl. 4.20 Primjer objekta u Realm bazi podataka

Spremanje i dohvaćanje objekata je u Realm-u jednostavno. Pri spremanju, uređivanju i brisanju je potrebno pozvati Realm-ovu metodu `write()` koja prima *closure* te u njemu definirati akcije koje se trebaju dogoditi. Metoda `write()` je Realm-ov mehanizam osiguravanja četiri svojstva transakcija u bazama podataka, a to su cjelovitost, dosljednost, izolacija i trajnost (*ACID properties*). Tijekom ove operacije, sve promjene se izvode u zatvorenom bloku i izvršavaju cjelovito, što znači da ili sve promjene budu uspješno primijenjene ili nijedna neće biti, čime se osigurava valjanost. Ovaj mehanizam također osigurava da su podaci uvijek u dosljednom stanju, izolirani od drugih operacija, te trajno pohranjeni čak i u slučaju nepredviđenih prekida rada sistema.

Kod čitanja je potrebno pozvati metodu `objects()` koja prima željeni tip podataka i vraća rezultate. Moguće je raditi dodatne operacije nad rezultatima kao što su filtriranje ili sortiranje.

Primjeri spremanja, brisanja i dohvaćanja objekata klase `MealStorageViewModel` prikazani su slikama (Sl. 4.21) (Sl. 4.22).

```

func save(meal: MealViewModel) {
    try! realm.write {
        let model = MealStorageViewModel(from: meal)
        realm.add(model)
    }
}

func delete(meal: MealViewModel) {
    guard let object = realm.object(ofType: MealStorageViewModel.self, forKey: meal.id) else { return }

    try! realm.write {
        realm.delete(object)
    }
}

```

Sl. 4.21 Spremanje i brisanje obroka

```

func fetchMeals(with date: Date) -> [MealViewModel] {
    let targetDateComponents = Calendar.current.dateComponents([.day, .month, .year], from: date)

    let meals = realm.objects(MealStorageViewModel.self).filter {
        let mealDateComponents = Calendar.current.dateComponents([.day, .month, .year], from: $0.date)

        let isConsumedOnTheSameDay = mealDateComponents.day == targetDateComponents.day &&
            mealDateComponents.month == targetDateComponents.month &&
            mealDateComponents.year == targetDateComponents.year

        return isConsumedOnTheSameDay
    }

    return meals.map { MealViewModel(from: $0) }
}

```

Sl. 4.22 Dohvaćanje obroka za određeni dan

4.5. Računanje preporučenih kalorija i nutrijenata

4.5.1. Računanje preporučenih kalorija

The Mifflin-St Jeor formula je formula za računanje stope metabolizma u mirovanju (RMR) također zvanu utrošak energije u mirovanju (REE). Ona je definirana kao vrijednost koju tijelo potroši u potpunom mirovanju [13]. Od raznih formula za računanje utroška energije u mirovanju se pokazala kao najpreciznija u mogućnosti da odredi vrijednost REE unutar 10% izmjerene [15].

Ova formula ovisi o dobi, visini i težini te je različita za žene i muškarce. Formula za muškarce se može vidjeti izrazom (1) dok formula za žene izrazom (2).

$$10 \cdot \text{težina [kg]} + 6.25 \cdot \text{visina [cm]} + 5 \cdot \text{dob [godine]} - 161 \quad (1)$$

$$10 \cdot \text{težina [kg]} + 6.25 \cdot \text{visina [cm]} + 5 \cdot \text{dob [godine]} + 5 \quad (2)$$

Za računanje ukupnog dnevnog utroška energije (TDEE), koji predstavlja dnevni preporučeni unos kalorija, prethodnu vrijednost REE dobivenu *Mifflin-St Jeor* formulom moramo pomnožiti sa faktorom aktivnosti (3).

$$REE * faktor_aktivnosti \quad (3)$$

Aktivnost korisnika se dijeli na pet skupina koje zajedno sa faktorima aktivnosti glase:

- Neaktivan – 1.2
- Malo aktivan – 1.375
- Srednje aktivan – 1.55
- Aktivan – 1.725
- Veoma aktivan – 1.9

4.5.2. Računanje potrebnih nutrijenata

Dnevne preporuke unosa nutrijenata su bazirane na dijetalnim referentnim unosima (DRI) na temelju dobi i težine. Pošto su to generalne vrijednosti namijenjene za širok spektar ljudi stvarne vrijednosti za individualce se u malom iznosu razlikuju od iznosa generiranog formulama.

Proteini

Preporučeni dnevni unos proteina se računa formulom (4) koja množi težinu korisnika sa faktorom koji je definiran sljedećom tablicom (Tablica 4.1 Faktor kod računanja preporučenog dnevnog unosa proteina) [16].

$$težina[kg] * faktor_za_proteine \quad (4)$$

Tablica 4.1 Faktor kod računanja preporučenog dnevnog unosa proteina

	Muškarci	Žene
1-3 godine	1.05	1.05

4-13 godina	0.95	0.95
14-18 godina	0.85	0.85
Više od 19 godina	0.8	0.8

Ugljikohidrati

Dnevni potrebni unos ugljikohidrata se pokazao jednak za sve dobne skupine kod muškaraca i žena te iznosi 130 grama [16].

Masti

Računanje potrebnog unosa masti je definiran kao raspon vrijednosti postotka ukupnog dnevnog utroška energije sljedećim iznosima [16]:

- 1-3 godine - 30-40%
- Više od 4 godine – 20-35%

U aplikaciji se uzima prosječan postotak ovog raspona te se množi sa ukupnim dnevnim utroškom energije (TDEE). Nakon množenja dobivamo iznos koji predstavlja broj kalorija koji je potrebno unijeti mastima. Kako bi dobili iznos masti u gramima potrebno je prethodni rezultat podijeliti sa 9 jer svaki gram masti iznosi 9 kalorija. Računanje potrebnog unosa masti je vidljivo formulom (5).

$$\frac{\left(\frac{\text{postotak_dnevnog_ukupnog_utroška_energije}}{100} * TDEE \right)}{9} \quad (5)$$

Vlakna

Dnevni preporučeni unos vlakana je izražen kao 14 grama na svakih 1000 kalorija [16]. Kako bi izračunali tu vrijednost potrebno je ukupni dnevni utrošak energije (TDEE) podijeliti sa 1000 i zatim pomnožiti sa 14 što se može vidjeti formulom (6).

$$\frac{TDEE}{1000} * 14 \quad (6)$$

Šećer

Dnevni preporučeni unos šećera iznosi 130 grama za sve dobne skupine za oba spola [16].

Natrij

Dnevni preporučeni unos natrija je izražen po dobnim skupinama [17]:

- 1-3 godine – 800mg
- 4-8 godina – 1000mg
- 9-18 godina – 1500mg
- Više od 18 godina – 2300mg

Kalij

Dnevni preporučeni unos kalija je izražen tablicom () [18]:

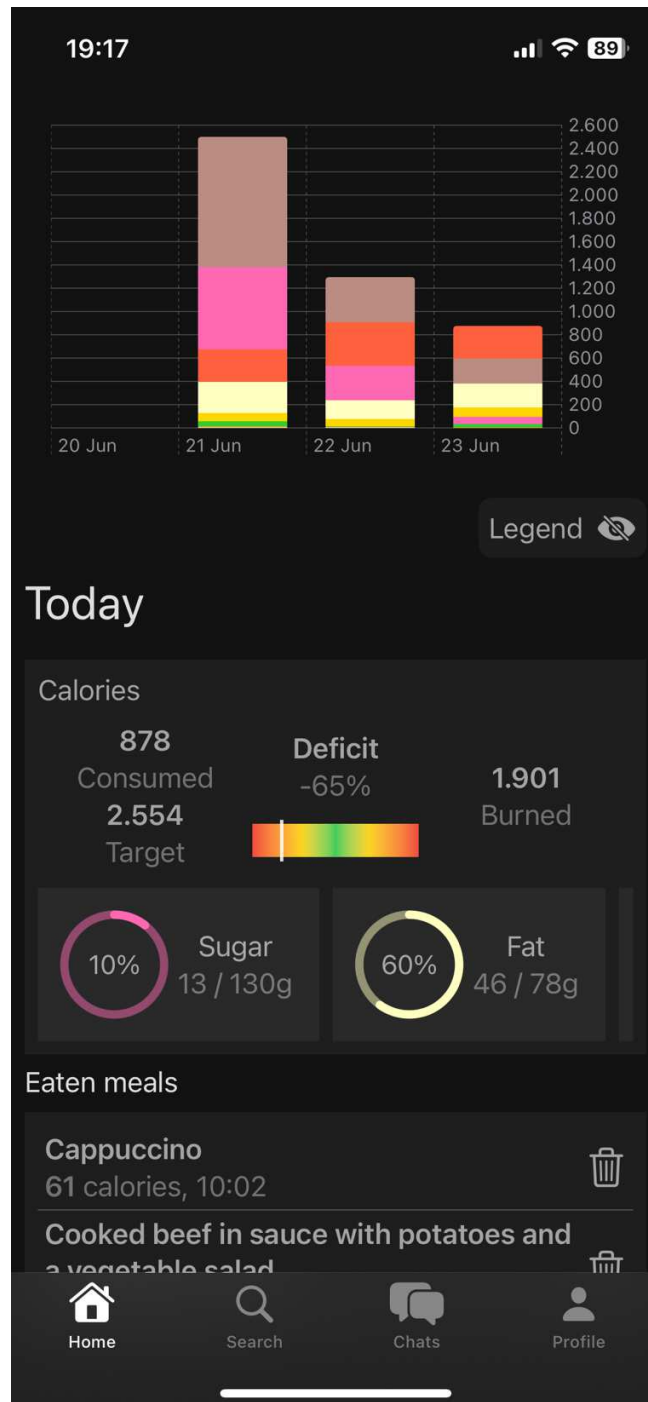
Tablica 4.2 Dnevni preporučeni iznos kalija

	Muškarci	Žene
1-3 godine	2000mg	2000mg
4-8 godina	2300mg	2300mg
9-13 godina	2500mg	2300mg
14-18 godina	3000mg	2300mg
Više od 18 godina	3400mg	2600mg

5. Implementacija ekrana

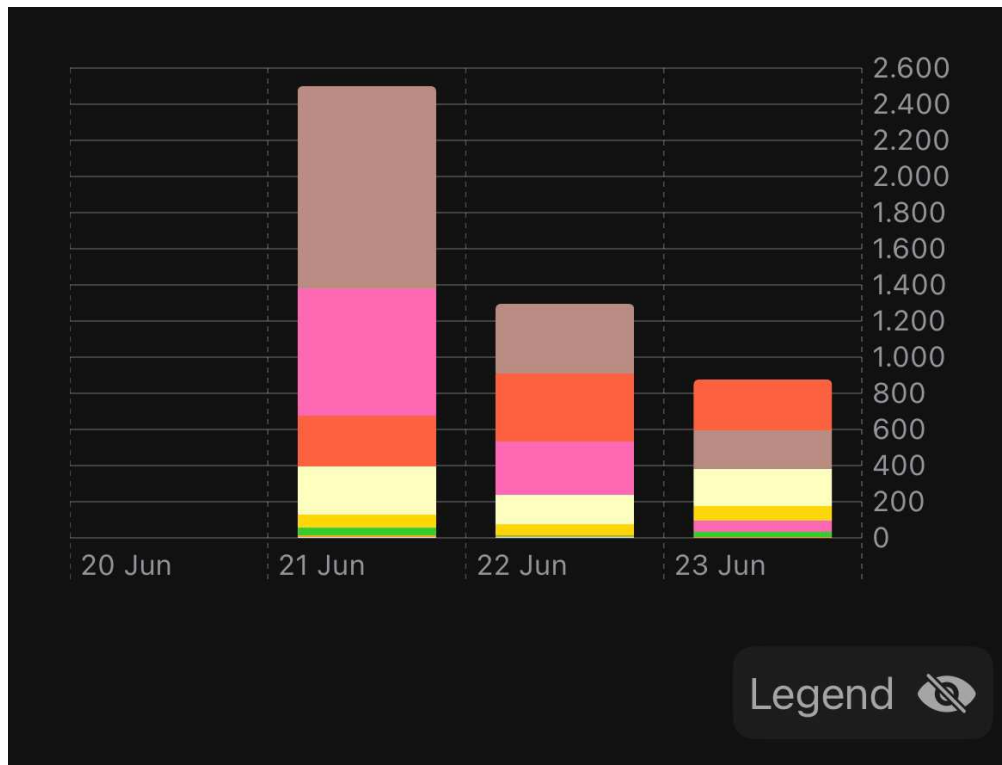
5.1. Početni pogled

Početni pogled je zadužen za prikazivanje informacija o korisnikovoj prehrani prikazan slikom (Sl. 5.1). Sastoji se od skupine pogleda od koji svaki korisniku daje bitne informacije.

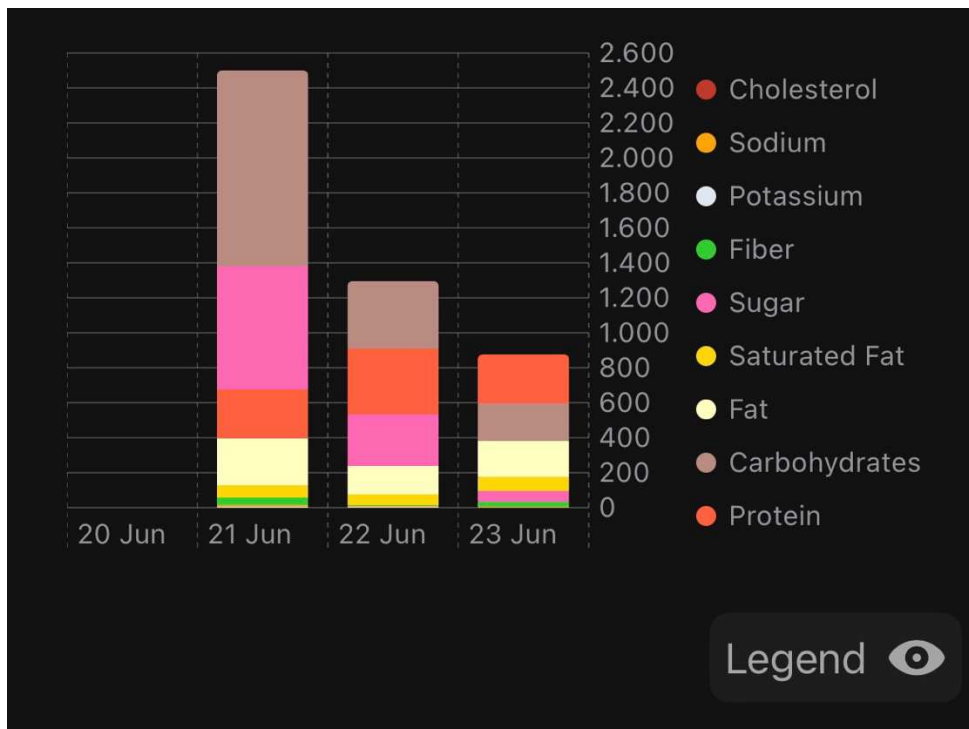


Sl. 5.1 Početni pogled

Prvi pogled ovog ekrana je graf prikazan slikom (Sl. 5.2) koji korisniku daje uvid u dnevno konzumiranje kalorija i nutrijenata. Graf je implementiran koristeći SwiftUI-ov Charts razvojni okvir koji pruža razne načine vizualizacije podataka u aplikacijama. Svaki stupac na grafu predstavlja jedan dan te su dnevno konzumirani nutrijenti označeni bojama na stupcu i poredani u odnosu od većeg prema manjem. Pritiskom na gumb *Legend* se prikazuje legenda grafa koja korisniku govori koji nutrijent označava svaka boja (Sl. 5.3).

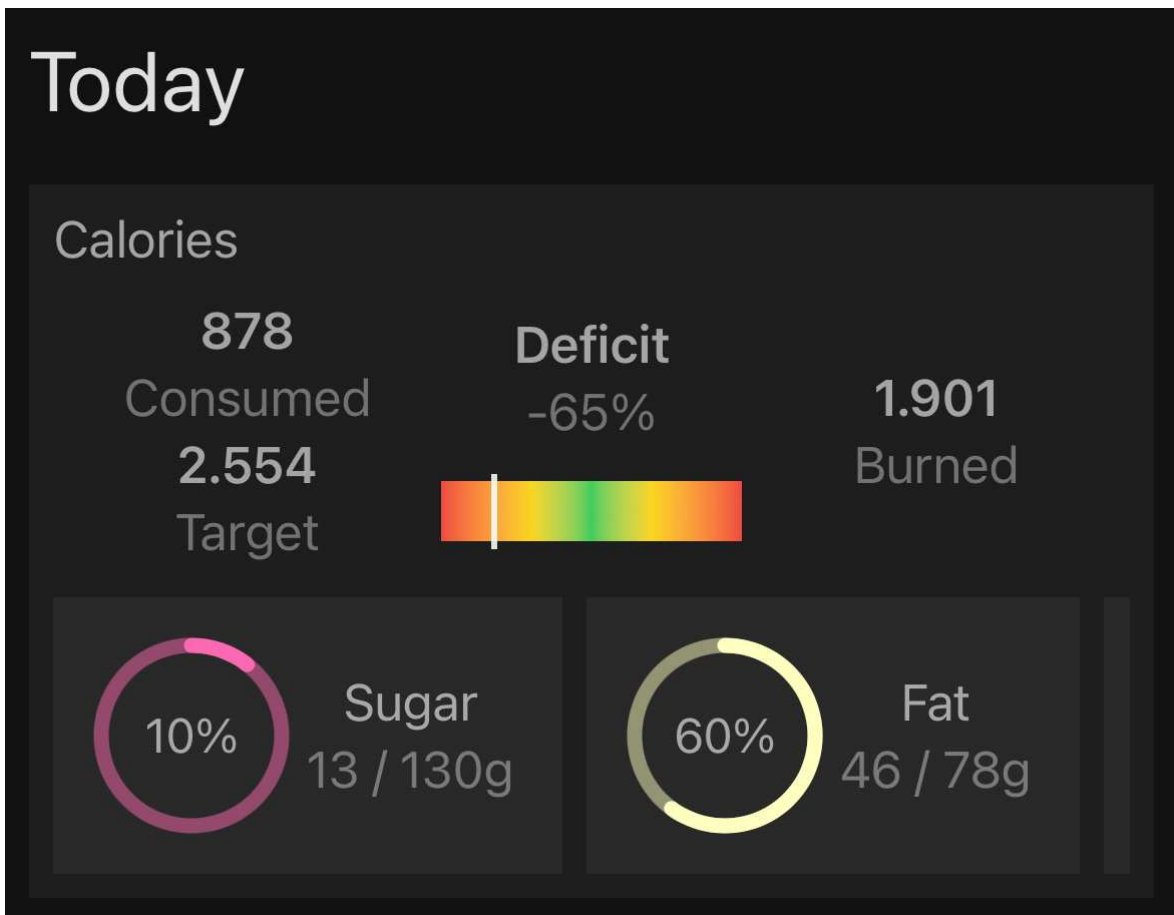


Sl. 5.2 Graf dnevnog unosa kalorija i nutrijenata bez legende



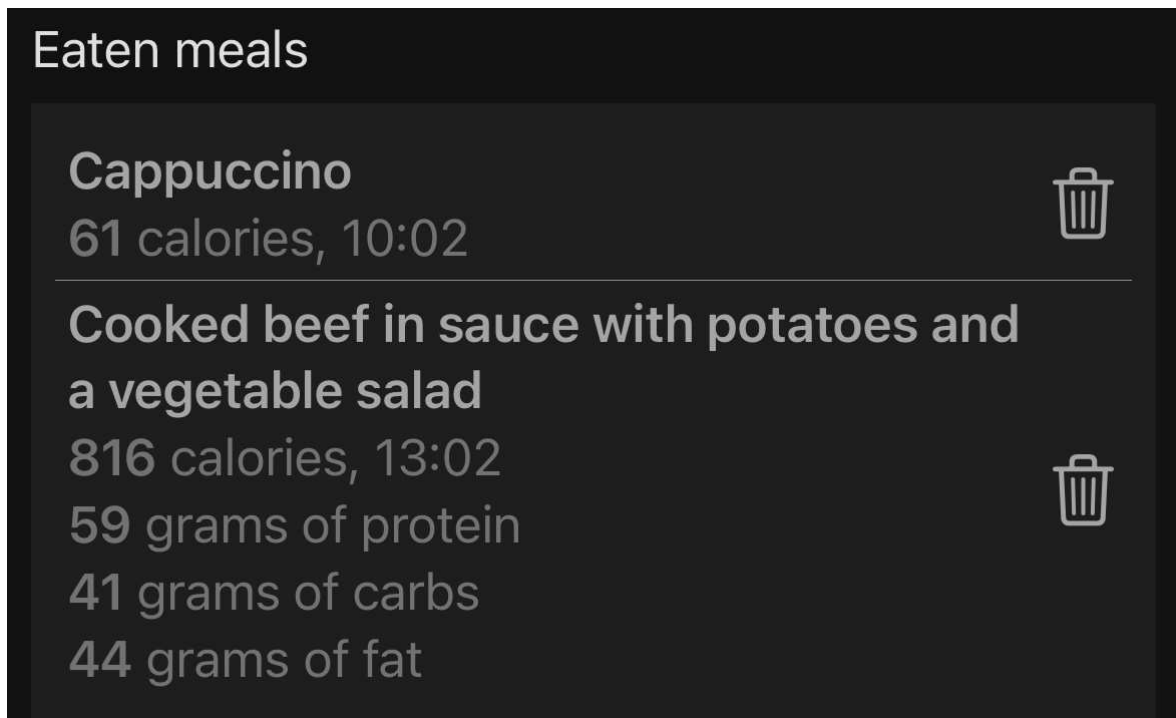
Sl. 5.3 Graf dnevnog unosa kalorija i nutrijenata sa legendom

Drugi pogled korisnika detaljnije informira o odabranom danu te je prikazan slikom (Sl. 5.4). Pri paljenju aplikacije su na ovom pogledu prikazane informacije o trenutnom danu, te se pritiskom na određeni stupac u grafu dnevnog unosa kalorija i nutrijenata može promijeniti odabrani dan. Kroz agregaciju podataka svih obroka u danu dobiva se ukupan broj kalorija koje je korisnik unio taj dan. Također se, uz kalorije, prati unesena količina svih nutrijenata. Koristeći podatke o fizičkim karakteristikama korisnika koje su unesene u pogled korisničkih karakteristika, računa se preporučeni dnevni iznos kalorija i nutrijenata za korisnika te se prikazuje na ekranu. Time korisnik može vidjeti koliko je u kalorijskom deficitu ili višku, te omjer konzumirane količine svakog nutrijenta s preporučenom.



Sl. 5.4 Pogled podataka za trenutni dan

Zadnji pogled ovog ekrana je pogled koji prikazuje sve konzumirane obroke za trenutni dan prikazan slikom (Sl. 5.5). On je jednostavna lista koja za svaki obrok prikazuje osnovne informacije kao što su ime, vrijeme, broj kalorija, proteine, ugljikohidrate i masti. Također, ovdje korisnik može izbrisati obrok ako je primijetio da je napravio pogrešku pritiskom na gumb za brisanje ili raširiti ćeliju pritiskom na ime čime se prikazuje više informacija za obrok.



Sl. 5.5 Pogled sa svim dnevnim obrocima sa normalnom i raširenom ćelijom

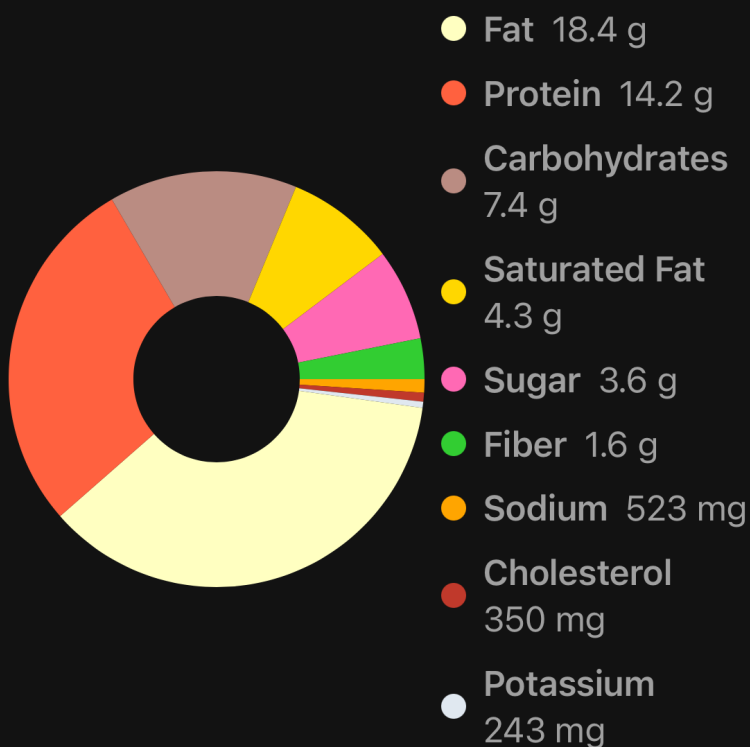
5.2. Pogled za pretraživanje

Pogled za pretraživanje prikazan slikom (Sl. 5.6) je zadužen za pretraživanje i dodavanje obroka u aplikaciju. Dizajniran je na intuitivan način na kojem su polja za upisivanje uvijek pri dnu ekrana i lako dostupna. Nakon što korisnik unese obrok i pritisne gumb za pretragu, informacije se prikazuju na više mjesta.

Omelette With A Vegetable Salad

246 calories

This meal is healthy as omelettes are a good source of protein and vegetables provide essential nutrients and fiber. Enjoy your meal!



Amount

Omelette: 100 g

Vegetable Salad: 100 g

Omelette with a vegetable salad



Home



Search



Chats



Profile

Sl. 5.6 Pogled za pretraživanje

Najbitnije informacije, koje korisnik lako razumije, prikazane su na vrhu pogleda. Prvi pogled, prikazan slikom (Sl. 5.7), sadrži ime obroka, broj kalorija i savjet. Savjet se generira pomoću velikog jezičnog modela koji zna koje je obroke korisnik konzumirao taj dan te na temelju toga formira odgovor.

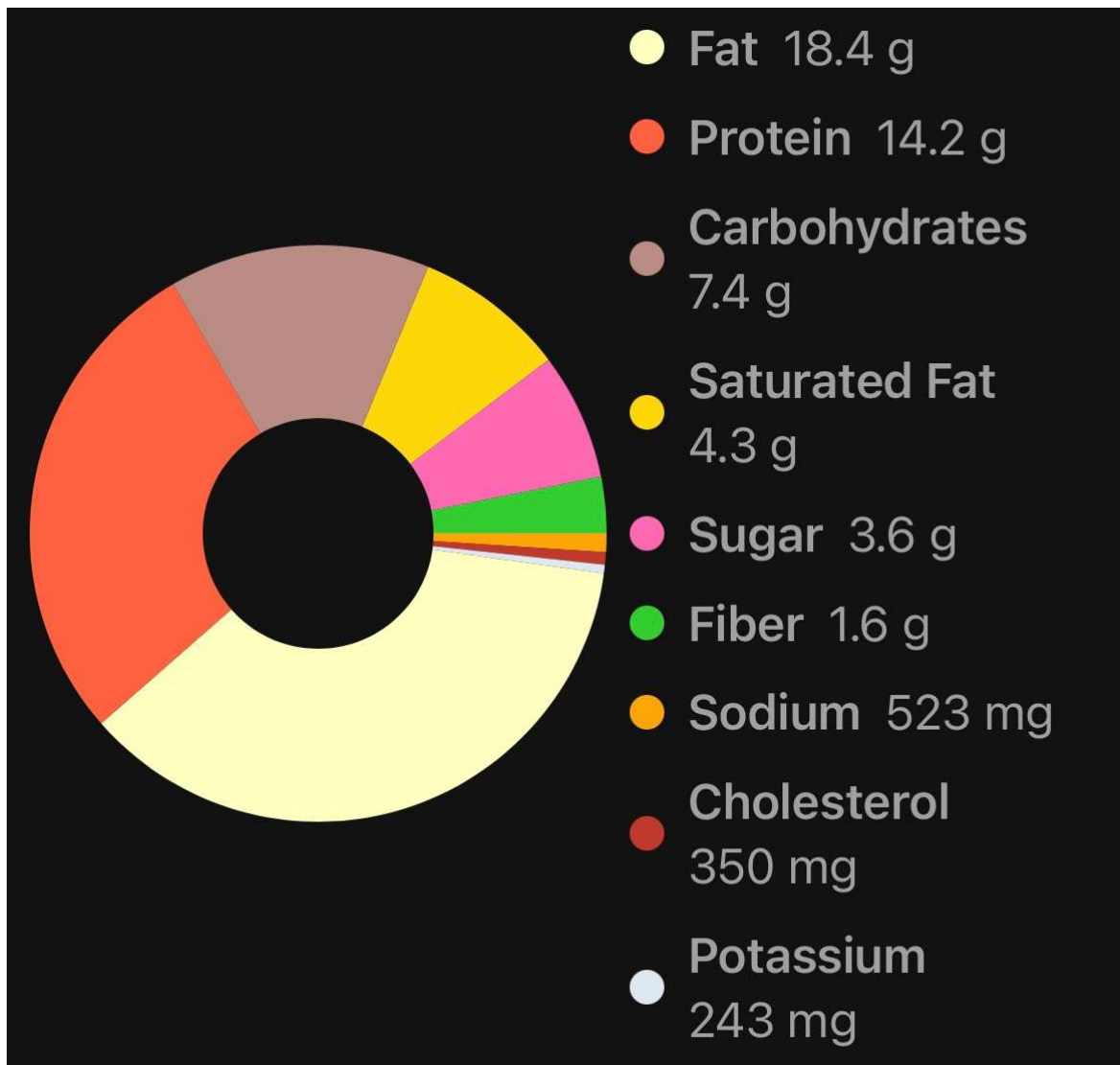
Omelette With A Vegetable Salad

246 calories

This meal is healthy as omelettes are a good source of protein and vegetables provide essential nutrients and fiber. Enjoy your meal!

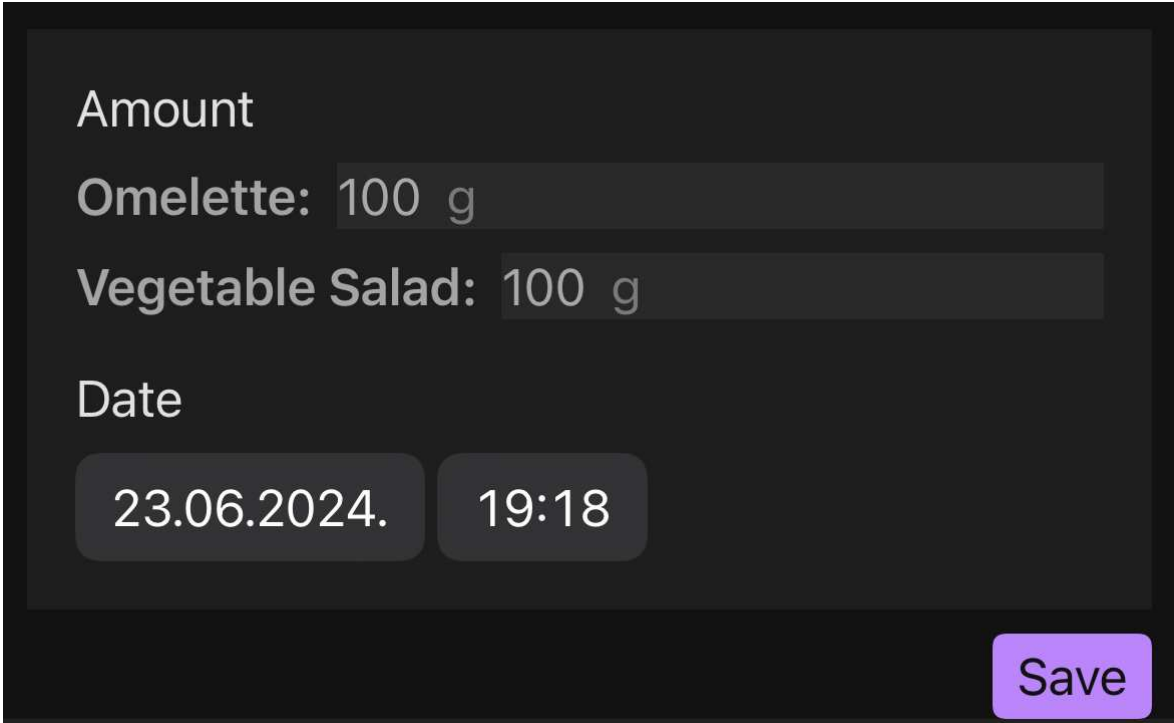
Sl. 5.7 Pogled s osnovnim informacijama obroka

Zatim slijede informacije koje su bitne, no zbog količine informacija teže za shvatiti, a to su količine nutrijenata u obroku. Kako bi korisnik mogao lako i intuitivno shvatiti ove informacije, razvijen je pogled koji te podatke vizualizira uz pomoć grafa i pored njega pruža informacije o točnom iznosu svakog nutrijenta. Pogled je prikazan slikom (Sl. 5.8).



Sl. 5.8 Graf sa iznosima nutrijenata

Na dnu ekrana je postavljen pogled u kojemu korisnik mijenja informacije o težini obroka te datumu i vremenu konzumacije prikazan slikom (Sl. 5.9). Ako obrok ima više namjernica korisnik će moći za svaku mijenjati težinu.



Amount

Omelette: 100 g

Vegetable Salad: 100 g

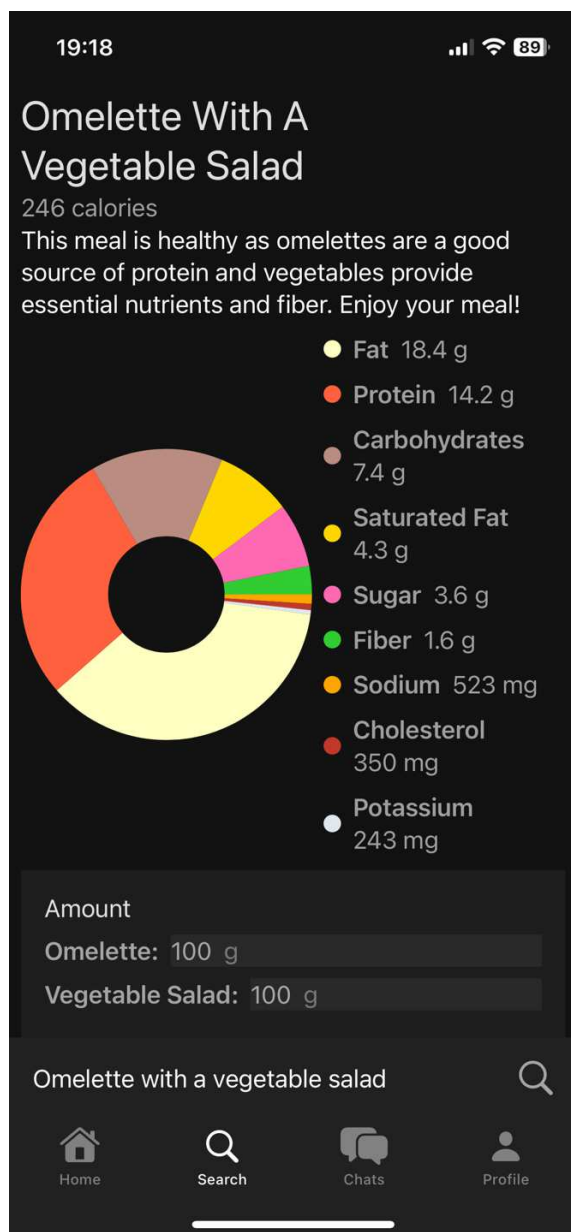
Date

23.06.2024. 19:18

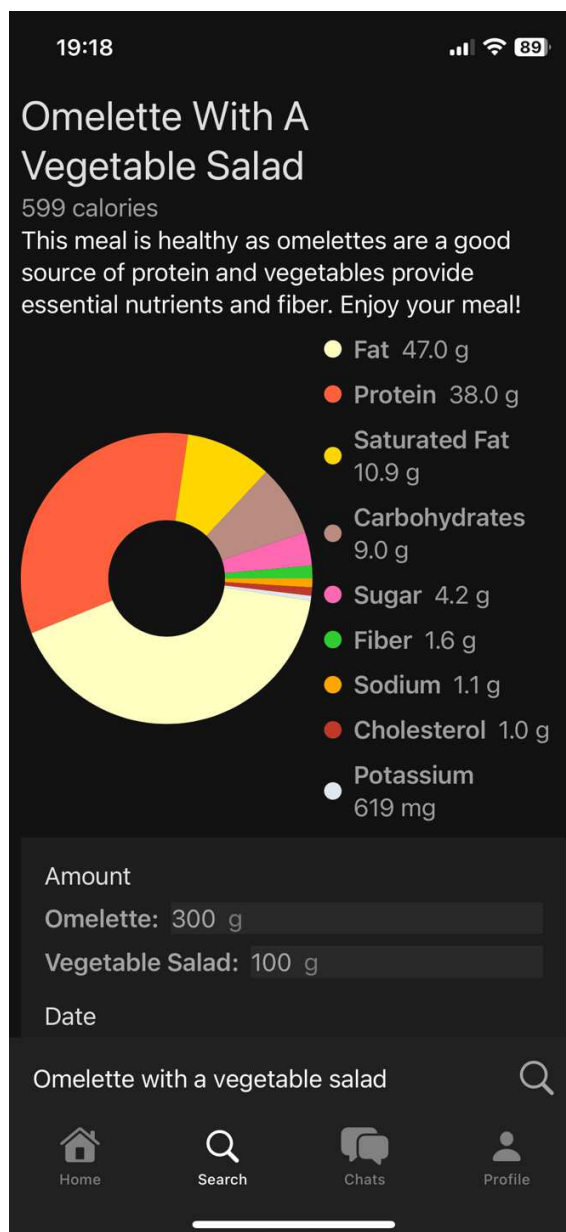
Save

Sl. 5.9 Pogled za mijenjanje težine i datuma

Promjenom težine u ovom pogledu se automatski mijenjaju sve druge vrijednosti na ostalim pogledima na ekranu. Usporedbom slika prije (Sl. 5.10) i poslije (Sl. 5.11) promjene se može vidjeti kako su se vrijednosti kalorija i nutrijenata, kao i graf promijenili. Promjene su automatske, za njih nije potrebno implementirati nikakvu dodatnu logiku jer se SwiftUI u pozadini brine u osvježavanju pogleda kada se vrijednost na njima promijeni.



Sl. 5.10 Ekran prije promjene težine omleta

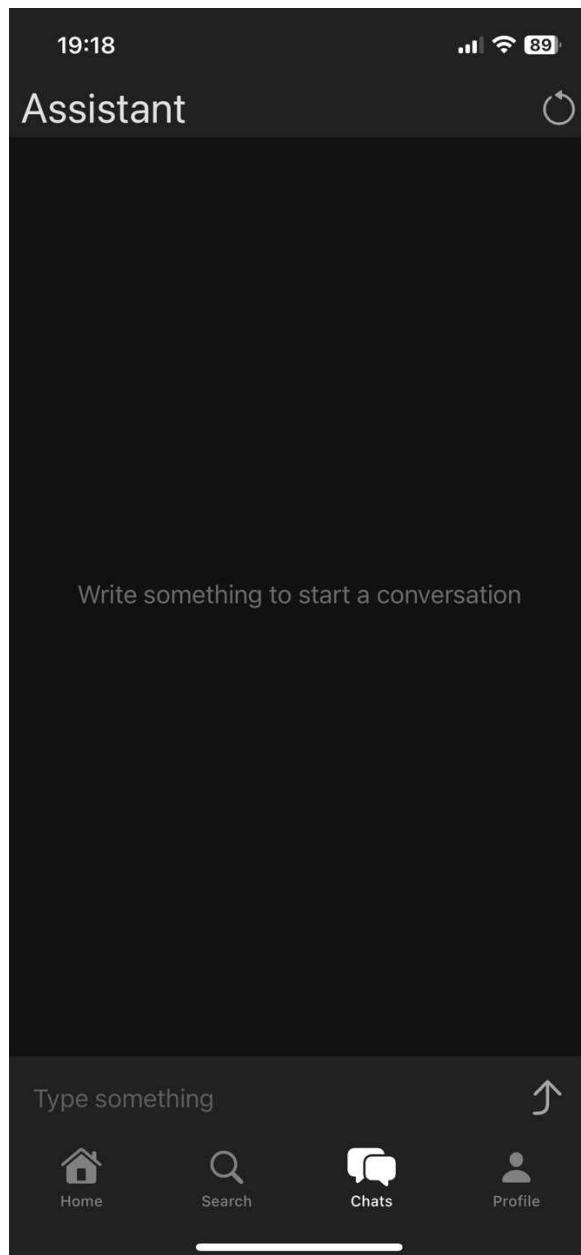


Sl. 5.11 Ekran nakon promjene težine omleta

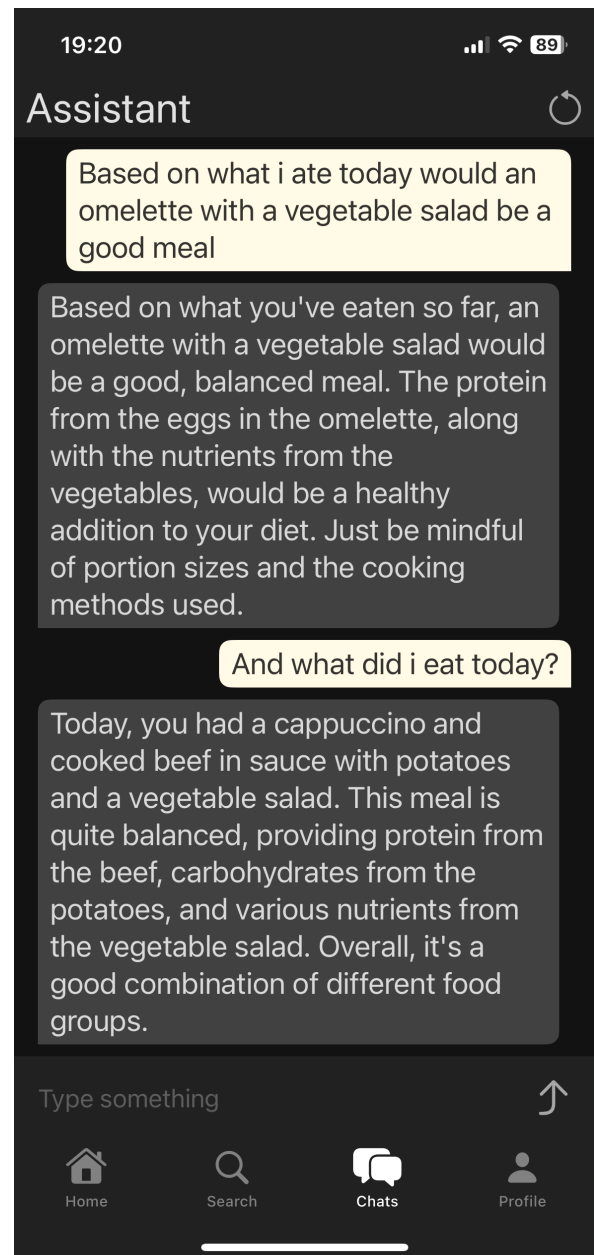
5.3. Pogled za razgovor

Pogled za razgovor, prikazan slikama (Sl. 5.12) i (Sl. 5.13), je implementiran kao klasičan ekran za razgovor gdje je polje za unos na dnu lako dostupno i ćelije od razgovora su poredane uzlazno, od najnovije do najstarije poruke. Ovaj ekran pruža korisniku mogućnost interakcije sa velikim jezičnim modelom koji se ponaša kao nutricionist. On ima pristup informacijama o prošlim korisnikovim obrocima u zadnja 3 dana. Korisnik ga zbog toga

može ispitivati o savjetima za prehranu, pomoći kod imenovanja obroka i raznim drugim stvarima.

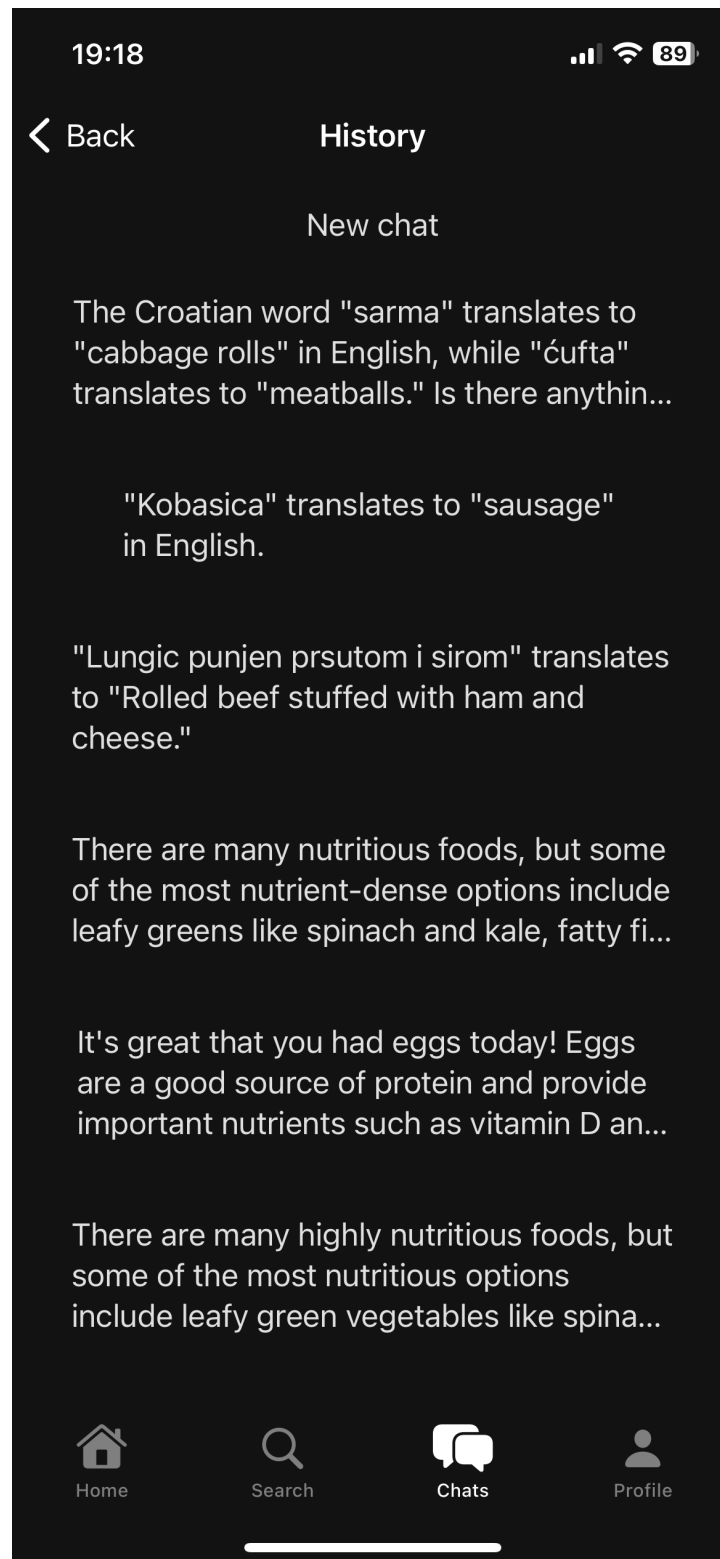


Sl. 5.12 Pogled za razgovor prije početka razgovora



Sl. 5.13 Pogled za razgovor sa porukama

Ovaj ekran također ima i povijest prikazanu slikom (Sl. 5.14), gdje korisnik može pregledati svoje prošle razgovore i vratiti se na njih ako se treba podsjetiti prijašnjeg razgovora ili ga nastaviti.



Sl. 5.14 Povijest razgovora

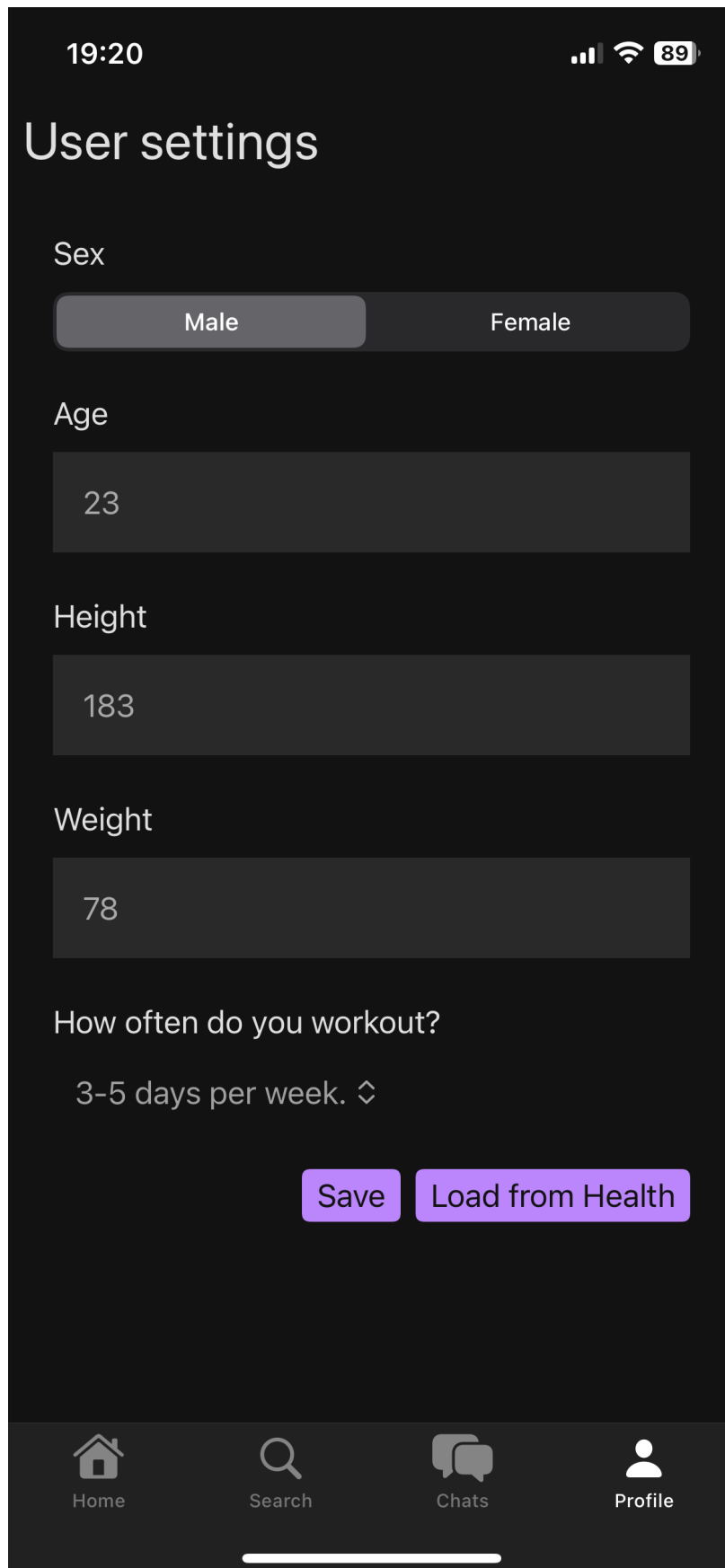
5.4. Pogled korisničkih karakteristika

Pogled korisničkih karakteristika je prikazan slikom (Sl. 5.15). Na ovom pogledu korisnik unosi svoje fizičke karakteristike kako bi aplikacija mogla izračunati dnevnu kalorijsku potrošnju i preporučene dnevne vrijednosti nutrijenata. Bez unosa ovih informacija početni pogled će imati umanjenu funkcionalnost jer pogled sa informacijama o trenutnom danu neće biti prikazan.

Karakteristike koje se definiraju su:

- Spol
- Dob
- Visina
- Težina
- Tjelesna aktivnost

Kako bi se olakšao unos, korisniku je omogućen uvoz svih informacija osim tjelesne aktivnosti iz HealthKit-a. Time, ako je korisnik već u aplikaciju Health koja je zadužena za prikaz svih informacija razvojnog okvira HealthKit na jednom mjestu unio svoje fizičke karakteristike, neće ih morati zasebno upisivati, već će biti preuzete iz HealthKit razvojnog okvira.



Sl. 5.15 Ekran korisničkih postavki

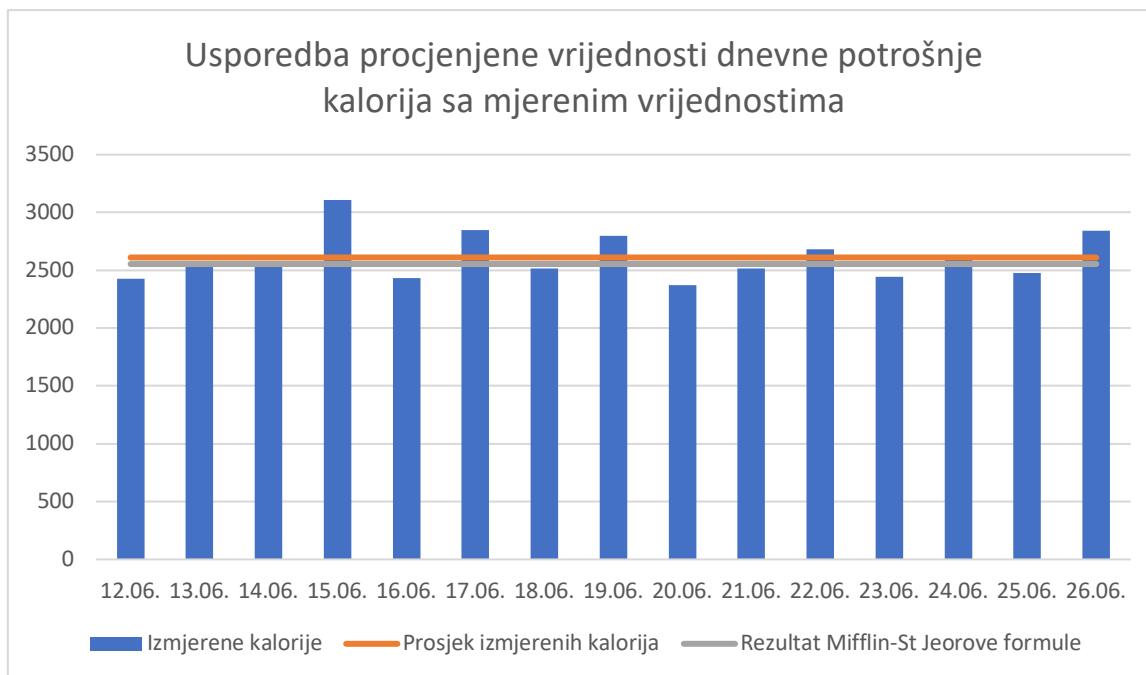
6. Rezultati i doradivanje

Korištenjem aplikacije možemo evaluirati točnost *Mifflin-St Jeorove* formule za računanje ukupnog dnevnog energetskeg utroška te ispitati korisnost velikih jezičnih modela u pružanju relevantnih informacija. Cilj je proći sve funkcionalnosti aplikacije, identificirati greške u kôdu i uočiti područja koja se mogu poboljšati. Posebna pažnja posvećuje se provjeri točnosti preporučenog dnevnog unosa kalorija te analizi odgovora velikog jezičnog modela.

6.1. Mjerenje dnevne potrošnje kalorija

Kako bi testirali točnost *Mifflin-St Jeorove* formule potrebno je usporediti njenu vrijednost sa podacima o dnevnim potrošenim kalorijama zabilježenim sa pametnim satom ili drugim uređajem koji ima sposobnost praćenja kalorija. Mjerenja prikazana u nastavku su zabilježena pametnim satom Apple Watch Series 7. On pruža precizne podatke o dnevnim aktivnostima i potrošenim kalorijama te nudi kvalitetne referentne točke za usporedbu podataka.

Podaci prikazani grafom su mjereni na način da je korisnik nosio pametni sat tokom cijelog dana i normalno radio sve aktivnosti. Možemo vidjeti da iznos varira od dana do dana ovisno o aktivnosti korisnika. Kada se izračuna prosjek izmjerenih vrijednosti dobivamo vrijednost koja je jako blizu procijenjene vrijednosti čime možemo zaključiti da je *Mifflin-St Jeorova* formula precizna i korisna.



6.2. Inženjerstvo uputa

Inženjerstvo uputa je proces dizajniranja i prerađivanja uputa kako bi od velikih jezičnih modela dobili odgovore strukturirane po želji. Ključan je za osiguravanje da model generira odgovore koji su relevantni, koherentni i poravnati sa funkcionalnim zahtjevima aplikacije.

6.2.1. Upiti kod pretraživanja

Kod pretraživanja obroka se veliki jezični model koristi za generiranje savjeta koji korisnika informira o zdravosti obroka. Veliki jezični model ima informaciju o konzumiranim obrocima za taj dan i obrok kojeg korisnik želi konzumirati. Ovdje želimo korisniku jasno prenijeti informacije bez nepotrebnog teksta koji zauzima dodatan prostor na ekranu. Testiranje se vršilo na način da se isti upit izvodio nekoliko puta za različite velike jezične modele i odgovori su se evaluirali po sljedećim kriterijima:

- Odgovor mora biti što kraći i direktno i jasno prenijeti sve informacije
- Ne smiju se spominjati prijašnji obroci u odgovoru
- Pri određivanju savjeta model treba u obzir uzeti i količinu obroka

Upute koje su bile postavljene kod prvog testiranja:

„You will be provided with a list of meals that the user ate today which can be empty and a meal that the user wants to eat. You should inform the user about the healthiness of the

chosen meal with the consideration of the previous meals that the user ate that day. The answer should be concise. Do not mention the eaten meals in the reply.“

Prvo testiranje je provedeno velikim jezičnim modelom *GPT 3.5 turbo*. U nastavku slijede prosječni odgovori.

Odgovor za jedno jaje:

The meal of egg you want to eat is a healthy choice considering your previous meal of chicken, pasta, and salad. Eggs are a good source of protein and various nutrients. Enjoy your meal!

Odgovor za jedan kilogram jaja:

Eating 1 kilogram of eggs in addition to the chicken, pasta, and salad would be excessive and not recommended for a balanced and healthy diet. It is important to maintain a variety of food groups in your meals to ensure proper nutritional intake.

Odgovor za 10 grama torte:

The meal you want to eat, a 10 gram piece of cake, is on the less healthy side compared to the chicken, pasta, and salad you have already eaten today.

Iz primjera se može vidjeti da odgovori nisu prihvatljivi. Odgovori bi za zdrave obroke, idealno, trebali biti kratki kako tekst ne bi zauzimao puno mjesta na ekranu te mogu biti dulji za nezdrave obroke kako bi se moglo objasniti zašto. Također, veliki jezični model je s ovim uputama pokazao manjak pažnje za težinu obroka te je klasificirao 10 grama torte kao nezdrav obrok iako nema puno utjecaja na zdravlje. Najbitnije, u uputama je eksplicitno navedeno da se u odgovorima ne smiju spominjati konzumirani obroci koji se ipak spominju.

Zatim slijede odgovori za iste upite uz korištenje velikog jezičnog modela *GPT 4 turbo*.

Odgovor za jedno jaje:

Eating an egg would be a healthy choice for you. It's rich in protein and would complement the nutrients from your previous meal without adding excessive calories.

Odgovor za jedan kilogram jaja:

Eating 1kg of eggs is excessive in terms of calorie and cholesterol intake, especially if you already had a balanced meal like chicken with pasta and salad today. Consider a lighter option or reducing the portion.

Odgovor za 10 grama torte:

Eating a small, 10-gram piece of cake after your previous meals today should be fine. Enjoy it in moderation!

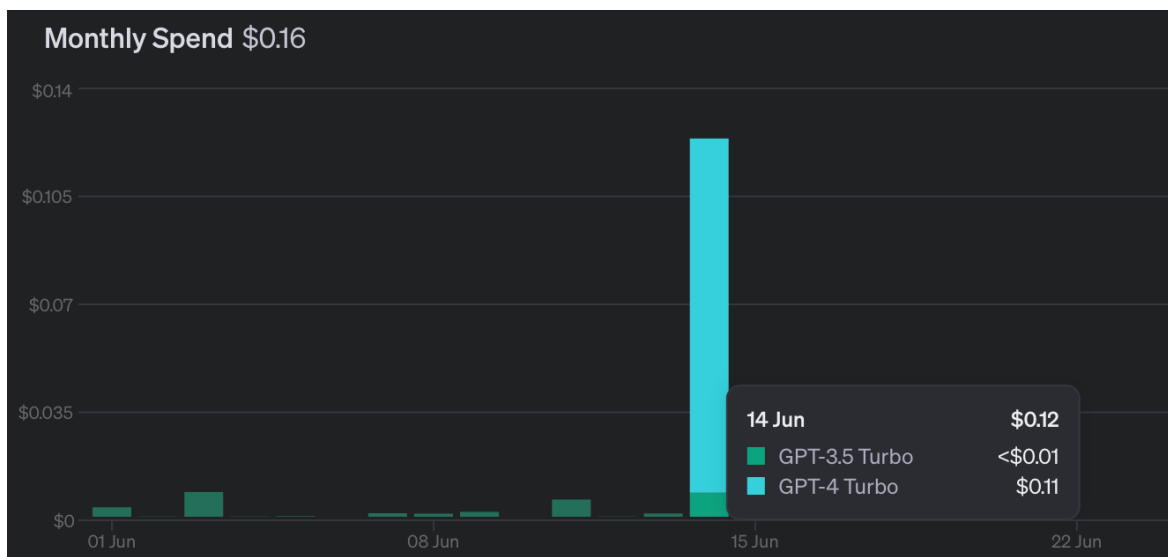
Novi poboljšani veliki jezični model *GPT 4 turbo* donosi bolje rezultate. U stanju je prepoznati veličinu porcije i prema njoj evaluirati utjecaj obroka na prehranu te bolje prati instrukcije koje su mu dane. No bitno je uzeti u obzir troškove odabranih modela.

U tablici (Tablica 6.1 Usporedba cijena velikih jezičnih modela) se mogu vidjeti cijene za dva najpopularnija OpenAI velika jezična modela, *GPT 4 turbo* i *GPT 3.5 turbo*.

Tablica 6.1 Usporedba cijena velikih jezičnih modela [19]

	Cijena ulaza	Cijena izlaza
gpt-3.5-turbo	US\$0.50 / 1M tokena	US\$1.50 / 1M tokena
gpt-4-turbo	US\$10.00 / 1M tokena	US\$30.00 / 1M tokena

Po tablici se može vidjeti da je *GPT 4 turbo* znatno skuplji. Ova razlika u cijeni se jasno vidi na grafu troška pod izbornikom „Usage“ na web stranici OpenAI web sučelja. Na dan uspoređivanja odgovora za ova dva velika jezična modela je trošak za *GPT 4 turbo* preko 10 puta veći nego za *GPT 3.5 turbo* što se može vidjeti na slici (Sl. 6.1).



Sl. 6.1 Graf troška za velike jezične modele

Zbog troška se *GPT 4 turbo* nije pokazao kao dobra opcija za ovaj projekt te je bilo potrebno istražiti načine na koje se odgovori modela mogu poboljšati.

Kako bismo adresirali probleme u odgovorima, potrebno je prilagoditi upute velikog jezičnog modela. OpenAI sugerira da je korisno odvojiti odjeljke u uputama za veliki jezični model koristeći simbole poput "###" ili „“”“. Ovaj pristup povećava preciznost rezultata. Također, važno je davati jasne i specifične upute koje model usmjeravaju prema točnim odgovorima [20].

Veliki jezični modeli bolje prate upute koje im govore što trebaju činiti, u odnosu na one koje im govore što ne smiju činiti, iako se mogu koristiti različite tehnike za zabrane. Istraživanja su pokazala da veliki jezični modeli bolje slijede upute kada im je kao nagrada za praćenje obećana velika svota novca [21], što se koristi kao tehnika za poštivanje zabrana u uputama.

Na kraju, korisno je u uputama navesti postupak i primjere kako bi se veliki jezični model usmjerio u pravom smjeru. Na taj način, povećava se točnost i kvaliteta rezultata koje veliki jezični model daje.

Nakon uređivanja uputa koristeći navedene principe one izgledaju ovako:

”

Instructions

Your task is to evaluate the healthiness of a meal that the user wants to eat. Keep the answers short for healthy meals and more descriptive for unhealthy ones.

I'm going to tip \$1000 if you don't mention consumed meals or the serving size.

Guidelines

1. Determine the healthiness of a meal based on its name.
2. Consider the serving size and its impact on health. Small quantities of otherwise unhealthy foods may have a negligible impact.
3. Prioritize providing feedback that helps the user make informed dietary choices.
4. Combine the information and return it in a friendly manner.

Examples

New meal: "50g of sugar"

This meal is unhealthy due to high sugar content.

New meal: "tea"

This meal is healthy, providing hydration and antioxidants.

New meal: "20kg of meat"

This meal is unhealthy due to excessive quantity, which can lead to various health issues.

Provided data

Consumed meals: %@

New meal: %@“

”

U nastavku su prikazani primjeri odgovora velikog jezičnog modela *GPT 3.5 turbo* na prethodne upute.

Primjeri odgovora za jaje:

This meal is healthy as eggs are a good source of protein and essential nutrients.

Primjer odgovora za jedan kilogram jaja:

This meal is healthy, as eggs are a good source of protein and other essential nutrients. However, consuming 1kg at once may not be ideal due to the large quantity. Moderation is key for a balanced diet.

Primjeri odgovora za 10 grama torte:

This meal is unhealthy due to the high sugar and fat content typically found in cake. It's best to enjoy desserts like this in moderation.

Kao što možemo vidjeti, mijenjanje uputa je rezultiralo puno boljim odgovorima. Model ne vraća dugačke odgovore osim kada je to potrebno, više ne spominje prethodne obroke i pokazuje djelomično razumijevanje težine obroka. Upozorava korisnika za prekomjernu količinu obroka za obroke koji su inače zdravi, ali i dalje ne klasificira male količine inače nezdravih obroka kao neznčajnim za zdravlje. Prema tome je inženjerstvo upita uspješno popraviti većinu neželjenih odgovora koji su se prije pojavljivali.

6.2.2. Upit kod razgovora

Pri razgovoru sa asistentom nije potrebno utjecati na odgovore velikog jezičnog modela u količini kao i kod pretraživanja. Pogled za razgovor je dizajniran kao ekran za razgovor i dugački odgovori ne predstavljaju problem. Za ovu funkcionalnost je bitno da veliki jezični model zna svoju odgovornost i korisnikove obroke od prošla 3 dana. Kod razgovora su velikom jezičnom modelu zadane sljedeće upute koje pružaju zadovoljavajuće odgovore:

”

Instructions

You're a nutritionist. You will be provided with a list of meals that the user ate for the last few days. You should tell the user about how healthy their diet is and give some

recommendations if it isn't. Make sure to answer any followup questions in a matter that a nutritionist would. No yapping.

Provided data

Consumed meals: %@

“

6.3. Buduća poboljšanja

Kroz testiranje programera i neovisnih korisnika otkriveni su neki nedostaci aplikacije, kao i funkcionalnosti koje bi se mogle dodati u budućnosti. Tijekom ovog procesa evaluacije, uočeno je nekoliko ključnih područja za poboljšanje koja mogu značajno unaprijediti korisničko iskustvo i preciznost podataka koje aplikacija pruža.

Dodatne karakteristične informacije za korisnike

Trenutno aplikacije koristi osnovne parametre kao što su dob, spol, visina, težina i razina fizičke aktivnosti. Međutim, određena fiziološka stanja kao što su trudnoća i dojenje, te medicinska stanja poput visokog krvnog tlaka, značajno utječu na energetske potrebe organizma. Implementacijom ovih dodatnih parametara, aplikacija bi mogla pružiti prilagođenije i točnije preporuke korisnicima koji se nalaze u ovim stanjima, čime bi se povećala ukupna preciznost i korisnost aplikacije. Također, uvođenje dodatnih mogućnosti za praćenje specifičnih dijetetskih potreba i ograničenja može pomoći korisnicima s posebnim prehrambenim zahtjevima.

Sažimanje informacija na početnom ekranu

Iako je početni ekran aplikacije dizajniran na način da korisniku pruža bitne informacije na intuitivan način, testiranja su pokazala da novi korisnici i dalje imaju poteškoća s razumijevanjem prikazanih podataka. Kako bi se olakšalo korisničko iskustvo, predlaže se korištenje velikog jezičnog modela koji bi automatski sažimao informacije i prikazivao samo one najvažnije za korisnika. Primjerice, veliki jezični model bi mogao identificirati ključne podatke o dnevnom unosu kalorija, preostalim makro nutrijentima i eventualnim

odstupanjima od preporučenih vrijednosti. Na taj način, korisnik bi bio bolje informiran i mogao bi brže donositi odluke o svojoj prehrani.

Praćenje trendova

Jedna od korisnih funkcionalnosti aplikacije Health je slanje obavijesti korisnicima kada se uoče promjene u određenim zdravstvenim trendovima, poput otkucaja srca u mirovanju ili balansa pri hodaњу. Korisnici su istaknuli da bi slična funkcionalnost bila korisna i u aplikaciji ovog rada. Uvođenjem funkcionalnosti praćenja prehrambenih trendova, aplikacija bi mogla korisnike obavještavati o nezdravim obrascima prehrane, kao što su pretjerani unos šećera ili nedostatak vlakana. Takve obavijesti bi korisnicima pomogle da na vrijeme prepoznaju i isprave nezdrave navike, čime bi se poboljšalo njihovo ukupno zdravlje.

Dodavanje prepoznavanja obroka po slici

Izlaskom novog OpenAI-ovog multi modalnog modela *GPT-4o* koji uz tekst pruža mogućnost slanja slika, videa i glasovnih zapisa može se implementirati prepoznavanje obroka sa slike. Ova funkcionalnost bi značajno poboljšala korisničko iskustvo, jer bi eliminirala potrebu za ručnim unosom podataka o prehrani. Također, preciznije prepoznavanje namirnica omogućilo bi točniji unos podataka, što bi rezultiralo boljim praćenjem i analizom prehrambenih navika korisnika.

Zaključak

Praćenje prehrane važan je aspekt zdravog načina života koji ima značajan utjecaj na cjelokupno zdravlje i dobrobit pojedinca. Korištenjem suvremenih tehnologija i najnovijih zdravstvenih istraživanja, ovaj proces, koji je tradicionalno uključivao ručno bilježenje, računanje i nerijetko pogađanje, može se znatno pojednostaviti i automatizirati. Aplikacija razvijena u sklopu ovog rada pokazala je kako se precizne formule za procjenu dnevnog unosa kalorija i nutrijenata mogu uspješno implementirati i kako one mogu biti iznimno korisne za korisnike.

Ova aplikacija koristi velike jezične modele i različita web sučelja kako bi korisnicima omogućila brzi pristup informacijama koje bi inače morali tražiti na internetu, čime se značajno štedi vrijeme i povećava učinkovitost. Korištenje velikih jezičnih modela pokazalo se korisnim u svakodnevnom životu, pružajući korisnicima vrijedne i točne informacije na jednostavan i pristupačan način.

Jedan od ključnih elemenata aplikacije je integracija razvojnog okvira HealthKit, koji omogućuje praćenje i analizu podataka o potrošnji kalorija i fizičkoj aktivnosti korisnika. Korištenje HealthKit-a omogućava prikupljanje preciznih i kontinuiranih podataka, što je ključno za točnost procjena i preporuka koje aplikacija pruža. Korištenjem baza podataka Realm i UserDefaults postignuta je efikasna pohrana i obrada podataka, čime je aplikacija postala još korisniji alat za sve koji žele unaprijediti svoje zdravlje i stil života.

Testiranjem od strane programera i neovisnih korisnika, otkriveni su određeni nedostaci, kao i mogućnosti za daljnje poboljšanje aplikacije. U budućnosti je moguće dodati dodatne parametre za praćenje korisnika, integrirati sustave za prepoznavanje obroka sa slika te uvesti obavijesti o promjenama u prehrambenim trendovima. Također, planira se daljnje unapređenje korisničkog iskustva kroz integraciju novih tehnologija i optimizaciju postojećih funkcionalnosti.

Ukratko, rad na ovoj aplikaciji pokazao je da se način na koji pratimo i upravljamo svojom prehranom može značajno unaprijediti korištenjem suvremenih tehnologija. Daljnji razvoj i implementacija predloženih poboljšanja imaju potencijal za daljnje poboljšanje korisničkog iskustva i pružanje sveobuhvatnog i personaliziranog alata za praćenje prehrane i zdravlja.

Literatura

- [1] Fernando Putra, *Learn UIKit (Part 1 of 3) — Introduction for SwiftUI Devs*, Medium (2023, prosinac). Poveznica: <https://fernandoptr.medium.com/learn-uikit-part-1-of-3-introduction-for-swiftui-devs-33cd7a05e765>; Pristupljeno 27. lipnja 2024.
- [2] Jeroen L., *UIKit vs. SwiftUI: How to Choose the Right Framework for Your App*, GetStream (2022, siječanj). Poveznica: <https://getstream.io/blog/uikit-vs-swiftui/>; Pristupljeno 27. lipnja 2024.
- [3] Bilescu Adrian, *A Pragmatic Guide to Clean Architecture on iOS*, Medium (prosina 2023). Poveznica: <https://adrian-bilescu.medium.com/a-pragmatic-guide-to-clean-architecture-on-ios-e58d19d00559>; Pristupljeno 27. lipnja 2024.
- [4] Mehar Chand, *Model View Controller VS Model View Presenter Using Example*, Medium (2023, prosinac). Poveznica: <https://medium.com/@mehar.chand.cloud/model-view-controller-vs-model-view-presenter-using-example-428b389b5158>; Pristupljeno 27. lipnja 2024.
- [5] Dmitry Lupich, *The Composable Architecture: Swift guide to TCA*, Medium (2023, kolovoz). Poveznica: <https://medium.com/@dmitrylupich/the-composable-architecture-swift-guide-to-tca-c3bf9b2e86ef>; Pristupljeno 27. lipnja 2024.
- [6] *CalorieNinjas*, CalorieNinjas. Poveznica: <https://calorieninjas.com>; Pristupljeno 27. lipnja 2024.
- [7] *API Documentation*, CalorieNinjas. Poveznica: <https://calorieninjas.com/api>; Pristupljeno 27. lipnja 2024.
- [8] *Assistants API overview*, OpenAI, Poveznica: <https://platform.openai.com/docs/assistants/overview>; Pristupljeno 27. lipnja 2024.
- [9] *How Assistants work*, OpenAI, Poveznica: <https://platform.openai.com/docs/assistants/how-it-works>; Pristupljeno 27. lipnja 2024.

- [10] *HealthKit*, Apple Developer Documentation, Poveznica: <https://developer.apple.com/documentation/healthkit/>; Pristupljeno 27. lipnja 2024.
- [11] *UserDefaults*, Apple Developer Documentation, Poveznica: <https://developer.apple.com/documentation/foundation/userdefaults>; Pristupljeno 27. lipnja 2024.
- [12] *Realm*, MongoDB, Poveznica: <https://www.mongodb.com/docs/realmsdks/swift/latest/>; Pristupljeno 27. lipnja 2024.
- [13] *Mifflin-St Jeor Equation*, Medscape, Poveznica: <https://reference.medscape.com/calculator/846/mifflin-st-jeor-equation>; Pristupljeno 28. lipnja 2024.
- [14] M D Mifflin, S T St Jeor, L A Hill, B J Scott, S A Daugherty, Y O Koh, *A new predictive equation for resting energy expenditure in healthy individuals*, American Journal of Clinical Nutrition, 1990.
- [15] David Frankenfield, Lori Roth-Yousey, Charlene Compher, *Comparison of predictive equations for resting metabolic rate in healthy nonobese and obese adults: a systematic review*, Journal of the American Dietetic Association, 2005.
- [16] Jennifer J. Otten, Jennifer Pizzi Hellwig, and Linda D. Meyers, *Dietary Reference Intakes*, Institute of Medicine, 2006.
- [17] *Sodium in diet*, Medline Plus, (lipanj 2022), Poveznica: <https://medlineplus.gov/ency/article/002415.htm>; Pristupljeno 28. lipnja 2024.
- [18] *Potassium in diet*, Medline Plus, (lipanj 2022), Poveznica: <https://medlineplus.gov/ency/article/002413.htm>; Pristupljeno 28. lipnja 2024.
- [19] *Pricing*, OpenAI, Poveznica: <https://openai.com/api/pricing/>; Pristupljeno 28. lipnja 2024.
- [20] *Best practices for prompt engineering with the OpenAI API*, OpenAI, Poveznica: <https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-the-openai-api>; Pristupljeno 28. lipnja 2024.

- [21] Sondos Mahmoud Bsharat, Aidar Myrzakhan, Zhiqiang Shen, *Principled Instructions Are All You Need for Questioning LLaMA-1/2, GPT-3.5/4*, Mohamed bin Zayed University of AI, 2023.

Sažetak

Mobilna aplikacija za pomoć u pravilnoj prehrani

Cilj diplomskog rada bio je implementirati mobilnu aplikaciju za praćenje prehrane koristeći formule za predviđanje preporučenog dnevnog unosa, razvojnog okvira HealthKit za praćenje potrošenih kalorija i velikih jezičnih modela za davanje savjeta korisniku. Kroz rad se spominju osnove implementacije aplikacija za operativni sustav iOS i razvojni okviri koji se mogu koristiti. Detaljno je objašnjena arhitektura aplikacije i integracija svake njene funkcionalnosti. Na kraju rada je provedeno testiranje preciznosti formula za preporučen dnevni unos kalorija koristeći mjerenja potrošenih kalorija pomoću pametnog sata te je objašnjeno kako se sa inženjerstvom upita može doći do željenih odgovora kod velikih jezičnih modela.

Ključne riječi: Mobilna aplikacija, iOS, HealthKit, veliki jezični modeli, OpenAI, pravilna prehrana, inženjerstvo upita, praćenje prehrane

Summary

A mobile application to help with proper nutrition.

The aim of this thesis was to implement a mobile application for tracking nutrition using formulas for predicting the recommended daily intake, the HealthKit framework for monitoring burned calories, and large language models for providing user advice. The work discusses the basics of implementing applications for the iOS operating system and the development frameworks that can be used. The architecture of the application and the integration of each of its functionalities are explained in detail. At the end of the work, the precision of the formulas for recommended daily calorie intake was tested using measurements of burned calories with a smartwatch, and it was explained how prompt engineering can achieve the desired answers with large language models.

Keywords: Mobile application, iOS, HealthKit, large language models, OpenAI, proper nutrition, prompt engineering, nutritional tracking

Skraćenice

MVP	<i>Model-View-Presenter</i>	Model-Pogled-Predstavljач
MVC	<i>Model-View-Controller</i>	Model-Pogled-Upravljač
TCA	<i>The Composable Architecture</i>	Kompozitna arhitektura
HTTP	<i>Hypertext Transfer Protocol</i>	Protokol prijenosa hiperteksta
API	<i>Application Programming Interface</i>	Programsko sučelje aplikacije
JSON	<i>JavaScript Object Notation</i>	JavaScript objektna notacija
RMR	<i>Resting metabolic rate</i>	Stopa metabolizma u mirovanju
REE	<i>Resting energy expenditure</i>	Utrošak energije u mirovanju
TDEE	<i>Total daily energy expenditure</i>	Ukupni dnevni utrošak energije

Privitak

Poveznica na GitHub repozitorij projekta:
<https://github.com/marinmarkelic/NutritionApp>