Razvoj mobilne aplikacije Budi zdrav

Tomljanović, Domagoj

VELEUČILIŠTE U RIJECI

Domagoj Tomljanović

RAZVOJ MOBILNE APLIKACIJE BUDI ZDRAV

završni rad

Rijeka, 2024.

VELEUČILIŠTE U RIJECI

Stručni preddiplomski studij Telematika

RAZVOJ MOBILNE APLIKACIJE BUDI ZDRAV

ZAVRŠNI RAD

MENTOR

izv.prof.dr.sc. Alen Jakupović, prof. struč. stud.

STUDENT

Domagoj Tomljanović

MBS: 2427000015/19

Rijeka, 2024.

Sažetak

Rad opisuje proces razvoja mobilne aplikacije unutar Android Studija, pružajući sveobuhvatan pregled koraka potrebnih za njeno kreiranje. Rad se fokusira na različite aspekte aplikacije, uključujući vizualne komponente koje omogućuju korisnicima praćenje dnevnog kalorijskog unosa, broja prijeđenih koraka i skeniranje barkodova za evidentiranje konzumiranih obroka unutar baze podataka. Aplikacija je dizajnirana s ciljem da korisnicima pruži intuitivno sučelje, koje olakšava praćenje i upravljanje prehrambenim navikama. Vizualni dijelovi aplikacije uključuju dijagrame uporabe, dijagrame aktivnosti, mockupove i stvarne prikaze ekrana, što omogućava korisnicima da bolje razumiju funkcionalnosti aplikacije. Pored vizualnog aspekta, rad obuhvaća detaljan prikaz Java koda koji je razvijen za implementaciju ključnih funkcionalnosti aplikacije. Opisuje se način na koji je kod strukturiran, s posebnim naglaskom na integraciju s Firestore bazom podataka, koja se koristi za pohranu korisničkih podataka.

Dodatno, aplikacija koristi Google API za dohvaćanje broja koraka koje korisnik odradi, čime se omogućava praćenje tjelesne aktivnosti. Ovaj sustav pruža korisnicima sveobuhvatan alat za praćenje zdravlja i kondicije, kombinirajući prehrambene podatke s podacima o tjelesnoj aktivnosti. Na kraju, rad donosi zaključke o korisnosti aplikacije te predlaže moguće smjerove za daljnji razvoj.

Ključne riječi: Android studio, Java, Firestore, Google API, mobilna aplikacija

Sadržaj

1. Uvod

U suvremenom dobu ubrzane svakodnevnice, potreba za balansiranjem osobnog zdravlja i fizičke aktivnosti postaje sve izraženija. S ciljem zadovoljenja tih potreba, razvijena je ova mobilna aplikacija koja je dizajnirana da olakša praćenje zdravstvenih i fitness ciljeva kako amaterima, tako i profesionalnim sportašima. Ova aplikacija omogućava korisnicima jednostavan način praćenja nutritivnog unosa, uključujući makro-nutrijente, te nudi alate za upravljanje tjelesnom težinom - bilo da je cilj smanjenje masnog tkiva, održavanje kilaže ili povećanje mišićne mase. Hrvatska, kao zemlja koja se suočava s izazovima pretilosti, zajedno s mnogim drugim zemljama, pruža idealno tržište za ovakvu aplikaciju. Aplikacija ne samo da korisnicima omogućuje da održavaju kontrolu nad svojom prehranom, već ih i aktivno vodi ka ostvarivanju njihovih zdravstvenih i fitness ciljeva. Uz integrirani skener barkodova, korisnici mogu lako unijeti informacije o hrani koju konzumiraju, dok integrirani brojač koraka pruža precizan prikaz njihove dnevne aktivnosti. Aplikacija nudi prilagođene savjete i strategije koje korisnicima pomažu da ostvare svoje ciljeve, bilo da se radi o gubitku težine ili povećanju tjelesne mase.

2. Osnovni koncepti razvoja mobilne aplikacije

2.1 Alati za razvoj

2.1.1 Android studio

Android Studio je glavni alat za izradu Android aplikacija, razvijen od strane Googlea. Ovaj alat nudi sve što je potrebno za razvoj aplikacija – od dizajniranja sučelja do pisanja koda i testiranja. Sve je tu na jednom mjestu, što značajno olakšava rad. Jedna od ključnih prednosti Android Studija je njegova sposobnost upravljanja različitim verzijama aplikacije, zahvaljujući integraciji s Gradleom[1]. Gradle automatizira komplicirane procese izrade, a na taj način omogućuje lakše upravljanje projektima. Uz to, Android Studio omogućuje pregled aplikacije na različitim uređajima, što osigurava da aplikacija radi jednako dobro na starijim telefonima kao i na najnovijim tabletima (Google Developers, 2023).

2.1.2 Java

Java je jedan od najpopularnijih programskih jezika, posebno kada je riječ o razvoju Android aplikacija. U ovom projektu, Java je bila glavni alat za izradu svih bitnih dijelova aplikacije. Svojom fleksibilnošću i jednostavnošću, omogućila je da se svi ključni dijelovi aplikacije razviju bez previše komplikacija. Ono što Javu čini idealnom za ovaj projekt je njezina fleksibilnost. Kod napisan u Javi može raditi na praktički svim Android uređajima zahvaljujući Java Virtual Machine-u (JVM[2]), što znači da je aplikacija lako prilagodljiva različitim verzijama Androida. Korištenje Jave omogućilo je brzu integraciju s Firebaseom za autentifikaciju korisnika i pohranu podataka, kao i jednostavno upravljanje korisničkim sučeljem. Ukratko, Java je omogućila da se ova aplikacija razvije brzo i efikasno, bez većih problema (Oracle, 2023).

[1] Gradle je alat za automatizaciju izgradnje projekata, koji omogućuje jednostavno upravljanje različitim verzijama aplikacija i automatizaciju procesa izgradnje.
[2] JVM je virtualni stroj koji omogućuje da se Java programi izvršavaju na različitim uređajima bez potrebe za izmjenama u kodu

2.2 Arhitektura mobilne aplikacije

2.2.1 XML

XML je bio neizostavan dio u izradi korisničkog sučelja ove aplikacije. Korištenje XML-a omogućilo je jednostavno i pregledno definiranje svih elemenata na ekranu, od gumbi do tekstualnih polja. Umjesto da se sve piše unutar koda u Javi, XML je omogućio da vizualni dio aplikacije bude odvojen, što je kasnije olakšalo prilagodbu i izmjene određenih dijelova aplikacije. Što se tiče XML-a, on je fleksibilan jezik za strukturiranje informacija, dizajniran tako da bude lako čitljiv i razumljiv. U svijetu Android razvoja, XML se koristi prvenstveno za dizajn korisničkog sučelja. Umjesto da se ručno pozicioniraju svi elementi u kodu, XML omogućuje da se layouti [3] definiraju kroz jednostavne oznake, što čini cijeli proces dizajniranja mnogo preglednijim, efikasnijim i zabavnijim. Zahvaljujući XML-u, svi dijelovi sučelja su bili jasno definirani i spremni za povezivanje s funkcionalnostima koje su implementirane u Javi. Također, XML datoteke omogućuju fleksibilnost u prilagodbi izgleda aplikacije za različite veličine zaslona i orijentacije uređaja, što je ključno za pružanje optimalnog korisničkog iskustva (W3Schools, 2024).

2.2.2 UML

UML dijagrami su bili važan alat u procesu planiranja i razvoja ove aplikacije. Korištenjem UML dijagrama, bilo je moguće vizualizirati različite dijelove sustava prije nego što su napisani u kod, što je smanjilo rizik od grešaka i zapravo, pomoglo u planiranju koraka pisanja koda nakon kreiranja istih. Dijagrami uporabe omogućili su definiranje funkcionalnosti koje aplikacija pruža korisnicima, jasno prikazujući kako korisnici stupaju u interakciju s aplikacijom. Na primjer, dijagrami uporabe su prikazivali kako korisnici unose podatke, pregledavaju rezultate ili se prijavljuju putem aplikacije. Korištenje UML dijagrama omogućilo je da se složeni koncepti i procesi u aplikaciji predstave na jasan i razumljiv način, što je

[3] Layout u kontekstu Android razvoja odnosi se na način na koji su elementi korisničkog sučelja raspoređeni na ekranu. Layout definira strukturu sučelja, određujući položaj i veličinu elemenata kao što su gumbi, tekstualna polja, slike i drugi vizualni objekti

olakšalo prelazak s faze planiranja na fazu implementacije (Diagrams.net, 2024; Nordeen, 2020).

2.3 Autentifikacija, sigurnost i baza podataka

2.3.1 Firebase autentifikacija

Autentifikacija i sigurnost su ključni aspekti ove aplikacije, posebno s obzirom na to da se radi o rukovanju osjetljivim korisničkim podacima. U ovom projektu, Firebase autentifikacija pokazala se kao rješenje za sigurno prijavljivanje korisnika i upravljanje njihovim podacima. Firebase autentifikacija omogućila je jednostavnu implementaciju više metoda prijave, uključujući prijavu putem e-pošte i lozinke, kao i putem Google Maila (Google Sign-In[4]). Korištenje Google Sign-Ina integrirano je u aplikaciju kako bi korisnicima omogućilo brz i siguran pristup njihovim profilima koristeći njihove postojeće Google račune. Implementacija ovih funkcionalnosti nije bila komplicirana ponajviše zahvaljujući Firebase SDK-u[5], koji je pružio sve potrebne alate za dodavanje funkcionalnosti prijave, registracije i odjave. Osim same prijave, Firebase je osigurao i sigurno pohranjivanje korisničkih podataka u Firestore bazi podataka. Korištenjem ovog sustava, podaci korisnika su šifrirani i zaštićeni, što je ključno za osiguranje privatnosti (Firebase, 2024).

2.3.2 Firestore baza podataka

Firestore je baza podataka u oblaku[6] koju pruža Firebase, dizajnirana za jednostavno pohranjivanje i sinkronizaciju podataka u stvarnom vremenu. U ovoj aplikaciji, Firestore je korišten za spremanje osobnih podataka korisnika, uključujući profile, fitness aktivnosti i prehrambene navike. Svaki korisnik u bazi podataka dobiva svoj jedinstveni ID, koji se

[4] Google Sign-In je usluga prijave koju pruža Google, omogućujući korisnicima da se prijave u aplikaciju koristeći svoje postojeće Google račune

[5] SDK (Software Development Kit) je skup alata i knjižnica koje omogućuju programerima razvoj softvera za određenu platformu.

[6] Oblak (cloud) odnosi se na pružanje usluga putem interneta, uključujući pohranu podataka i računalne resurse, koji su dostupni s bilo kojeg mjesta i uređaja, bez potrebe za lokalnim hardverom.

automatski generira prilikom registracije. Taj ID koristi se za identifikaciju korisnika i povezivanje svih njegovih podataka unutar baze. Unutar kolekcije[7] "Users", svaki korisnik ima vlastitu mapu (document) koja sadrži podatke specifične za njega. Unutar te mape, podaci su dalje organizirani u različite podkolekcije (subcollections), kao što su "Activities" i "Nutrition". Na ovaj način, svi podaci vezani uz jednog korisnika su centralizirani i lako dostupni. Kada korisnik unese nove podatke, aplikacija koristi generirani ID kako bi pohranila te informacije u odgovarajuću mapu unutar baze. Na primjer, ako korisnik dodaje novu fitness aktivnost, ta aktivnost se sprema u podkolekciji "Activities" unutar mape koja odgovara tom korisniku. Podaci se unose pomoću metoda kao što su set()[8] ili add()[9], što omogućuje brzo i efikasno pohranjivanje i kasniji dohvat podataka.

2.4 Integracija s vanjskim API-jem

2.4.1 Google API

Google API-jevi[10] omogućuju programerima da integriraju različite Googleove usluge u svoje aplikacije. Ovi API-jevi pružaju širok spektar funkcionalnosti, od rada s kartama i geolokacijom, do upravljanja korisničkim podacima i zdravljem. U ovoj aplikaciji, Google API je korišten za integraciju brojača koraka s podacima iz Samsung Healtha[11], omogućujući korisnicima praćenje svojih fitness aktivnosti izravno unutar aplikacije. Kako bi aplikacija mogla pristupiti podacima o broju koraka iz Samsung Healtha, korisnik mora dati odobrenje za korištenje svojih podataka putem Google API-ja. Kada korisnik pokrene aplikaciju i zatraži pristup podacima o koracima, aplikacija šalje zahtjev Google API-ju. Nakon što korisnik potvrdi odobrenje, aplikacija može dohvatiti podatke iz Samsung Healtha i prikazati ih

[7] U Firestoreu, podaci se organiziraju u kolekcije (collections) i dokumente (documents). Kolekcije su skupovi dokumenata, dok su dokumenti osnovne jedinice podataka koje sadrže informacije u obliku polja (fields).
[8] Metoda set() u Firestoreu koristi se za pohranjivanje podataka u određeni dokument unutar kolekcije. Ako dokument već postoji, podaci će biti ažurirani; ako ne postoji, bit će kreiran novi dokument.
[9] Metoda add() u Firestoreu koristi se za dodavanje novog dokumenta u kolekciju s automatski generiranim jedinstvenim ID-om
[10] API je sučelje koje omogućuje aplikacijama da komuniciraju međusobno ili s vanjskim uslugama, olakšavajući razmjenu podataka i izvršavanje radnji.
[11] Samsung Health je aplikacija za praćenje zdravlja i fitnessa koja omogućuje korisnicima bilježenje i praćenje svojih aktivnosti, prehrambenih navika i zdravstvenih parametara na njihovim Samsung uređajima.

korisniku. Tehnički, proces funkcionira tako da se koristi OAuth 2.0[12] protokol, koji omogućuje siguran prijenos podataka između aplikacije i Googleovih usluga. OAuth 2.0 prvo traži od korisnika da se autentificira i odobri pristup, nakon čega se generira pristupni token. Ovaj token omogućuje aplikaciji pristup podacima o koracima dokle god traje sesija. Podaci o koracima dohvaćeni putem Google API-ja zatim se koriste za prikazivanje korisnikovih statistika unutar aplikacije. Ova funkcionalnost omogućuje korisnicima da prate svoje ciljeve i uspoređuju ih kroz vrijeme (Sosa, 2023).

[12] OAuth 2.0 je standardizirani protokol za autorizaciju koji omogućuje trećim stranama da dobiju ograničeni pristup resursima korisnika na vanjskim uslugama bez potrebe za dijeljenjem lozinki. Osigurava siguran prijenos i kontrolu pristupa podacima

3 Objektno orijentirana analiza

Unutar objektno orijentirane analize su upisani scenariji uporabe, na temelju njih izrađeni UML dijagram uporabe i dijagram aktivnosti.

3.1 Dijagram uporabe

Aplikacija BeHealthy ima sljedeće funkcionalnosti:

- **Register:** Registracija pomoću korisničkih podataka; email i lozinka
- **Login:** Korisnik se logira u aplikaciju putem podataka koje je koristio u registraciji/ili se može logirati pomoću Google računa
- **Personal data**: Korisnik upisuje svoje osobne podatke i definira cilj (gubitak, održavanje ili povećanje kilaže)
- **Personal data info button**: Tekstualno pojašnjenje dijela funkcionalnosti aplikacije
- **My Nutrition:** Ključni dio aplikacije koji korisniku pruža mogućnost upisa kalorija za tri obroka, kao i mogućnost skeniranja barkoda za svaki od tih obroka
- **Nutrition:** Omogućuje pregled unesenoga preko „My Nutritiona“
- **More**: Omogućuje dva dodatna pregleda;
    - **Progress:** Praćenje napretka u odnosu na zadani cilj
    - **Goals:** Zadavanje ciljeva
- **Barcode scan:** Aplikacija ima funkcionalnost prepoznavanja barkoda. Odnosno, pretrage baze podataka sukladno već spremljenim barkodovima i pripadajućim unosima. Kao i spremanje novih namirnica uz barkod koji je skeniran
- **Steps history:** Prikazuje korisniku broj ostvarenih koraka za tekući i prethodne dane

Slika 1 Dijagram uporabe korisnik

Izvor: autor

3.2 Dijagrami aktivnosti

U aplikaciji za praćenje unosa kalorija korisnik ima iduće funkcionalnosti:

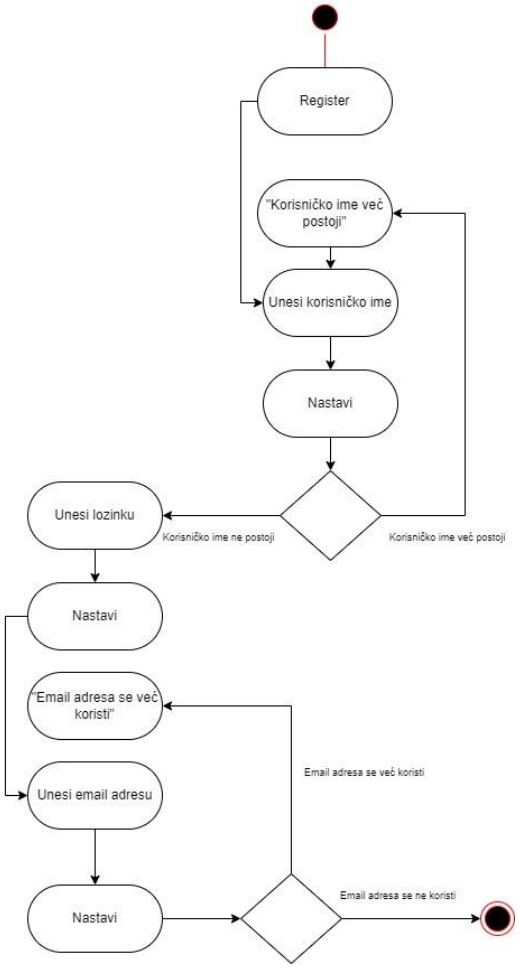
3.2.1 Register

1. Korisnik odabire navedenu tipku
2. Ispisuju mu se polja za upis korisničkom imena, lozinke i email adrese
3. Korisnik potvrđuje navedeni unos

4. Radi se provjera postoji li već korisnik s navedenim korisničkim imenom ili lozinkom
5. Ako postoji, javlja se prigodna poruka. Ako ne postoji, potvrđuje se registracija uz uputu da je istu potrebno dovršiti putem zaprimljenog maila

Opis dijagrama: Korisnik se putem ove funkcije registrira u aplikaciji. Njegovi podaci se šalju u bazu podataka i svaki put kada korisnik pristupa aplikaciji, aplikacija provjerava je li navedeno korisničko ime uz točno tu lozinku dostupno u bazi podataka. Ako je, korisnik pristupa svojem korisničkom računu. Prilikom registracije, ako korisnik upiše korisničko ime ili email adresu koja već postoji u bazi podataka dobije poruku koja mu upućuje na isto i ponudi mu reset lozinke (ako je upisana ista email adresa).

Slika 2 Dijagram aktivnosti Register



Izvor: autor

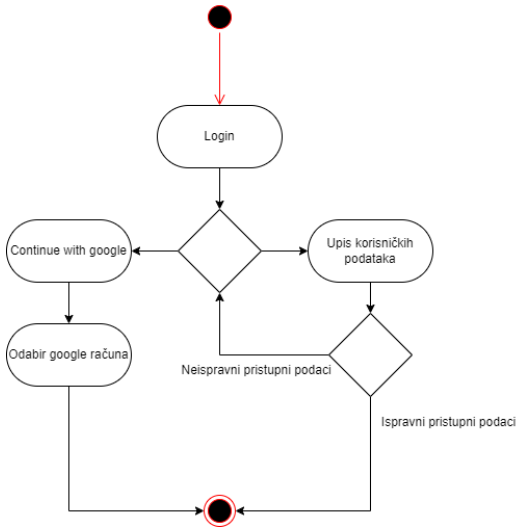
3.2.2 Login

1. Korisnik odabire navedenu tipku
2. Pojavljuje mu se polje za upis korisničkog imena/emaila i lozinke
3. Pojavljuje se i tipka *Continue with Google*
4. Ako upiše korisničke podatke s kojima se registrirao u aplikaciju pristupa aplikaciji
5. Ako odabere *Google login* i odabere važeći *Google account* također pristupa aplikaciji

Opis dijagrama:

Korisnik ovu funkciju koristi kako bi s podacima preko kojih se registrirao koristeći tipku *'register'* pristupio aplikaciji. Sve informacije koje je korisnik prethodno upisivao u kreirani korisnički račun će ga dočekati kada se opet logira u aplikaciju. Ako korisnik upiše nepostojeće korisničko ime/email ili postojeće ali uz krivu lozinku, dobiti će pripadajuću poruku. Također, korisnik ima napredniju funkciju, pristupanje aplikacije pomoću Google računa. Nakon odabira navedenoga, korisnik mora odabrati svoj Google račun (ili se logirati u isti ako već nije) i na taj način također može pristupiti aplikaciji.

Slika 3 Dijagram aktivnosti Login



Izvor: autor

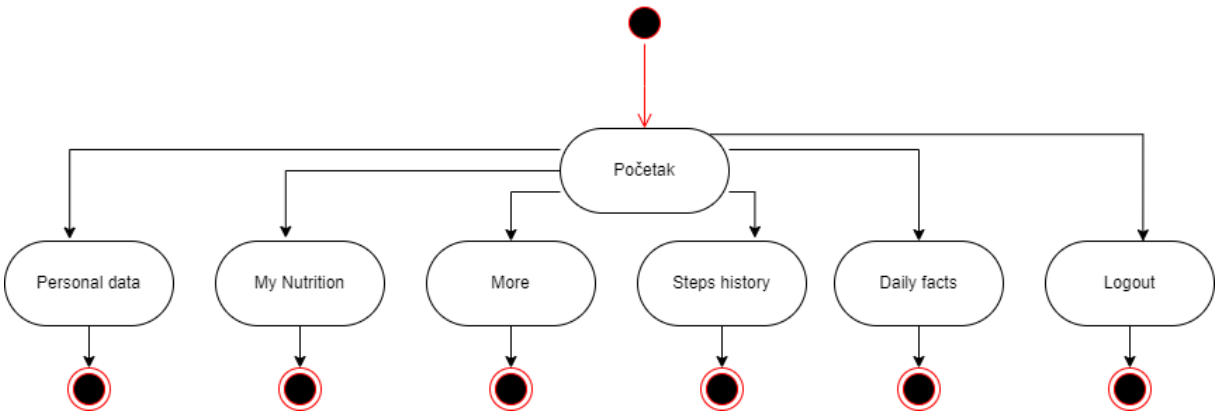
3.2.3 Main Activity

1. Korisnik pristupa glavnom sučelju aplikacije
2. Unutar istoga može odabrati 6 različitih smjerova za daljnje korištenje aplikacije
3. *Personal data*, *My nutrition*, *More*, *Steps history*, *Daily facts* i *Logout*

Opis dijagrama:

Dijagram prikazuje mogućnosti, odnosno smjerove korisnika unutar glavnog sučelja aplikacije.

Slika 4 Dijagram aktivnosti Main Activity



Izvor: Autor

3.2.4 Personal data

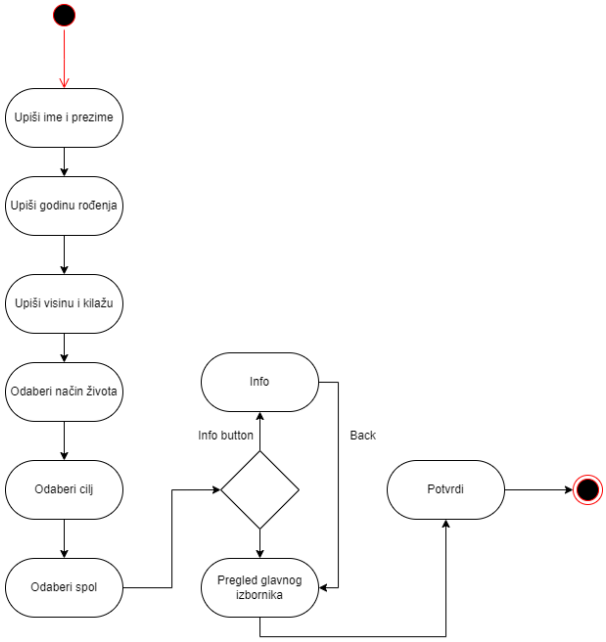
1. Korisnik iz glavnog izbornika odabire funkcionalnost 'Personal data'
2. Nakon pristupa navedenom dijelu aplikacije, korisnik upisuje osobne podatke
3. Ime i prezime
4. Godište rođenja
5. Visinu i kilažu
6. Odabire jedan od četiri predložena 'Načina života'
7. Odabire jedan od tri cilja (Održavanje kilaže, gubitak ili dobivanje na kilaži)

Opis dijagrama:

U ovome pregledu, korisnik upisuje osnovne podatke putem koje će se u konačnici računati njegov BMI, dnevna potreba za kalorijama i putem kojeg će se pratiti njegov napredak. Također, korisnik uz odabire tipova aktivnosti (koje pomažu u izračunu dnevne kalorijske potrošnje) odabire i jedan od tri dostupna cilja kako bi aplikacija mogla korisniku izračunati dnevno potrebni unos kalorija kako bi se isti cilj ostvario. Korisnik se iz ovog pregleda može vratiti preko 'Home' tipke u osnovni pregled aplikacije. Također, korisnik ima

„Info button“ koji mu daje dodatno pojašnjenje same aplikacije i njenih funkcionalnosti.

Slika 5 Dijagram aktivnosti Personal data



Izvor: Autor

3.2.5 My Nutrition

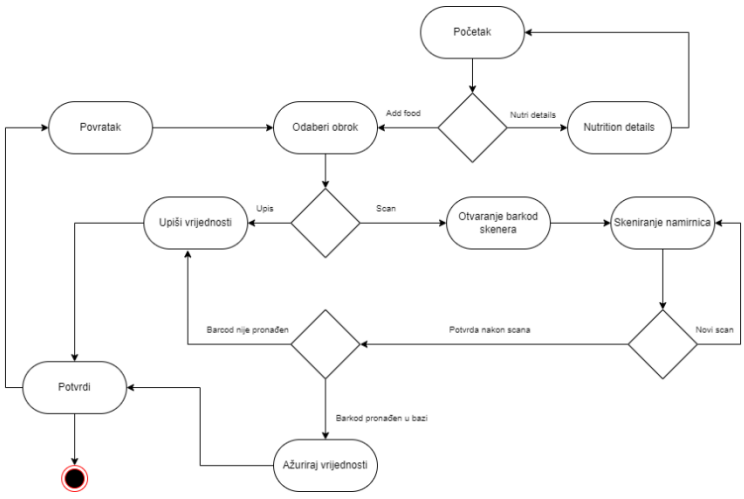
1. Korisnik odabire navedeno klikom na gumb 'My nutrition'
2. Korisnik ima predefinirano 3 obroka (Meal 1, Meal 2, Meal 3) u koje može upisivati kalorijske vrijednosti obroka koje je konzumirao. Uz to, može unijeti i makro-nutrijente (proteine, masti i ugljikohidrate)

3. Za svaki meal je omogućeno korištenje kamere kako bi se moglo, putem skena barkoda, upisati kalorijska vrijednost za određeni obrok i time vrijednost tog barkoda sačuvati u bazu podataka za buduće unose
4. Korisnik ima 'Nutrition' gumb koji kada otvori, dobije uvid u unesene makro-nutrijente. Iz ovog dijela aplikacije korisnik se može vratiti korak unatrag ili na glavno sučelje aplikacije

Opis dijagrama:

Korisnik unutar ovoga pregleda može za tri dnevna obroka upisati kalorijsku vrijednost hrane koju konzumira. Uz to, ima mogućnost upisati makro-nutrijente (proteine, ugljikohidrate i masti). Kako bi mogao preciznije pratiti namirnice koje konzumira, omogućeno je da za svaki obrok može koristiti barkod skener. Naime, kada korisnik odradi skeniranje putem svoje kamere, navedenom barkodu daje vrijednost u obliku kalorija i/ili makro-nutrijenata. Na taj način će korisnik svaki idući put kada skenira isti barkod, dobiti upisano točno te vrijednosti u svoju aplikaciju.

Slika 6 Dijagram aktivnosti My Nutrition



Izvor: autor

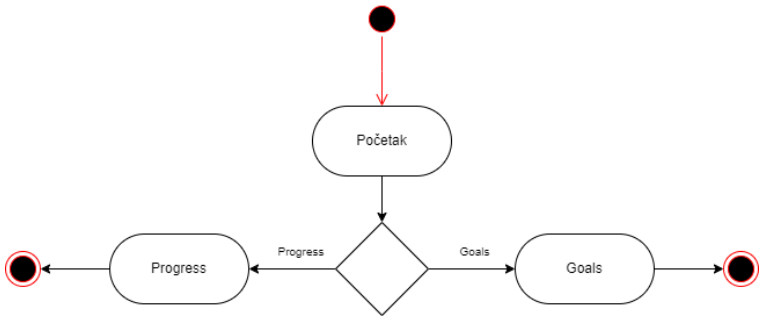
3.2.6 More

1. Korisnik može kliknuti na gumb „More“ u kojem mu se otvara dodatni pregled
2. Korisnik unutar istoga može odabrati „Progress“ ili „Goals“

Opis dijagrama:

Dijagram pokazuje aktivnost „More“ koja je međukorak kako bi se došlo do idućih aktivnosti.

Slika 7 Dijagram aktivnosti More



Izvor: Autor

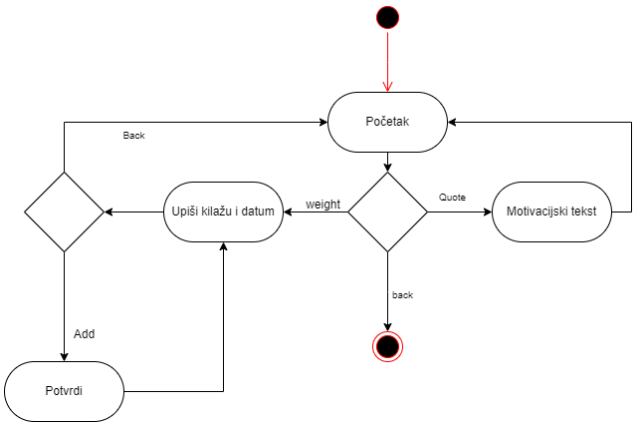
3.2.7 Progress

1. Korisnik pristupa Progressu kroz „More“ pregled, a unutar istoga ima dva smjera
2. Funkcionalnost 'Weight' u koju korisnik upisuje trenutnu kilažu i datum
3. Gumb 'Motivation quote' koji generira motivacijski tekst
4. „Back“ gumb za povratak u prethodni pregled

Opis dijagrama:

Korisnik unutar aktivnosti ima mogućnost upisa vlastite kilaže kao i datuma. Uz to, ima mogućnost generiranja motivacijskog teksta.

Slika 8 Dijagram aktivnosti Progress



Izvor: Autor

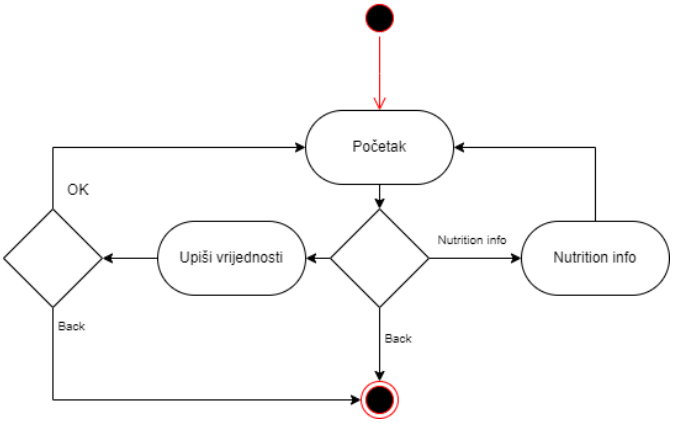
3.2.8 Goals

1. Korisnik pristupa Goalsu putem 'More' pregleda
2. Korisnik upisuje trenutnu kilažu, bira cilj i upisuje željenu kilažu
3. Korisnik može odabrati „Nutrition info“ gumb
4. Korisnik ima back tipku za povratak u „More“ pregled

Opis dijagrama:

Korisnik unutar pregleda ima mogućnost upisati svoju trenutnu, željenu ili buduću kilažu i svoj cilj. Navedena polja može neograničeni broj puta ažurirati. Uz to, može se vratiti u „More“ izbornik ali može i generirati dodatni tekst putem gumba „Nutrition info“

Slika 9 Dijagram aktivnosti Goals



Izvor: Autor

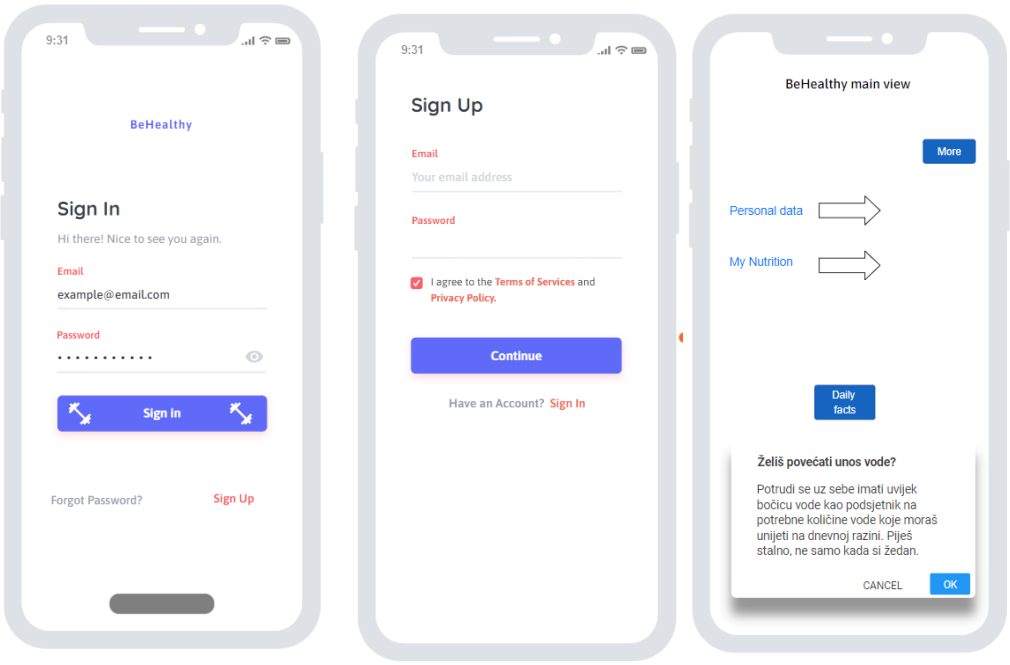
4 Mockup-ovi

Niže su prikazani mockupovi, koji su bili ključni za razvoj mobilne aplikacije. Naime, kroz izradu istih se kreirala i početna ideja kako će sučelje u konačnici izgledati. U osnovi, niže navedeni set mockupova demonstrira osnovnu navigaciju kroz aplikaciju kao što je registracija, pristup aplikaciji i korištenje njezinih osnovnih funkcija: praćenje kilaže i nutritivnog unosa.

4.1 Login & Register & Main view

Na slikama niže su prikazane početne ideje kako bi početno sučelje same aplikacije trebalo izgledati.

Slika 10 Mockupovi Login, Register i Main view

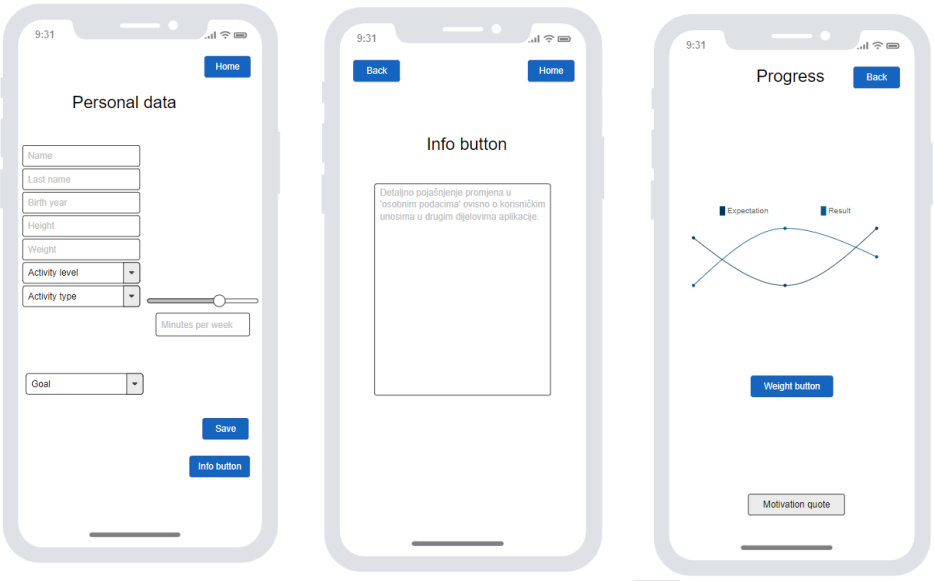


Izvor: Autor

4.2 Personal data & Info button & Progress

Nakon što se korisnik logira i pristupi aplikaciji, niže prikazani prozori koji bi korisniku dali mogućnost upisa osobnih podataka, ali kroz koje bi mogao pratiti svoj napredak.

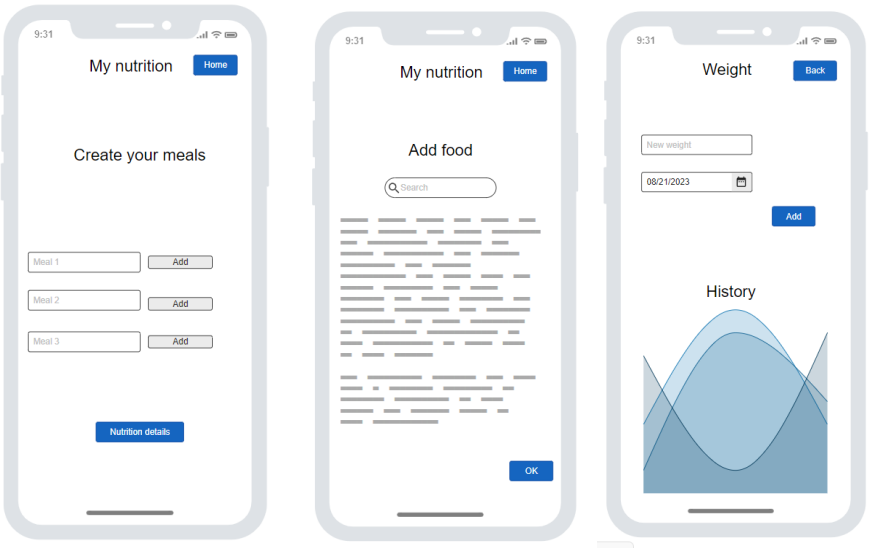
Slika 11 Mockupovi Personal dana, info button i progress



Izvor: Autor

4.3 My Nutrition & Progress

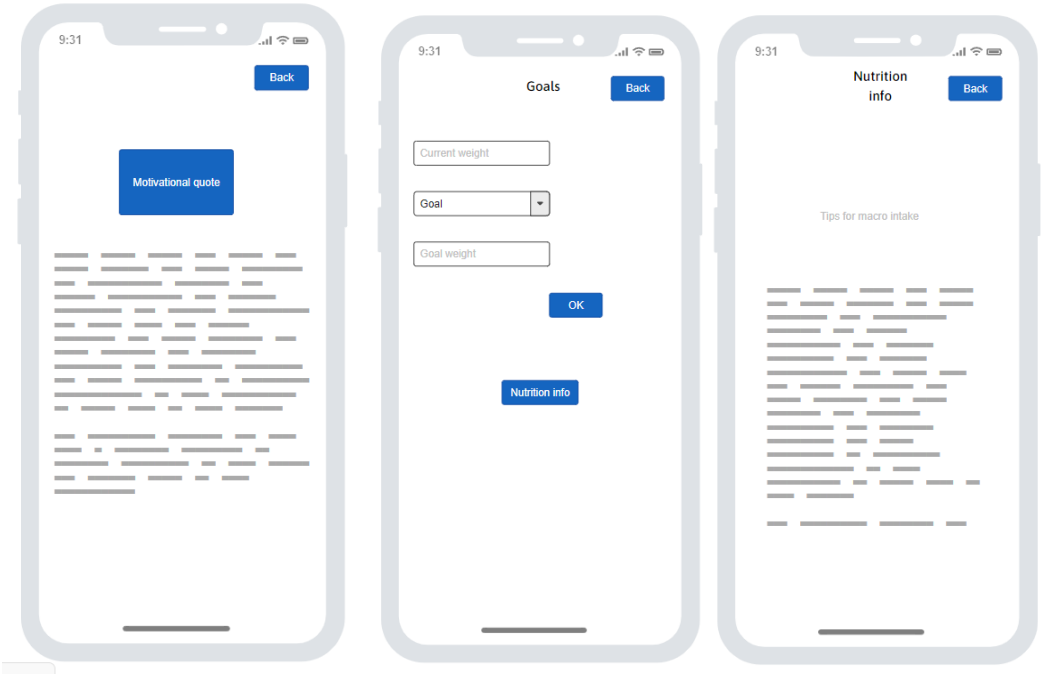
Slika 12 Mockupovi My Nutrition i progress



Izvor: Autor

4.4 Motivational quote & goals & nutrition info

Slika 13 Motivational quote, goals i nutrition info



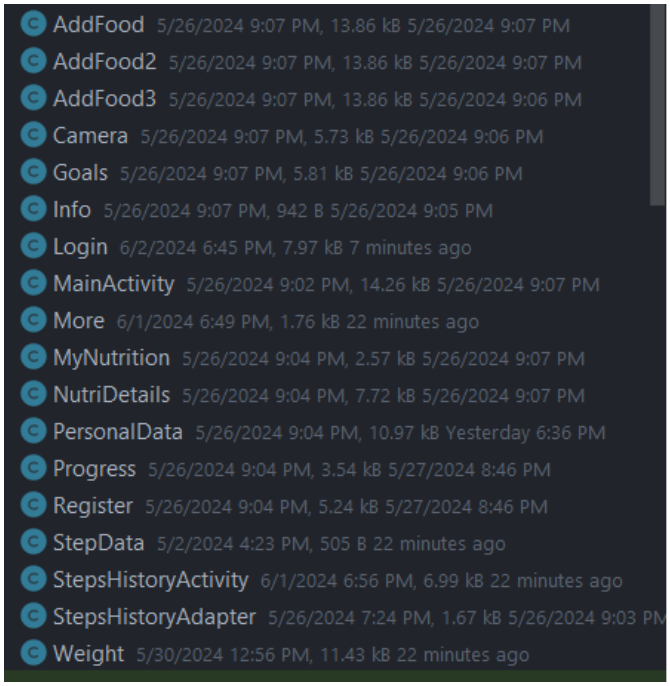
Izvor: Autor

5 Programski kod

Niže se prikazuju i pojašnjavaju ključni dijelovi koda aplikacije. Ovaj segment rada je ključan za razumijevanje praktične primjene koncepata koji su obuhvaćeni u prethodnim poglavljima. Kroz komentare iznad kodova se pruža jasan uvid u logiku rješenja i to ne nužno na način da prikazujemo kompletan programski kod, već fokusirajući se na one najvažnije, krucijalne dijelove. Dijelovi koji se ponavljaju između različitih klasa nisu posebno navođeni za svaku klasu, kao ni slične logike koje se koriste unutar aplikacije, čime se izbjegava redundancija i omogućava lakše praćenje ključnih aspekata implementacije. Osim toga, ova sekcija služi i kao vodič za daljnje unaprjeđenje projekta.

5.1 Java klase & XML

Slika 14 Prikaz programskih klasa



Izvor: Autor

5.2 Login

Login dio koda koristi dvije različite mogućnosti kako bi se omogućila prijava korisnika u aplikaciju. Naime, korisnik ima mogućnost prijavu u aplikaciju odraditi putem podataka koje je koristio prilikom registracije u aplikaciju ili pomoću svojega Google računa. Svakako je preporuka da se aplikaciji pristupa putem Google računa kako bi se iskoristio puni potencijal aplikacije (brojač koraka funkcionira preko Google API servisa za koji je potrebno pristupiti aplikaciji putem Google računa). Pristup aplikaciji preko korisničkih podataka koristi firebase autentikaciju korisnika. Kod vrši provjeru je li korisnik logiran ili nije, ako je logiran, automatski će se preusmjeriti na mainActivity, ako nije, korisnik se može logirati. Pristup preko Google računa omogućuje korisniku odabir Google računa na temelju već prijavljenih preko njegovog mobilnog uređaja ili, ako nema trenutno prijavljenih, korisnik može upisati Google korisničke podatke kako bi pristupio aplikaciji.

Klasa Login u Android aplikaciji omogućava autentifikaciju korisnika putem emaila i Google računa koristeći Firebase. Sadrži elemente poput TextInputEditText za unos, Button za aktiviranje prijave i ProgressBar za prikaz procesa. Korištenjem FirebaseAuth i GoogleSignInClient, klasa upravlja sigurnom i praktičnom prijavom, dok RC_SIGN_IN identifikator pomaže u obradi rezultata Google SignIn aktivnosti.

Slika 15 Klasa Login

```
public class Login extends AppCompatActivity {
    2 usages
    TextInputEditText editTextEmail, editTextPassword;
    2 usages
    Button buttonLogin, buttonGoogleSignIn;
    6 usages
    FirebaseAuth mAuth;
    5 usages
    ProgressBar progressBar;
    2 usages
    TextView textView;
    2 usages
    GoogleSignInClient mGoogleSignInClient;
    2 usages
    private static final int RC_SIGN_IN = 9001;
```

Izvor: Autor

Nakon toga imamo prikazane dvije metode: onStart i onCreate. Metoda onStart automatski provjerava je li korisnik već prijavljen kada se aplikacija pokrene. Ako je korisnik prijavljen, aplikacija prikazuje poruku o tome i preusmjerava ga na glavni ekran, sprječavajući povratak na ekran za prijavu. Metoda onCreate postavlja početni izgled aplikacije i povezuje elemente sučelja s u pozadini. Sakriva se naslovna traka za više prostora na ekranu i definiraju se gumbi i tekstualna polja za unos podataka. Također, postavlja se opcija za prijavu putem Google računa i link za navigaciju na ekran za registraciju novih korisnika.

Slika 16 Metoda onStart i onCreate

```
public void onStart() {
    super.onStart();
    FirebaseUser currentUser = mAuth.getCurrentUser();
    if (currentUser != null) {
        Log.d( tag: "UserInfo",  msg: "Logged in user: " + currentUser.getUid());
        Toast.makeText( context: this,  text: "Already logged in", Toast.LENGTH_SHORT).show();
        Intent intent = new Intent(getApplicationContext(), MainActivity.class);
        startActivity(intent);
        finish();
    }
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (getSupportActionBar() != null) {
        getSupportActionBar().hide();
    }
    setContentView(R.layout.activity_login);

    mAuth = FirebaseAuth.getInstance();
    editTextEmail = findViewById(R.id.email);
    editTextPassword = findViewById(R.id.password);
    buttonLogin = findViewById(R.id.btn_login);
    progressBar = findViewById(R.id.progressBar);
    textView = findViewById(R.id.registerNow);
    buttonGoogleSignIn = findViewById(R.id.btn_google_sign_in);
```

Izvor: Autor

Ovaj dio koda omogućava korisnicima da klikom na tekst prelaze na stranicu za registraciju te omogućuje prijavu putem Google računa. Kada korisnik klikne na tekst, aplikacija ga automatski vodi na novu stranicu za registraciju i zatvara trenutnu stranicu. Također, postavljena je opcija za prijavu pomoću Googlea, gdje korisnici mogu koristiti svoj Google račun za pristup aplikaciji. Klikom na odgovarajući gumb, pokreće se proces prijave preko Googlea.

Slika 17 Pristup aplikaciji putem Google računa

```
textView.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent(getApplicationContext(), Register.class);
        startActivity(intent);
        finish();
    }
});

GoogleSignInOptions gso = new GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
        .requestIdToken("139510997517-170k0l90fv3oq0upjnk876njquefs42b.apps.goog...")
        .requestEmail()
        .build();


mGoogleSignInClient = GoogleSignIn.getClient( activity: this, gso);


buttonGoogleSignIn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent signInIntent = mGoogleSignInClient.getSignInIntent();
        startActivityForResult(signInIntent, RC_SIGN_IN);
    }
});
```

Izvor: Autor

Iduća faza programskog koda nam prikazuje događanja nakon što korisnik klikne gumb za prijavu. Prvo imamo pokazivanje indikatora napretka, aplikacija zatim preuzima email i lozinku koje je korisnik unio. Ako nije došlo do unosa emaila i/ili lozinke, aplikacija će korisniku prikazati poruku da to mora učiniti i uz to zaustaviti proces prijave korisnika.

Slika 18 Slušač na gumbu za pristupanje aplikaciji

```
buttonLogin.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        progressBar.setVisibility(View.VISIBLE);
        String email, password;
        email = String.valueOf(editTextEmail.getText());
        password = String.valueOf(editTextPassword.getText());

        if (TextUtils.isEmpty(email)) {
            Toast.makeText( context: Login.this,  text: "Enter email", Toast.LENGTH_SHORT).show();
            progressBar.setVisibility(View.GONE);
            return;
        }

        if (TextUtils.isEmpty(password)) {
            Toast.makeText( context: Login.this,  text: "Enter password", Toast.LENGTH_SHORT).show();
            progressBar.setVisibility(View.GONE);
            return;
        }
```

Izvor: Autor

Ovaj dio koda, kao nastavak na prethodni, odgovoran je za proces prijave korisnika pomoću unesenih emaila i lozinke. Nakon što korisnik pritisne gumb za prijavu i unese svoje podatke, aplikacija pokušava prijaviti korisnika koristeći Firebase autentifikaciju. Ako je prijava uspješna prikazuje se prigodna poruka i korisnika se preusmjerava na glavni ekran. Ako nije, uz poruku o neuspjehu prijave, korisniku se pruža mogućnost da opet pokuša pristupiti aplikaciji.

Slika 19 Firebase autentifikacija

```
mAuth.signInWithEmailAndPassword(email, password)
        .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                progressBar.setVisibility(View.GONE);
                if (task.isSuccessful()) {
                    Log.d( tag: "UserInfo",  msg: "Logged in user: " + mAuth.getCurrentUser().getUid());
                    Toast.makeText(getApplicationContext(),  text: "Login Successful", Toast.LENGTH_SHORT).show();
                    Intent intent = new Intent(getApplicationContext(), MainActivity.class);
                    startActivity(intent);
                    finish();
                } else {
                    Toast.makeText( context: Login.this,  text: "Authentication failed.",
                            Toast.LENGTH_SHORT).show();
```

Izvor: Autor

Nakon što se korisnik pokuša prijaviti se s Googleom, aplikacija provjerava je li prijava bila uspješna. Ako je uspješna, korisnikov Google račun se koristi za automatsku prijavu u aplikaciju preko Firebase-a. Ako prijava nije uspjela, korisniku se prikazuje poruka o grešci. Ova funkcionalnost osigurava integraciju s Google prijavama unutar aplikacije.

Slika 20 Provjera uspješnosti Google prijave

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    // Rezultat Google prijave
    if (requestCode == RC_SIGN_IN) {
        Task<GoogleSignInAccount> task = GoogleSignIn.getSignedInAccountFromIntent(data);
        handleSignInResult(task);
    }
}

1 usage
private void handleSignInResult(Task<GoogleSignInAccount> completedTask) {
    try {
        GoogleSignInAccount account = completedTask.getResult();
        if (account != null) {
            firebaseAuthWithGoogle(account.getIdToken());
        }
    } catch (Exception e) {
        Log.w( tag: "GoogleSignIn",  msg: "signInResult:failed code=" + e.getMessage());
        Toast.makeText( context: this,  text: "Google sign in failed", Toast.LENGTH_SHORT).show();
```

Izvor: Autor

Zadnji dio koda login klase i dalje definira detalje vezane za Google prijavu. Naime, niže se korisnicima omogućava da se prijave koristeći svoj Google račun. Kada korisnik daje dopuštenje, aplikacija dobiva poseban identifikacijski broj (ID token) od Googlea. Taj broj se koristi za provjeru identiteta korisnika preko Firebase servisa. Ako je sve u redu i prijava je uspješna, korisnika se automatski preusmjerava na glavnu stranicu aplikacije.

Slika 21 Google ID token

```
private void firebaseAuthWithGoogle(String idToken) {
    AuthCredential credential = GoogleAuthProvider.getCredential(idToken,  accessToken: null);
    mAuth.signInWithCredential(credential)
            .addOnCompleteListener( activity: this, new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    if (task.isSuccessful()) {
                        Log.d( tag: "UserInfo",  msg: "signInWithCredential:success");
                        FirebaseUser user = mAuth.getCurrentUser();
                        if (user != null) {
                            Log.d( tag: "UserInfo",  msg: "Logged in user: " + user.getUid());
                            Toast.makeText(getApplicationContext(),  text: "Google sign in successful", Toast.LENGTH_SHORT).show();
                            Intent intent = new Intent(getApplicationContext(), MainActivity.class);
                            startActivity(intent);
                            finish();
                        }
                    } else {
                        Log.w( tag: "UserInfo",  msg: "signInWithCredential:failure", task.getException());
                        Toast.makeText(getApplicationContext(),  text: "Google sign in failed", Toast.LENGTH_SHORT).show();
```

Izvor: Autor

5.3 Register

Ovaj dio koda provjerava je li korisnik već ulogiran, ako je, onda ga automatski prebacuje na mainActivity. Također, dodan je dio da se korisnik može prebaciti na „login“ aktivnost. Dio koda koji vrši provjeru je li korisnik već ulogiran. Ako je, automatski ga prebacuje na glavni zaslon. Uz to, dio koda omogućuje da se korisnik automatski prebaci na drugu aktivnost (login).

Slika 22 Provjera korisnika prije procesa registracije

```
public void onStart() {
    super.onStart();
    FirebaseUser currentUser = mAuth.getCurrentUser();
    if (currentUser != null) {
        Intent intent = new Intent(getApplicationContext(), MainActivity.class);
        startActivity(intent);
        finish();}
}
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (getSupportActionBar() != null) {
        getSupportActionBar().hide();}
    setContentView(R.layout.activity_register);
    db = FirebaseFirestore.getInstance();

    mAuth = FirebaseAuth.getInstance();
    editTextEmail = findViewById(R.id.email);
    editTextPassword = findViewById(R.id.password);
    buttonReg = findViewById(R.id.btn_register);
    progressBar = findViewById(R.id.progressBar);
    textView = findViewById(R.id.loginNow);

    textView.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent intent = new Intent(getApplicationContext(), Login.class);
            startActivity(intent);
            finish();
```

Izvor:Autor

Drugi dio je prikazan u dvije slike niže. Kod „sluša“ korisnika, odnosno prati je li pritisnuo gumb, progress bar ima pokretnu funkciju u trenutcima kada se informacije provjeravaju s baze podataka, preko toast poruke se korisniku daje do znanja je li upisao sve informacije u polja. Ostatak koda radi spremanje korisnika u bazu podataka, odrađuje spremanje njegovog jedinstvenog ID-a koji se koristi i kao jedinstveni ID za spremanje podataka u bazu (kako bi svaki korisnik imao svoje podatke).

Slika 23 Slušač na gumb za registraciju

```
buttonReg.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        progressBar.setVisibility(View.VISIBLE);
        String email, password;
        email = String.valueOf(editTextEmail.getText());
        password = String.valueOf(editTextPassword.getText());

        if (TextUtils.isEmpty(email)) {
            Toast.makeText( context: Register.this,  text: "Enter email", Toast.LENGTH_SHORT).show();
            return;
        }

        if (TextUtils.isEmpty(password)) {
            Toast.makeText( context: Register.this,  text: "Enter password", Toast.LENGTH_SHORT).show();
            return;
        }
```

Izvor: Autor

Nastavak koda prikazuje spremanje u bazu podataka na temelju jedinstvenih korisničkih podataka.

Slika 24 Spremanje u bazu podataka nakon registracije

Izvor: Autor

5.4 Main Activity

Pojašnjenje ove aktivnosti započinje s dvije linije koda koje imaju za ideju omogućiti korištenje Google API servisa kako bi se moglo doći do podataka o broju koraka korisnika. Ove dvije linije koda postavljaju zahtjeve za dozvole unutar aplikacije. Prvi kod je za pristup Google Fit servisima, a drugi za dozvole koje omogućavaju prepoznavanje korisnikove aktivnosti. Ovi kodovi omogućavaju aplikaciji da precizno identificira i obradi odgovore korisnika na zahtjeve za dozvolama.

Slika 25 Google API servisi

```
1 usage
private static final int RC_GOOGLE_FIT_PERMISSIONS_REQUEST_CODE = 1;
2 usages
private static final int RC_ACTIVITY_RECOGNITION_PERMISSION = 101;
```

Izvor: Autor

Kao nastavak na gornje pojašnjenje, ovaj dio definira opcije za pristup Google Fit podacima, specifično za broj koraka. Ako aplikacija nema potrebne dozvole za praćenje aktivnosti korisnika, zahtjeva se dozvola od korisnika. Ako su dozvole već odobrene, provjerava se postoje li potrebne Google Fit dozvole pomoću funkcije checkGoogleFitPermissions. Ovime se osigurava da aplikacija može koristiti podatke o korisničkoj aktivnosti samo ako ima dozvole.

Slika 26 Google dozvole za korištenje fitness opcija

```
fitnessOptions = FitnessOptions.builder()
        .addDataType(DataType.TYPE_STEP_COUNT_DELTA, FitnessOptions.ACCESS_READ)
        .build();


if (ContextCompat.checkSelfPermission( context: this, Manifest.permission.ACTIVITY_RECOGNITION)
        != PackageManager.PERMISSION_GRANTED) {

    ActivityCompat.requestPermissions( activity: this,
            new String[]{Manifest.permission.ACTIVITY_RECOGNITION},
            RC_ACTIVITY_RECOGNITION_PERMISSION);
} else {

    checkGoogleFitPermissions();
}
```

Izvor: Autor

Ova metoda provjerava ima li aplikacija odobrenje za pristup Google Fit podacima korisnika. Ako korisnikov Google račun nema potrebne dozvole, metoda traži te dozvole putem dijaloga za odobrenje. To čini koristeći funkciju GoogleSignIn.requestPermissions. Ako su dozvole već prisutne, metoda pokreće funkciju readStepCountData kako bi dobila podatke o

broju koraka korisnika. Ovo osigurava da aplikacija pristupa korisničkim podacima o zdravlju samo uz njegov pristanak.

Slika 27 Provjera odobrenja za pristup Google Fit podacima

```
2 usages
private void checkGoogleFitPermissions() {
    GoogleSignInAccount account = GoogleSignIn.getAccountForExtension( context: this, fitnessOptions);
    if (!GoogleSignIn.hasPermissions(account, fitnessOptions)) {
        GoogleSignIn.requestPermissions(
                activity: this,
                RC_GOOGLE_FIT_PERMISSIONS_REQUEST_CODE,
                account,
                fitnessOptions);
    } else {
        readStepCountData();
    }
}
```

Izvor: Autor

Ključan dio koda za osiguravanje podataka o broju koraka koji je korisnik odradio nalazi se niže. Naime, kod čita podatke o broju koraka korisnika iz Google Fit servisa za tekući dan na način da prvo postavlja vremenske granice za čitanje podataka: od početka do kraja tekućeg dana. Zatim kreira zahtjev za čitanje podataka koji traži podatke o broju koraka u zadanom vremenskom intervalu. Kod zatim pokreće zadatak čitanja podataka koristeći Google Fit API, vezan za korisnikov Google račun. Kada zadatak završi, provjerava je li uspješno dobio podatke. Ako jest, prolazi kroz dobivene podatke, zbraja korake i prikazuje ukupan broj koraka na korisničkom sučelju. Ako nema dostupnih podataka o koracima ili ako čitanje podataka nije uspjelo, prikazuje se odgovarajuća poruka korisniku

Slika 28 Čitanje podataka o broju koraka

```
public void onComplete(@NonNull Task<DataReadResponse> task) {
    Log.d( tag: "FitnessActivity", msg: "Task completed");
    if (task.isSuccessful()) {
        DataReadResponse response = task.getResult();
        if (response != null && !response.getDataSets().isEmpty()) {
            int stepCount = 0;
            for (com.google.android.gms.fitness.data.DataSet dataSet : response.getDataSets()) {
                for (com.google.android.gms.fitness.data.DataPoint dataPoint : dataSet.getDataPoints()) {
                    for (Field field : dataPoint.getDataType().getFields()) {
                        stepCount += dataPoint.getValue(field).asInt();
                        Log.d( tag: "FitnessActivity", msg: "Field: " + field.getName() + " Value: " + dataPoint.getValue(field)
                    }
                }
            }
            Log.d( tag: "FitnessActivity", msg: "Total steps: " + stepCount);
            stepCountText.setText(String.valueOf(stepCount));
        } else {
            Log.d( tag: "FitnessActivity", msg: "No data available");
            Toast.makeText( context: MainActivity.this, text: "No step count data available", Toast.LENGTH_SHORT).show();
        }
    } else {
        Log.d( tag: "FitnessActivity", msg: "Failed to read step data", task.getException());
        Toast.makeText( context: MainActivity.this, text: "Failed to read step count data", Toast.LENGTH_SHORT).show();
    }
}
```

Izvor: Autor

Budući da korisnik može unutar aplikacije vidjeti podatke za prethodnih sedam dana, dio koda niže definira načine kako bi se isto izvršilo.

Slika 29 Povijesni prikaz broja koraka

```
private void readStepHistory() {
    Calendar cal = Calendar.getInstance();
    cal.setTime(new Date());
    cal.set(Calendar.HOUR_OF_DAY, 0); // Postavi na početak dana
    cal.set(Calendar.MINUTE, 0);
    cal.set(Calendar.SECOND, 0);
    cal.set(Calendar.MILLISECOND, 0);
    long endTime = cal.getTimeInMillis();  // Kraj dana
    cal.add(Calendar.DAY_OF_YEAR, amount: -7);  // Uzmi podatke za posljednjih 7 dana
    long startTime = cal.getTimeInMillis();  // Početak dana

    DataReadRequest readRequest = new DataReadRequest.Builder()
            .read(DataType.TYPE_STEP_COUNT_DELTA)
            .setTimeRange(startTime, endTime, TimeUnit.MILLISECONDS)
            .build();

    GoogleSignInAccount account = GoogleSignIn.getAccountForExtension( context: this, fitnessOptions);

    Task<DataReadResponse> task = Fitness.getHistoryClient( activity: this, account).readData(readRequest);

    task.addOnCompleteListener(new OnCompleteListener<DataReadResponse>() {
        @Override
        public void onComplete(@NonNull Task<DataReadResponse> task) {
            if (task.isSuccessful()) {
                DataReadResponse response = task.getResult();
                if (response != null && !response.getDataSets().isEmpty()) {
                    int stepCount = 0;
                    for (com.google.android.gms.fitness.data.DataSet dataSet : response.getDataSets()) {
                        for (com.google.android.gms.fitness.data.DataPoint dataPoint : dataSet.getDataPoints()) {
                            for (Field field : dataPoint.getDataType().getFields()) {
                                stepCount += dataPoint.getValue(field).asInt();
                                Log.d( tag: "FitnessActivity", msg: "Field: " + field.getName() + " Value: " + dataPoint.getValue(field).asInt());
                            }
                        }
                    }
                    Log.d( tag: "FitnessActivity", msg: "Total steps in last 7 days: " + stepCount);
                    Toast.makeText( context: MainActivity.this, text: "Total steps in last 7 days: " + stepCount, Toast.LENGTH_SHORT).show();
                } else {
                    Log.d( tag: "FitnessActivity", msg: "No data available");
                    Toast.makeText( context: MainActivity.this, text: "No step count data available for last 7 days", Toast.LENGTH_SHORT).show();
                }
            } else {
                Log.d( tag: "FitnessActivity", msg: "Failed to read step data", task.getException());
                Toast.makeText( context: MainActivity.this, text: "Failed to read step count data for last 7 days", Toast.LENGTH_SHORT).show();
```

Izvor: Autor

Korisnik na glavnom zaslonu aplikacije ima mogućnost preko gumba generirati tekst koji mu pomaže u ostvarenju ciljeva. Koji će se tekst prikazivati definira kod niže.

Slika 30 Niz sugestija

```
// Lista daily factsa
dailyFacts = new ArrayList<>();
dailyFacts.add("Stay hydrated: Always carry a water bottle with you to ...");
dailyFacts.add("Diverse diet: Consume a varied diet rich in fruits, veg...");
dailyFacts.add("Regular exercise: Set goals for weekly physical activit...");
dailyFacts.add("Adequate sleep: Prioritize getting 7-9 hours of quality...");
dailyFacts.add("Mindful eating: Pay attention to portion sizes and eat ...");
dailyFacts.add("Stress management: Practice relaxation techniques such ...");
dailyFacts.add("Limit processed foods: Minimize your intake of highly p...");
dailyFacts.add("Social connections: Cultivate meaningful relationships ...");
dailyFacts.add("Sun protection: Apply sunscreen and wear protective clo...");
dailyFacts.add("Stay curious: Engage in lifelong learning and try new a...");
```

Izvor: Autor

Kako bi svakim klikom aplikacija generirala novi tekst, morali smo kreirati 'slušač' koji će na svako korištenje gumba korisniku generirati novu poruku.

Slika 31 Slušač za generiranje teksta

```
dailyFactsButton = findViewById(R.id.daily_facts_button);
dailyFactsText = findViewById(R.id.daily_facts_text);
stepCountText = findViewById(R.id.step_count);

dailyFactsButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        showNextFact(); //
    }
});
```

Izvor: Autor

Ostatak programskog koda ove klase omogućuje korisnike odjavu iz Google računa i otvaranje ostalih dijelova aplikacije putem više pojedinačnih intenta.

5.5 Steps

U sljedećem segmentu detaljnije prolazimo kroz tri klase koje omogućuju prikaz podataka o broju koraka korisnika. Dio koda za navedeno je već prikazan kroz klasu MainActivity, a ostatak pojašnjen niže.

5.5.1 Step Data

Klasa StepData je model podataka koji čuva informacije o datumu i broju koraka koje je korisnik napravio. Ima dva privatna atributa: date za datum i stepCount za broj koraka. Također, klasa ima funkciju za ispisivanje ovih informacija u jednostavnom formatu putem metode "toString".

Slika 32 Klasa StepData

```
public class StepData {
    3 usages
    private String date;
    3 usages
    private int stepCount;

    1 usage
    public StepData(String date, int stepCount) {
        this.date = date;
        this.stepCount = stepCount;
    }

    public String getDate() { return date; }

    1 usage
    public int getStepCount() { return stepCount; }

    @Override
    public String toString() { return "Date: " + date + ", Steps: " + stepCount; }
```

Izvor: Autor

5.5.2 Steps History Activity

Pojašnjenje ove klase je razdijeljeno u tri cjeline. Prva pojašnjava linije koda prikazane na slici niže. Za početak, kod sakriva naslovnu traku na vrhu ekrana, postavlja "layout" koji definira kako stranica izgleda i priprema listu za prikaz podataka o koracima. Postavlja i dugme koje, kada se pritisne, vodi korisnika nazad na glavnu stranicu aplikacije. Možemo reći da ovaj dio koda definira vizualne detalje i priprema dijelove za prikaz ali ne radi dohvaćanje podataka koji će biti prikazani.

Slika 33 Vizualni detalji klase Steps History Activity

```
protected void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (getSupportActionBar() != null) {
        getSupportActionBar().hide();
    }
    setContentView(R.layout.steps_history);


    recyclerView = findViewById(R.id.recycler_view_steps_history);
    recyclerView.setLayoutManager(new LinearLayoutManager( context: this));


    stepCounts = new ArrayList<>();


    readStepHistory();

    BackButton = findViewById(R.id.Back_button);
    BackButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) { openMainActivity(); }
    });
}
```

Izvor: Autor

Drugi dio koda radi glavninu posla u ovoj klasi, on koristi Google Fit API za prikupljanje podataka o broju koraka korisnika tijekom posljednjih 7 dana. Prvo se stvara zahtjev za pristup podacima, a zatim se povezuje s Google Fit servisom koristeći korisnički račun. Kod definira razdoblje od sedam dana unatrag od trenutnog dana i traži podatke za to

razdoblje. Nakon slanja zahtjeva, se obrađuje odgovor: ako je zadatak uspješno izvršen, podaci se obrađuju i sortiraju po datumu, a zatim se prikazuju u komponenti "RecyclerView". Ako dođe do greške, ispisuje se poruka o grešci. Ovo omogućava korisnicima da vizualno prate svoje aktivnosti preko korisničkog sučelja aplikacije. Pojedinačni dijelovi koda su dodatno pojašnjeni unutar samoga koda.

Slika 34 Povezivanje s Google

```
private void readStepHistory() {
    // Kreiranje fitness opcija
    FitnessOptions fitnessOptions = FitnessOptions.builder()
            .addDataType(DataType.TYPE_STEP_COUNT_DELTA, FitnessOptions.ACCESS_READ)
            .build();

    // Povezivanje s Google Fit servisom
    GoogleSignInAccount account = GoogleSignIn.getAccountForExtension( context: this, fitnessOptions);

    // Postavljanje vremenskog raspona na prethodnih 7 dana
    Calendar cal = Calendar.getInstance();
    cal.setTime(new Date());
    long endTime = cal.getTimeInMillis();
    cal.add(Calendar.DAY_OF_YEAR,  amount: -7);
    long startTime = cal.getTimeInMillis();

    // Kreiranje zahtjeva za čitanje podataka
    DataReadRequest readRequest = new DataReadRequest.Builder()
            .read(DataType.TYPE_STEP_COUNT_DELTA)
            .setTimeRange(startTime, endTime, TimeUnit.MILLISECONDS)
            .build();
```

Izvor: Autor

Slika 35 Obrada rezultata i konvertiranje podataka

```
// Obrada rezultata
task.addOnCompleteListener(new OnCompleteListener<DataReadResponse>() {
    @Override
    public void onComplete(@NonNull Task<DataReadResponse> task) {
        if (task.isSuccessful()) {
            DataReadResponse response = task.getResult();
            if (response != null && response.getDataSets() != null && !response.getDataSets().isEmpty()) {
                Map<String, Integer> stepsByDate = new HashMap<>();
                for (DataPoint dp : response.getDataSets().get(0).getDataPoints()) {
                    long startTimeMillis = dp.getStartTime(TimeUnit.MILLISECONDS);
                    String date = formatDate(startTimeMillis);
                    int steps = dp.getValue(Field.FIELD_STEPS).asInt();
                    if (stepsByDate.containsKey(date)) {
                        stepsByDate.put(date, stepsByDate.get(date) + steps);
                    } else {
                        stepsByDate.put(date, steps);
                    }
                }
                // Konvertiranje podataka u format StepData
                for (Map.Entry<String, Integer> entry : stepsByDate.entrySet()) {
                    stepCounts.add(new StepData(entry.getKey(), entry.getValue()));
                }
                // Sortiranje liste po datumima u rastućem redoslijedu
                Collections.sort(stepCounts, (sd1, sd2) -> {
                    Date date1 = parseDate(sd1.getDate());
                    Date date2 = parseDate(sd2.getDate());
                    if (date1 != null && date2 != null) {
                        return date1.compareTo(date2);
                    }
                    return 0;
                });
                // Oznaka da su podaci učitani
                isDataLoaded = true;
                // Kreiranje i postavljanje adaptera ako su podaci učitani
                if (isDataLoaded) {
                    adapter = new StepsHistoryAdapter(stepCounts);
                    recyclerView.setAdapter(adapter);
                }
            }
        } else {
            Log.e( tag: "StepsHistoryActivity",  msg: "Error reading step data", task.getException());
        }
```

Izvor: Autor

Posljednji dio koda ove klase izvršava tri funkcije: pretvara vrijeme iz milisekundi u format datuma koji je čitljiv, a to uspijeva koristeći „SimpleDataFormat“. Pretvara tekstualni datum natrag u objekt „Date“ i omogućava korisniku povratak na glavnu aktivnost koristeći „Intent“.

Slika 36 Simple data format

```
1 usage
private String formatDate(long timeInMillis) {
    SimpleDateFormat formatter = new SimpleDateFormat( pattern: "dd.MM.yyyy", Locale.ENGLISH);
    return formatter.format(new Date(timeInMillis));
}


2 usages
private Date parseDate(String dateStr) {
    try {
        SimpleDateFormat formatter = new SimpleDateFormat( pattern: "dd.MM.yyyy", Locale.ENGLISH);
        return formatter.parse(dateStr);
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}

1 usage
public void openMainActivity(){
    Intent intent = new Intent( packageContext: this, MainActivity.class);
    startActivity(intent);
}
```

Izvor: Autor

5.5.3 Steps History Adapter

Ovaj kod predstavlja adapter za prikaz liste koraka u mobilnoj aplikaciji. Adapter organizira kako će se svaki zapis o koracima prikazivati unutar liste: za svaki zapis postavlja datum i broj koraka. Koristi se poseban format datoteka za dizajn svakog zapisa, koji se automatski učitava i prikazuje. Ovaj postupak omogućava korisnicima aplikacije da lako pregledavaju svoju aktivnost po danima.

Slika 37 Klasa Steps History Adapter

```
public class StepsHistoryAdapter extends RecyclerView.Adapter<StepsHistoryAdapter.StepViewHolder>

    3 usages
    private List<StepData> stepDataList;

    1 usage
    public StepsHistoryAdapter(List<StepData> stepDataList) { this.stepDataList = stepDataList; }

    @NonNull
    @Override
    public StepViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View itemView = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.item_steps_history, parent, attachToRoot: false);
        return new StepViewHolder(itemView);
    }

    @Override
    public void onBindViewHolder(@NonNull StepViewHolder holder, int position) {
        StepData stepData = stepDataList.get(position);
        holder.dateTextView.setText(stepData.getDate());
        holder.stepsCountTextView.setText(String.valueOf(stepData.getStepCount()));
    }

    @Override
    public int getItemCount() { return stepDataList.size(); }

    4 usages
    public static class StepViewHolder extends RecyclerView.ViewHolder {
        2 usages
        public TextView dateTextView;
        2 usages
        public TextView stepsCountTextView;

        1 usage
        public StepViewHolder(View view) {
            super(view);
            dateTextView = view.findViewById(R.id.date_text_view);
            stepsCountTextView = view.findViewById(R.id.steps_count_text_view);
        }
```

Izvor: Autor

5.6 More

Možemo reći da ovaj dio koda i aplikacije postoji kao međukorak kako bi se došlo do dodatnih funkcionalnosti same aplikacije. Pomoću ovoga koda se upravlja navigacijom unutar aplikacije kroz različite aktivnosti pomoću gumba. Svaki gumb na ekranu More je povezan s akcijom koja otvara novu aktivnost: gumb za ciljeve (GoalsButt) otvara aktivnost Goals, gumb

za napredak (Progress) vodi do aktivnosti Progress, a gumb za početni ekran (HomeButton) vraća korisnika na glavnu aktivnost (MainActivity)

Slika 38 Progress i Goals tipke

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (getSupportActionBar() != null) {
        getSupportActionBar().hide();
    }
    setContentView(R.layout.activity_more);

    GoalsButt = (Button) findViewById(R.id.goals_button);
    GoalsButt.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) { openGoals(); }
    });



    Progress = (Button) findViewById(R.id.progress_button);
    Progress.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) { openProgress(); }
    });
    HomeButton = (Button) findViewById(R.id.home_button);
    HomeButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) { openMainActivity(); }
    });
```

Izvor: Autor

5.7 Personal data

Kako bi svaki korisnik imao svoje osobne podatke moramo imati dio koda koji omogućuje korisniku da se diferencira od ostalih korisnika. Slike koda niže će u detalje pojasniti na koji način se u aplikaciji unose osobni podaci korisnika i kako isti komuniciraju s bazom podataka gdje se u konačnici i spremaju.

Ovaj dio koda u Android aplikaciji postavlja elemente korisničkog sučelja za odabir spola i aktivnosti. Dva checkboxa omogućavaju korisniku da izabere spol (muški ili ženski), dok dva spinnera nude opcije za odabir razine aktivnosti i ciljeva kroz padajuće liste. Ove liste se pune predefiniranim opcijama koje se nalaze u resursima aplikacije. Na kraju, automatski se pokušava učitati postojeće korisničke podatke iz baze podataka.

Slika 39 Odabir spola i aktivnosti

```
maleCheckbox = findViewById(R.id.male_checkbox);
femaleCheckbox = findViewById(R.id.female_checkbox);

Spinner activityLevelSpinner = findViewById(R.id.activity_level_spinner);
Spinner goalTypeSpinner = findViewById(R.id.goal);

ArrayAdapter<CharSequence> activityLevelAdapter = ArrayAdapter.createFromResource(
        context: this,
        R.array.activity_levels,
        android.R.layout.simple_spinner_item
);
activityLevelAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
activityLevelSpinner.setAdapter(activityLevelAdapter);

ArrayAdapter<CharSequence> goalAdapter = ArrayAdapter.createFromResource(
        context: this,
        R.array.goal,
        android.R.layout.simple_spinner_item
);
goalAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
goalTypeSpinner.setAdapter(goalAdapter);

loadPersonalDataFromFirestore();
```

Izvor: Autor

Kako je gore napisano, ako postoje podaci u bazi podataka, aplikacija će pokušati iste dohvatiti i smjestiti ih u odgovarajuća mjesta. Time postižemo da kada korisnik jednom upiše podatke, isti budu evidentirani dok god ih on ne odluči ažurirati ili obrisati. Kod niže učitava osobne podatke korisnika iz Firestore baze podataka, ako je korisnik prijavljen.

Slika 40 Dohvat podataka iz Firestore baze podataka

```
private void loadPersonalDataFromFirestore() {
    if (user != null) {
        String userId = user.getUid();
        db.collection( collectionPath: "users").document(userId) DocumentReference
                .collection( collectionPath: "personal_data").document( documentPath: "user_data")
                .get() Task<DocumentSnapshot>
                .addOnSuccessListener(documentSnapshot -> {
                    if (documentSnapshot.exists()) {
                        String name = documentSnapshot.getString( field: "name");
                        String lastName = documentSnapshot.getString( field: "last_name");
                        String birthYear = documentSnapshot.getString( field: "birth_year");
                        String height = documentSnapshot.getString( field: "height");
                        String weight = documentSnapshot.getString( field: "weight");
                        String activityLevel = documentSnapshot.getString( field: "activity_level");
                        String goalType = documentSnapshot.getString( field: "goal_type");
                        String gender = documentSnapshot.getString( field: "gender");
                        EditText nameEditText = findViewById(R.id.name_edit_text);
                        nameEditText.setText(name);
                        EditText lastNameEditText = findViewById(R.id.last_name_edit_text);
                        lastNameEditText.setText(lastName);
                        EditText birthYearEditText = findViewById(R.id.birth_year_edit_text);
                        birthYearEditText.setText(birthYear);
                        EditText heightEditText = findViewById(R.id.height_edit_text);
                        heightEditText.setText(height);
                        EditText weightEditText = findViewById(R.id.weight_edit_text);
                        weightEditText.setText(weight);
                        Spinner activityLevelSpinner = findViewById(R.id.activity_level_spinner);
                        setSelectedSpinnerValue(activityLevelSpinner, activityLevel);
                        Spinner goalTypeSpinner = findViewById(R.id.goal);
                        setSelectedSpinnerValue(goalTypeSpinner, goalType);
                        if (gender.equals("Male")) {
                            maleCheckbox.setChecked(true);
                        } else if (gender.equals("Female")) {
                            femaleCheckbox.setChecked(true);
                        }
                    }
                })
                .addOnFailureListener(e -> {
                    Toast.makeText( context: PersonalData.this, text: "Failed to load personal data", Toast.LENGTH_SHORT).show();
                });
    }
```

Izvor: Autor

Metoda savePersonalDataToFirestore služi za spremanje korisničkih podataka u Firestore bazu podataka. Kada korisnik popuni formu i potvrdi unos, metoda prikuplja sve unesene podatke i sprema ih u bazu pod korisničkim identifikatorom (userId).

Slika 41 Metoda savePersonalDataToFirestore

```
private void savePersonalDataToFirestore(String name, String lastName, String birthYear, String height, String weight,
                                         String activityLevel, String goalType, String gender) {
    if (user != null) {
        String userId = user.getUid();

        Map<String, Object> personalData = new HashMap<>();
        personalData.put("name", name);
        personalData.put("last_name", lastName);
        personalData.put("birth_year", birthYear);
        personalData.put("height", height);
        personalData.put("weight", weight);
        personalData.put("activity_level", activityLevel);
        personalData.put("goal_type", goalType);
        personalData.put("gender", gender);

        db.collection( collectionPath: "users").document(userId) DocumentReference
                .collection( collectionPath: "personal_data").document( documentPath: "user_data")
                .set(personalData) Task<Void>
                .addOnSuccessListener(aVoid -> {
                    Toast.makeText( context: PersonalData.this, text: "Personal data saved successfully", Toast.LENGTH_SHORT).show();
                })
                .addOnFailureListener(e -> {
                    Toast.makeText( context: PersonalData.this, text: "Failed to save personal data", Toast.LENGTH_SHORT).show();
                });
    }
```

Izvor: Autor

5.8 Info

Jednostavan dio koda koji omogućuje povratak u PersonalData i postavlja se izgled aktivnosti koji je definiran pomoću XML layouta. Unutar istoga se korisniku prikazuju dodatne informacije o samoj aplikaciji.

Slika 42 Gumb za povratak i postavljanje izgleda

```
protected void onCreate(Bundle savedInstanceState) {


    super.onCreate(savedInstanceState);
    if (getSupportActionBar() != null) {
        getSupportActionBar().hide();
    }
    setContentView(R.layout.activity_info);

    Button back =(Button) findViewById(R.id.back_button);
    back.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) { openPersonalData(); }
    });
```

Izvor: autor

5.9 Progress

Progress, kao što i ime kaže prati napredak korisnika aplikacije. Stoga, kod ima za zadatak korisniku prikazati na koji način se kreće njegova kilaža u odnosu na njegovo očekivanje. Metoda loadAndDisplayData služi za učitavanje i prikazivanje korisničkih ciljeva vezanih za težinu iz Firestore baze podataka. Nakon što metoda provjeri da je korisnik prijavljen (user != null), koristi se korisnikov jedinstveni identifikator (userId) za pristup njegovim zapisima unutar kolekcije "users" u Firestoreu. Specifično, metoda traži dokument "user_goals" unutar podkolekcije "goals" koji sadrži korisnikove ciljeve. Ako se dokument pronađe, unutar aplikacije se prikazuje ciljana i trenutna kilaža.

Slika 43 metoda LoadAndDisplayData

```
private void loadAndDisplayData() {
    if (user != null) {
        String userId = user.getUid();
        db.collection( collectionPath: "users").document(userId) DocumentReference
                .collection( collectionPath: "goals").document( documentPath: "user_goals")
                .get() Task<DocumentSnapshot>
                .addOnSuccessListener(documentSnapshot -> {
                    if (documentSnapshot.exists()) {
                        String expectation = documentSnapshot.getString( field: "goal_weight");
                        String reality = documentSnapshot.getString( field: "current_weight");

                        expectationText.setText("Expectation: " + expectation +"kg");
                        realityText.setText("Reality: " + reality +"kg");
                    }
                })
                .addOnFailureListener(e -> {

                });
    }
}
```

Izvor: Autor

Kako bi korisnik, kada mu ponestane motivacije, dobio poruku podrške, omogućeno je generirane motivacijskih poruka pomoću koda niže. Motivacijske poruke su zapisane u strings.xml datoteci, a korisnik pomoću gumba u aplikaciji povlači unaprijed definirani tekst.

Slika 44 Gumb za generiranje motivacijskog sadržaja

```
motivationalQuoteButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        int index = counter.getAndIncrement() % motivationalQuotes.length;
        String quote = motivationalQuotes[index];
        motivationalQuoteText.setText(quote);
    }
});
```

Izvor: Autor

5.10 Weight

Korisnik unutar ove klase ima dvije osnovne mogućnosti: upisati kilažu i datum na koji se navedena kilaža odnosi i pratiti kretanje vlastite kilaže, za sve unesene datume u bazi pomoću grafa koji je dostupan u aplikaciji. Naime, korisničko sučelje sadrži graf koji ima mogućnost prikaza kilaže kroz četiri različita vremenska perioda: jedan mjesec, tri mjeseca, dvanaest mjeseci i potpuni prikaz (prikazuje kompletnu povijest).

Kako bi se u bazu podataka spremili podaci pod ispravnim datumom, potrebno je u kodu omogućiti korisniku odabir datuma. Ovaj kod upravlja klikom na dateText element. Kad korisnik klikne na dateText, ako je datePicker vidljiv, uzima se odabrani datum i prikazuje kao tekst u dateText, a datePicker se sakriva. Ako datePicker nije vidljiv, postaje vidljiv i dateText se mijenja u "OK". Na ovaj način korisnik može birati datum i vidjeti ga prikazanog kao tekst.

Slika 45 Odabir datuma

```
dateText = findViewById(R.id.date_text);
datePicker = findViewById(R.id.date_picker);
addWeightButton = findViewById(R.id.add_weight_button);
weightEditText = findViewById(R.id.new_weight_edit_text);
lineChart = findViewById(R.id.line_chart);

Locale.setDefault(Locale.ENGLISH);

dateText.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (datePicker.getVisibility() == View.VISIBLE) {
            int day = datePicker.getDayOfMonth();
            int month = datePicker.getMonth() + 1;
            int year = datePicker.getYear();

            String selectedDate = String.format(Locale.ENGLISH, format: "%02d/%02d/%d", month, day, year);
            dateText.setText(selectedDate);

            datePicker.setVisibility(View.GONE);
        } else {
            datePicker.setVisibility(View.VISIBLE);
            dateText.setText("OK");
        }
    }
});
```

Izvor: Autor

Nakon što korisnik klikne na gumb, dohvaća se datum iz datePickera i težina iz tekstualnog polja weightEditText. Ako polje za težinu nije prazno, stvara se mapa (weightData)

koja sadrži težinu i datum, te se ti podaci spremaju u Firestore bazu podataka. Nakon što se spreme podaci, ažurira se graf. Također, ažurira se trenutna kilaža u dva dijela aplikacije (goals i personal data)

Slika 46 Spremanje podataka o težini u bazu podataka

```
public void onClick(View v) {
    int day = datePicker.getDayOfMonth();
    int month = datePicker.getMonth() + 1;
    int year = datePicker.getYear();

    String selectedDate = String.format(Locale.ENGLISH, format: "%02d/%02d/%d", month, day, year);
    String weight = weightEditText.getText().toString();

    if (weight.isEmpty()) {
        Toast.makeText( context: Weight.this, text: "Enter weight", Toast.LENGTH_SHORT).show();
    } else {
        Map<String, Object> weightData = new HashMap<>();
        weightData.put("weight", weight);
        weightData.put("date", selectedDate);

        db.collection( collectionPath: "users").document(userId).collection( collectionPath: "weight").add(weightData)
                .addOnSuccessListener(aVoid -> {
                    Toast.makeText( context: Weight.this, text: "Data has been saved.", Toast.LENGTH_SHORT).show();
                    loadChartData( days: -1);
                })
                .addOnFailureListener(e -> Toast.makeText( context: Weight.this, text: "Error saving data.", Toast.LENGTH_SHORT).show());

        Map<String, Object> goalData = new HashMap<>();
        goalData.put("current_weight", weight);
        db.collection( collectionPath: "users").document(userId).collection( collectionPath: "goals").document( documentPath: "user_goals").update(goalData);

        Map<String, Object> personalData = new HashMap<>();
        personalData.put("weight", weight);
        db.collection( collectionPath: "users").document(userId).collection( collectionPath: "personal_data").document( documentPath: "user_data").update(personalData);
    }
```

Izvor: Autor

Kao što je spomenuto u pojašnjenju iznad, Postoji graf koji se ažurira nakon određenih upisa koje odrađujemo. Ovaj kod učitava podatke o korisnikovoj težini iz baze podataka i prikazuje ih na grafikonu. Prvo, dohvaća sve zapise o težini korisnika i sortira ih po datumu. Ako je zadan određeni broj dana (days), filtrira podatke kako bi prikazao samo težinu iz tog razdoblja. Zatim, svaki zapis o težini dodaje na grafikon, prikazujući kako se težina mijenjala tijekom vremena.

Slika 47 Dohvat podataka o težini korisnika iz baze podataka

```
private void loadChartData(int days) {
    db.collection( collectionPath: "users").document(userId).collection( collectionPath: "weight") CollectionReference
            .orderBy( field: "date", Query.Direction.ASCENDING) Query
            .get() Task<QuerySnapshot>
            .addOnCompleteListener(task -> {
                if (task.isSuccessful() && task.getResult() != null) {
                    List<Entry> entries = new ArrayList<>();
                    SimpleDateFormat sdf = new SimpleDateFormat( pattern: "MM/dd/yyyy", Locale.ENGLISH);
                    Date latestDate = null;

                    for (DocumentSnapshot document : task.getResult().getDocuments()) {
                        String dateStr = document.getString( field: "date");
                        String weightStr = document.getString( field: "weight");
                        if (dateStr != null && weightStr != null) {
                            try {
                                float weight = Float.parseFloat(weightStr);
                                Date date = sdf.parse(dateStr);
                                if (date != null) {
                                    entries.add(new Entry(date.getTime(), weight));
                                    if (latestDate == null || date.after(latestDate)) {
                                        latestDate = date;
                                    }
                                }
                            } catch (Exception e) {
                                e.printStackTrace();
                            }
                        }
                    }

                    if (entries.isEmpty()) {
                        Toast.makeText( context: Weight.this, text: "No data to display.", Toast.LENGTH_SHORT).show();
                        return;
                    }
```

Izvor: Autor

Ako nema podataka ili se dogodi greška, prikazuje se poruka korisniku. Nakon što su podaci spremni, kreira se grafikon s tim podacima, postavlja boje i izgled grafikona, te se prikazuje u aplikaciji. Na taj način, korisnik može vidjeti svoju težinu kroz različite periode.

Slika 48 Kreiranje grafa za prikaz dohvaćenih podataka

```
        if (days > 0 && latestDate != null) {
            Calendar calendar = Calendar.getInstance();
            calendar.setTime(latestDate);
            calendar.add(Calendar.DAY_OF_YEAR, -days);
            Date thresholdDate = calendar.getTime();

            List<Entry> filteredEntries = new ArrayList<>();
            for (Entry entry : entries) {
                if (new Date((long) entry.getX()).after(thresholdDate)) {
                    filteredEntries.add(entry);
                }
            }
            entries = filteredEntries;
        }

        if (entries.isEmpty()) {
            Toast.makeText( context Weight.this,  text "No data to display for the selected period.", Toast.LENGTH_SHORT).show();
            return;
        }

        Collections.sort(entries, (e1, e2) -> Float.compare(e1.getX(), e2.getX()));

        LineDataSet dataSet = new LineDataSet(entries,  label "Weight Over Time");
        dataSet.setColor(Color.BLUE);
        dataSet.setValueTextColor(Color.BLACK);

        LineData lineData = new LineData(dataSet);
        lineChart.setData(lineData);

        lineChart.getXAxis().setValueFormatter(new DateAxisValueFormatter(entries));
        lineChart.getXAxis().setPosition(XAxis.XAxisPosition.BOTTOM);
        lineChart.getXAxis().setGranularity(1f);
        lineChart.getXAxis().setGranularityEnabled(true);
        lineChart.getXAxis().setLabelRotationAngle(-45); // Rotate labels for better visibility
        lineChart.invalidate();
    } else {
        Toast.makeText( context Weight.this,  text "Error loading data.", Toast.LENGTH_SHORT).show();
    }
})
.addOnFailureListener(e -> Toast.makeText( context Weight.this,  text "Error loading data.", Toast.LENGTH_SHORT).show());
```

Izvor: Autor

Kako bi korisnik znao čitati informacije na grafu, potrebno je imati datume koji se prilagođavaju realnim datumima koji su odabrani prilikom upisa težine u bazu. DateAxisValueFormatter formatira datume za prikaz na osi x grafikona. Ova klasa omogućava da se datumi prikazuju jasno i u odgovarajućem formatu na grafikonu.

Slika 49 Prilagodba datuma za prikaz na grafu

```
private class DateAxisValueFormatter extends ValueFormatter {
    1 usage
    private final List<Entry> entries;
    5 usages
    private final SimpleDateFormat dateFormat;

    1 usage
    DateAxisValueFormatter(List<Entry> entries) {
        this.entries = entries;
        if (!entries.isEmpty()) {

            long start = (long) entries.get(0).getX();
            long end = (long) entries.get(entries.size() - 1).getX();
            long span = end - start;
            if (span > 31536000000L) {
                this.dateFormat = new SimpleDateFormat( pattern: "MMM yy", Locale.ENGLISH);
            } else if (span > 2592000000L) {
                this.dateFormat = new SimpleDateFormat( pattern: "MMM dd", Locale.ENGLISH);
            } else {
                this.dateFormat = new SimpleDateFormat( pattern: "MM/dd", Locale.ENGLISH);
            }
        } else {
            this.dateFormat = new SimpleDateFormat( pattern: "MM/dd", Locale.ENGLISH);
        }
    }
```

Izvor: Autor

Zadnje metode iz ove klase definiraju kako će se datumi prikazivati na osi x grafikona. Metoda getFormattedValue formatira vrijednosti osi x grafikona u datumski format koristeći postavljeni SimpleDateFormat. Metoda configureChartAxis osigurava da su datumi na osi x grafikona jasno vidljivi i čitljivi tako što definira pozicionirane, granularnost i rotaciju.

Slika 50 Prikaz datuma na grafu

```
        public String getFormattedValue(float value) {
            return dateFormat.format(new Date((long) value));
        }
    }

    1 usage
    private void configureChartAxis() {
        XAxis xAxis = lineChart.getXAxis();
        xAxis.setPosition(XAxis.XAxisPosition.BOTTOM);
        xAxis.setGranularity(1f);
        xAxis.setGranularityEnabled(true);
        xAxis.setLabelRotationAngle(-45);
    }
```

Izvor: Autor

5.11 Goals

Unutar klase goals imamo metodu koja korisniku omogućuje pregled korisnih savjeta o prehrani. Niže je prikazan kod u kojem se prolazi kroz petlju kako bi se pokazali svi korisni savjeti dostupni u aplikaciji.

Slika 51 Generiranje prehrambenih savjeta

```
private void showNextTip() {
    if (currentTipIndex < nutritionTips.length) {
        tipTextView.setText(nutritionTips[currentTipIndex]);
        currentTipIndex = (currentTipIndex + 1) % nutritionTips.length;
    }
}
```

Izvor: Autor

5.12 My Nutrition

Klasa MyNutrition predstavlja jednu od ključnih komponenti unutar aplikacije, omogućujući korisnicima pristup različitim funkcionalnostima vezanim za unos makro-nutrijenata i kalorija. Uz ovu klasu su povezane još tri klase koje korisnicima omogućuju unos makro-nutrijenata i kalorija u svoj prehrambeni dnevnik, skeniranje barkodova uz korištenje kamere i pogled na ukupni unos kao na ukupnu potrošnju kroz tekući dan. Za početak, korisniku je omogućeno da preko gumba uđe na onaj obrok koji bi htio kreirati/ažurirati, odradi povratak na glavno sučelje aplikacije ili pogleda ukupni kalorijski unos i potrošnju za taj dan.

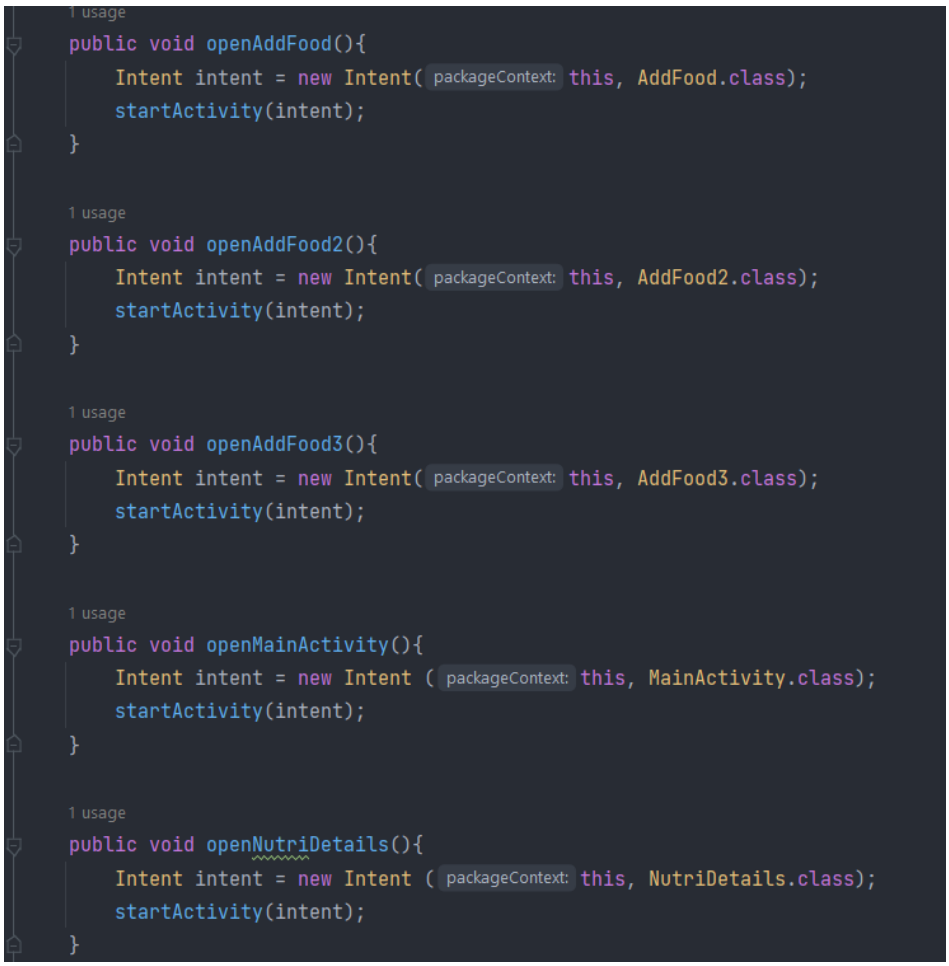
Slika 52 Ulazak u pregled za dodavanje kalorija

```
NutriDet = (Button) findViewById(R.id.nutrition_details_button);
NutriDet.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) { openNutriDetails(); }
});

AddFoodbtn1 = (Button) findViewById(R.id.add_button_1);
AddFoodbtn2 = (Button) findViewById(R.id.add_button_2);
AddFoodbtn3 = (Button) findViewById(R.id.add_button_3);
Home = (Button) findViewById(R.id.home_button);

Home.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) { openMainActivity(); }
});

AddFoodbtn1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) { openAddFood(); }
});

AddFoodbtn2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) { openAddFood2(); }
});

AddFoodbtn3.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) { openAddFood3(); }
});
```

Izvor: Autor

Ovisno o pritisnutom gumbu, preko intenta se korisniku pokreće neka od aktivnosti vidljivih u kodu niže.

Slika 53 Pokretanje aktivnosti AddFood



Izvor: Autor

5.13 Add Food

Naredni dio rada detaljno pojašnjava i analizira ključne dijelove koda koji omogućuju unos i praćenje prehrambenih podataka unutar aplikacije. Poseban fokus je na korištenju funkcionalnosti skeniranja barkoda, kao i na spremanju podataka u firebase.

Ovaj kod unosi podatke o obroku u bazu podataka. Prvo, provjerava jesu li kalorije unesene. Ako nisu, prikazuje poruku da se unesu kalorije. Ako je skeniran barkod, koristi metodu checkAndUpdateDocumentWithBarcode za ažuriranje ili dodavanje zapisa temeljenog na barkodu. Ako barkod nije skeniran, koristi metodu checkAndUpdateDocumentWithoutBarcode za ažuriranje ili dodavanje zapisa bez barkoda.

Slika 54 Spremanje unesenih kalorija u bazu podataka

```
private void saveDataToDatabase() {
    String calories = caloriesEditText.getText().toString();
    String currentDate = dateEditText.getText().toString();
    String mealName = "Meal1_" + new SimpleDateFormat( pattern: "MM", Locale.getDefault()).format(Calendar.getInstance().getTime());

    if (calories.isEmpty()) {
        Toast.makeText( context: this,  text: "Enter calories", Toast.LENGTH_SHORT).show();
    } else {
        if (scannedBarcode != null) {
            checkAndUpdateDocumentWithBarcode(currentDate, calories, mealName);
        } else {
            checkAndUpdateDocumentWithoutBarcode(currentDate, calories, mealName);
        }
    }
}
```

Izvor: Autor

Kako bi podaci bili ispravno spremljeni u bazu podataka, kod niže radi pretraživanje baze podataka i traži dokumente u kolekciji "meals" koji odgovaraju datumu "currentDate", nemaju barkod i imaju isti naziv obroka (mealName). Ako pronađe takav dokument, s njegovim ID-em poziva metodu updateDocument preko koje ažurira podatke. Ako ne pronađe dokument, poziva metodu addNewDocument kako bi stvorio novi zapis u bazi podataka s unesenim podacima.

Slika 55 Pretraga baze i ažuriranje postojećih unosa

```
private void checkAndUpdateDocumentWithoutBarcode(String currentDate, String calories, String mealName) {
    db.collection( collectionPath: "users").document(userId).collection( collectionPath: "meals") CollectionReference
            .whereEqualTo( field: "entry_date", currentDate) Query
            .whereEqualTo( field: "barcode",  value: null)
            .whereEqualTo( field: "meal_name", mealName)
            .get() Task<QuerySnapshot>
            .addOnCompleteListener(task -> {
                if (task.isSuccessful()) {
                    List<DocumentSnapshot> documents = task.getResult().getDocuments();
                    if (!documents.isEmpty()) {
                        DocumentSnapshot existingDocument = documents.get(0);
                        String documentId = existingDocument.getId();
                        Log.d( tag: "AddFood",  msg: "Found existing document with ID: " + documentId);
                        updateDocument(documentId, calories,  updateLastEntry: true);  // Treći parametar dodan
                    } else {
                        Log.d( tag: "AddFood",  msg: "No existing document found, adding new.");
                        addNewDocument(currentDate, calories, mealName);
                    }
                }
            });
}
```

Izvor: Autor

Ovisno o tome koja je metoda pozvana, izvršava se neki od ova dva koda niže.

Slika 56 Metode UpdateDocument i AddNewDocument

```
private void updateDocument(String documentId, String calories, boolean updateLastEntry) {
    Map<String, Object> updateData = new HashMap<>();
    updateData.put("calories", calories);
    updateData.put("proteins", proteinsEditText.getText().toString());
    updateData.put("fats", fatsEditText.getText().toString());
    updateData.put("carbs", carbsEditText.getText().toString());

    if (updateLastEntry) {
        updateData.put("last_entry", Calendar.getInstance().getTime());
    }

    db.collection( collectionPath: "users").document(userId).collection( collectionPath: "meals").document(documentId) DocumentReference
            .update(updateData) Task<Void>
            .addOnSuccessListener(aVoid -> {
                Toast.makeText( context: this,  text: "Barcode data has been updated.", Toast.LENGTH_SHORT).show();
                clearFields();
            })
            .addOnFailureListener(e -> Toast.makeText( context: this,  text: "Failed to update barcode data.", Toast.LENGTH_SHORT).show());
}

3 usages
private void addNewDocument(String currentDate, String calories, String mealName) {
    Map<String, Object> mealData = new HashMap<>();
    mealData.put("calories", calories);
    mealData.put("proteins", proteinsEditText.getText().toString());
    mealData.put("fats", fatsEditText.getText().toString());
    mealData.put("carbs", carbsEditText.getText().toString());
    mealData.put("entry_date", currentDate);
    mealData.put("meal_name", mealName);
    mealData.put("last_entry", Calendar.getInstance().getTime());
    mealData.put("barcode", scannedBarcode);

    db.collection( collectionPath: "users").document(userId).collection( collectionPath: "meals") CollectionReference
            .add(mealData) Task<DocumentReference>
            .addOnSuccessListener(documentReference -> {
                Toast.makeText( context: this,  text: "Data has been saved.", Toast.LENGTH_SHORT).show();
                clearFields();
            })
```

Izvor: Autor

Ovaj kod dohvaća podatke o obroku iz baze podataka na temelju skeniranog barkoda. Ako postoje zapisi s tim barkodom, uzima najnoviji zapis i postavlja njegove vrijednosti (kalorije, proteine, masti i ugljikohidrate) u polja za unos. Ako nema zapisa s tim barkodom, učitava najnovije podatke o obroku. Ako dođe do greške pri dohvaćanju podataka, prikazuje poruku o neuspjehu.

Slika 57 Učitavanje podataka putem barkoda

```
private void loadMealDataByBarcode(String barcode) {
    db.collection( collectionPath: "users").document(userId).collection( collectionPath: "meals") CollectionReference
            .whereEqualTo( field: "barcode", barcode) Query
            .get() Task<QuerySnapshot>
            .addOnCompleteListener(task -> {
                if (task.isSuccessful()) {
                    List<DocumentSnapshot> documents = task.getResult().getDocuments();
                    if (!documents.isEmpty()) {
                        Collections.sort(documents, (d1, d2) -> {
                            Date date1 = d1.getDate( field: "last_entry");
                            Date date2 = d2.getDate( field: "last_entry");
                            if (date1 == null || date2 == null) return 0;
                            return date2.compareTo(date1);
                        });

                        DocumentSnapshot latestDocument = documents.get(0);
                        String calories = latestDocument.getString( field: "calories");
                        String proteins = latestDocument.getString( field: "proteins");
                        String fats = latestDocument.getString( field: "fats");
                        String carbs = latestDocument.getString( field: "carbs");

                        caloriesEditText.setText(calories);
                        proteinsEditText.setText(proteins);
                        fatsEditText.setText(fats);
                        carbsEditText.setText(carbs);
                    } else {
                        loadLatestMealData();
                    }
                } else {
                    Log.e( tag: "AddFood",  msg: "Failed to load meal data by barcode: " + task.getException().getMessage());
                    Toast.makeText( context: this,  text: "Failed to load meal data by barcode.", Toast.LENGTH_SHORT).show();
                }
            });
}
```

Izvor: Autor

Kod niže omogućava otvaranje kamere za skeniranje barkoda i obradu rezultata skeniranja. Metoda openCamera pokreće aktivnost kamere i šalje naziv obroka kao dodatni podatak (mealType). Kada se kamera zatvori, metoda onActivityResult provjerava rezultat skeniranja. Ako je skeniranje uspješno, dohvaća skenirani barkod i učitava podatke o obroku s tim barkodom (loadMealDataByBarcode). Ako barkod nije skeniran, učitava najnovije podatke o obroku (loadLatestMealData).

Slika 58 Pokretanje kamere uređaja

```
public void openCamera() {
    Intent intent = new Intent( packageContext: this, Camera.class);
    intent.putExtra( name: "MEAL_TYPE", mealType); // Dodajemo mealType u intent
    startActivityForResult(intent, REQUEST_CODE_CAMERA);
}

2 usages
private static final int REQUEST_CODE_CAMERA = 101;

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == REQUEST_CODE_CAMERA && resultCode == RESULT_OK) {
        scannedBarcode = data.getStringExtra( name: "SCANNED_BARCODE");
        if (scannedBarcode != null) {
            loadMealDataByBarcode(scannedBarcode);
        } else {
            loadLatestMealData();
        }
    }
```

Izvor: Autor

5.14 Camera

Za optimalno korištenje kamere u aplikaciji, ključno je da svi dijelovi koda budu precizno organizirani. Posebno je važno osigurati da aplikacija pravilno pamti iz kojeg "meala" je kamera otvorena, kako bi se osigurao povratak na ispravnu klasu i pravilno spremanje podataka u bazu. Kod omogućava da se informacije o obroku i skenirani barkod pravilno povežu s odgovarajućim unosom u bazi podataka. Nakon snimanja, aplikacija se vraća u klasu iz koje je kamera pokrenuta, osiguravajući da podaci budu spremljeni na točno mjesto, dok skenirani barkod služi kao ključna informacija za prepoznavanje i ažuriranje obroka u bazi podataka.

Ovaj kod vraća korisnika na prethodnu stranicu ovisno o vrsti obroka (mealType). Ako je mealType "meal1", "meal2" ili "meal3", otvara odgovarajuću stranicu (AddFood, AddFood2, AddFood3)

Slika 59 Povratak u prethodnu aktivnost

```
private void openPreviousActivity() {
    Intent intent;
    if (mealType != null) {
        switch (mealType) {
            case "meal1":
                intent = new Intent( packageContext: this, AddFood.class);
                break;
            case "meal2":
                intent = new Intent( packageContext: this, AddFood2.class);
                break;
            case "meal3":
                intent = new Intent( packageContext: this, AddFood3.class);
                break;
            default:
                intent = new Intent( packageContext: this, AddFood.class);
                break;
        }
        startActivity(intent);
    } else {
        intent = new Intent( packageContext: this, AddFood.class);
        startActivity(intent);
    }
    finish();
```

Izvor: Autor

Ako korisnik odluči skenirati proizvod, metoda niže pokreće navedeni proces.

Slika 60 Skeniranje putem kamere uređaja

```
2 usages
private void startScan() {
    IntentIntegrator integrator = new IntentIntegrator( activity: this);
    integrator.setOrientationLocked(false);
    integrator.initiateScan();
    Log.d( tag: "Camera",  msg: "Starting barcode scan...");
}
```

Izvor: Autor

Nakon što je izvršeno skeniranje barkoda, korisniku se prikazuje dohvaćena vrijednost. Korisnik može napraviti novo skeniranje ili može prihvatiti vrijednost.

Slika 61 Prikaz dohvaćene vrijednosti barkoda

```
private void showScannedBarcodeDialog(String barcodeContents) {
    AlertDialog.Builder builder = new AlertDialog.Builder( context: this);
    builder.setTitle("Scanned Barcode");
    builder.setMessage("Scanned Barcode: " + barcodeContents);
    builder.setPositiveButton( text: "OK", (dialog, which) -> {

        openAddFoodWithBarcode(barcodeContents, mealType);
    });
    builder.setNegativeButton( text: "Rescan", (dialog, which) -> {

        startScan();
    });
    builder.setCancelable(false);
    builder.show();
}
```

Izvor: Autor

Ako je korisnik koristio funkcionalnost skeniranja barkoda i prihvatio skenirani barkod, informacija o navedenom barkodu mora biti proslijeđena natrag u klasu koja omogućuje unos informacija u bazu podataka. Ovaj proces se izvršava pomoću sljedećeg koda.

Slika 62 Otvaranje obroka putem barkoda

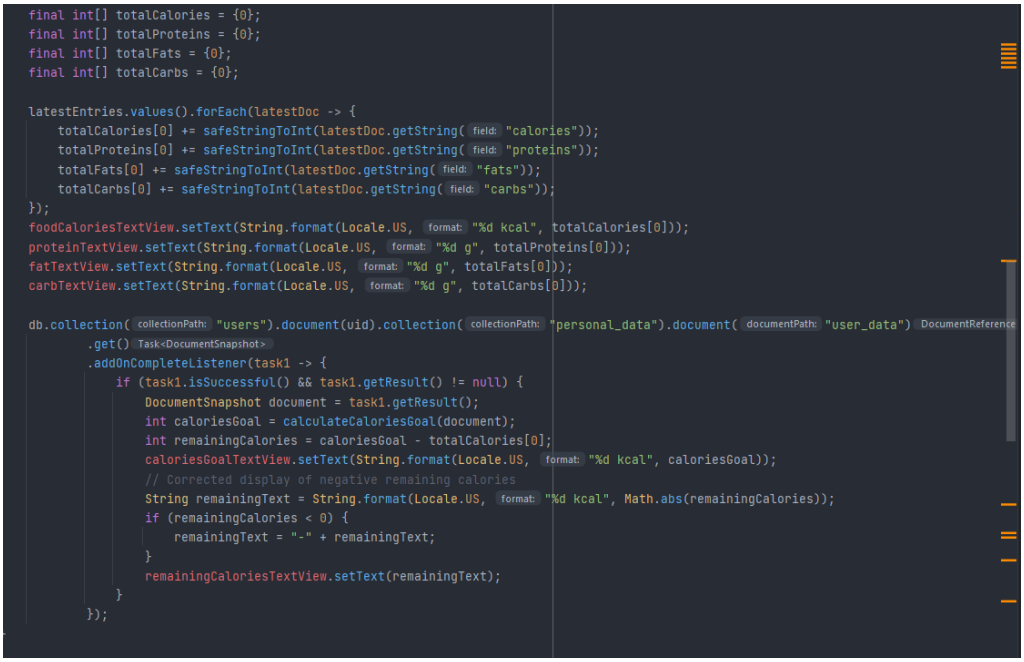
```
private void openAddFoodWithBarcode(String barcodeContents, String mealType) {
    Intent intent;
    if (mealType != null) {
        switch (mealType) {
            case "meal1":
                intent = new Intent( packageContext: this, AddFood.class);
                break;
            case "meal2":
                intent = new Intent( packageContext: this, AddFood2.class);
                break;
            case "meal3":
                intent = new Intent( packageContext: this, AddFood3.class);
                break;
            default:
                intent = new Intent( packageContext: this, AddFood.class);
                break;
        }
    } else {
        intent = new Intent( packageContext: this, AddFood.class);
    }
    intent.putExtra( name: "SCANNED_BARCODE", barcodeContents);
    intent.putExtra( name: "MEAL_TYPE", mealType);
    startActivity(intent);
}
```

Izvor: Autor

5.15 Nutrition details

Klasa "Nutrition Details" pruža korisniku objedinjeni pregled svih unesenih obroka i njihove nutritivne vrijednosti. Unutar ove klase, korisnik može vidjeti svoju ukupnu dnevnu potrošnju kalorija, izračunatu na temelju unesenih obroka, razine aktivnosti, postavljenih ciljeva i indeksa tjelesne mase. Ova klasa omogućava korisniku praćenje prehrambenih unosa u odnosu na njihove osobne ciljeve i tjelesnu aktivnost, pružajući sveobuhvatan uvid u nutritivni status i napredak. Ovaj kod učitava i prikazuje podatke o unosu hrane korisnika iz Firebase Firestore baze podataka te izračunava ukupni unos kalorija, proteina, masti i ugljikohidrata za tekući dan. Također, učitava osobne podatke korisnika i prikazuje njegov dnevni cilj kalorija te preostale kalorije. Ako bismo ušli dublje u analizu koda, uz gornje opisani dio, možemo reći da kod prvo radi provjeru korisnika kako bi dohvatio trenutno aktivnog korisnika. Nakon toga, dohvaća podatke o obrocima te iste filtrira kako bi pronašao najnovije unose. Tek nakon što je napravio te korake odrađuje izračun ukupnih vrijednosti i prikaz istih u textView.

Slika 63 Prilagodba podataka za prikaz u aplikaciji



```
final int[] totalCalories = {0};
final int[] totalProteins = {0};
final int[] totalFats = {0};
final int[] totalCarbs = {0};

latestEntries.values().forEach(latestDoc -> {
    totalCalories[0] += safeStringToInt(latestDoc.getString( field: "calories"));
    totalProteins[0] += safeStringToInt(latestDoc.getString( field: "proteins"));
    totalFats[0] += safeStringToInt(latestDoc.getString( field: "fats"));
    totalCarbs[0] += safeStringToInt(latestDoc.getString( field: "carbs"));
});
foodCaloriesTextView.setText(String.format(Locale.US, format: "%d kcal", totalCalories[0]));
proteinTextView.setText(String.format(Locale.US, format: "%d g", totalProteins[0]));
fatTextView.setText(String.format(Locale.US, format: "%d g", totalFats[0]));
carbTextView.setText(String.format(Locale.US, format: "%d g", totalCarbs[0]));

db.collection( collectionPath: "users").document(uid).collection( collectionPath: "personal_data").document( documentPath: "user_data") DocumentReference
        .get() Task<DocumentSnapshot>
        .addOnCompleteListener(task1 -> {
            if (task1.isSuccessful() && task1.getResult() != null) {
                DocumentSnapshot document = task1.getResult();
                int caloriesGoal = calculateCaloriesGoal(document);
                int remainingCalories = caloriesGoal - totalCalories[0];
                caloriesGoalTextView.setText(String.format(Locale.US, format: "%d kcal", caloriesGoal));
                // Corrected display of negative remaining calories
                String remainingText = String.format(Locale.US, format: "%d kcal", Math.abs(remainingCalories));
                if (remainingCalories < 0) {
                    remainingText = "-" + remainingText;
                }
                remainingCaloriesTextView.setText(remainingText);
            }
        });
```

Izvor: Autor

Kako bi aplikacija uopće mogla prikazati kalorijske potrebe, potrebno je u kodu imati matematičku logiku koja na temelju dostupnih informacija radi izračun. Metoda calculateCaloriesGoal dohvaća korisnikove podatke iz baze podataka. Zatim te podatke prosljeđuje metodi calculateBasalMetabolism, koja koristi formulu za izračun bazalnog metabolizma i prilagođava ga razini aktivnosti korisnika kako bi dobila ukupni dnevni kalorijski cilj. Zajedno, ove dvije metode omogućuju izračun dnevnog kalorijskog cilja temeljenog na osobnim podacima i razini aktivnosti korisnika.

Slika 64 Izračun potrebnog kalorijskog unosa

```
private int calculateCaloriesGoal(DocumentSnapshot document) {
    String goalType = document.getString( field: "goal_type");
    int birthYear = Integer.parseInt(document.getString( field: "birth_year"));
    String height = document.getString( field: "height");
    String weight = document.getString( field: "weight");
    String activityLevel = document.getString( field: "activity_level");
    String gender = document.getString( field: "gender");
    return calculateBasalMetabolism(birthYear, height, weight, activityLevel, gender);
}

1 usage
private int calculateBasalMetabolism(int birthYear, String height, String weight, String activityLevel, String gender) {
    int age = 2023 - birthYear;
    double heightInCm = Double.parseDouble(height);
    double weightInKg = Double.parseDouble(weight);
    double bmr = (10 * weightInKg) + (6.25 * heightInCm) - (5 * age) + (gender.equals("Female") ? -161 : 5);
    if (activityLevel.equals("Lightly Active")) bmr *= 1.375;
    else if (activityLevel.equals("Moderately Active")) bmr *= 1.55;
    else if (activityLevel.equals("Very Active")) bmr *= 1.725;
    else bmr *= 1.2;
    return (int) bmr;
}
```

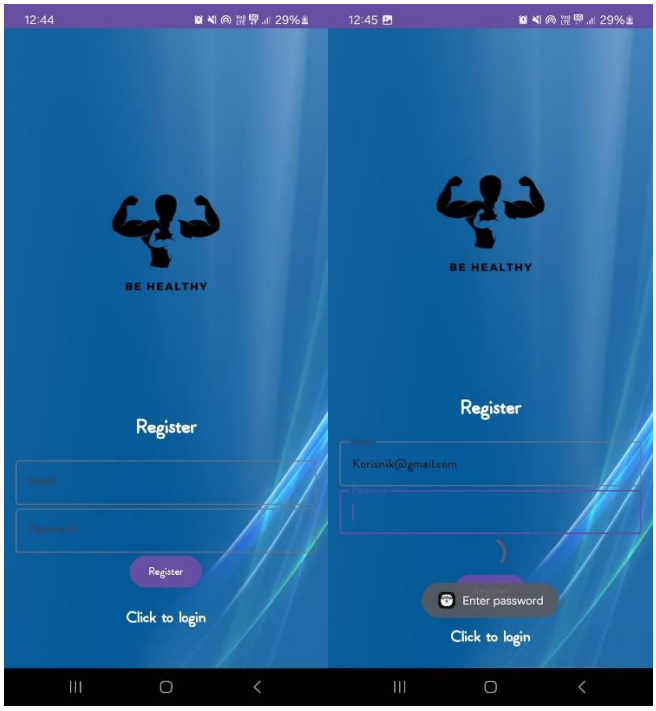
Izvor: Autor

6 Aplikacija

Ovo poglavlje posvećeno je vizualnom dizajnu i korisničkom sučelju aplikacije. Detaljno će se opisati izgled svake stranice aplikacije, uključujući navigacijske elemente, tipke, obrasce za unos podataka i grafičke prikaze. Priložene slike zaslona pružit će jasan uvid u raspored i funkcionalnost sučelja, dok će opisi naglasiti odluke koje pridonose intuitivnosti i estetskoj privlačnosti aplikacije.

Glavno registracijsko sučelje omogućuje korisnicima kreiranje novog računa ako ga već nemaju, ili ako ne žele koristiti Google račun za pristup aplikaciji. Kao što je prikazano na donjoj slici, korisnik ima mogućnost unosa potrebnih podataka poput e-mail adrese i lozinke, uz opciju "click to login" za prebacivanje na sučelje za prijavu. U primjeru je prikazano što se događa kada korisnik ne unese lozinku prilikom registracije - aplikacija prikazuje obavijest o nedostatku podataka. Ista se situacija događa ako korisnik ne unese e-mail adresu. Ova funkcionalnost osigurava da korisnici unesu sve potrebne informacije prije nego što dovrše registraciju.

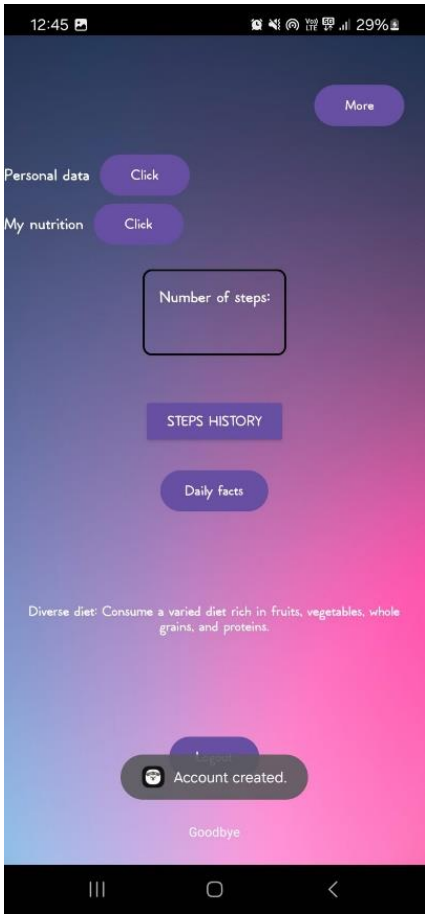
Slika 65 Login i Register prozori aplikacije



Izvor: Autor

Nakon što korisnik unese podatke u oba polja, kreira se korisnički račun i korisnik se preusmjerava na glavno sučelje aplikacije prikazano u nastavku. Budući da se radi o novom korisničkom računu, korisniku još nisu evidentirani broj koraka niti osobni podaci. Unatoč tome, niže je prikazano glavno korisničko sučelje koje vidi svaki korisnik nakon registracije i pristupa aplikaciji.

Slika 66 Glavno sučelje aplikacije

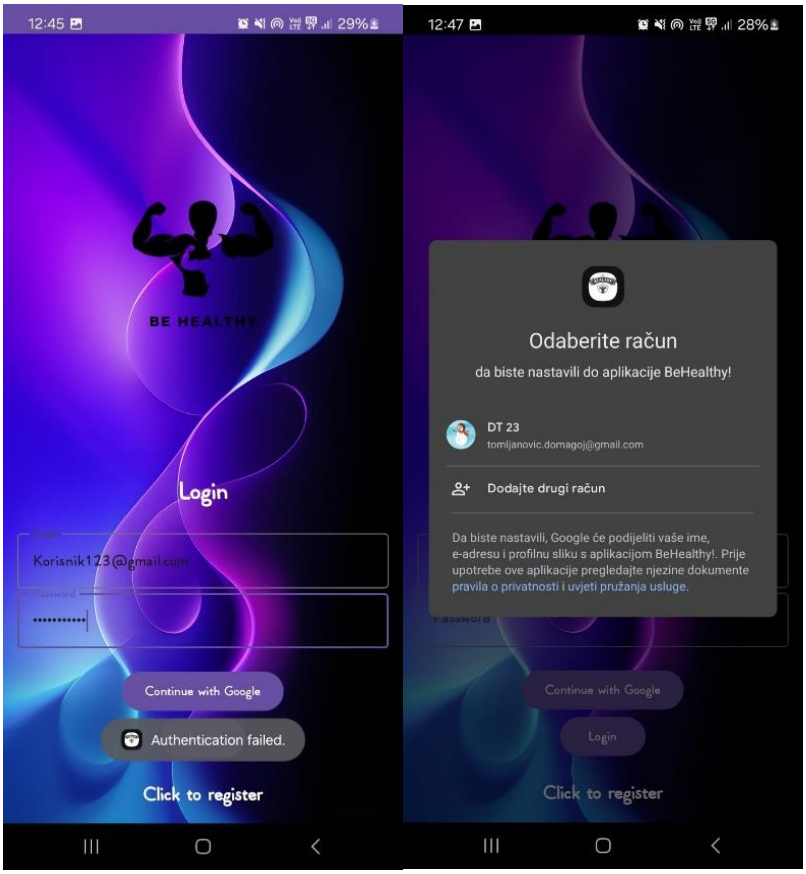


Izvor: Autor

Kako bi korisnici imali pristup svim funkcionalnostima koje aplikacija pruža, preporučuje se povezivanje računa s Google računom. Brojač koraka unutar aplikacije integriran je putem Google API-ja, koji zahtijeva korištenje Google računa za omogućavanje

praćenja koraka. Osim korištenja Google računa, korisnici moraju dati odobrenje za pristup podacima kako bi aplikacija mogla pravilno funkcionirati. Prva slika prikazuje ponašanje aplikacije kada korisnik unese pogrešnu lozinku, dok druga slika pokazuje reakciju aplikacije u slučaju ispravne lozinke. Nakon uspješnog pristupa aplikaciji, korisniku se nudi mogućnost odabira Google računa za daljnju integraciju.

Slika 67 Pristup aplikaciji i odabir Google računa

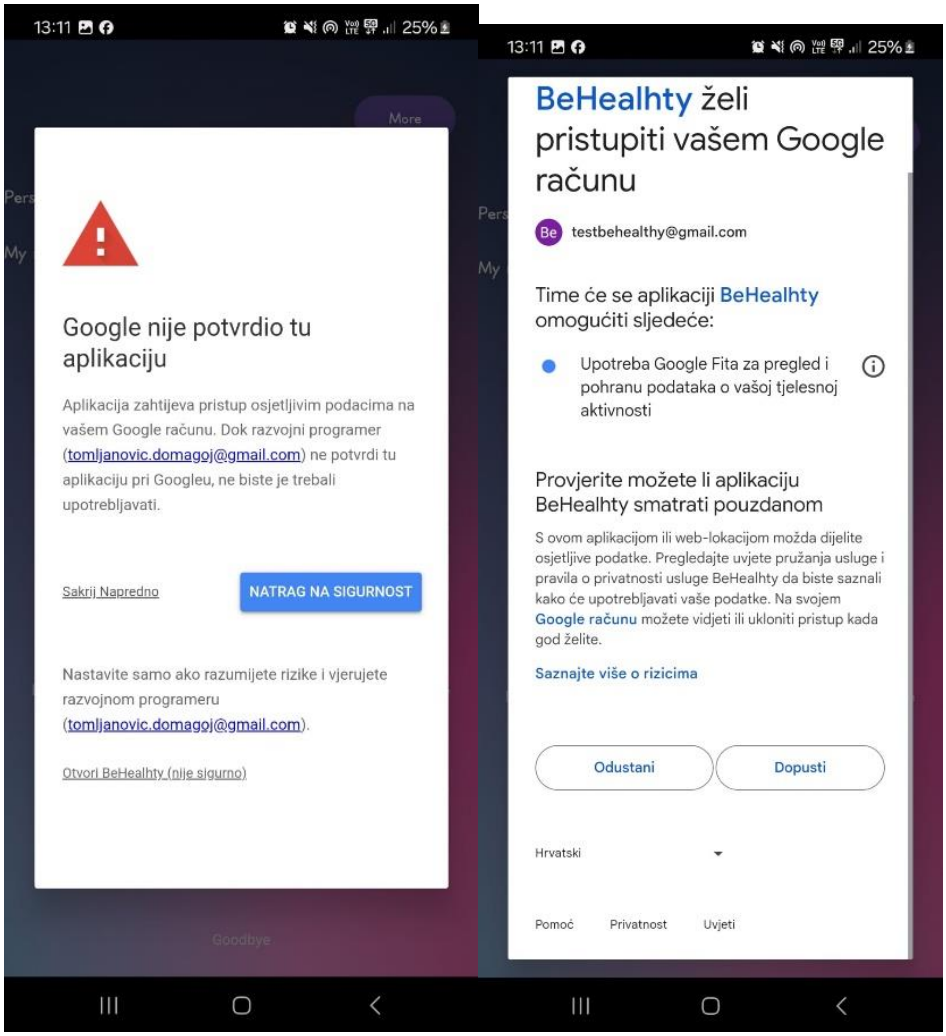


Izvor: Autor

Kao što je već navedeno, nije dovoljno da korisnik samo odabere svoj Google račun; on također mora dati odobrenje kako bi aplikacija mogla pristupiti podacima koje Google posjeduje. Bez ovog odobrenja nije moguće prikazivati broj koraka korisnika. Koraci se bilježe

putem Google aplikacije, a mobilna aplikacija zapravo preuzima te podatke i prikazuje ih unutar svog sučelja.

Slika 68 Google potvrda



Izvor: Autor

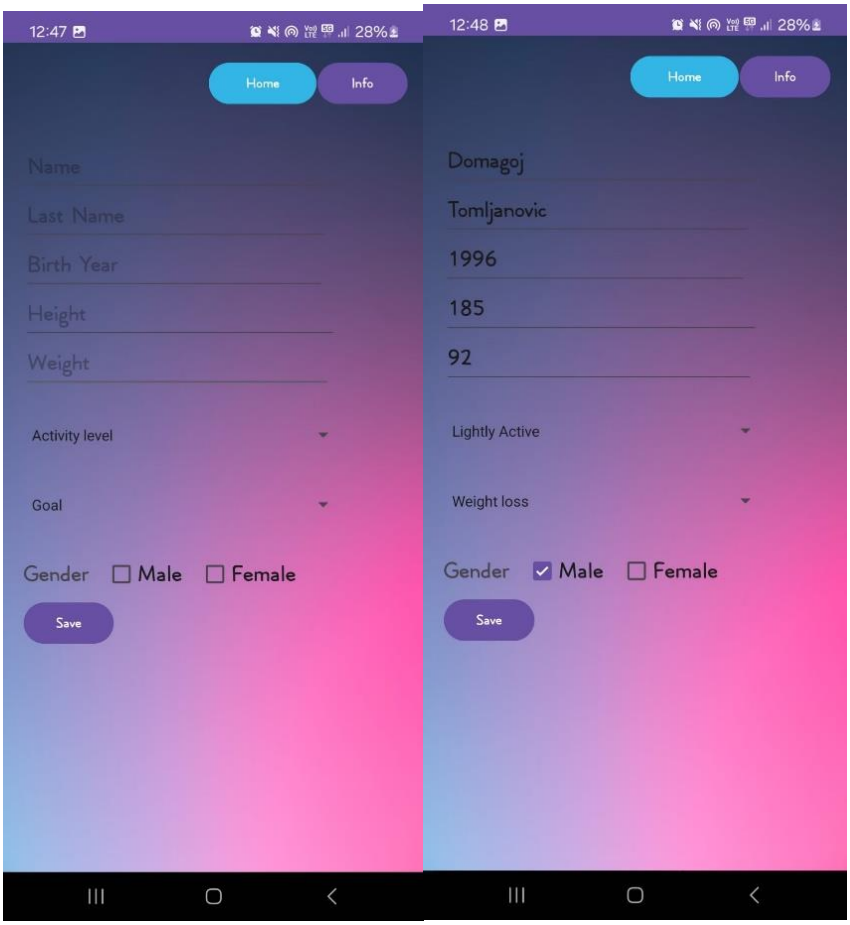
Korisnik je dao potrebna odobrenja i sada može koristiti sve funkcionalnosti aplikacije. Na sljedećoj slici prikazano je glavno sučelje aplikacije. Možemo primijetiti broj koraka u sredini ekrana, koji se ažurira u stvarnom vremenu.

Slika 69 Sučelje aplikacije nakon pristupa postojećeg korisnika

Izvor: Autor

Prvi korak u korištenju aplikacije je unos osobnih podataka. Prva slika prikazuje prazna polja i ono što korisnik vidi kada pristupi dijelu aplikacije "Personal Data". Druga slika prikazuje ispunjena polja nakon što je korisnik unio svoje podatke. Nakon spremanja, ovi podaci se šalju u bazu podataka i povezuju s korisnikom koji je trenutno prijavljen u aplikaciju. Svaki korisnik nakon prijave u aplikaciju dobije jedinstveni UID u Firebase bazi podataka, što omogućava aplikaciji da zna kojem korisniku pripadaju uneseni ili ažurirani podaci.

Slika 70 Unos osobnih podataka

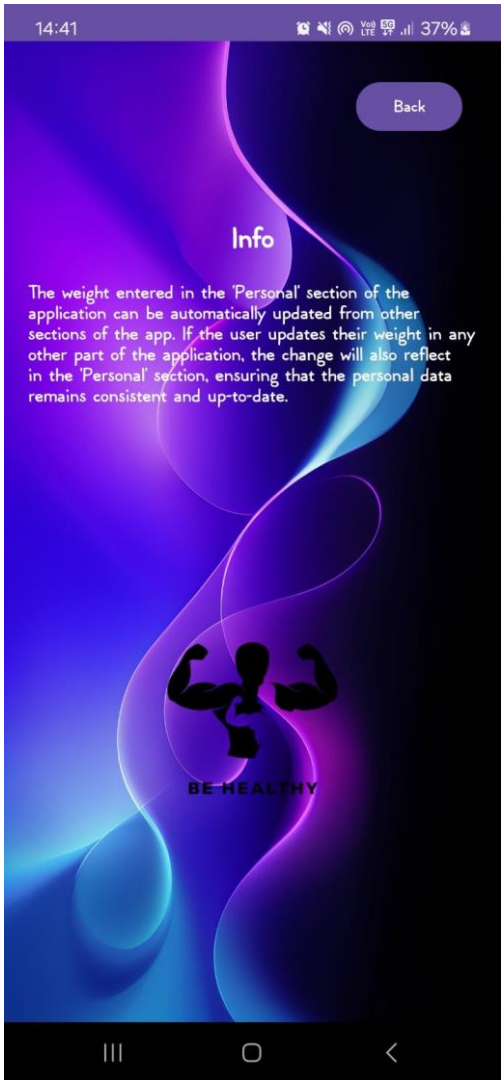


Izvor: Autor

Kako bismo osigurali da korisnik uvijek ima ispravne i ažurirane podatke, aplikacija automatski ažurira osobne podatke bez obzira na to u kojem dijelu aplikacije je unesena nova kilaža korisnika. Ova funkcionalnost omogućava korisnicima da unesu svoju težinu u bilo kojem dijelu aplikacije, a da se ti podaci automatski preslikaju u sekciji "Personal Data".

Korisnik je o ovoj značajki informiran putem info prozora unutar "Personal Data" dijela aplikacije, čime se osigurava transparentnost i jasnoća u vezi s načinom ažuriranja podataka.

Slika 71 Info prikaz

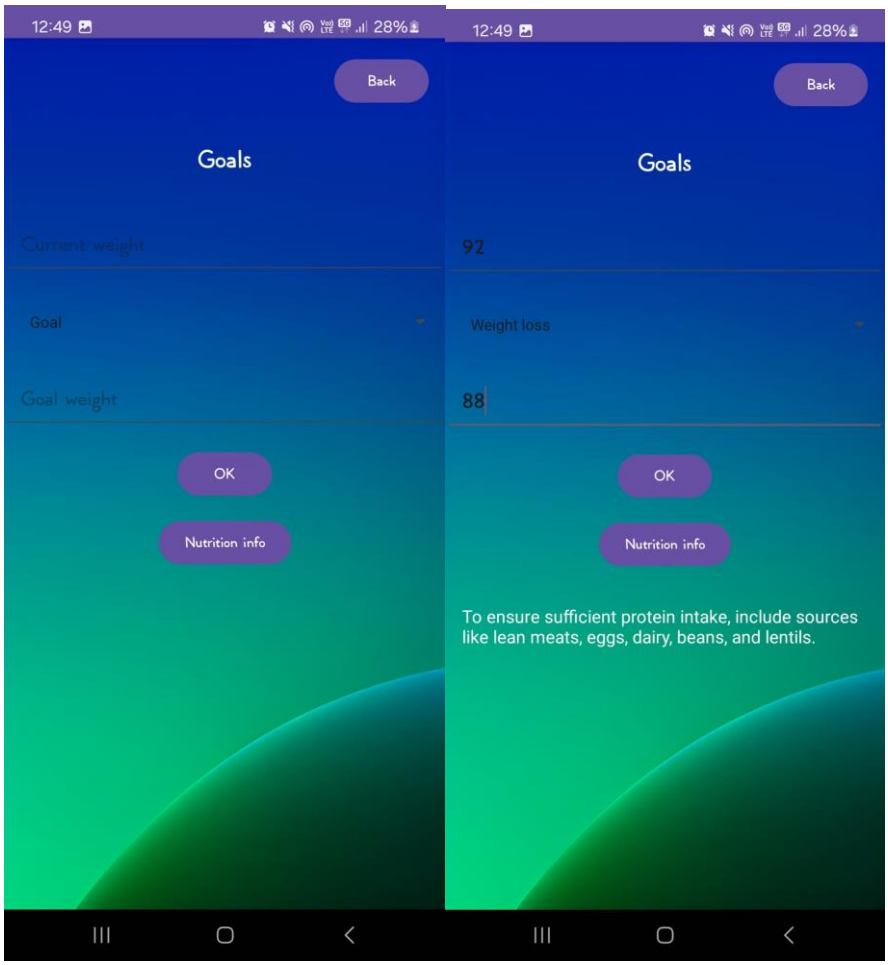


Izvor: Autor

Kao što je prethodno navedeno, kilaža se može ažurirati iz različitih dijelova aplikacije, a jedan od tih dijelova je sekcija "Goals". Odmah nakon što korisnik spremi svoje osobne podatke, aplikacija ga automatski preusmjerava na novo sučelje u kojem potvrđuje svoju trenutnu težinu, unosi svoj cilj te definira svoju ciljanu težinu. Ova funkcionalnost osigurava

da korisnik odmah nakon registracije ili ažuriranja osobnih podataka može postaviti svoje fitness ciljeve i pratiti svoj napredak prema tim ciljevima. Uz navedeno, korisnik ima gumb „Nutrition info“ preko kojih može generirati korisne savjete koje će mu olakšati ostvariti cilj i steći dobre navike.

Slika 72 Postavljanje ciljeva

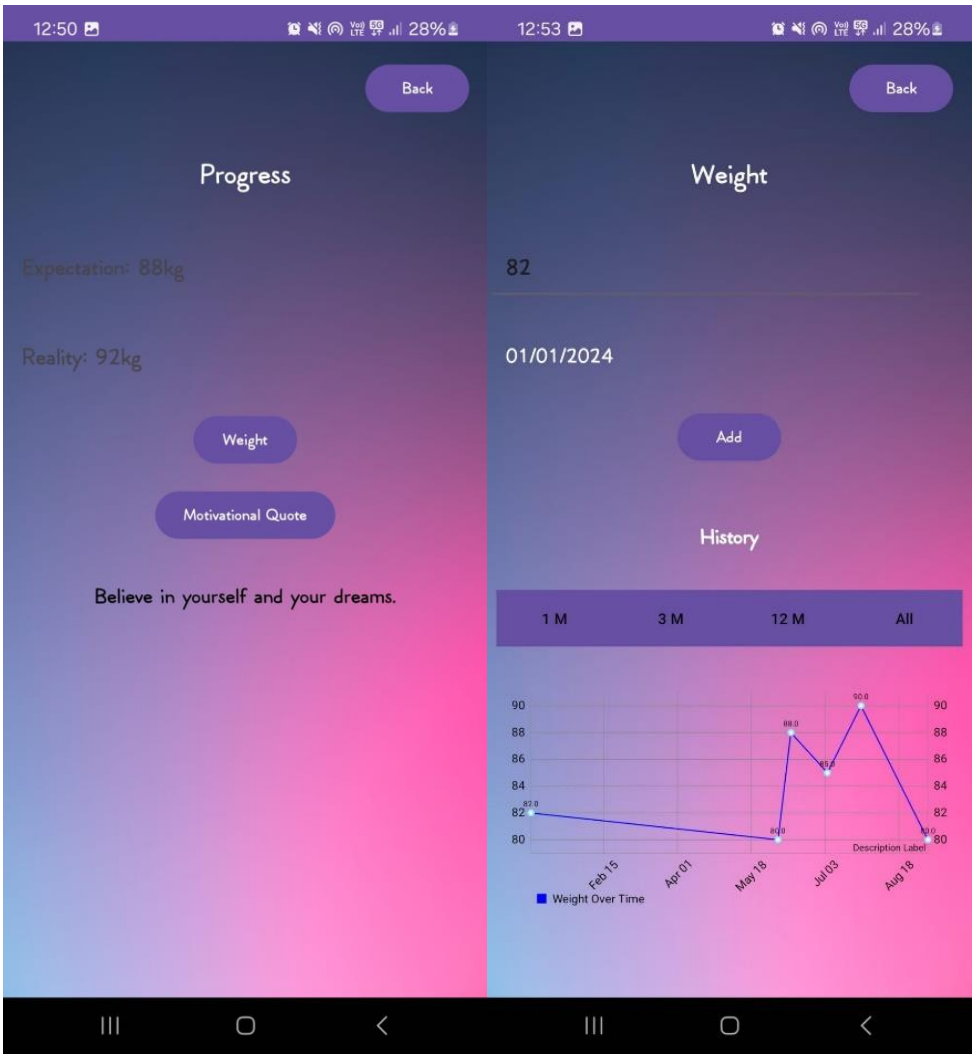


Izvor: Autor

Kako bi korisnici imali detaljan uvid u svoje promjene težine, aplikacija nudi funkcionalnosti "Progress" i "Weight". Putem funkcionalnosti "Progress", korisnici mogu pratiti svoju trenutnu težinu u odnosu na cilj koji su postavili. Uz to, imaju dvije opcije: generirati motivacijske citate unutar aplikacije ili, ukoliko dođe do promjene u težini, pritiskom

na gumb "Weight" otvoriti novo sučelje za unos promjene. Kada korisnici otvore funkcionalnost "Weight", prikazuje im se sučelje u kojem mogu unijeti novu težinu i datum promjene. Datum unosa omogućuje praćenje promjena težine kroz vrijeme putem grafa koji je dostupan unutar aplikacije. Graf pruža mogućnost pregleda povijesnih podataka ovisno o odabranom periodu (jedan mjesec, tri mjeseca, dvanaest mjeseci i svi unosi). Ovisno o odabranom razdoblju, graf se prilagođava i prikazuje relevantne podatke samo za taj period. Ove funkcionalnosti omogućuju korisnicima da na jednostavan i pregledan način prate svoj napredak i ostanu motivirani na putu postizanja svojih ciljeva.

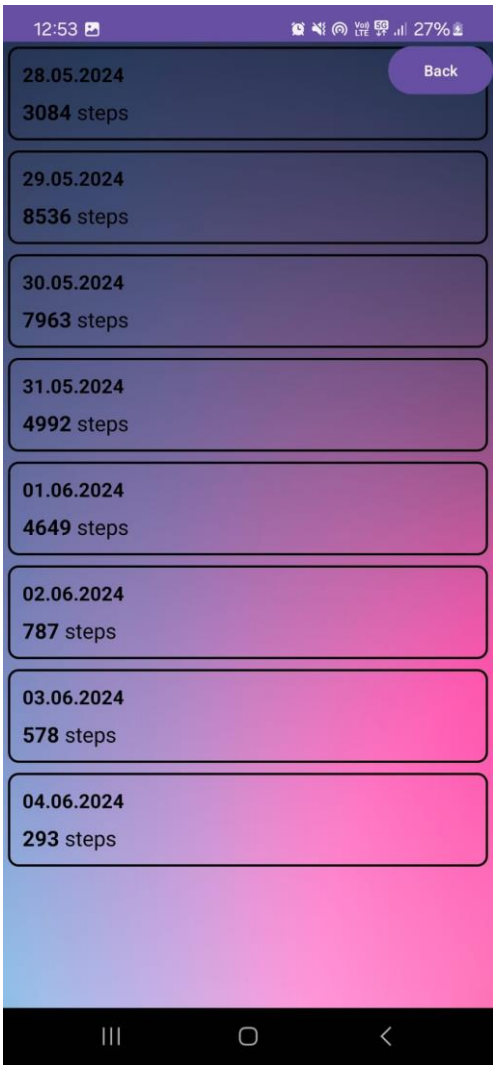
Slika 73 Praćenje napretka i unos nove kilaže



Izvor: Autor

Aplikacija sadrži povijesni prikaz broja koraka, omogućujući korisnicima praćenje svoje tjelesne aktivnosti kroz vrijeme. Ovaj pregled omogućuje uvid u trendove, pomažući korisnicima da bolje razumiju i poboljšaju svoje navike kretanja.

Slika 74 Povijesni prikaz broja koraka

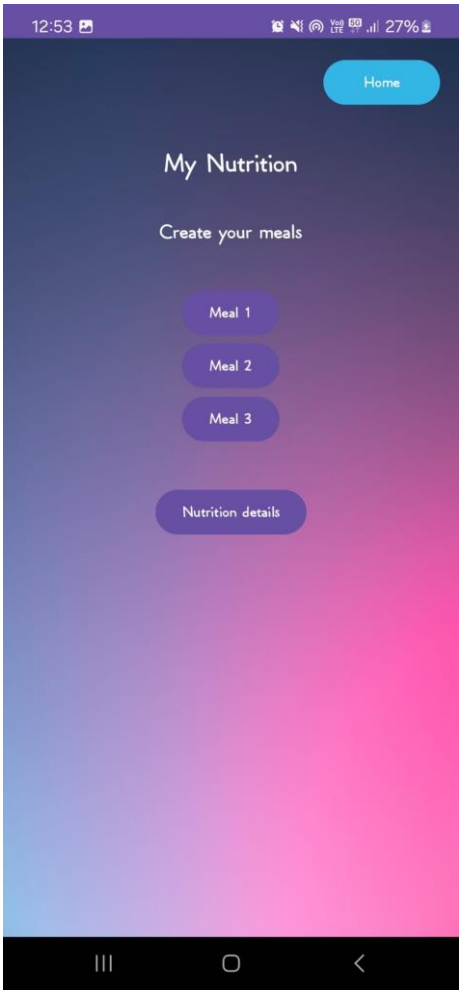


Izvor: Autor

U nastavku će biti prikazane ključne funkcionalnosti aplikacije za praćenje kalorijskog unosa kroz nekoliko slika. Aplikacija omogućuje unos kalorija kroz tri različita obroka.

Pritiskom na gumb „Nutrition details“, korisnici mogu pregledati ukupno unesene kalorije i makro-nutrijente. Dodatno, unutar opcije za dodavanje obroka, korisnici mogu skenirati barkod putem kamere kako bi automatski unijeli nutritivne vrijednosti za određeni obrok.

Slika 75 Odabir obroka za unos

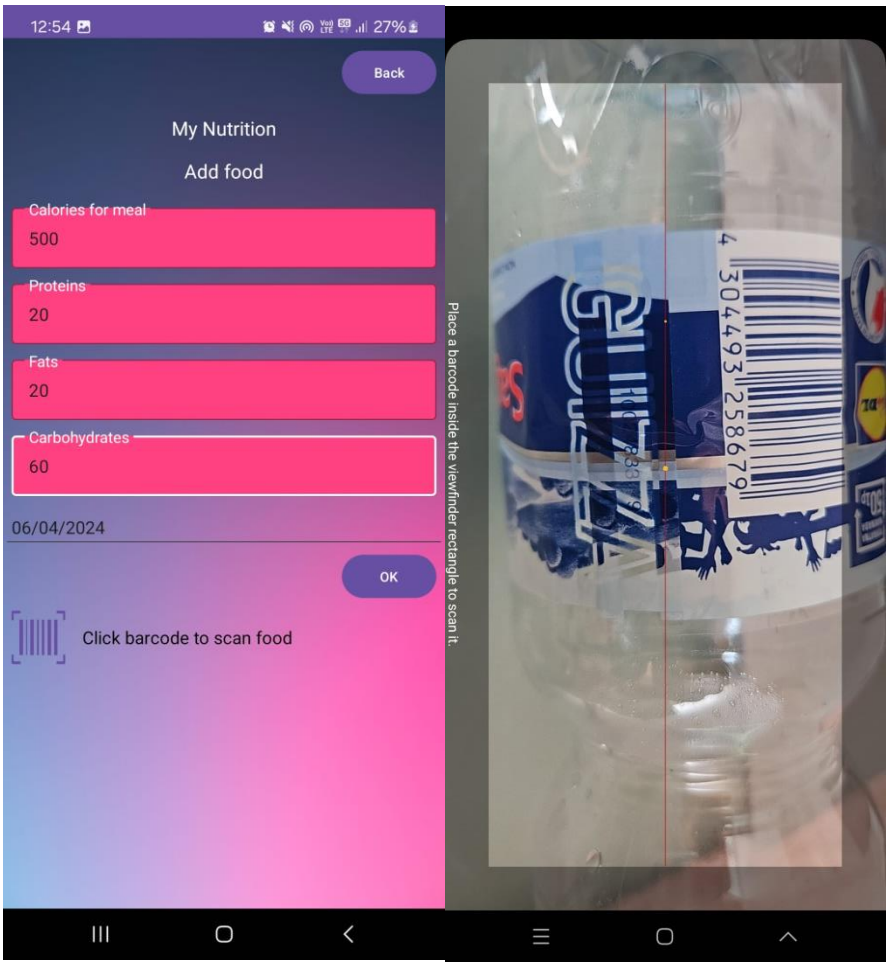


Izvor: Autor

Pritiskom na gumb „AddFood“, korisniku se otvaraju polja za unos kalorijskih vrijednosti i makro-nutrijenata. Korisnik može unijeti vrijednosti u svako od polja, a nakon spremanja podataka, ti podaci se pohranjuju u bazu podataka. Za taj obrok i taj datum stvara se

unos u bazi. Kada korisnik ponovno pristupi istom obroku (za isti datum), prikazuju mu se prethodno unesene vrijednosti koje se povlače iz baze podataka. Ako korisnik unese nove vrijednosti, postojeći unos će biti ažuriran tim novim podacima. Kako bi korisnici mogli koristiti barkodove za unos nutritivnih vrijednosti, kreiran je poseban gumb za pristup ovoj opciji. Nakon pritiska na gumb za skeniranje barkoda, otvara se kamera i započinje proces prepoznavanja barkoda, kao što je prikazano na slici ispod.

Slika 76 Unos kalorija, makro-nutrijenata i skeniranje barkoda

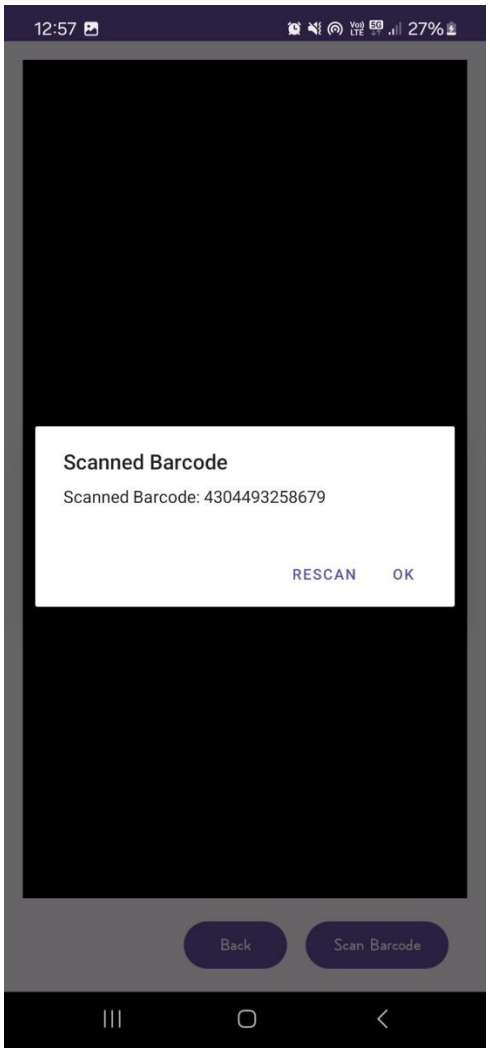


Izvor: Autor

Ako je skeniranje uspješno obavljeno, korisniku se prikazuje prozor s prikazom skeniranog barkoda. Korisnik može odabrati novo skeniranje ili potvrditi skenirani barkod. Ako potvrdi, bit će preusmjeren na prozor „AddFood“ gdje će unijeti vrijednosti za taj barkod, a

zatim će se te informacije pohraniti u bazu podataka. Svaki put kada korisnik skenira isti barkod, bez obzira na datum, aplikacija će povući točno one vrijednosti koje su prethodno definirane za taj barkod. Ako korisnik ažurira te vrijednosti u aplikaciji, one će se automatski ažurirati i u bazi podataka.

Slika 77 Skenirani barkod

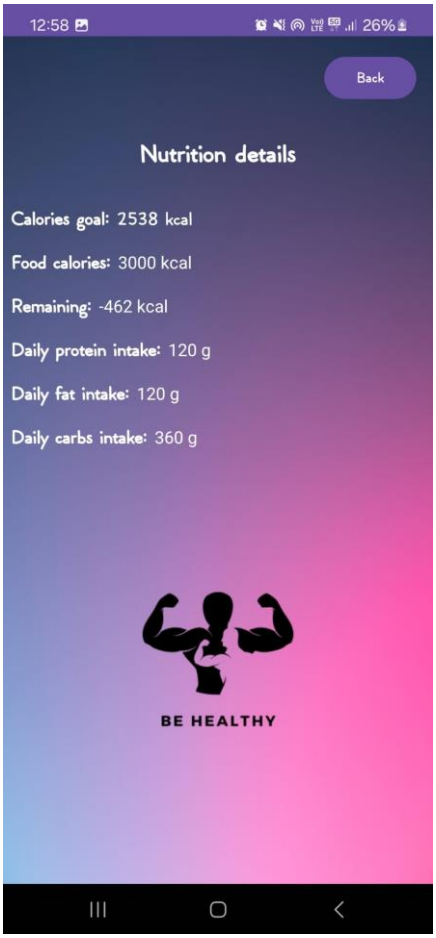


Izvor: Autor

Finalne informacije unesene u aplikaciju prikazuju se u detaljima prehrane. Korisnik može vidjeti svoj cilj, koji je izračunat na temelju njegove visine, težine, razine aktivnosti i zadanog cilja. Uz to, korisnik može vidjeti ukupno unesene kalorije, makronutrijente i preostali

broj kalorija. Ako preostali broj kalorija ima negativan predznak, to znači da je korisnik unio više kalorija nego što je predviđeno.

Slika 78 Nutritivni detalji



Izvor: Autor

7 Zaključak

U zaključku, možemo reći da razvijena aplikacija, koja služi za praćenje dnevnog unosa kalorija i kalorijske potrošnje, predstavlja korisno i funkcionalno rješenje za one koji žele poboljšati svoje zdravlje i tjelesnu kondiciju. Međutim, unatoč svojim korisnim značajkama, ova aplikacija također ostavlja prostora za daljnje poboljšanje i razvoj.

Aplikacija ima potencijalnu primjenu u različitim fitness i prehrambenim režimima te može koristiti različitim skupinama korisnika:

- Praćenje kalorijskog unosa: Korisnicima omogućuje jednostavno praćenje unosa kalorija putem unosa obroka, pomažući im u postizanju svojih ciljeva u vezi s tjelesnom težinom
- Planiranje prehrane: Korisnici mogu koristiti aplikaciju za planiranje obroka koji odgovaraju njihovim ciljevima, bilo da žele izgubiti kilograme, dobiti na težini ili održavati postojeću težinu
- Kalorijska potrošnja: Prati kalorijsku potrošnju, pomažući korisnicima bolje razumjeti kako njihova tjelesna aktivnost utječe na njihove ciljeve u vezi s prehranom
- Ciljevi u vezi s prehranom: Aplikacija može poslužiti kao alat za postavljanje i praćenje ciljeva vezanih uz prehranu i fitness, što pomaže korisnicima da ostanu motivirani
- Praćenje broja koraka: Korisnici mogu pratiti broj koraka, što dodatno pomaže u razumijevanju njihove tjelesne aktivnosti
- Skeniranje hrane putem barkoda: Omogućuje jednostavno unošenje nutritivnih vrijednosti skeniranjem barkoda hrane
- Vizualna privlačnost i jednostavnost: Aplikacija je dizajnirana da bude vizualno privlačna i jednostavna za korištenje, čime se potiče redovita uporaba

Daljnji razvoj ovakve aplikacije bio bi koristan iz nekoliko razloga:

- Povećanje funkcionalnosti: Aplikacija može biti proširena dodatnim značajkama kao što su praćenje mikro-nutrijenata, generiranje personaliziranih planova prehrane i vježbanja te integracija s dodatnim uređajima (pametni satovi, vage itd.)
- Unapređenje korisničkog iskustva: Kontinuirano poboljšanje korisničkog sučelja i iskustva čini aplikaciju privlačnijom i jednostavnijom za upotrebu, što potiče korisnike da je redovito koriste
- Analiza podataka: Analiza podataka koje korisnici unose može pomoći u pružanju personaliziranih preporuka i informacija koje su usklađene s ciljevima korisnika.
- Edukacija i praćenje napretka: Aplikacija može pružiti dodatne edukativne materijale i alate koji pomažu korisnicima da bolje razumiju svoje potrebe u vezi s prehranom i fitnessom te da prate svoj napredak na dugoročnoj osnovi

Što se tiče osobnog doprinosa u kreiranju aplikacije, važno je naglasiti da je ulaganje vlastitog truda, vremena i znanja u razvoj aplikacije značajno. Moja predanost i rad na kreiranju aplikacije su ključni za njezinu funkcionalnost i korisnost. S kontinuiranim učenjem i nadogradnjom aplikacije sigurno mogu doprinijeti daljnjem usavršavanju aplikacije i omogućiti joj da bolje služi korisnicima.

8 Popis literature

1. Diagrams.net (2024), Create Diagrams and Flowcharts. https://app.diagrams.net/ (25.07.2024.)
2. Firebase (2024), Cloud Firestore Documentation. https://firebase.google.com/docs/firestore (22.08.2024.)
3. Google Developers (2024), Google Fit API Documentation. https://developers.google.com/fit/ (04.08.2024.)
4. Nordeen, A. (2020), Learn UML in 24 Hours. BPB Publications. (04.08.2024.)
5. Oracle (2023), Java Documentation. Oracle. https://www.oracle.com/java/ (25.07.2024.)
6. Sosa, H. (2023), Understanding How to Access Google APIs. https://dev.to/iamhectorsosa/understanding-how-to-access-google-apis-1593 (25.07.2024.)
7. W3Schools (2024), XML Tutorial. https://www.w3schools.com/xml/default.asp (25.07.2024.)

9 Popis slika