

SVEUČILIŠTE U ZAGREBU  
FAKULTET ORGANIZACIJE I INFORMATIKE  
VARAŽDIN

Daniel Škrlac

# MOBILNA APLIKACIJA ZA PLANIRANJE I PRAĆENJE TJELESNE TEŽINE

PROJEKT

## TEORIJA BAZA PODATAKA

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Daniel Škrlac

Matični broj: 0016149236

Studij: Informacijsko i programsко inženjerstvo

## MOBILNA APLIKACIJA ZA PLANIRANJE I PRAĆENJE TJELESNE TEŽINE

PROJEKT

Mentor:

dr. sc. Bogdan Okreša Đurić

Varaždin, siječanj 2024.

*Daniel Škrlac*

**Izjava o izvornosti**

Izjavljujem da je ovaj projekt izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

*Autor potvrdio prihvaćanjem odredbi u sustavu FOI Radovi*

---

## Sažetak

U ovom radu istražuje se implementacija aplikacije *KeepFit*, koja je usmjeren na unaprjeđenje zdravstvenih navika korisnika. Ključni elementi ove aplikacije uključuju integraciju temporalnih i aktivnih baza podataka, što omogućuje efikasno upravljanje i analizu zdravstvenih podataka kroz vremenski element u samoj shemi baze podataka. Implementacija baze podataka ostvarena je korištenjem PostgreSQL-a, koji nudi podršku za JSON tip podataka, omogućujući fleksibilnost u pohrani i obradi složenih struktura podataka. Razvoj API-ja za aplikaciju *KeepFit* proveden je korištenjem Node.js okruženja dok je na strani klijenta, za izgradnju mobilne aplikacije korišten Kotlin uz Android operacijski sustav.

**Ključne riječi:** API, Node.js, Android, Kotlin, PostgreSQL

# Sadržaj

<b>1. Uvod</b>	1
<b>2. Opis aplikacijske domene</b>	2
<b>3. Teorijski uvod</b>	4
3.1. Temporalne baze podataka	4
3.2. Aktivne baze podataka	5
3.3. JSON format u relacijskom modelu podataka	6
<b>4. Model baze podataka</b>	8
<b>5. Implementacija</b>	10
5.1. Prikaz dijagrama slučajeva korištenja	10
5.2. Implementacija API-ja	11
5.3. Mobilna aplikacija	15
5.4. Baza podataka	16
<b>6. Primjeri korištenja</b>	19
<b>7. Zaključak</b>	25
<b>Popis literature</b>	26
<b>Popis slika</b>	27
<b>Popis tablica</b>	28
<b>Popis isječaka koda</b>	29
<b>1. Prilog</b>	31

# 1. Uvod

Aplikacija *KeepFit* predstavlja rješenje namijenjeno korisnicima koji teže zdravom načinu života i žele aktivno pratiti svoje zdravstvene ciljeve. Ova aplikacija je razvijena s ciljem pružanja jednostavnog načina pristupa praćenju i upravljanju zdravstvenim navikama - prehranom i fizičkom aktivnošću. Aplikacija obuhvaća integraciju temporalnih i aktivnih baza podataka uz korištenje integriranog tipa podataka *JSON* u okruženju *PostgreSQL-a*.

## 2. Opis aplikacijske domene

Aplikacija *KeepFit* omogućuje olakšani pristup zdravom načinu života i praćenju individualnih ciljeva korisnika. Aplikacija omogućuje korisnicima da prate svoje zdravlje, postavljaju i prate svoje ciljeve, bilježe prehrambene navike, vježbanje te prate svoj napredak kroz vježbanje i prehranu.

Glavni segment aplikacije vezan je uz definiranje ciljeva od strane korisnika, korisnici mogu definirati tri različite vrste ciljeva - ciljevi vezani uz gubitak odnosno dobitak tjelesne težine, ciljevi vezani uz nutricionizam odnosno to su ciljevi koji definiraju optimalnu konzumaciju hrane koja upotpunjava unos potrebnih hranjivih sastojaka te sami ciljevi vezani uz redovno vježbanje ili provođenje različitih aktivnosti.

Sustav ne samo što omogućuje postavljanje i praćenje dugoročnih ciljeva već korisnicima pruža i priliku za svakodnevno detaljno praćenje zdravstvenih indikatora. Na dnevnoj bazi, korisnici mogu unijeti i ažurirati informacije poput razine šećera u krvi, trenutnog krvnog tlaka, težine, broja sati sna ili otkucaja srca. Ova mogućnost dnevnog praćenja omogućuje korisnicima da budu u konstantnom doticaju sa svojim zdravljem. Sustav pruža trenutne informacije o njihovom zdravstvenom statusu na dnevnoj razini, čime će im pomoći u prilagodbi navika ili aktivnosti kako bi bolje uskladili svoje ponašanje s postavljenim zdravstvenim ciljevima. Detaljno praćenje također pruža osnovu za donošenje važnih odluka o vlastitom zdravstvenom stanju i potrebama za potencijalnim poboljšanjem životnih navika donošenjem ispravnih odluka na temelju analize podataka.

Korisnici mogu detaljno pratiti unesenu hranu za svaki pojedinačni dan, uz navođenje sastojaka, kalorijske vrijednosti i druge nutritivne informacije. Paralelno s tim, korisnici imaju pristup svim svojim provedenim aktivnostima, što im omogućuje potpuni uvid u svoj dnevni režim prehrane i vježbanja. Svakodnevno, korisnici imaju mogućnost unosa osobnih bilješki ili podsjetnika koji su povezani s njihovim dnevnim rutinama. Mogu zabilježiti važne detalje, osjećaje ili podsjetnike o prehrani, vježbama ili zdravstvenom stanju. Ove bilješke pružaju korisnicima mogućnost da dodaju kontekst ili specifičnosti svojim dnevnim aktivnostima, pomažući im da bolje razumiju svoje navike i motivaciju.

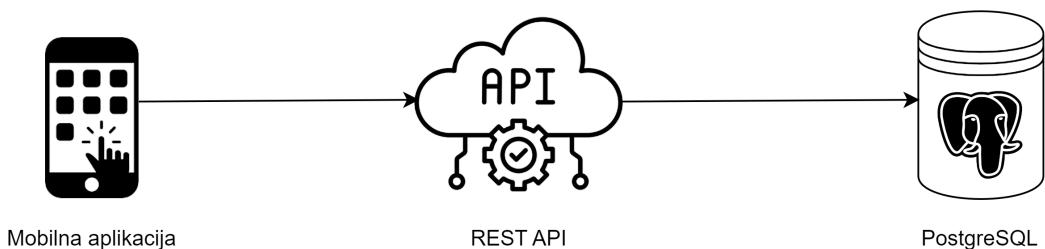
Sposobnost detaljnog praćenja prehrane, vježbanja i zdravstvenih pokazatelja te mogućnost bilježenja osobnih podsjetnika čine ovu aplikaciju sveobuhvatnim pristupom u svakodnevnom održavanju zdravih navika. *KeepFit* potiče korisnike na kontinuirano unaprjeđivanje njihovih navika i životnog stila.

U samim zahtjevima aplikacije, postoji raznolikost vrsta entiteta poput ciljeva, prehrambenih unosa, zdravstvenih pokazatelja i dnevnih aktivnosti, s različitim atributima i varijacijama. Tradicionalna relacijska baza podataka s njenom strogom strukturom zapisa može postati ograničavajuća za ovu vrstu aplikacije. Specifičnost ove aplikacije leži u potrebi za prilagodbom i fleksibilnošću u pohrani podataka. Umjesto klasičnih tablica s fiksnim stupcima, *KeepFit* koristi koncepte NoSQL baza podataka u obliku *JSON* tipa podataka.

*KeepFit* koristi PostgreSQL bazu podataka, pružajući mogućnost rada s integriranim

*JSON* tipom podataka što je ključno za aplikaciju koja zahtijeva fleksibilnost u pohrani i upravljanju raznolikim podacima koji variraju po vrstama i atributima. *PostgresSQL* baza podataka je integrirana s *Node.js* izvršnim *JavaScript* okruženjem zbog jednostavnosti implementacije asinkronog-neblokirajućeg programskog kôda. Za olakšanu izradu API-ja, odabran je *Express.js* okvir koji je jednostavan za korištenje i pruža čitav niz alata i funkcija za brzu izradu API-ja.

Mobilna aplikacija je razvijena za Android operacijski sustav, preciznije aplikacija je podržana od Android 27 API-ja odnosno Android 8.1 verzije. Za razvoj mobilne aplikacije korišten je programski jezik *Kotlin* iz razloga kako pruža jednostavno sučelje za implementaciju asinkronog programiranja u obliku koncepta korutina. Za implementaciju sučelja mobilne aplikacije korišten je *Jetpack Compose* alat koji nam omogućuje reaktivno programiranje samog sučelja mobilne aplikacije. Prilikom same implementacije korišten je *MVVM* uzorak dizajna.



Slika 1: Arhitektura *KeepFit* aplikacije [autorski rad]

### 3. Teorijski uvod

Aplikacija *KeepFit* koristi koncepte temporalnih baza podataka, omogućavajući praćenje vremenskih podataka poput povijesti zdravstvenih pokazatelja i dnevnika aktivnosti. Također, implementira i koncepte aktivnih baza podataka kroz uporabu okidača. Osim toga, aplikacija uključuje i koncepte *NoSQL* baza podataka, koristeći polustrukturirane podatke unutar relacijskog modela podataka.

#### 3.1. Temporalne baze podataka

Glavni koncept temporalnih baza podataka je temporalni podatak. Razlika između temporalnih i netemporalnih podataka je u tome što se temporalni podaci zasnivaju na određenom vremenskom periodu koji izražava kada su bili važeći ili pohranjeni u bazi podataka. Podaci koje pohranjuju konvencionalne baze podataka smatraju se važećima u trenutnom vremenu. Kada se podaci u takvoj bazi podataka mijenjaju, brišu ili umetnu, stanje baze podataka se prepisuje kako bi se formiralo novo stanje te prethodno stanje baze podataka, prije bilo kakvih promjena, više nije dostupno. Povezivanjem vremena s podacima, moguće je pohraniti različita stanja baze podataka. [1]

Vrijeme se može tumačiti kao vrijeme valjanosti (kada se je određeni podatak *stvorio* ili je *postao* istiniti u stvarnosti, engl. *valid time*) ili vrijeme transakcije (kada su podaci uneseni u bazu podataka, engl. *transaction time*). Također, postoji i korisnički definiran (engl. *user defined*) temporalni podatak kojega ne možemo svrstati u niti jednu od već navedenih kategorija. Na primjer atribut *datum\_rođenja* ne predstavlja vrijednost kada je određeni podatak postao istinit u bazi podataka i ne predstavlja trenutak kada je podatak zapisan u bazi podataka. Razne vrste temporalnih podataka omogućuju definiranje različitih vrsta implementacija temporalnih baza podataka. [1]:

- **Povjesna (engl. *historical*) baza podataka** - Pohranjuje podatke u kontekstu valjanog vremena.
- **Baza podataka za povrat unazad (engl. *rollback*)** - Pohranjuje podatke u kontekstu vremena transakcije.
- **Dvovremenska (engl. *bitemporal*) baza podataka** - Kombinira oba pristupa, omogućujući kontekst vremena valjanosti i vremena transakcije.

Najčešći primjeri temporalnih baza podataka su [1]:

- **Zdravstveni sustavi:** Lječnici trebaju povijest zdravlja pacijenata za pravilnu dijagnozu. Najčešće se pamte informacije porasta određenih identifikatora zdravlja pacijenata
- **Sustavi osiguranja:** Informacije o zahtjevima za osiguranje, povijesti nesreća.

- **Sustavi rezervacija:** U sustavima rezervacija, kao što su hoteli, zrakoplovne kompanije potrebne su informacije o datumima i vremenima kada su rezervacije važeće.
- **Znanstvene baze podataka:** U znanstvenim bazama podataka, podaci prikupljeni iz eksperimenata uključuju vrijeme kada je svaki podatak izmijeren.
- **Podatkovna baza fakulteta:** U podatkovnoj bazi fakulteta, vrijeme se može uključiti u semestar i godinu, povijest ocjena studenata itd.

Prednosti upotrebe temporalnih baza podataka [1], [2]:

- **Analiza povijesti podataka:** Omogućava analizu podataka kroz povijest u svrhu donošenja odluka temeljenih na podacima.
- **Korisničko iskustvo:** Pruža mogućnost povećanja korisničkog iskustva kako korisnik može raditi sa trenutnim, povijesnim ili čak podacima u raznim vremenskim intervalima.
- **Verzioniranje podataka:** Omogućava usporedbu različitih verzija podataka.

Nedostatci upotrebe temporalnih baza podataka [1], [2]:

- **Složenost implementacije:** Kompleksnija implementacija.
- **Povećana potreba za pohranom:** Čuvanje višestrukih verzija podataka može značajno povećati potrebe za većom pohranom.
- **Složenost upita:** Može dovesti do složenijih upita.
- **Performanse:** Složeniji upiti i veći volumen podataka mogu utjecati na performanse aplikacije.

## 3.2. Aktivne baze podataka

Aktivne baze podataka podržavaju mehanizme koji im omogućuju automatski odgovor na događaje koji se odvijaju unutar ili izvan samog sustava baze podataka. Aktivne baze podataka koriste pravila bazirana na događajima, uvjetima i akcijama (ECA-pravila) za upravljanje reakcijama na različite situacije. Najčešće aktivne baze podataka omogućuju definiranje reaktivnih koncepata unutar sustava baze podataka no neke implementacije bazi podataka (npr. MongoDB) omogućuju reaktivno programiranje i na razini programskoga kôda [3]:

- **Događaj (engl. event):**
  - Predstavlja određeni događaj koji se dogodio unutar baze podataka.
  - Primjeri uključuju ažuriranje podataka, pristup određenom podatku itd.

- **Uvjet (engl. condition):**

- Nakon događaja, sustav provjerava određeni uvjet.
- Na primjer, u slučaju ažuriranja zapisa o cijeni dionice, uvjet može biti provjera da li je nova cijena viša od zadane granice.

- **Akcija (engl. action):**

- Ako je uvjet zadovoljen, izvršava se određena akcija.
- U primjeru s dionicama, to može biti automatsko slanje obavijesti investitorima.

Prednosti korištenja aktivnih baza podataka su [3]:

- **Automatsko ažuriranje podataka:** Kada se određeni podaci promijene, aktivna baza podataka automatski ažurira povezane podatke ili izvršava određene akcije.
- **Osiguranje usklađenosti podataka:** Ako se određeni uvjeti promijene, baza podataka može automatski primijeniti pravila kako bi osigurala da podaci ostanu usklađeni.
- **Nadgledavanje:** Baze podataka mogu nadgledati određene uvjete ili aktivnosti i slati obavijesti ili izvješća administratorima.
- **Automatska provedba složenih poslovnih pravila:** Koristi se za automatsko provođenje kompleksnih poslovnih pravila i logike unutar baze podataka.

Nedostatci korištenja aktivnih baza podataka su [3]:

- **Performanse:** Utjecaj na performanse sustava pri čestim događajima.
- **Implementacija:** Potrebno je napredno poznavanje svih koncepata baze podataka za pravilnu implementaciju.
- **Neželjeni sporedni efekti:** Složene interakcije između pravila mogu uzrokovati nepredvidive sporedne efekte.
- **Pronalaženje pogrešaka:** Složenost u otklanjanju implicitno nastalih pogrešaka.

### 3.3. JSON format u relacijskom modelu podataka

JSON (*JavaScript Object Notation*) format je postao popularan u prijenosu i pohrani podataka zbog svoje fleksibilnosti. Korištenje JSON-a u relacijskim bazama podataka omogućava spremanje polustrukturiranih podataka unutar relacijske sheme [4].

Prednosti korištenja JSON formata u relacijskim bazama su [4]:

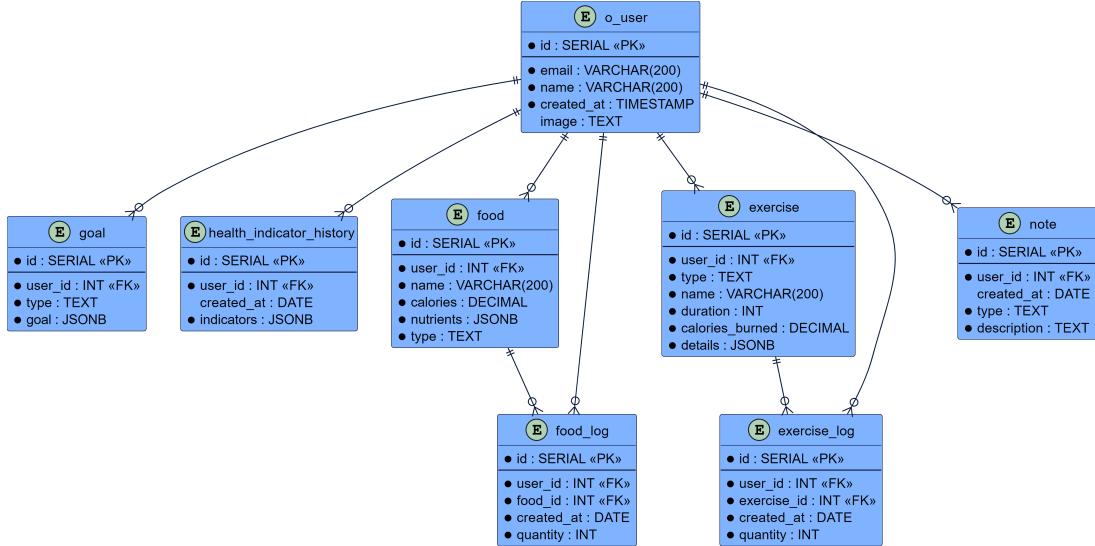
- **Fleksibilnost sheme:** Omogućava lako prilagođavanje sheme podataka, što je korisno u brzom razvoju i dinamičkim aplikacijama kao i u projektima koji su u ranoj fazi razvoja te nemaju već fiksno definiran model podataka.
- **Performanse:** JSON omogućava pohranu raznovrsnih podataka, uključujući hijerarhijske strukture, što je korisno za složene podatke koji nisu lako normalizirani.
- **Jednostavnost:** JSON format je lako čitljiv i jednostavan za korištenje, što olakšava razvoj i održavanje.
- **Interoperabilnost:** JSON se lako može koristiti u raznim programskim jezicima i platformama, što olakšava integraciju i razmjenu podataka.

Nedostatci korištenja JSON formata u relacijskim bazama su [4]:

- **Složenost Upita:** Upiti mogu postati složeniji kada se rade s JSON strukturama unutar relacijskih baza.
- **Performanse:** Obrada i indeksiranje JSON podataka može biti sporije u usporedbi s tradicionalnim relacijskim podacima.
- **Ograničenja u validaciji Podataka:** JSON format ne nudi iste razine validacije i integriteta podataka kao što to omogućuje tradicionalni relacijski model.

## 4. Model baze podataka

Model baze podataka pruža strukturu za bilježenje i povezivanje ključnih informacija o korisniku. Centralni entitet *o\_user* predstavlja osnovu oko koje se povezuju ciljevi, prehrambene navike, vježbe te zdravstveni pokazatelji. Entitet *o\_user* sadrži attribute *id name, email, image, created\_at*.



Slika 2: ERA dijagram *KeepFit* aplikacije [autorski rad]

Jedan korisnik može imati više ciljeva. Korisnik može postaviti različite ciljeve (kao što su zdravlje, fitness ili prehrambeni ciljevi) i svaki od tih ciljeva povezan je s jednim korisnikom. Ova veza omogućuje korisniku da prati svoje ciljeve u različitim područjima. Također, postoji ograničenje da jedan korisnik može kreirati samo tri cilja, iz svake kategorije jedan. Entitet *goal* sadrži attribute *id, user\_id, type* te *goal*. Atribut *goal* predstavlja specifična svojstva cilja ovisno o samome tipu *type*.

Povijest zdravstvenih pokazatelja također je povezana s korisnikom *o\_user*. Korisnik može bilježiti svoje zdravstvene pokazatelje (poput težine, tlaka ili drugih parametara) kroz različita vremenska razdoblja. Ova veza omogućuje korisniku praćenje i analizu njihovog zdravstvenog stanja kroz vrijeme. Entitet *health\_indicator\_history* sadrži attribute *id, user\_id, created\_at* te *indicators*. Atribut *indicators* predstavlja specifična svojstva zdravstvenog pokazatelja ovisno o samome tipu *type*.

Također, bilješke *note* su vezane uz korisnika. Korisnik može stvarati bilješke ili zapise povezane sa vremenski određenim informacijama ili događajima. Entitet bilješke sadrži ograničenje jedinstvenosti s obzirom na attribute *user\_id, created\_at* i *type*, u jednoj vremenskoj jedinici (određeni dan u godini) jedan korisnik može imati maksimalno onoliko bilješki koliko ima različitih tipova *type*.

Hrana i vježbe također su povezane s korisnikom. Korisnik može unositi različite stavke hrane *food* te bilježiti različite vježbe *exercise* koje izvodi. Svaka stavka hrane i vježba povezana je s jednim korisnikom, omogućujući praćenje prehrambenih navika i vježbanja, aktivnosti.

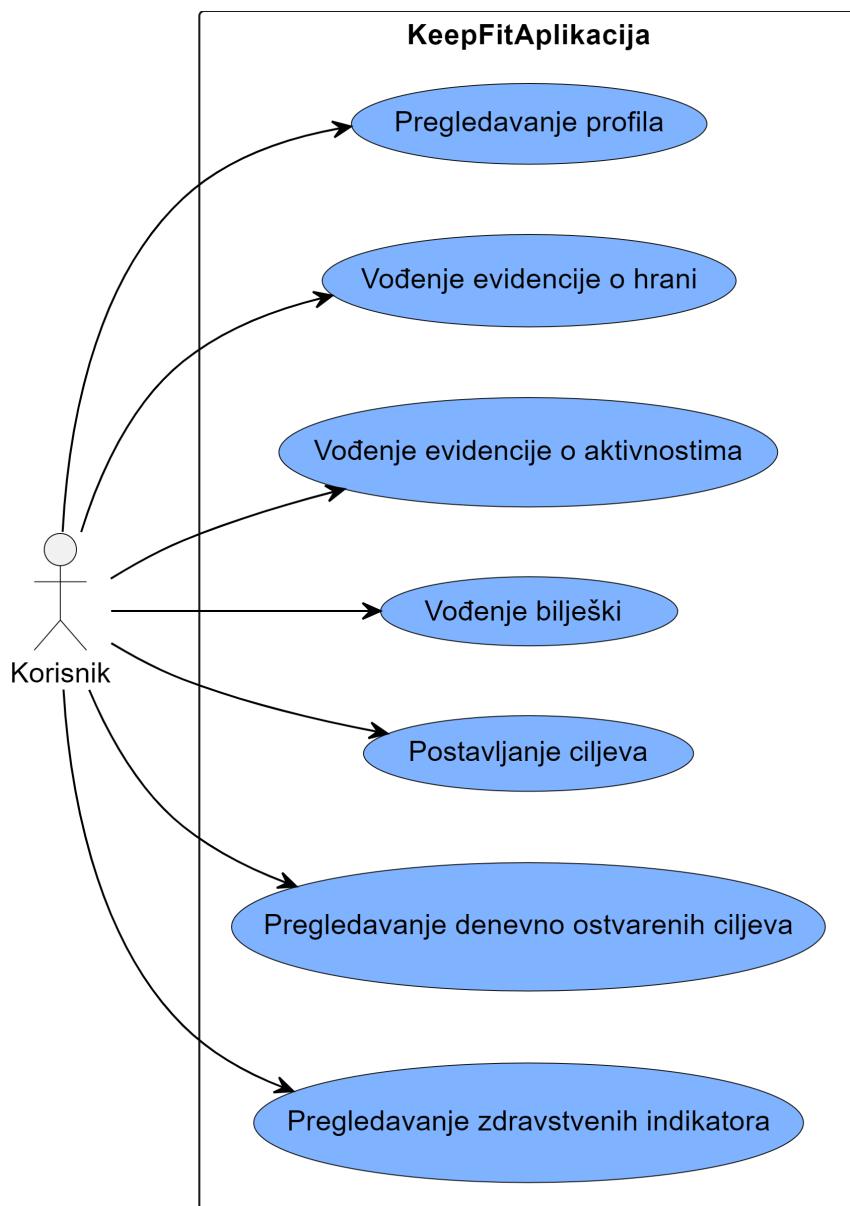
Entitet *food* sadrži atribute *id*, *user\_id*, *name*, te *calories*, *type* te *nutrients*. Atribut *nutrients* predstavlja specifična svojstva hrane, u ovom slučaju svi atributi su jednaki neovisno o tipu *textit{type}*. Entitet *exercise* sadrži atribute *id*, *user\_id*, *name*, te *calories\_burned*, *type* te *details*. Atribut *details* predstavlja specifična svojstva vježbe ovisno o samome tipu *type*. Uz hranu *food* i vježbe *exercise* vežu se i entiteti *food\_log* i *exercise\_log* koji predstavljaju vremenski definiranu relaciju između unesene hrane i same vježbe odnosno aktivnosti. Entitet *food\_log* sadrži atribute *id*, *user\_id*, *food\_id* te *created\_at*, te *quantity*. Entitet *exercise\_log* sadrži atribute *id*, *user\_id*, *exercise\_id*, te *created\_at* te *quantity*.

## 5. Implementacija

U ovom poglavlju detaljno će biti opisana implementacija API-ja i baze podataka za mobilnu aplikaciju *KeepFit*. Razmotrit će se način na koji se API povezuje s mobilnom aplikacijom i kako se podaci pohranjuju, obrađuju. Također, prikazan je osnovni dijagram slučajeva korištenja.

### 5.1. Prikaz dijagrama slučajeva korištenja

U nastavku je prikazan dijagram slučajeva korištenja za aplikaciju *KeepFit*. Dijagram sadrži sljedeće elemente:



Slika 3: Dijagram slučajeva korištenja *KeepFit* aplikacije [autorski rad]

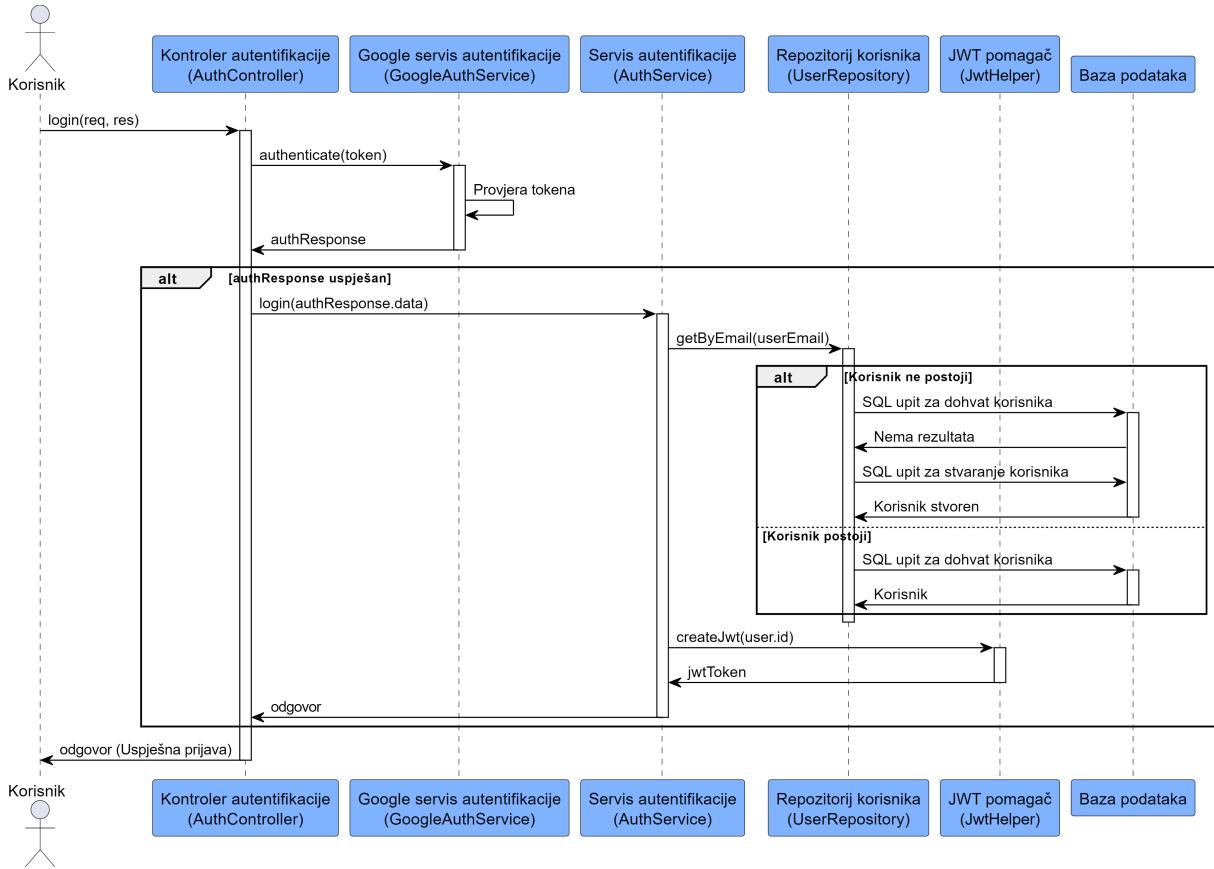
Glavne elemente dijagrama slučajeva korištenja *KeepFit* aplikacije možemo definirati na sljedeći način:

- **Korisnik:** Predstavlja korisnika aplikacije koji se koristi različitim funkcijama sustava.
- **Slučajevi korištenja:**
  - Pregledavanje profila: Omogućava korisniku da pregleda i ažurira svoj profil.
  - Vođenje evidencije o hrani: Korisnik može unijeti i pratiti podatke o konzumiranoj hrani.
  - Vođenje evidencije o aktivnostima: Omogućava korisniku da zabilježi svoje fizičke aktivnosti.
  - Vođenje bilješki: Korisnik može kreirati bilješke.
  - Postavljanje ciljeva: Korisnik može postaviti ciljeve za svoje zdravlje i aktivnosti.
  - Pregledavanje dnevnih ostvarenih ciljeva: Omogućava korisniku pregleđ napretka prema dnevnim ciljevima.
  - Pregledavanje zdravstvenih indikatora: Korisnik može pratiti i pregledavati povijest svojih zdravstvenih pokazatelja.

## 5.2. Implementacija API-ja

U tablici su prikazane specifične rute API-ja zajedno s kratkim opisima njihove funkcionalnosti. Ove rute omogućuju korisnicima da upravljaju svojim profilima, ciljevima, zdravstvenim pokazateljima, bilješkama, vježbama i unosom prehrane, čime se osigurava sveobuhvatno praćenje i upravljanje osobnim ciljevima u vođenju zdravog načina života. Za implementaciju API-ja korišten je *Node.js* zajedno s *Express.js* okvirom.

Autentifikacija korisnika u *KeepFit* aplikaciji implementirana je korištenjem Google OAuth API servisa. Nakon uspješne autentifikacije putem Googlea, *KeepFit* API generira JWT (*JSON Web Token*), koji se koristi za daljnju autentifikaciju unutar aplikacije, osiguravajući sigurnost korisničkih podataka. Samu implementaciju autentifikacije unutar *KeepFit* aplikacije možemo prikazi putem sljedećeg dijagrama slijeda:



Slika 4: Dijagram slijeda autentifikacije u *KeepFit* aplikaciji [autorski rad]

Tablica 1: Pregled API ruta za *KeepFit* aplikaciji [autorski rad]

API ruta	Opis
/api/v1/auth/login	Autentifikacija prijave
/api/v1/user/current	Dohvat trenutnih detalja korisnika
/api/v1/user/current/goal	Dohvat, stvaranje ili ažuriranje ciljeva za trenutnog korisnika
/api/v1/user/current/health-indicator	Dohvat ili stvaranje zdravstvenih pokazatelja za trenutnog korisnika
/api/v1/user/current/note/:id	Dohvat, stvaranje, ažuriranje ili brisanje bilješki za trenutnog korisnika
/api/v1/user/current/exercise	Dohvat, stvaranje, ažuriranje ili brisanje vježbi za trenutnog korisnika
/api/v1/user/current/food	Dohvat, stvaranje, ažuriranje ili brisanje namirnica za trenutnog korisnika
/api/v1/user/current/food-log	Dohvat, stvaranje ili brisanje zapisa o hrani za trenutnog korisnika
/api/v1/user/current/exercise-log	Dohvat, stvaranje ili brisanje zapisa o vježbama za trenutnog korisnika

Ruta `/api/v1/auth/login` koristi se za autentifikaciju prijave. Ova ruta omogućuje koris-

nicima da se sigurno prijave u svoje KeepFit korisničke račune, autentifikacije se vrši preko vanjskog Google API servisa. Ruta */api/v1/user/current* omogućuje dohvatanje trenutnih detalja korisnika. Kroz ovu rutu, korisnici mogu pristupiti svojim osobnim podacima i informacijama o profilu. Ruta */api/v1/user/current/goal* omogućuje dohvatanje, stvaranje ili ažuriranje ciljeva za trenutnog korisnika. Ova ruta je ključna za postavljanje i praćenje osobnih ciljeva unutar aplikacije. Ruta */api/v1/user/current/health-indicator* koristi se za dohvatanje ili stvaranje zdravstvenih pokazatelja za trenutnog korisnika. Ruta */api/v1/user/current/note/:id* omogućuje dohvatanje, stvaranje, ažuriranje ili brisanje bilješki za trenutnog korisnika. Ova funkcionalnost je korisna za bilježenje podsjetnika vezanih uz vježbanje i prehranu. Ruta */api/v1/user/current/exercise* se koristi za dohvatanje, stvaranje, ažuriranje ili brisanje vježbi za trenutnog korisnika. Korisnici kroz ovu rutu mogu upravljati svojim rutinama. Ruta */api/v1/user/current/food* omogućuje dohvatanje, stvaranje, ažuriranje ili brisanje namirnica za trenutnog korisnika. Ova ruta je ključna za praćenje i upravljanje prehrane. Ruta */api/v1/user/current/food-log* služi za dohvatanje, stvaranje ili brisanje zapisa o hrani za trenutnog korisnika. Kroz nju korisnici mogu voditi dnevnik svoje prehrane. Ruta */api/v1/user/current/exercise-log* omogućuje dohvatanje, stvaranje ili brisanje zapisa o vježbama za trenutnog korisnika. Ovo omogućuje korisnicima da detaljno prate svoje aktivnosti.

---

```

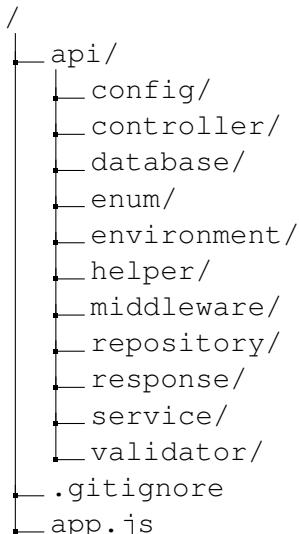
1 async get(req, res) {
2     const response = await this.#goalService.get(req.body.userId)
3     return response.success
4         ? res.status(response.statusCode).json(response)
5         : res.status(response.statusCode).json(response)
6 }
7
7 async get(userId) {
8     const userGoals = await this.#goalRepository.getAllByUserId(userId)
9     if (userGoals.length == 0)
10        return ResponseBuilder.createErrorResponse({})
11     const fetchedGoals = {
12         weight_goal: null,
13         nutrients_goal: null,
14         fitness_goal: null
15     }
16     userGoals.forEach(goal => {
17         if (goal.type == GoalType.WEIGHT) {
18             fetchedGoals.weight_goal = goal
19         } else if (goal.type == GoalType.NUTRITION) {
20             fetchedGoals.nutrients_goal = goal
21         } else if (goal.type == GoalType.FITNESS) {
22             fetchedGoals.fitness_goal = goal
23         }
24     })
25     return ResponseBuilder.createSuccessResponse(fetchedGoals)
26 }
27
27 async getAllById(userId) {
28     const sql = 'SELECT * FROM goal WHERE user_id = $1;'
29     const result = await this.#db.query(sql, [userId])
30     return result.rows
31 }

```

---

Isječak koda 1: Prikaz troslojne arhitekture API-ja *KeepFit* aplikacije

Samu strukturu API dijela projekta možemo prikazati na sljedeći način:



## 5.3. Mobilna aplikacija

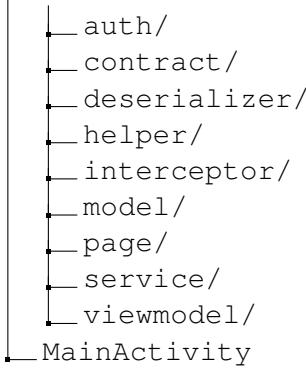
Razvoj mobilne aplikacije *KeepFit* bio je usmjeren na platformu Android, posebice od verzije API-ja 27, odnosno Android 8.1. Izbor jezika za razvoj pao je na *Kotlin*. *Kotlin* je izuzetno pogodan za pisanje asinkronog kôda, zahvaljujući svojoj implementaciji korutina. Za implementaciju korisničkog sučelja, odabran je *Jetpack Compose*, alat za izradu deklarativnih UI komponenta u Android-u. Korištenje *Jetpack Compose* alata omogućuje definiranje reaktivnih sučelja čime je sam razvoj znatno ubrzan. Za mrežnu komunikaciju s API-jem korištena je *Retrofit* biblioteka. Za vizualizaciju podataka korištena je biblioteka *co.yml:ycharts:2.1.0*. Autentifikacija korisnika implementirana je putem Google-a koristeći *com.google.android.gms:play-services-auth:20.7.0*.

```
1 val foodResponse by foodViewModel.foodResponse.observeAsState(null)
2 foodViewModel.tryGetFoodLog(
3     LocalDate.now().toString().format(DateTimeFormatter.ISO_DATE),
4     onSucceed = {
5         checkLoadingState()
6     },
7     onFailed = {
8         isLoading = false
9         Toast.makeText(
10             context,
11             foodViewModel.apiMessage.value.toString(),
12             Toast.LENGTH_LONG
13         ).show()
14     }
15 )
16
17 fun tryGetFoodLog(date: String, onFailed: () -> Unit, onSucceed: () -> Unit = {}) {
18     viewModelScope.launch {
19         try {
20             val response = foodService.getLogs(Auth.authUserData!!.jwt, date)
21             if (response.isSuccessful) {
22                 val apiResponse = response.body()
23                 _foodResponse.value = apiResponse?.data
24                 reportApiMessage(apiResponse!!.message)
25                 onSucceed()
26             }
27         } catch (exception: Exception) {
28             reportApiMessage("Service currently not available")
29             onFailed()
30         }
31     }
32 }
```

Isječak koda 2: Prikaz MVVM arhitekture mobilnog dijela KeepFit aplikacije

Samu strukturu mobilne aplikacije možemo prikazati na sljedeći način:

```
app/
└── manifests/
    └── AndroidManifest.xml
```



## 5.4. Baza podataka

Sama baza podataka integrira koncepte temporalne baze podataka u kontekstu definiranja vremena transakcije pojedinog zapisa u svrhu kreiranja *povijesnih* tablica. Implementacija povijesnih tablica može se ostvariti kreiranjem nove tablice koja sadrži sve atribute definirane u prvobitnoj instance te tablice ili definiranjem referentnog integriteta između prvobitne tablice i povijesna tablice, kreiranjem vanjskog ključa čija je zadaća zatim povezati te dvije tablice ili ukoliko nije potrebna relacija *jedan-prema-više* sama povijesna tablica može sadržavati sve potrebne informacije.

U shemi *KeepFit* aplikacije, vremenska dimenzija ugrađena je u različite tablice, kao što su *o\_user*, *health\_indicator\_history*, *food\_log*, *exercise\_log* i *note*.

Tablica *o\_user* sadrži stupac *created\_at* koji bilježi trenutak stvaranja korisničkog računa. To omogućuje aplikaciji da zadrži informaciju o tome kada je račun prvi put stvoren.

Tablica *health\_indicator\_history* sadrži vremensku oznaku *created\_at* koja se koristi za praćenje povijesti zdravstvenih indikatora korisnika. To omogućuje praćenje promjena u zdravstvenom stanju korisnika tijekom vremena. Sama tablica *health\_indicator\_history* je samo nadogradiva (engl. *append-only*).

Tablice *food\_log* i *exercise\_log* koriste vremenske označke za praćenje kada su određene namirnice konzumirane ili vježbe izvedene. Vremenska dimenzija omogućuje korisnicima da vide svoje prehrambene i tjelesne navike u određenom vremenskom okviru. Tablice *food\_log* i *exercise\_log* omogućuju i brisanje i izmjenu unesenih podataka.

Tablica *note* koristi atribut *created\_at* kako bi zabilježila kada je bilješka napravljena. Ovo omogućuje korisnicima da vide kronološki kontekst svojih bilješki vezanih uz hranu i vježbanje. Te također *created\_at* atribut se koristi za definirane ograničenja gdje korisnik može u jednom danu kreirati samo onoliko bilješki koliko smo definirano različitim *type* tipova bilješki.

Korištenje ovih temporalnih aspekata u bazi podataka omogućuje *KeepFit* aplikaciji da pruži korisnicima uvid u njihov osobni razvoj i napredak kroz vrijeme.

---

```

1  CREATE TABLE health_indicator_history (
2      id SERIAL PRIMARY KEY,
3      user_id INT REFERENCES o_user(id) NOT NULL,
4      created_at DATE NOT NULL,
5      indicators JSONB NOT NULL
6  );
7
7  CREATE TABLE exercise_log (
8      id SERIAL PRIMARY KEY,
9      user_id INT REFERENCES o_user(id) NOT NULL,
10     exercise_id INT REFERENCES exercise(id) ON DELETE CASCADE NOT NULL,
11     created_at DATE NOT NULL,
12     quantity INT NOT NULL DEFAULT 1,
13     UNIQUE (user_id, exercise_id, created_at)
14 );
15
15 CREATE TABLE food_log (
16     id SERIAL PRIMARY KEY,
17     user_id INT REFERENCES o_user(id) NOT NULL,
18     food_id INT REFERENCES food(id) ON DELETE CASCADE NOT NULL,
19     created_at DATE NOT NULL,
20     quantity INT NOT NULL DEFAULT 1,
21     UNIQUE (user_id, food_id, created_at)
22 );

```

---

Isječak koda 3: SQL kod za stvaranje tablica *health\_indicator\_history*, *food\_log*, *exercise\_log*

Implementacija aktivne baze podataka prikazana je u priloženom SQL kodu, definirana je funkcija *update\_weight\_history\_on\_goal\_patch()* koja se izvršava kada se stvori ili ažurira zapis u tablici *goal*. Ako je tip cilja ‘Weight’, funkcija izvlači novu težinu iz JSONB objekta cilja i ubacuje taj podatak u tablicu *health\_indicator\_history*. Ovo osigurava da svaka promjena u ciljevima vezanim za težinu bude zabilježena u povijesti zdravstvenih indikatora.

Okidač (engl. *trigger*) *trigger\_update\_weight\_on\_goal\_patch* postavljen je tako da nakon svakog umetanja ili ažuriranja cilja izvršava gore navedenu funkciju. Ovo omogućuje aplikaciji da automatski prati promjene vezane za težinu korisnika bez potrebe za *ručnim* ažuriranjem tablica.

---

```

1 CREATE OR REPLACE FUNCTION update_weight_history_on_goal_patch()
2 RETURNS TRIGGER AS $$ 
3 DECLARE
4     new_weight DECIMAL;
5 BEGIN
6     IF NEW.type = 'Weight' THEN
7         new_weight := (NEW.goal->>'current_weight')::DECIMAL;
8
9         INSERT INTO health_indicator_history (user_id, created_at, indicators)
10        VALUES (NEW.user_id, CURRENT_TIMESTAMP, jsonb_build_object('current_weight',
11        ↳ new_weight));
12    END IF;
13
14    RETURN NEW;
15 END;
16 $$ LANGUAGE plpgsql;
17
18 CREATE OR REPLACE TRIGGER trigger_update_weight_on_goal_patch1
19 AFTER INSERT OR UPDATE OF goal ON goal
20 FOR EACH ROW
21 EXECUTE FUNCTION update_weight_history_on_goal_patch();

```

---

Isječak koda 4: Korištenje triggera za automatsko ažuriranje povijesti težine u aktivnoj bazi podataka

U razvoju relacijskih baza podataka, posebno kada je riječ o upravljanju složenim ili heterogenim skupom podataka, često se susrećemo s izazovom kako efikasno strukturirati i upravljati takvim podacima. Tip podatka JSONB u PostgreSQL bazi podataka pruža rješenje za ovaj problem. JSONB, što označava JSON Binary, format je koji omogućava pohranu JSON objekata u binarno optimiziranom formatu. Ovaj pristup omogućuje brže čitanje i indeksiranje podataka u usporedbi s klasičnim JSON tipom podatka jer ne zahtijeva ponovno parsiranje pri svakom čitanju, što rezultira poboljšanom performansom upita [5].

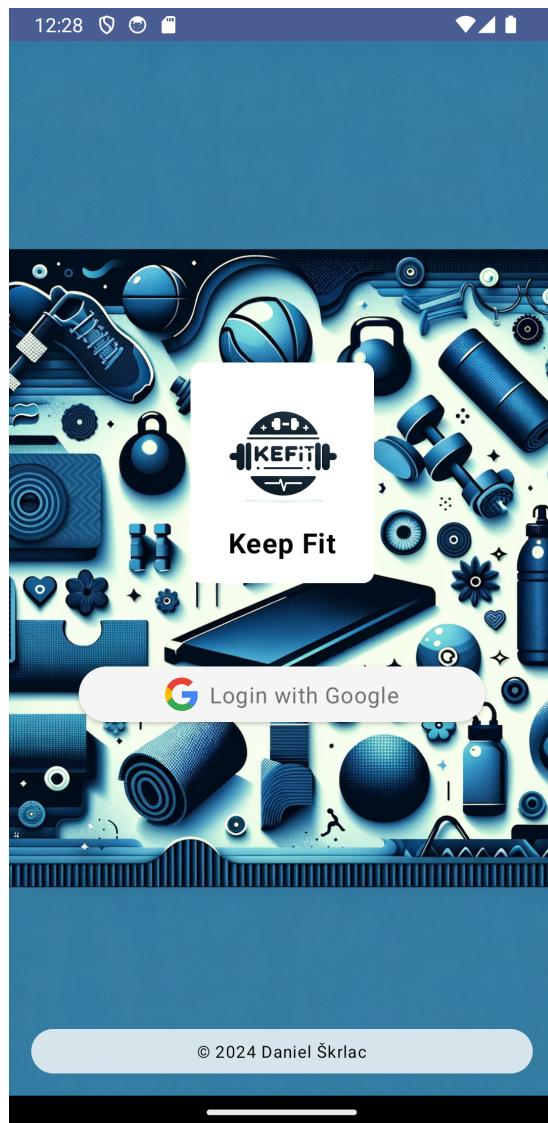
Primjena JSONB tipa podatka osobito je korisna u tablicama kao što je na primjer tabela *health\_indicator\_history* itd., gdje svaki unos sadrži skup specifičnih podataka koji se mogu značajno razlikovati ovisno o kontekstu. Implementacija takvog modela u klasičnoj relacijskoj shemi rezultirala bi brojnim zapisima s mnogo `NULL` vrijednosti, jer svaki atribut zahtijeva zaseban stupac.

Na primjer u tablici *health\_indicator\_history*, JSONB se koristi za praćenje različitih zdravstvenih indikatora koji se mogu promijeniti tijekom vremena. Ovo uključuje podatke kao što su krvni tlak, razina šećera u krvi, tjelesna masa itd. Korištenjem JSONB tipa podatka, svi ovi podaci se mogu efikasno pohraniti i ažurirati u jednom stupcu, čime se pojednostavljuje upravljanje podacima i omogućuje brže dohvaćanje povijesnih podataka za analizu.

Korištenje JSONB tipa podataka u PostgreSQL bazi podataka pokazuje se izuzetno korisnim u scenarijima gdje su performanse čitanja podataka od ključne važnosti. To je osobito važno u aplikacijama koje zahtijevaju brzo dohvaćanje složenih podataka, a gdje tradicionalni pristup zahtijeva izvođenje upita nad višestrukim tablicama.

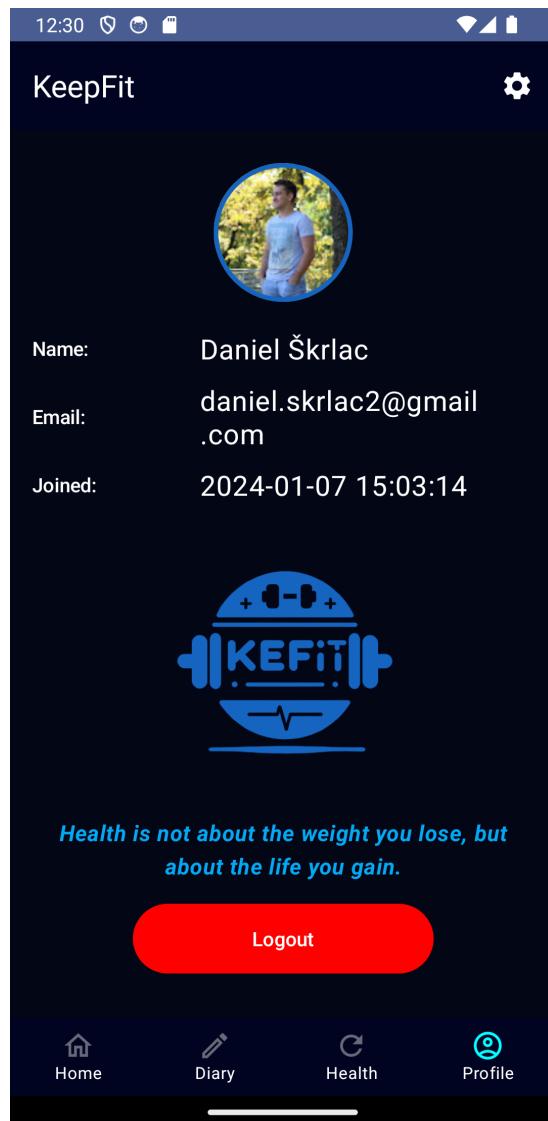
## 6. Primjeri korištenja

U nastavku su prikazane ključne funkcionalnosti *KeepFit* mobilne aplikacije.



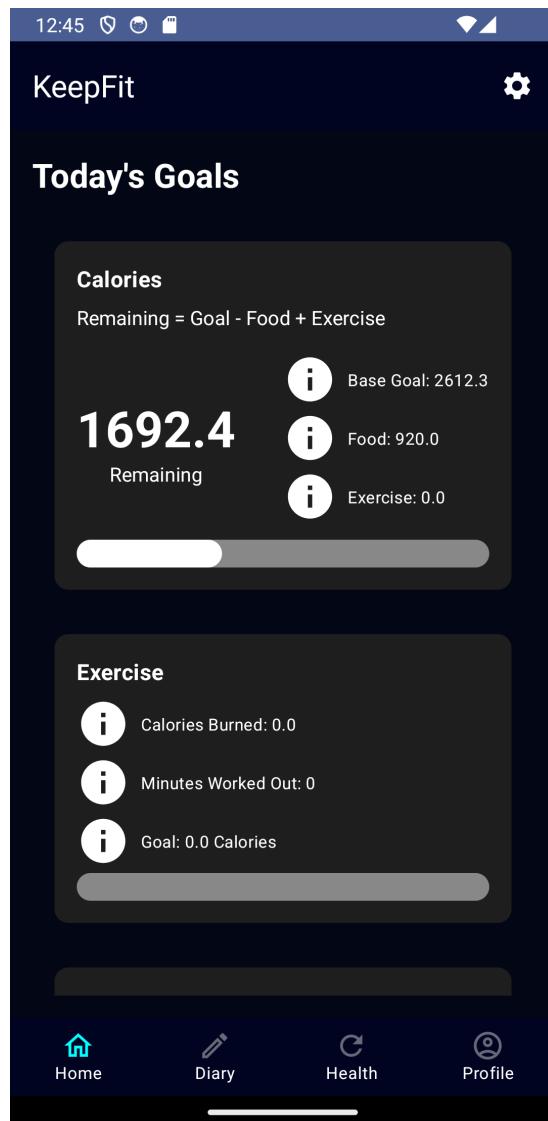
Slika 5: Prikaz login funkcionalnosti *KeepFit* aplikacije [autorski rad]

Na početnom ekranu mobilne aplikacije *KeepFit* prikazan je logo aplikacije uz opciju za prijavu pomoću Google računa.



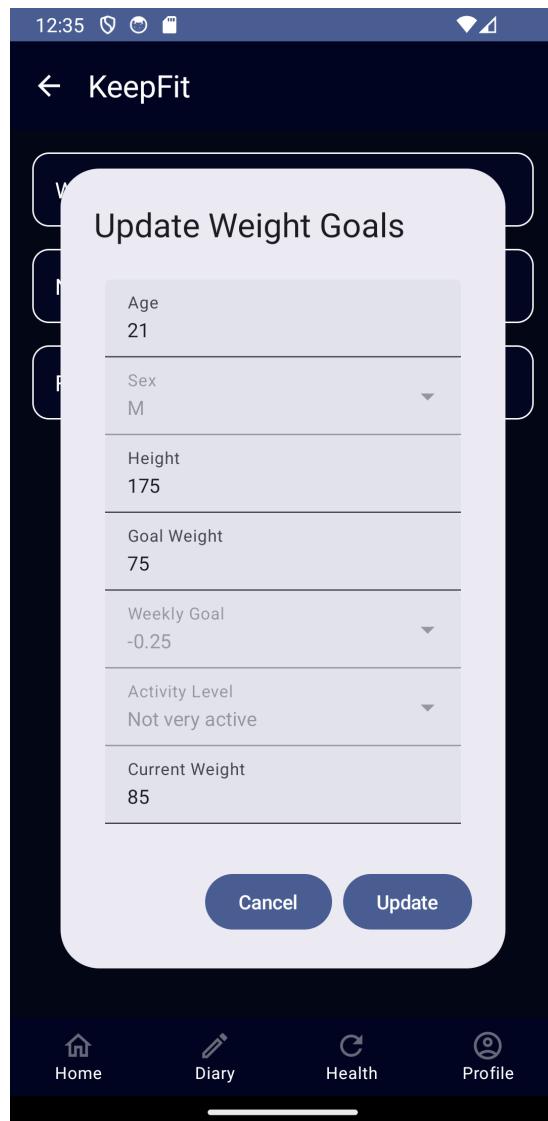
Slika 6: Prikaz ekrana pregleda profila *KeepFit* aplikacije [autorski rad]

Na prikazanom ekranu profilne stranice mobilne aplikacije *KeepFit*, korisnici mogu vidjeti svoje osnovne informacije, uključujući ime, email i datum i vrijeme pridruživanja. U sredini ekrana je profilna slika korisnika. Na dnu ekrana nalazi se gumb za odjavu.



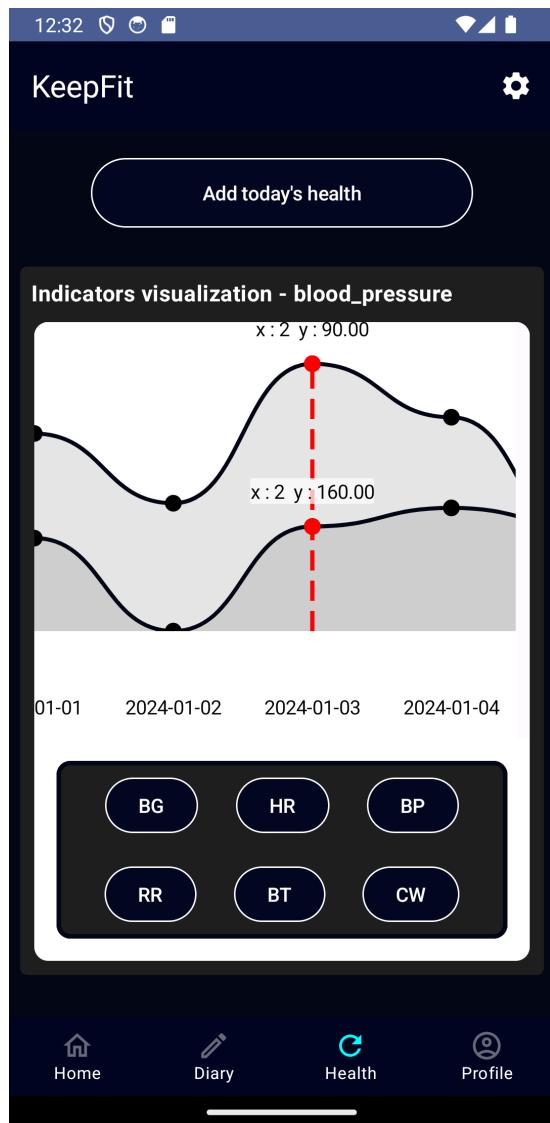
Slika 7: Prikaz ekrana stranice *KeepFit* aplikacije [autorski rad]

Na prikazanom ekranu, korisnici aplikacije *KeepFit* mogu pratiti svoje dnevne ciljeve povezane s kalorijama, vježbanjem te dnevnim unosom potrebnih nutrijenata.



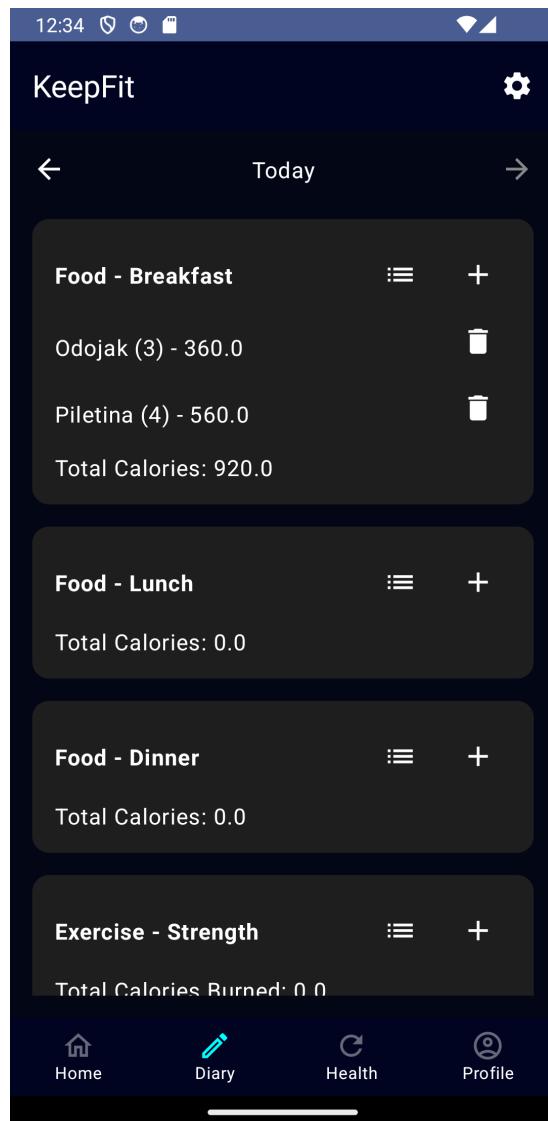
Slika 8: Prikaz ekrana za ažuriranje ciljeva korisnika *KeepFit* aplikacije [autorski rad]

Ekran prikazuje sučelje unutar aplikacije *KeepFit* gdje korisnici mogu ažurirati svoje ciljeve vezane uz tjelesnu masu. Korisniku su na raspolaganju opcije za unos ili promjenu informacija kao što su dob, spol, visina, trenutna i ciljana težina, tjedni cilj gubitka ili dobitka težine te razina fizičke aktivnosti.



Slika 9: Prikaz ekrana za pregled zdravstvenih pokazatelja *KeepFit* aplikacije [autorski rad]

Na ekranu je prikazana zdravstvena sekcija aplikacije *KeepFit* gdje korisnici mogu dodati i vizualizirati različite zdravstvene pokazatelje. Vidljiv je graf koji prikazuje mjerena krvnog tlaka tijekom određenog vremenskog razdoblja. Korisnici mogu odabrati i prikazati druge indikatore zdravlja, kao što su razina glukoze u krvi (BG), brzina otkucaja srca (HR), disanje (RR), tjelesna temperatura (BT) i trenutna težina (CW), pomoću interaktivnih gumba na ekranu.



Slika 10: Prikaz ekrana za pregled dnevnika *KeepFit* aplikacije [autorski rad]

Ekran prikazuje dnevnik unosa hrane, vježbanja te dodavanja bilješki u aplikaciji *KeepFit*. Na ekranu su kategorije obroka poput doručka, ručka i večere, s mogućnošću dodavanja i brisanja stavki hrane. Za svaku kategoriju obroka prikazan je ukupan broj kalorija. Također se prikazuju podaci o vježbanju, uključujući ukupan broj potrošenih kalorija te bilješke koje se nalaze na dnu ekrana.

## 7. Zaključak

U procesu izrade aplikacije *KeepFit*, upotreba temporalnih i aktivnih baza podataka, zajedno s integracijom JSON tipa podataka, pokazala se kao dobra kombinacija za implementaciju aplikacije usmjerenе na zdravlje. Temporalne baze podataka pružile su mogućnost za praćenje promjena u zdravstvenim pokazateljima korisnika kroz vrijeme. Aktivne baze podataka, s druge strane, automatizirale su procese unutar aplikacije, smanjujući potrebu za ručnim ažuriranjima promjena u bazi podataka. Integracija JSONB tipa podataka unutar PostgreSQL baze podataka omogućila je visoku razinu fleksibilnosti u pohrani i obradi podataka čime u budućnosti, samim povećanjem korisnika, možemo težiti prema podatkovnoj modularnosti mobilne aplikacije. Jedan od izazova implementacije bio je implementacija dodatne validacije za osiguranje integriteta podataka spremljenih u ovom formatu te je bilo potrebno definirati priлагodjene funkcije za efikasno upravljanje JSONB podacima kao što je na primjer ažuriranje podataka u JSON formatu.

# Popis literature

- [1] C. J. Date, H. Darwen i N. A. Lorentzos, *Time and relational theory: temporal databases in the relational model and SQL*, eng. Waltham, MA: Morgan Kaufmann, 2014., ISBN: 978-0-12-800675-7 978-0-12-800631-3.
- [2] P. Revesz, „Temporal Databases,” en, *Introduction to Databases: From Biological to Spatio-Temporal*, serija Texts in Computer Science, P. Revesz, ur., London: Springer, 2010., str. 67–79, ISBN: 978-1-84996-095-3. DOI: 10.1007/978-1-84996-095-3\_5.
- [3] „Active database systems,” en, *ACM Computing Surveys*, sv. 31, br. 1, str. 63–103, 3. 1999., Publisher: ACM, ISSN: 0360-0300. DOI: 10.1145/311531.311623.
- [4] S. Asad, *An introduction to working with JSON data in PostgreSQL*, en, 9. 2023.
- [5] *8.14. JSON Types*, en, 11. 2023.

# Popis slika

1.	Arhitektura <i>KeepFit</i> aplikacije [autorski rad] . . . . .	3
2.	ERA dijagram <i>KeepFit</i> aplikacije [autorski rad] . . . . .	8
3.	Dijagram slučajeva korištenja <i>KeepFit</i> aplikacije [autorski rad] . . . . .	10
4.	Dijagram slijeda autentifikacije u <i>KeepFit</i> aplikaciji [autorski rad] . . . . .	12
5.	Prikaz login funkcionalnosti <i>KeepFit</i> aplikacije [autorski rad] . . . . .	19
6.	Prikaz ekrana pregleda profila <i>KeepFit</i> aplikacije [autorski rad] . . . . .	20
7.	Prikaz ekrana stranice <i>KeepFit</i> aplikacije [autorski rad] . . . . .	21
8.	Prikaz ekrana za ažuriranje ciljeva korisnika <i>KeepFit</i> aplikacije [autorski rad] . . .	22
9.	Prikaz ekrana za pregled zdravstvenih pokazatelja <i>KeepFit</i> aplikacije [autorski rad]	23
10.	Prikaz ekrana za pregled dnevnika <i>KeepFit</i> aplikacije [autorski rad] . . . . .	24

# **Popis tablica**

1.	Pregled API ruta za <i>KeepFit</i> aplikaciji [autorski rad] . . . . .	12
----	--	----

# Popis isječaka koda

1.	Prikaz troslojne arhitekture API-ja <i>KeepFit</i> aplikacije . . . . .	14
2.	Prikaz MVVM arhitekture mobilnog dijela <i>KeepFit</i> aplikacije . . . . .	15
3.	SQL kod za stvaranje tablica <i>health_indicator_history</i> , <i>food_log</i> , <i>exercise_log</i> . .	17
4.	Korištenje triggera za automatsko ažuriranje povijesti težine u aktivnoj bazi podataka . . . . .	18

## **Prilozi**

## 1. Prilog

Radu je priložen programski kôd u *.zip* arhivi. Programski kôd dostupan je i na *poveznici*. Također, Overleaf projekt dostupan je na *poveznici*.