Android aplikacija za praćenje unosa kalorija i tjelesnih aktivnosti

Paprić, Jovana

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

Stručni prijediplomski studij Računarstvo

Android aplikacija za praćenje unosa kalorija i tjelesnih aktivnosti

Završni rad

Jovana Paprić

Osijek, 2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1S: Obrazac za ocjenu završnog rada na stručnom prijediplomskom studiju

Ocjena završnog rada na stručnom prijediplomskom studiju

Ime i prezime pristupnika:	Jovana Paprić
Studij, smjer:	Stručni prijediplomski studij Računarstvo
Mat. br. pristupnika, god.	AR4870, 27.07.2021.
JMBAG:	0165090897
Mentor:	prof. dr. sc. Krešimir Nenadić
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	doc. dr. sc. Krešimir Romić
Član Povjerenstva 1:	prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 2:	Robert Šojo, univ. mag. ing. comp.
Naslov završnog rada:	%naziv_rada%
Znanstvena grana završnog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada:	Kao dio ovog zadatka potrebno je izraditi Android aplikaciju pomoću koje će korisnici moći pratiti unos kalorija u organizam i obavljanje tjelesnih aktivnosti, što uključuje i spavanje. Potrebno je napraviti usporedbu izrađene aplikacije s nekoliko postojećih sličnih rješenja te naglasiti prednosti i nedostatke izrađene aplikacije u usporedbi s postojećim rješenjima. Detaljno opisati postupa izrade aplikacije kao i izrađene funkcionalnosti. Omogućiti korisniku nekakav oblik praćenja napretka u obliku statistike. Kao potporu radu aplikacije potrebno je projektirati i izraditi bazu

Datum ocjene pismenog dijela završnog rada od strane mentora:	08.07.2024.
Ocjena pismenog dijela završnog rada od strane mentora:	Izvrstan (5)
Datum obrane završnog rada:	15.7.2024.
Ocjena usmenog dijela završnog rada (obrane):	Izvrstan (5)
Ukupna ocjena završnog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije završnog rada čime je pristupnik završio stručni prijediplomski studij:	15.07.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 15.07.2024.

Ime i prezime Pristupnika:	Jovana Paprić
Studij:	Stručni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	AR4870, 27.07.2021.
Turnitin podudaranje [%]:	6

Ovom izjavom izjavljujem da je rad pod nazivom**: Android aplikacija za praćenje unosa kalorija i tjelesnih aktivnosti**

izrađen pod vodstvom mentora prof. dr. sc. Krešimir Nenadić

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

SADRŽAJ

1. UVOD

U današnjem brzom i zahtjevnom životnom stilu, mnogi ljudi često zanemaruju brigu o vlastitom zdravlju. Uz sve veći broj kancelarijskih poslova, rad od kuće i online poslova, fizička aktivnost se smanjuje, što dovodi do raznih zdravstvenih problema i nezadovoljstva osobnim izgledom. Ova moderna dinamika stvara začarani krug u kojem se ljudi suočavaju s izazovima održavanja zdravog načina života.

Cilj ovog završnog rada je razvoj Android aplikacije koja pomaže korisnicima u njihovom putovanju prema zdravstvenim i tjelesnim ciljevima. Aplikacija nudi alate za praćenje unosa kalorija, hrane, tekućine i napretka u težini. Također, korisnicima pruža mogućnost praćenja svojih osjećaja tijekom procesa i načina za unaprjeđenje njihovog zdravstvenog putovanja. Dodatno, aplikacija sadrži kutak s receptima kako bi inspirirala korisnike u pripremi izbalansiranih obroka.

Za izradu aplikacije korišteni su programski jezik Kotlin, a za izradu korisničkog sučelja korišten je Jetpack Compose. Integrirani su Spoonacular API (eng. *Application Programming Interface*) za generiranje nasumičnih recepata i Postman za testiranje API-ja. Za autentikaciju i pohranu podataka korištena je Firebase baza podataka.

Struktura završnog rada uključuje sljedeća navedena poglavlja. Drugo poglavlje, Pregled postojećih rješenja, pruža pregled postojećih rješenja koja su služila kao uzor u izradi. Treće poglavlje, Zahtjevi za programsko rješenje, opisuje korisničke zahtjeve i funkcionalnosti. Četvrto poglavlje, Alati i tehnologije, opisuje tehničke značajke i korištene tehnologije. Peto poglavlje, Izrada programskog rješenja, fokusira se na tijek izrade programskog rješenja. Šesto poglavlje, Zaključak, sadrži zaključna razmatranja i doprinos rada.

1.1. Zadatak završnog rada

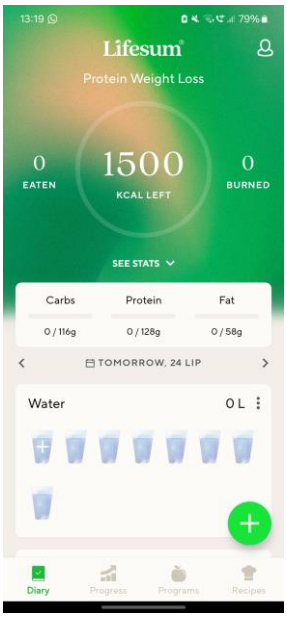
Kao dio ovog zadatka potrebno je izraditi Android aplikaciju pomoću koje će korisnici moći pratiti unos kalorija u organizam i obavljanje tjelesnih aktivnosti, što uključuje i spavanje. Potrebno je napraviti usporedbu izrađene aplikacije s nekoliko postojećih sličnih rješenja te naglasiti prednosti i nedostatke izrađene aplikacije u usporedbi s postojećim rješenjima. Detaljno opisati postupak izrade aplikacije kao i izrađene funkcionalnosti. Omogućiti korisniku nekakav oblik praćenja napretka u obliku statistike. Kao potporu radu aplikacije potrebno je projektirati i izraditi bazu podataka te opisati postupak izrade baze.

2. PREGLED POSTOJEĆIH RJEŠENJA

U ovom poglavlju prikazan je kratak pregled nekoliko popularnih postojećih aplikacija koje su služile kao inspiracija u izradi programskog rješenja. Ove aplikacije nude razne funkcionalnosti koje korisnicima olakšavaju postizanje zdravstvenih i tjelesnih ciljeva. Cilj ovog poglavlja je prikazati funkcionalnosti ovih aplikacija i usporediti njihove značajke, a prikazane će biti: Lifesum, MyFitnessPal, Caloriegram, Fitia i Yazio.

2.1. Lifesum

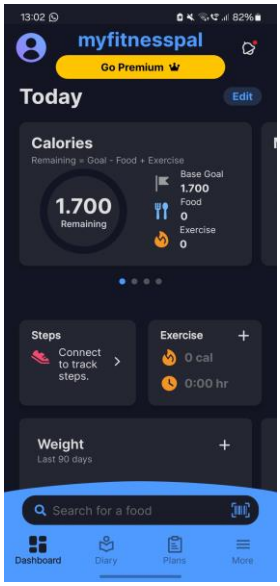
Lifesum nudi personaliziranu podršku za zdravu prehranu i postizanje ciljeva mršavljenja. Aplikacija sadrži dnevnik hrane, brojač kalorija i praćenje makronutrijenata. Nudi planove prehrane poput keto, mediteranske i visoko proteinske dijete te opcije povremenog posta (eng. *intermittent fasting*). Integrira se s fitnes uređajima za dubinsko praćenje zdravlja i nudi „*Life Score"* test za personalizirane preporuke. Također nudi planove obroka s popisima namirnica i primjere recepata. [1] Na slici 2.1. prikazano je sučelje aplikacije Lifesum.



Sl. 2.1. Sučelje aplikacije Lifesum

2.2. MyFitnessPal

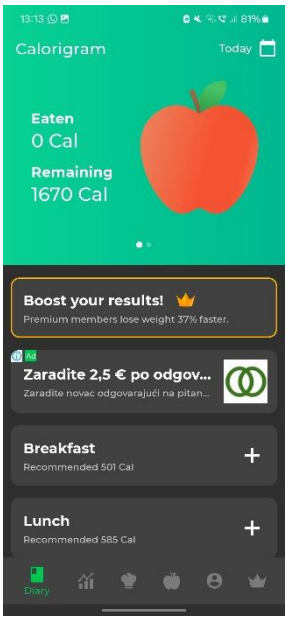
MyFitnessPal je sveobuhvatan alat za praćenje prehrane, fitnessa i postizanje zdravstvenih ciljeva. Pruža dnevnik hrane, brojač kalorija, uređaj za praćenje makronutrijenata i planer obroka. Uz ovu aplikaciju može se pratiti napredak u kondiciji, prilagoditi ciljeve za mršavljenje ili održavanje težine, i povećati svijest o prehrambenim navikama. MyFitnessPal također nudi puno recepata za zdravu prehranu i povezivanje s aktivnom zajednicom korisnika. [2] Na slici 2.2. prikazano je sučelje aplikacije MyFitnessPal.



Sl. 2.2. Sučelje aplikacije MyFitnessPal

2.3. CalorieGram

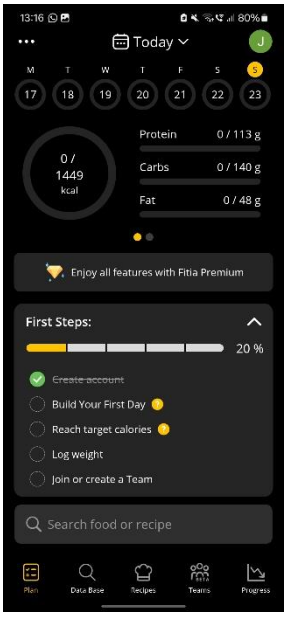
CalorieGram je aplikacija koja se fokusira na jednostavno praćenje kalorijskog unosa. Korisnici mogu unositi obroke, a aplikacija izračunava kalorije i prati dnevni unos. Aplikacija je dizajnirana da bude korisnički jednostavna, s brzim unosom podataka i jasnim prikazom kalorijskog statusa. CalorieGram je koristan alat za one koji žele brzo i učinkovito pratiti svoje kalorijske ciljeve. [3] Na slici 2.3. prikazano je sučelje aplikacije CalorieGram.



Sl. 2.3. Sučelje aplikacije CalorieGram

2.4. Fitia

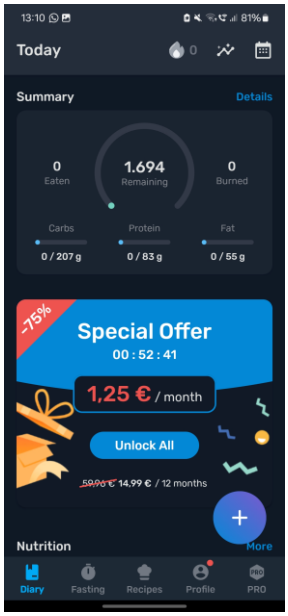
Fitia je aplikacija za brojanje kalorija i planiranje obroka. Sadrži puno provjerenih proizvoda i zdravih recepata te omogućuje izračun ciljanih kalorija i makronutrijenata za mršavljenje ili dobivanje mišića. Mogu se prilagoditi prehrambeni ciljevi i birati između jednostavnih namirnica ili detaljnih recepata. Ima automatski popis za kupovinu i personalizirane recepte prema potrebama korisnika. Također ima implementirane podsjetnike za obroke i unos vode. [4] Na slici 2.4. prikazano je sučelje aplikacije Fitia.



Sl. 2.4. Sučelje aplikacije Fitia

2.5. Yazio

Yazio je aplikacija za brojanje kalorija koja podržava post (eng. *intermittent fasting*) naizmjeničnim danima za učinkovito mršavljenje bez strogih dijeta. Omogućava brojanje kalorija i sadrži dnevnik hrane s praćenjem posta za metode poput 16:8 ili 5:2. Ima veliku bazu podataka hrane, recepte, automatsko praćenje aktivnosti i podsjetnike za unos vode. [5] Na slici 2.5. prikazano je sučelje aplikacije Yazio.



Sl. 2.5. Sučelje aplikacije Yazio

.

3. ALATI I TEHNOLOGIJE

Razvoj ove aplikacije oslanja se na niz naprednih alata i tehnologija koji omogućuju efikasan i učinkovit proces razvoja. U ovom poglavlju detaljno je opisan svaki od korištenih alata i tehnologija, naglašavajući njihove ključne karakteristike i uloge u razvoju aplikacije. Korišteni alati i tehnologije su:

- Android studio
- Kotlin
- Jetpack Compose
- Firebase
- Postman
- Spoonacular

3.1. Android Studio

Android Studio je službeno integrirano razvojno okruženje (IDE) za Android platformu. Razvila ga je tvrtka Google, Android Studio pruža sveobuhvatan skup alata za razvoj, testiranje i otklanjanje pogrešaka Android aplikacija. Neke od ključnih značajki uključuju:

- Napredni uređivač koda: Pruža inteligentne prijedloge, automatsko dovršavanje koda i analizu sintakse, što značajno ubrzava proces razvoja.
- Alati za dizajn korisničkog sučelja: Omogućuje vizualno dizajniranje sučelja s podrškom za pregled u stvarnom vremenu na različitim veličinama ekrana.
- Ugrađeni emulatori: Omogućuju testiranje aplikacija na različitim Android uređajima i konfiguracijama bez potrebe za fizičkim uređajem.
- Integracija s verzionim sustavima: Pruža podršku za Git, što olakšava praćenje verzija koda i suradnju među razvojnim timom. [7]

3.2. Kotlin

Kotlin je moderni programski jezik koji je razvila tvrtka JetBrains. Kotlin se prvotno razvio kao alternativa postojećim programskim jezicima za razvoj na Java Virtual Machine (JVM), ali je s vremenom postao sveprisutan jezik za razvoj aplikacija na različitim platformama. Nekoliko ključnih značajki i karakteristika Kotlin jezika:

- Kompatibilnost s postojećim sustavima: Kotlin je kompatibilan s Java platformom, što znači da se Kotlin kod može koristiti zajedno s postojećim Java projektima. Ovo omogućuje postupno usvajanje Kotlina u postojeće aplikacije i lakšu integraciju s postojećim Java bibliotekama i alatima.
- Sigurnost i stabilnost: Kotlin je dizajniran s naglaskom na sigurnost i stabilnost. Pruža dodatne sigurnosne mehanizme poput provjere *null* vrijednosti na razini jezika, što pomaže smanjiti greške u izvršavanju vezane uz *null* vrijednosti.
- Kompaktnost i izražajnost: Kotlin jezik je čist i izražajan, što znači da možete postići više s manje koda. Sintaksa je jednostavna i čitljiva, što olakšava razumijevanje i održavanje koda.
- Višestruka upotreba: Kotlin se može koristiti za razvoj različitih vrsta aplikacija, uključujući Android aplikacije, web aplikacije, serverske aplikacije, desktop aplikacije i još mnogo toga. Također podržava više platformi poput JVM-a, Androida, JavaScripta i Native-a.
- Funkcionalno programiranje: Kotlin podržava funkcionalno programiranje kao jedan od svojih ključnih paradigmi. Omogućuje definiranje funkcija visokog reda, lambda izraza, imutabilnih struktura podataka i ostalih funkcionalnih konstrukcija.
- Interoperabilnost: Kotlin se može koristiti zajedno s Java kodom bez problema. To omogućuje postupno usvajanje Kotlina u postojeće Java projekte, kao i zajedničko korištenje Java i Kotlin koda u istom projektu.

Kotlin je postao popularan izbor za razvoj Android aplikacija zbog svojih modernih značajki, sigurnosti i jednostavnosti upotrebe. Međutim, sve više se koristi i za druge vrste aplikacija zbog svoje fleksibilnosti i kompatibilnosti s različitim platformama. [8]

3.3. Jetpack Compose

Jetpack Compose je moderni UI alat za razvoj aplikacija na Android platformi, koji donosi revolucionarni pristup izgradnji korisničkog sučelja. Ova biblioteka koristi deklarativni pristup za stvaranje korisničkog sučelja, omogućujući programerima da intuitivno i reaktivno opisuju izgled i ponašanje korisničkog sučelja u različitim situacijama.

U središtu Jetpack Composea nalaze se *composable* funkcije, koje su osnovni gradivni elementi korisničkog sučelja. Ove funkcije mogu se sastojati od drugih *composable* funkcija, čime se stvaraju složene hijerarhije i omogućuje modularan i fleksibilan razvoj korisničkog sučelja.

Prednosti korištenja Jetpack Composea uključuju:

- Povećana produktivnost programera: Brži i jednostavniji razvoj korisničkog sučelja.
- Manje grešaka tijekom razvoja: Deklarativni pristup smanjuje mogućnost pogrešaka.
- Brže izmjene UI-ja: Omogućava dinamičko ažuriranje korisničkog sučelja bez potrebe za velikim promjenama koda.
- Jetpack Compose se besprijekorno integrira s drugim Jetpack bibliotekama

Glavna prednost Jetpack Composea je njegova jednostavnost i fleksibilnost. Omogućuje brze izmjene i dinamičko ažuriranje korisničkog sučelja, što čini razvoj aplikacija učinkovitijim i manje podložnim pogreškama. [9] [10]

3.4. Firebase

Firebase je sveobuhvatna platforma za pomoć u razvoju mobilnih i web aplikacija koja pruža širok spektar alata i usluga za backend razvoj. U ovom projektu, korištene su sljedeće Firebase usluge:

- *Firebase Authentication*: Ova usluga omogućava sigurno upravljanje prijavama i registracijama korisnika putem različitih metoda autentikacije, uključujući e-poštu, zaporku, društvene mreže i jednokratne zaporke.
- *Cloud Firestore*: Firestore je NoSQL baza podataka (vrsta baze podataka koja pohranjuje i upravlja podacima na fleksibilne, skalabilne načine bez upotrebe tradicionalnih shema temeljenih na tablicama) koja omogućava pohranu i sinkronizaciju podataka u stvarnom vremenu među korisničkim uređajima. Pruža fleksibilne mogućnosti upita i skalabilnost, što ga čini idealnim za razvoj mobilnih i web aplikacija.
- *Firebase Analytics*: Ova usluga pruža detaljne analitičke podatke o korištenju aplikacije, omogućujući programerima bolje razumijevanje korisničkog ponašanja i optimizaciju aplikacije prema stvarnim potrebama korisnika.

Firebase funkcionira kao *Backend-as-a-Service* (BaaS), što znači da pruža kompletnu infrastrukturu i alate za upravljanje podacima bez potrebe za upravljanjem vlastitim serverima.

Firebase također pruža napredne sigurnosne postavke, kao što su pravila za kontrolu pristupa podacima, te omogućava laku integraciju različitih metoda autentikacije radi osiguranja sigurnog pristupa korisničkim računima. [11] [12]

3.5. Postman

Postman je API platforma (eng. *Application Programming Interface*, skup definiranih pravila i funkcija koji omogućuju aplikacijama komunikaciju i razmjenu podataka međusobno ili s drugim sustavima) za izradu i korištenje API-ja. Postman pojednostavljuje svaki korak životnog ciklusa API-ja i usmjerava suradnju tako da se može brže stvarati bolji API. Alat je za testiranje API-ja koji omogućuje slanje HTTP zahtjeva (eng. *Hypertext Transfer Protocol*, protokol za prijenos hipertekstualnih dokumenata preko mreže koji se koristi za komunikaciju između klijenta i servera na World Wide Webu) i pregled odgovora na intuitivan način. Programeri ga koriste kako bi pojednostavili proces testiranja API-ja pružajući korisničko sučelje za postavljanje zahtjeva, pregledavanje odgovora i otklanjanje pogrešaka. Ključne značajke Postmana uključuju:

- Korisničko sučelje: Intuitivno sučelje koje omogućuje jednostavno kreiranje, slanje i organiziranje HTTP zahtjeva.
- Testiranje API-ja: Pruža mogućnost automatiziranog testiranja API krajnjih točaka, što olakšava identifikaciju i ispravljanje pogrešaka.
- Otklanjanje pogrešaka: Nudi skripte prije i nakon zahtjeva koje omogućuju da se prilagode zahtjevi i automatiziraju određeni zadaci pomoću kojih se može pojednostaviti tijek rada, brzo identificirati i otkloniti sve probleme s API-jem. [13]

3.6. Spoonacular

Spoonacular je API za pristup velikoj bazi podataka recepata, nutritivnih informacija i planova prehrane. Korištenjem Spoonacular API-ja omogućava se:

- Pristup raznolikim receptima: API omogućuje dohvaćanje raznih recepata koje korisnici mogu koristiti kao inspiraciju za kuhanje.
- Nutritivne informacije: Pruža detaljne nutritivne informacije za razne namirnice, što pomaže korisnicima u praćenju kalorijskog unosa.
- Personalizacija: Omogućuje prilagodbu rezultata prema korisnikovim preferencijama i potrebama, čime se poboljšava korisničko iskustvo.

Korištenjem ovih alata i tehnologija, ova aplikacija pruža sveobuhvatno i integrirano rješenje za praćenje prehrane, unosa tekućine, tjelesne aktivnosti i napretka, čime se korisnicima olakšava postizanje njihovih zdravstvenih ciljeva. [14]

4. IZRADA PROGRAMSKOG RJEŠENJA

U ovom poglavlju detaljno je opisan proces izrade ovog programskog rješenja. Prikazani su koraci od početne faze planiranja, preko dizajna i implementacije, pa sve do testiranja i finalizacije aplikacije. Proces je počeo izradom mockupa aplikacije, koji je ključan za olakšavanje vizualnog dizajna i planiranje izgleda aplikacije. Zatim je opisana arhitektura aplikacije, koja je od velike važnosti za organizaciju koda i održivost projekta. Nadalje, kroz opis implementacije, prikazano je kako su korišteni alati i tehnologije spomenuti u prethodnim poglavljima. Na kraju, prikazano je kako su ispunjeni svi funkcionalni zahtjevi aplikacije i proces testiranja aplikacije.

4.1. Mockup

Prvi korak u izradi programskog rješenja bio je napraviti mockup aplikacije koji je kreiran koristeći vodeći alat za kolaborativni dizajn za izradu smislenih proizvoda - Figmu. [15] Mockup omogućava vizualizaciju korisničkog sučelja prije same implementacije, što olakšava planiranje rasporeda elemenata, boja, tipografije i interakcija unutar aplikacije. Korištenjem mockupa, proces izrade korisničkog sučelja i dizajna aplikacije bio je olakšan i ubrzan. Na slici 4.1. prikazan je mockup aplikacije.

Sl. 4.1. Mockup aplikacije

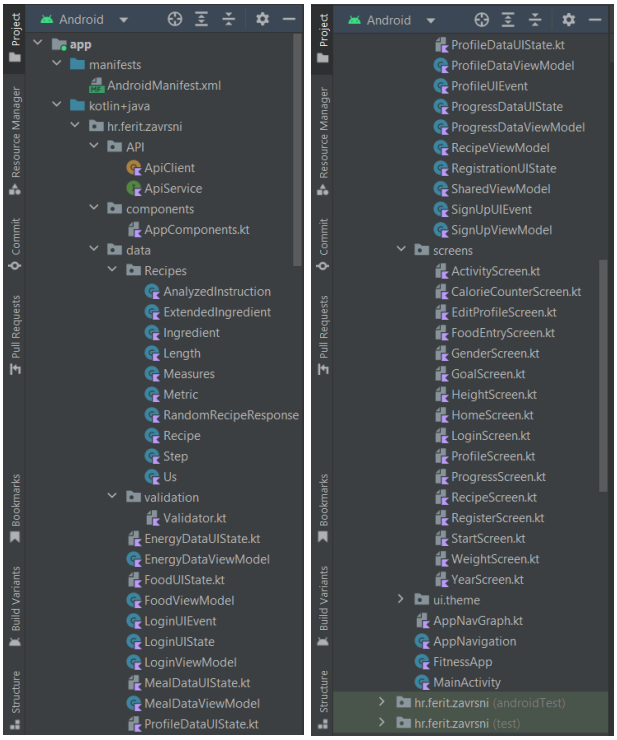
4.2. Arhitektura aplikacije

Arhitektura aplikacije organizirana je i podijeljena u nekoliko direktorija, od kojih svaki ima specifičnu svrhu. Prvi je API direktorij, koji sadrži definirane zahtjeve i pozive prema API-ju korištenom za recepte. Ovaj direktorij omogućava jednostavno upravljanje i održavanje API interakcija. [16]

Direktorij *components* uključuje komponente aplikacije koje su osmišljene da budu višekratno upotrebljive čime se smanjuje obim koda u drugim dijelovima aplikacije. To doprinosi čitljivosti i održivosti koda.

U direktoriju data nalaze se UI stanja za svaki ViewModel, zajedno s odgovarajućim ViewModelima. Ovdje je smješten i *validator* unutar poddirektorija *validation*, koji definira uvjete za validaciju korisničkih podataka prilikom prijave i registracije. Također, unutar ovog direktorija nalazi se i poddirektorij *Recipes*, koji sadrži odgovor API-ja i sve potrebne definirane klase za pravilno funkcioniranje API poziva.

Direktorij *screens* sadrži sve ekrane aplikacije. Na slici 4.2. prikazana je arhitektura aplikacije i organizacija direktorija u Android Studio alatu.



Sl. 4.2.Arhitektura aplikacije

4.3. Implementacija funkcionalnosti

4.3.1. Registracija i prijava korisnika

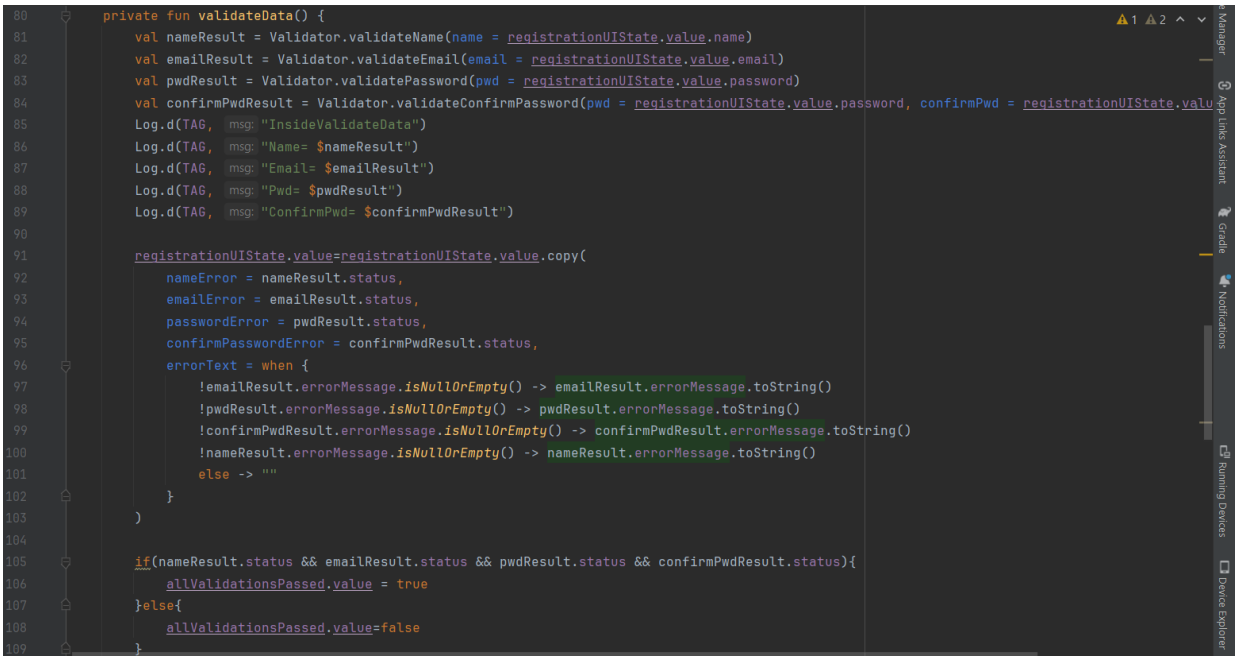
Registracija i prijava korisnika implementirane su pomoću *Firebase Authentication* servisa. Korisnik unosi svoje podatke (ime, e-poštu, zaporku, potvrdu zaporke) putem jednostavnog korisničkog sučelja. *Firebase Authentication* osigurava da su podaci korisnika sigurni i pravilno validirani. Validator korisnicima prikazuje povratne informacije u slučaju greške ili neispravnog unosa. Nakon uspješne registracije, korisniku se osigurava profil u aplikaciji i može se prijaviti svojim podacima koje je unio.

Slika 4.3. prikazuje funkciju koja služi za kreiranje korisnika u Firebase autentikacijskom sustavu koristeći email i zaporku. Ako je kreiranje korisnika uspješno, ažurira se profil korisnika s imenom koristeći *UserProfileChangeRequest*. Nakon ažuriranja profila, dohvaća podatke o profilu i navigira na GenderScreen. U slučaju greške pri kreiranju korisnika, bilježi poruku o grešci u log.

```
116    private fun createUserInFirebase(email: String, password: String, name: String, navController: NavController) {
117        singUpInProgress.value = true
118
119        val firebaseAuth = FirebaseAuth.getInstance()
120        firebaseAuth.createUserWithEmailAndPassword(email, password)
121            .addOnCompleteListener { task ->
122                singUpInProgress.value = false
123                if (task.isSuccessful) {
124                    val user = firebaseAuth.currentUser
125                    val profileUpdates = UserProfileChangeRequest.Builder()
126                        .setDisplayName(name)
127                        .build()
128                    user?.updateProfile(profileUpdates)
129                        ?.addOnCompleteListener { it: Task<Void!>
130                            ProfileDataViewModel().getProfileData()
131                            navController.navigate(route = AppNavigation.GenderScreen.route)
132                        }
133                }
134            }
135            .addOnFailureListener { e ->
136                Log.d(TAG, msg: "Registration failed: ${e.message}")
137            }
138    }
```

Sl. 4.3. Funkcija za kreiranje korisnika u bazi

Slika 4.4. prikazuje funkciju za provjeru valjanosti unesenih podataka u registracijskoj formi. Ukoliko neki od unesenih podataka ne zadovoljava uvjete validatora, ažurira se stanje korisničkog sučelja i ispisuje se odgovarajuća poruka.

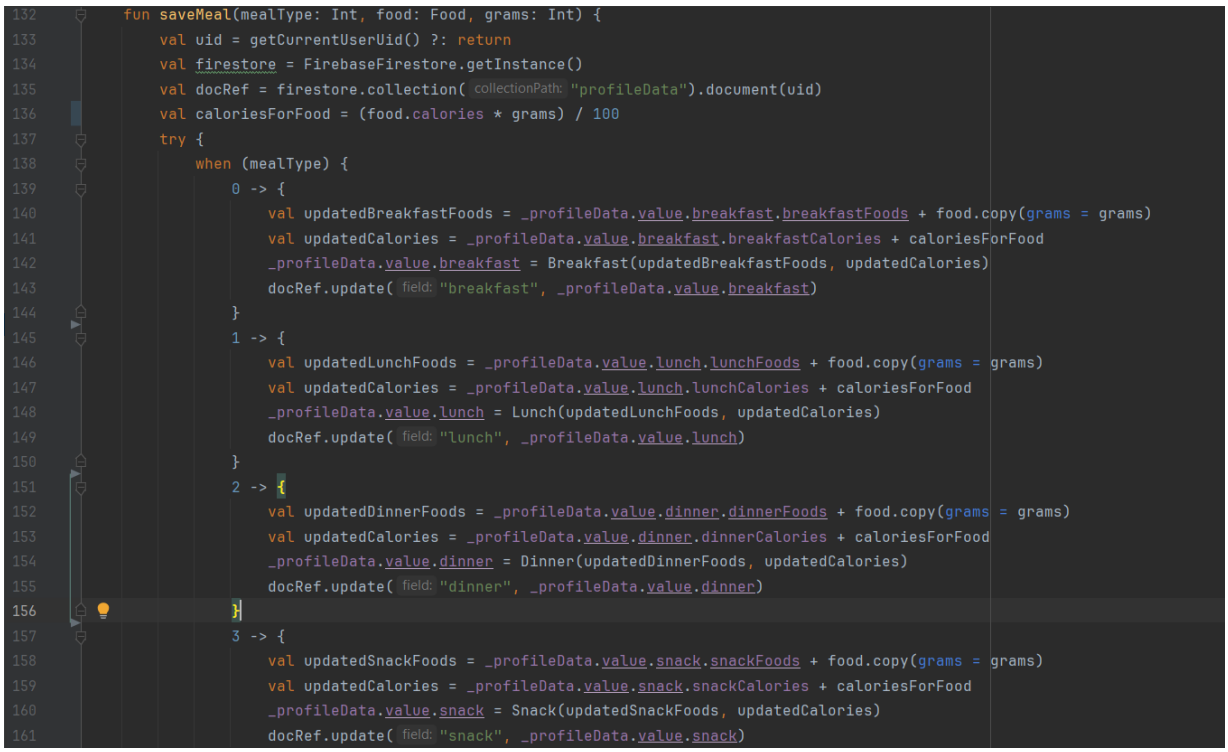


```
private fun validateData() {
    val nameResult = Validator.validateName(name = registrationUIState.value.name)
    val emailResult = Validator.validateEmail(email = registrationUIState.value.email)
    val pwdResult = Validator.validatePassword(pwd = registrationUIState.value.password)
    val confirmPwdResult = Validator.validateConfirmPassword(pwd = registrationUIState.value.password, confirmPwd = registrationUIState.valu
    Log.d(TAG, msg: "InsideValidateData")
    Log.d(TAG, msg: "Name= $nameResult")
    Log.d(TAG, msg: "Email= $emailResult")
    Log.d(TAG, msg: "Pwd= $pwdResult")
    Log.d(TAG, msg: "ConfirmPwd= $confirmPwdResult")

    registrationUIState.value=registrationUIState.value.copy(
        nameError = nameResult.status,
        emailError = emailResult.status,
        passwordError = pwdResult.status,
        confirmPasswordError = confirmPwdResult.status,
        errorText = when {
            !emailResult.errorMessage.isNullOrEmpty() -> emailResult.errorMessage.toString()
            !pwdResult.errorMessage.isNullOrEmpty() -> pwdResult.errorMessage.toString()
            !confirmPwdResult.errorMessage.isNullOrEmpty() -> confirmPwdResult.errorMessage.toString()
            !nameResult.errorMessage.isNullOrEmpty() -> nameResult.errorMessage.toString()
            else -> ""
        }
    )

    if(nameResult.status && emailResult.status && pwdResult.status && confirmPwdResult.status){
        allValidationsPassed.value = true
    }else{
        allValidationsPassed.value=false
    }
```

Sl. 4.4. Funkcija za validaciju podataka pomoću validatora

4.3.2. Praćenje kalorijskog unosa

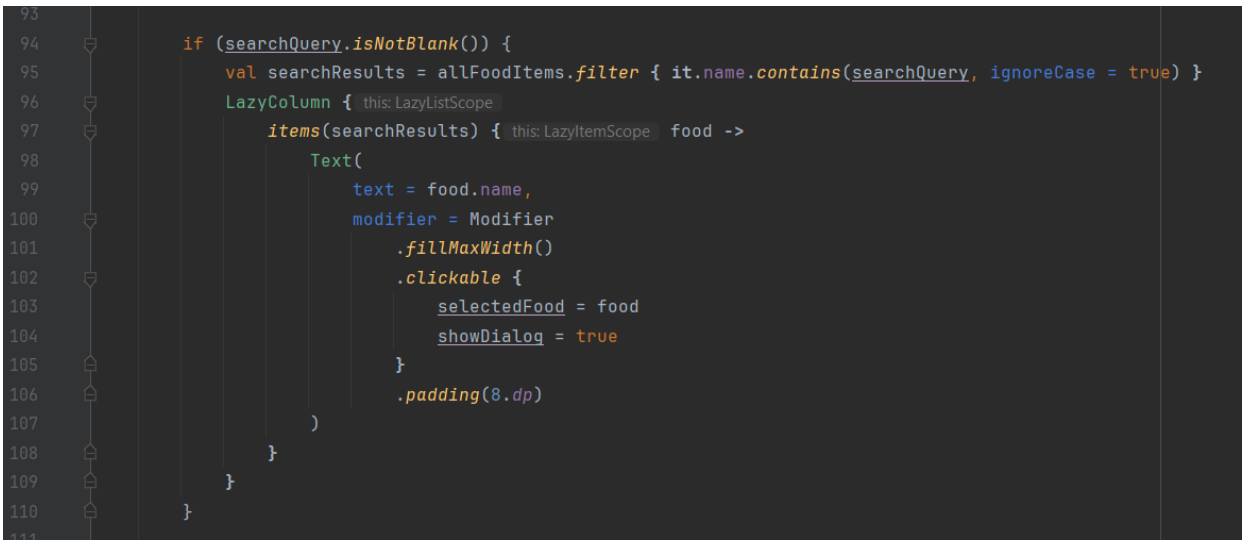
Praćenje kalorijskog unosa realizirano je pomoću Cloud Firestore baze podataka. Korisnici mogu unositi hranu koju konzumiraju tijekom dana, a aplikacija izračunava ukupan kalorijski unos i prikazuje ga u odnosu na korisnikov dnevni cilj. Svi podaci o unosu hrane pohranjuju se u Firestore. Funkcionalnost unosa hrane omogućuje korisnicima pretraživanje baze namirnica i unos količine konzumirane hrane. Pretraga je implementirana tako da se, dok korisnik upisuje željenu namirnicu, iz baze učitavaju namirnice koje odgovaraju unesenim slovima u traci za pretraživanje. Ako određena namirnica nije dostupna u bazi, korisnik ima mogućnost dodavanja novih namirnica unosom njihovih nutritivnih vrijednosti. Slika 4.5. prikazuje funkciju zaslužnu za pohranu obroka u bazu podataka. Prvo se pokušava dobiti UID korisnika, te ukoliko on ne postoji ne poduzimaju se nikakve akcije. Nakon toga, funkcija inicijalizira instancu Firestore baze podataka i referencu na dokument koji sadrži podatke korisničkog profila. Funkcija izračunava kalorije na temelju vrste hrane i unesene količine. Ovisno o vrsti obroka, ažurira podatke za doručak, ručak, večeru ili užinu. Ako dođe do greške tijekom procesa ažuriranja, bilježi grešku u logu. Nakon što se obrok uspješno pohrani, funkcija poziva *getMealData* kako bi osvježila podatke o obrocima.



```
fun saveMeal(mealType: Int, food: Food, grams: Int) {
    val uid = getCurrentUserUid() ?: return
    val firestore = FirebaseFirestore.getInstance()
    val docRef = firestore.collection( collectionPath: "profileData").document(uid)
    val caloriesForFood = (food.calories * grams) / 100
    try {
        when (mealType) {
            0 -> {
                val updatedBreakfastFoods = _profileData.value.breakfast.breakfastFoods + food.copy(grams = grams)
                val updatedCalories = _profileData.value.breakfast.breakfastCalories + caloriesForFood
                _profileData.value.breakfast = Breakfast(updatedBreakfastFoods, updatedCalories)
                docRef.update( field: "breakfast", _profileData.value.breakfast)
            }
            1 -> {
                val updatedLunchFoods = _profileData.value.lunch.lunchFoods + food.copy(grams = grams)
                val updatedCalories = _profileData.value.lunch.lunchCalories + caloriesForFood
                _profileData.value.lunch = Lunch(updatedLunchFoods, updatedCalories)
                docRef.update( field: "lunch", _profileData.value.lunch)
            }
            2 -> {
                val updatedDinnerFoods = _profileData.value.dinner.dinnerFoods + food.copy(grams = grams)
                val updatedCalories = _profileData.value.dinner.dinnerCalories + caloriesForFood
                _profileData.value.dinner = Dinner(updatedDinnerFoods, updatedCalories)
                docRef.update( field: "dinner", _profileData.value.dinner)
            }
            3 -> {
                val updatedSnackFoods = _profileData.value.snack.snackFoods + food.copy(grams = grams)
                val updatedCalories = _profileData.value.snack.snackCalories + caloriesForFood
                _profileData.value.snack = Snack(updatedSnackFoods, updatedCalories)
                docRef.update( field: "snack", _profileData.value.snack)
```

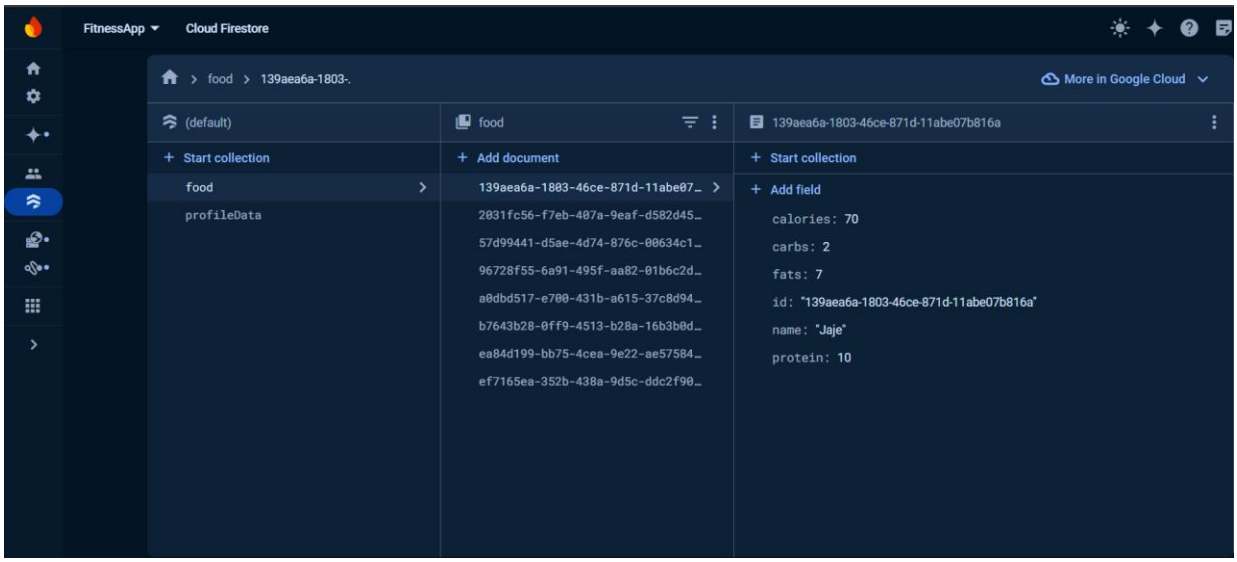
Sl. 4.5. Funkcija za dodavanje hrane u određeni obrok

Na slici 4.6. prikazan je dio koda koji pokazuje rezultate pretraživanja hrane na temelju korisničkog unosa. Prvo se provjerava *searchQuery* i ukoliko nije prazan, nastavlja s filtriranjem hrane. Filtrira sve dostupne stavke hrane (*allFoodItems*) kako bi pronašao one čiji naziv sadrži upit za pretraživanje, ignorirajući velika i mala slova. Za prikazivanje filtriranih rezultata hrane koristi se *LazyColumn*. Kada korisnik klikne na naziv hrane, ta se hrana postavlja kao odabrana (*selectedFood*). Postavlja *showDialog* na *true* kako bi se prikazao dijalog za unos količine odabrane hrane.



```
if (searchQuery.isNotBlank()) {
    val searchResults = allFoodItems.filter { it.name.contains(searchQuery, ignoreCase = true) }
    LazyColumn { this: LazyListScope
        items(searchResults) { this: LazyItemScope food ->
            Text(
                text = food.name,
                modifier = Modifier
                    .fillMaxWidth()
                    .clickable {
                        selectedFood = food
                        showDialog = true
                    }
                    .padding(8.dp)
            )
        }
    }
}
```

Sl. 4.6. Pretraga i prikaz dostupnih namirnica



Sl. 4.7. Baza podataka hrane

Slika 4.7 prikazuje trenutno stanje baze podataka koja pohranjuje namirnice, odnosno prikazuje trenutačno dostupne namirnice u sustavu.

```
26    fun addFood(food: Food) {
27        viewModelScope.launch { this: CoroutineScope
28            try {
29                foodCollection.document(food.id).set(food).await()
30            } catch (e: Exception) {
31                e.printStackTrace()
32            }
33        }
34    }
35
      20py03
36    private fun getAllFoodItems() {
37        viewModelScope.launch { this: CoroutineScope
38            val db = FirebaseFirestore.getInstance()
39            val foodCollection = db.collection( collectionPath: "food")
40             try {
41                val querySnapshot = foodCollection.get().await()
42                _allFoodItems.value = querySnapshot.documents
43                    .map { document ->
44                        document.toObject(Food::class.java)!!
45                    }
46            } catch (e: Exception) {
47                Log.d( tag: "error",  msg: "getAllFoodItems: $e")
48                emptyList<Food>()
49            }
50        }
51    }
52}
```

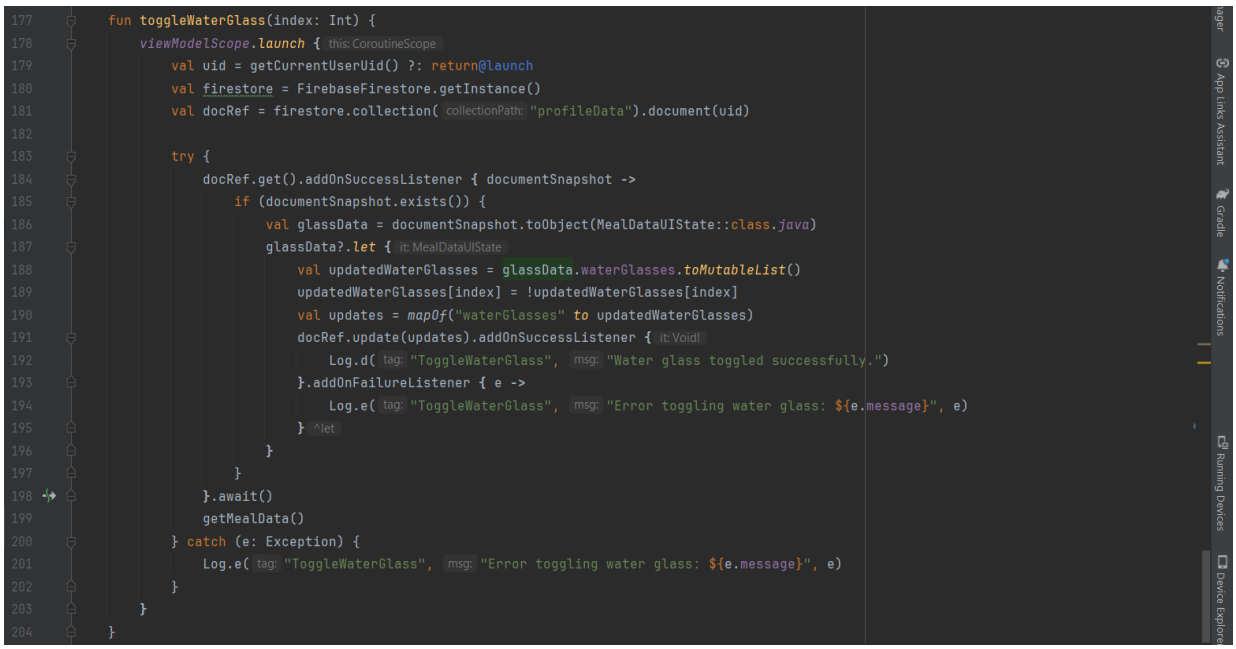
Sl. 4.8. Funkcija za dodavanje novih namirnica i funkcija koja dohvaća svu hranu iz baze [17]

Funkcija *addFood* zaslužena je za dodavanje nove namirnice u kolekciju hrane u Firestore bazi podataka. Funkcija *getAllFoodItems* dohvaća sve stavke hrane iz Firestore baze podataka. Ako je dohvaćanje uspješno, mapira dokumente na objekte tipa *Food* i postavlja vrijednost

_allFoodItems. U suprotnom, bilježi poruku o grešci u log i vraća praznu listu. Navedene funkcije prikazane su na slici 4.8.

4.3.3. Praćenje unosa tekućine

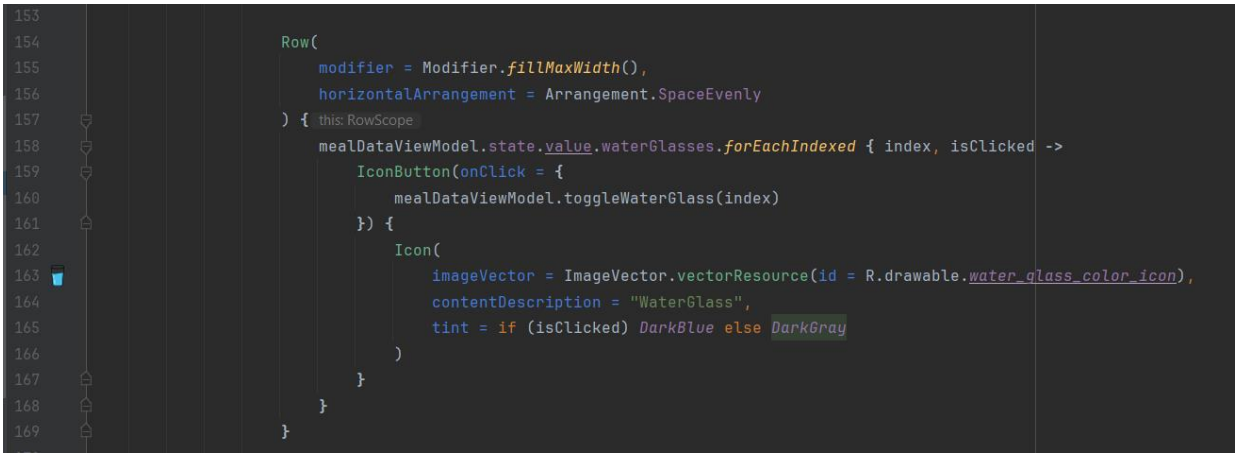
Praćenje unosa tekućine je implementirano pomoću jednostavnog sučelja koje prikazuje sedam čaša vode u jednom redu. Klikom na čašu mijenja se njeno stanje i korisnik bilježi unos tekućine, a aplikacija prati ukupni unos tijekom dana. Ukoliko korisnik ne ispuni svoj unos tekućine za taj dan postavljene su notifikacije kako bi podsjetile korisnika na potrebu za unosom tekućine. Funkcija na slici 4.9. mijenja status čaše vode (popijena/nepopijena) za korisnika u Firestore bazi podataka. Pokušava dohvatiti trenutni dokument korisničkog profila iz Firestore-a. Ako dokument postoji, pretvara ga u objekt tipa *MealDataUIState*. Ako je dokument uspješno pretvoren, mijenja status čaše vode na danom indeksu u popisu *waterGlasses*. Zatim ažurira dokument korisničkog



Sl. 4.9. Funkcija koja mijenja stanje čaše

profila u Firestore-u s novim popisom. Bilježi uspješno ažuriranje u logu ili grešku u slučaju neuspjeha. Nakon ažuriranja, poziva *getMealData* kako bi osvježila podatke o obrocima.

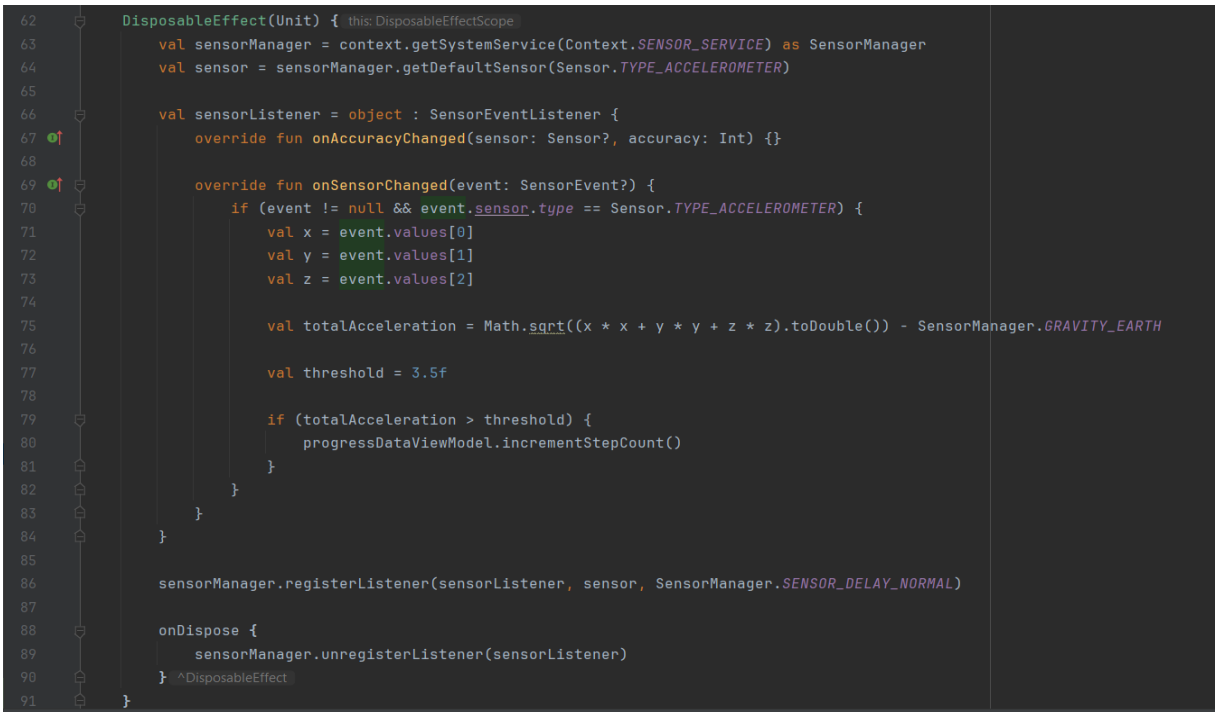
Slika 4.10. prikazuje dio koda koji predstavlja red ikona čaša vode na početnom ekranu koje se dinamički prikazuju ovisno o stanju *waterGlasses*. Za svaku čašu vode prikazuje *IconButton*, koji reagira na klik korisnika. Klik na ikonu poziva funkciju *toggleWaterGlass*, proslijeđujući indeks čaše vode. Kada je *isClicked* jednako *true*, ikona će biti obojana u tamnoplavu boju, inače će biti siva. Boja se kontrolira pomoću *tint* atributa u *Icon* komponenti.



Sl. 4.10. Izgled čaša na ekranu

4.3.4. Praćenje tjelesne aktivnosti kroz dan

Praćenje tjelesne aktivnosti koristi senzore uređaja za brojanje koraka. Na promjenu akceleracije uređaja iznad određenog praga u smjeru x,y,z poziva se funkcija koja povećava broj koraka. Pomoću tih podataka aplikacija prati korisnikovu dnevnu aktivnost. Podaci se ažuriraju u stvarnom vremenu i pohranjuju u Firestore. Slika 4.11. prikazuje dio koda koji koristi *DisposableEffect* kako bi integrirao upravljanje senzorom ubrzanja (eng. *Accelerometer*) u aplikaciji. Dobiva se *SensorManager* i referenca na akcelerometar. Stvara se novi *SensorEventListener* koji sluša promjene na senzoru ubrzanja. Svaki put kad senzor ubrzanja emitira novi događaj aktivira se *onSensorChanged*. Dobivaju se vrijednosti ubrzanja po osima, izračunava se ukupno ubrzanje koristeći Euklidsku normu i postavlja se prag na 3.5f. Ako je ukupno ubrzanje veće od praga, poziva se funkcija koja povećava broj koraka. Brzinu osvježavanja senzora određuje *SENSOR_DELAY_NORMAL*. Kada se komponenta ili ekran uklone ili se više ne koristi, poziva se *onDispose*. Odjavi se *sensorListener* iz *sensorManager* kako bi se oslobodili resursi senzora.



```
DisposableEffect(Unit) {
    val sensorManager = context.getSystemService(Context.SENSOR_SERVICE) as SensorManager
    val sensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)

    val sensorListener = object : SensorEventListener {
        override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {}

        override fun onSensorChanged(event: SensorEvent?) {
            if (event != null && event.sensor.type == Sensor.TYPE_ACCELEROMETER) {
                val x = event.values[0]
                val y = event.values[1]
                val z = event.values[2]

                val totalAcceleration = Math.sqrt((x * x + y * y + z * z).toDouble()) - SensorManager.GRAVITY_EARTH

                val threshold = 3.5f

                if (totalAcceleration > threshold) {
                    progressDataViewModel.incrementStepCount()
                }
            }
        }
    }

    sensorManager.registerListener(sensorListener, sensor, SensorManager.SENSOR_DELAY_NORMAL)

    onDispose {
        sensorManager.unregisterListener(sensorListener)
    }
}
```

Sl. 4.11. Praćenje kretanja uređaja [18]

4.3.5. Praćenje napretka

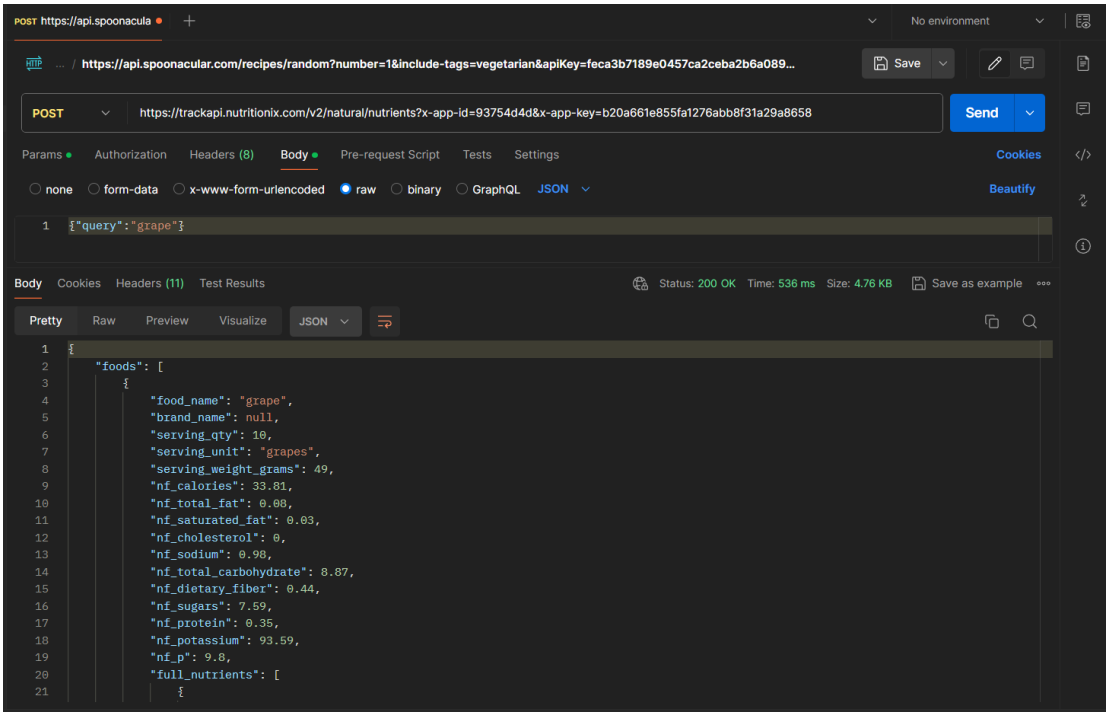
Korisnici mogu unositi svoju početnu, trenutnu i ciljanu težinu kako bi pratili napredak. Osim toga, mogu unositi bilješke o svojim osjećajima i načinima za bolji napredak kako bi im cijeli proces bio lakši. Ovi podaci se također pohranjuju u Firestore na gumb i prikazuju korisnicima. Slika 4.12. prikazuje funkcije za spremanje težina i bilješki u bazu podataka.

```
private suspend fun saveNotesToFirestore(uid: String, note1: String, note2: String) {
    val db = FirebaseFirestore.getInstance()
    val notes = hashMapOf(
        "note1" to note1,
        "note2" to note2
    )

    try {
        db.collection( collectionPath: "profileData").document(uid).update(notes as Map<String, Any>).await()
        Log.d( tag: "success",  msg: "Notes updated successfully.")
    } catch (e: FirebaseFirestoreException) {
        Log.d( tag: "error",  msg: "saveNotesToFirestore: $e")
    }
}
20py03
private suspend fun saveWeightsToFirestore(uid: String, beforeWeight: String, currentWeight: String, afterWeight: String) {
    val db = FirebaseFirestore.getInstance()
    val weights = hashMapOf(
        "beforeWeight" to beforeWeight,
        "currentWeight" to currentWeight,
        "afterWeight" to afterWeight
    )

    try {
        db.collection( collectionPath: "profileData").document(uid).update(weights as Map<String, Any>).await()
        Log.d( tag: "success",  msg: "Weights updated successfully.")
    } catch (e: FirebaseFirestoreException) {
        Log.d( tag: "error",  msg: "saveWeightsToFirestore: $e")
    }
}
```

Sl. 4.12. Funkcije za spremanje težine i osjećaja

4.3.6. Kutak s receptima

Kutak s receptima koristi Spoonacular API za generiranje inspirativnih ideja za kuhanje. Svaki put kada korisnik otvori ovaj dio aplikacije, nasumično se prikazuje deset novih recepata. Za realizaciju ovog dijela aplikacije korišten je Postman, koji je značajno olakšao testiranje API-ja i organizaciju potrebnih direktorija na temelju odgovora. Na slici 4.13. prikazan je odgovor koji je dobiven nakon slanja zahtjeva za Spoonacular API.



Sl. 4.13. Odgovor postmana

Slika 4.14. prikazuje objekt ApiClient koji služi za kreiranje Retrofit instance za implementaciju ApiService-a. [19]

```
package hr.ferit.zavrsni.API

import ...

20py03
object ApiClient {
    private const val BASE_URL = "https://api.spoonacular.com/"

    val retrofit: Retrofit = Retrofit.Builder() Retrofit.Builder
        .baseUrl(BASE_URL) Retrofit.Builder
        .addConverterFactory(GsonConverterFactory.create())
        .build()

}
```

Sl. 4.14. ApiClient

ApiService definira kako će se izvršavati pozivi API-ja. Slika 4.15. prikazuje sučelje ApiService koje je Retrofit sučelje.

```
1   package hr.ferit.zavrsni.API
2
3   import ...
6
    20py03
7   interface ApiService {
        20py03
8       @GET("recipes/random")
9       suspend fun getRandomRecipes(
10          @Query("apiKey") apiKey: String,
11          @Query("number") number: Int,
12      ): RandomRecipeResponse
13
14  }
```

Sl. 4.15. ApiService

Na slici 4.16. prikazan je RecipeScreen koji je strukturiran pomoću komponente RecipeList. Komponenta RecipeList određuje način prikazivanja recepata na ekranu.

```
21  @Composable
22  fun RecipeScreen(navController: NavController) {
23      val recipeViewModel: RecipeViewModel = viewModel()
24      val recipesState = recipeViewModel.recipes.collectAsState()
25
26      LaunchedEffect(Unit) { this: CoroutineScope
27          recipeViewModel.fetchRecipes( apiKey: "feca3b7189e0457ca2ceba2b6a089ee6")
28      }
29
30      Column(
31          modifier = Modifier
32              .fillMaxSize()
33              .background(color = White)
34              .padding(16.dp)
35      ) { this: ColumnScope
36
37          RecipeList(recipesState.value, recipeViewModel)
38      }
39
40      Box(
41          modifier = Modifier
42              .fillMaxSize(),
43          contentAlignment = Alignment.BottomCenter
44      ) { this: BoxScope
45          Footer(navController)
46      }
47  }
48
```

Sl. 4.16. RecipeScreen

4.4. Testiranje i otklanjanje grešaka

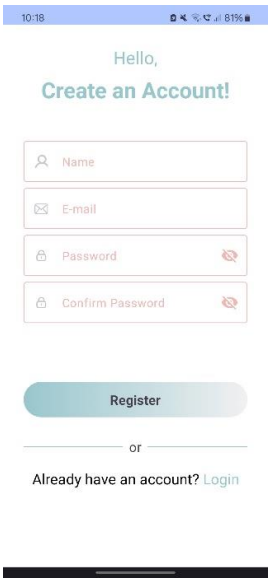
Testiranje i otklanjanje pogrešaka su ključni aspekti razvoja bilo koje aplikacije, osiguravajući da aplikacija radi ispravno, pruža dobro korisničko iskustvo, te je sigurna i stabilna. Način na koji se testirala aplikacija tijekom razvoja je korištenje emulatora koji pokrene aplikaciju na mobilnom uređaju, gdje su testirani različiti scenariji korištenja. Testiranje se provodilo nakon svake nove implementirane funkcionalnosti kako bi se osiguralo da svi dijelovi aplikacije rade ispravno i da nema nepredviđenih problema. U slučaju otklanjanja pogrešaka, koristili su se prijelomne točke i čitanje logcata (alat u Android operativnom sustavu koji bilježi i prikazuje poruke sustava i aplikacija u stvarnom vremenu za svrhe otklanjanja pogrešaka). Prijelomne točke su omogućile zaustavljanje izvođenja koda na određenim linijama kako bi se mogao detaljno pregledati trenutni stanje varijabli i tijek izvođenja. Logcat je pružao detaljne zapise o izvođenju aplikacije, uključujući informacije o greškama, upozorenjima i informacijama o radu aplikacije.

5. PREGLED RADA APLIKACIJE

U ovom poglavlju definirani su funkcionalni i nefunkcionalni zahtjevi za programsko rješenje. Funkcionalni zahtjevi definiraju specifične funkcionalnosti koje aplikacija mora imati kako bi ispunila svoje ciljeve, dok nefunkcionalni zahtjevi definiraju uvjete pod kojima aplikacija mora funkcionirati kako bi pružila optimalno korisničko iskustvo. Cilj je osigurati da aplikacija zadovolji potrebe korisnika i pruži intuitivno, pouzdano i efikasno iskustvo korištenja. U funkcionalne zahtjeve, koji će biti opisani u ovom poglavlju, ubraja se: registracija i prijava korisnika, praćenje kalorijskog unosa i unos hrane, praćenje unosa tekućine, praćenje tjelesne aktivnosti kroz dan, praćenje napretka te kutak s receptima. U nefunkcionalne zahtjeve, koji će biti samo spomenuti, ubraja se: sigurnost, performanse, pouzdanost, korisničko iskustvo.

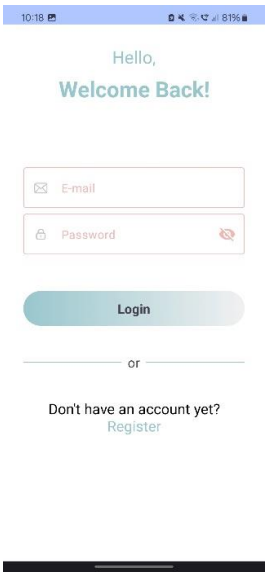
5.1. Registracija i prijava korisnika

Registracija i prijava korisnika ima značajnu ulogu u korištenju aplikacije. Registracijom, odnosno kreiranjem korisničkog računa, korisnicima se omogućava pristup svim funkcionalnostima aplikacije. Prilikom registracije, korisnici unose ime, e-poštu, zaporku i potvrdu zaporke, a nakon toga odgovaraju na nekoliko pitanja o svom trenutnom fizičkom stanju (spol, visina, težina, dob, razina aktivnosti i cilj koji žele postići). Na temelju tih informacija kreira se korisnikov profil u aplikaciji s odgovarajućim preporukama za energetski unos kroz dan. [6] Korisnik podatke unesene pri registraciji može mijenjati kasnije na profilu. Tijekom registracije provjeravaju se uneseni podaci, poput ispravnosti formata e-pošte i duljine zaporke, kako bi se osigurala sigurnost i integritet korisničkih računa. Prilikom prijave, postojeći korisnici mogu se prijaviti pomoću e-pošte i zaporke unesenim pri registraciji, čime im se omogućuje pristup njihovom korisničkom računu i svim spremljenim podacima. Slika 5.1. prikazuje izgled ekrana za registraciju korisnika. Ukoliko korisnik već ima račun, može kliknuti plavi tekst Login koji ga vodi na ekran za prijavu.



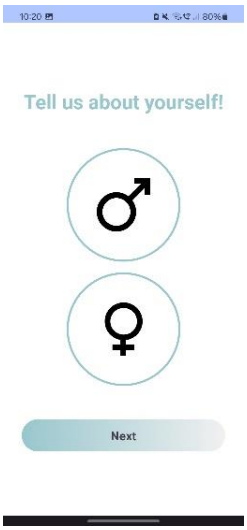
Sl. 5.1. Izgled ekrana za registraciju

Ukoliko korisnik nema račun, ima opciju kliknuti plavi tekst Register pri dnu ekrana koji vodi na ekran za registraciju. Slika 5.2. prikazuje izgled ekrana pri prijavi korisnika.



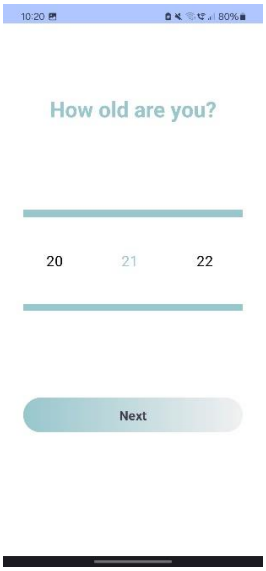
Sl. 5.2. Izgled ekrana za prijavu

Na ekranu za odabir spola nalaze se dvije ikone u krugovima koji na klik mijenjaju boju predstavljaći tako korisnikov odabir. Slika 5.3. prikazuje ekran na kojem korisnik bira spol.



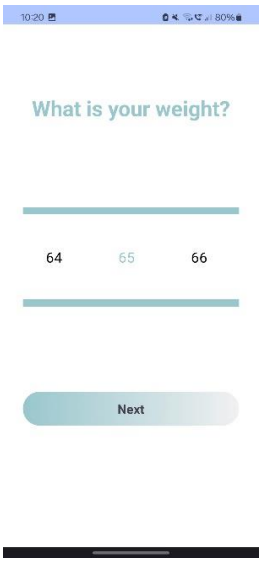
Sl. 5.3. Izgled ekrana za odabir spola

Sučelje za odabir godina uključuje interaktivni birač brojeva koji korisnik može pomicati. Kada korisnik dođe do svojih godina, treba kliknuti na odgovarajući broj. Na klik broj postaje zelen što označava korisnikov odabir. Slika 5.4. prikazuje kako igleda sučelje za odabir godina korisnika.



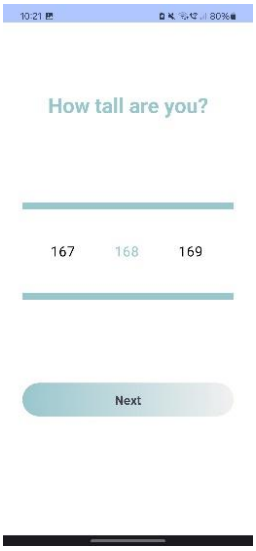
Sl. 5.4. Izgled ekrana za odabir godina

Slika 5.5. prikazuje ekran za odabir mase korisnika. Masa korisnika se odabire na isti način kao i godine, pomoću interaktivnog birača te klika na određeni broj.



Sl. 5.5. Izgled ekrana za odabir težine

Visina se bira na isti način kao masa korisnika i godine. Na slici 5.6. prikazan je izgled ekrana za odabir visine korisnika.



Sl. 5.6. Izgled ekrana za odabir visine

Na slici 5.7. prikazan je ekran za odabir razine aktivnosti korisnika. Na ekranu je postavljeno pet opcija i korisnik može odabrati samo jednu. Na klik se boja mijenja u sivu što označava korisnikov odabir.

Sl. 5.7. Izgled ekrana za odabir razine aktivnosti

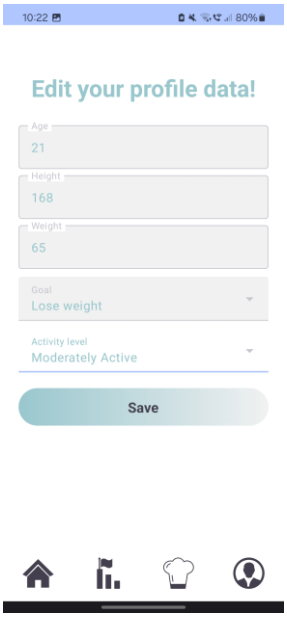
Korisnik bira cilj na isti način kao i razinu aktivnosti. Slika 5.8. prikazuje ekran za odabir korisnikovog cilja.

Sl. 5.8. Izgled ekrana za odabir cilja

Sučelje profila korisnika prikazano je na slici 5.9. koje sadrži izračune TDEE (eng. *Total Daily Energy Expenditure*) i preporučeni unos makronutrijenata na temelju unesenih podataka. Sučelje prikazuje sve relevantne informacije o korisniku, uključujući izračunate vrijednosti, te nudi opcije za odjavu i uređivanje podataka profila.

Sl. 5.9. Izgled korisnikovog profila nakon registracije

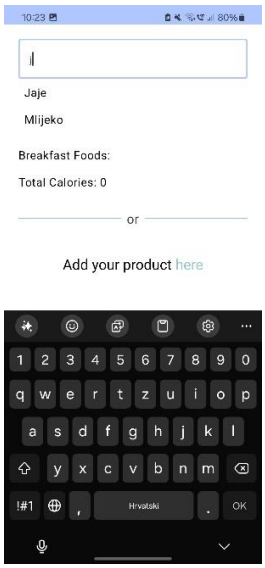
Korisniku je omogućeno mijenjanje informacija na profilu kao što su godine, visina, težina, cilj i razina aktivnosti. Slika 5.10. prikazuje izgled ekrana za uređivanje korisničkih podataka, gdje su prikazani trenutni podaci profila korisnika u poljima za unos.



Sl. 5.10. Izgled ekrana za uređivanje korisnikovih podataka

5.2. Praćenje kalorijskog unosa

Registrirani i prijavljeni korisnici imaju mogućnost praćenja vlastitog kalorijskog cilja tijekom dana. Kalorijski cilj za svakog korisnika izračunava se na temelju osobnih parametara i ciljeva koje žele postići. Praćenje kalorijskog unosa vrši se pretraživanjem namirnica u bazi podataka aplikacije, odabirom željene namirnice i unosom količine u gramima. Na temelju navedenog postupka dodavanja se izračunava ukupan kalorijski unos za taj obrok. Ako određena namirnica ne postoji u bazi podataka, korisnicima je omogućeno dodavanje novih namirnica unosom nutritivnih vrijednosti na sto grama. Na početnom ekranu prikazuju se kalorijske vrijednosti svih obroka zasebno i prikazuje se ukupna unesena kalorijska vrijednost u odnosu na korisnikov kalorijski cilj. Slika 5.11. prikazuje izgled ekrana za unos namirnica, gdje korisnik može pretraživati namirnice koje želi dodati u odabrani obrok. Sučelje omogućuje pretraživanje namirnica koje odgovaraju unesenim slovima, prikazujući korisniku relevantne rezultate za odabir.

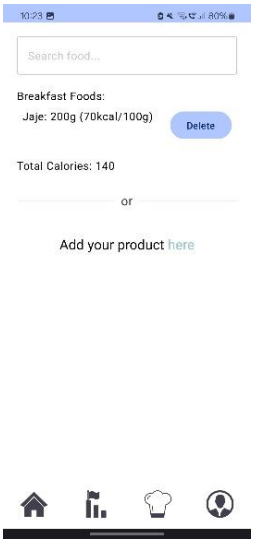


Sl. 5.11. Izgled ekrana prilikom pretraživanja namirnica

Na slici 5.12. prikazan je ekran s dijalogom za odabir količine namirnice. Nakon što korisnik pretraži i odabere željenu namirnicu, otvara se dijalog koji omogućuje korisniku da odabere željenu količinu te namirnice. Na temelju odabrane količine, izračunavaju se kalorije za tu namirnicu i zbrajaju se s ostalim kalorijama u odabranom obroku.

Sl. 5.12. Izgled ekrana pri odabiru količine namirnice

Kada korisnik doda namirnicu, na ekranu je prikazana namirnica zajedno s unesenom količinom, uz dodani gumb za brisanje. Niže na ekranu nalazi se ukupan izračun kalorija za taj obrok. Slika 5.13. prikazuje izgled ekrana nakon pretrage i unosa količine odabrane namirnice.

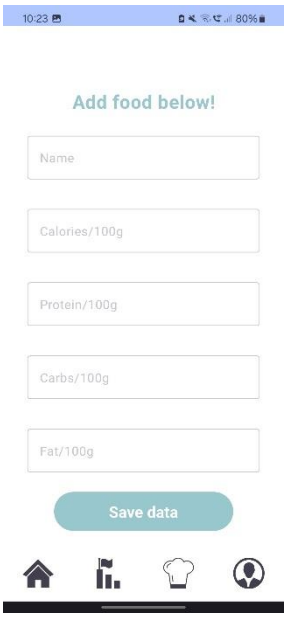


Sl. 5.13. Izgled ekrana nakon dodavanja namirnice

Na početnom ekranu aplikacije nalazi se izračunati dnevni unos kalorija korisnika, traka za praćenje napretka kalorija, informacije o makronutrijentima te praćenje unosa vode u čašama. Ispod ovih informacija nalaze se četiri kvadrata koje simboliziraju različite obroke. Svaki kvadrat prikazuje naziv obroka zajedno s brojem kalorija koje taj obrok sadrži. Slika 5.14. prikazuje sučelje početnog ekrana aplikacije.

Sl. 5.14. Izgled početnog ekrana sa izračunatim kalorijskim vrijednostima

Slika 5.15. prikazuje ekran za unos nove namirnice, gdje korisnik može dodati novu namirnicu unoseći njeno ime i nutritivne vrijednosti na sto grama. Nakon što korisnik unese potrebne informacije, pritiskom na gumb „*Save data*“ namirnica se sprema u bazu podataka. Nakon toga korisnik može pretražiti tu namirnicu i dodati je u svoje obroke.



Sl. 5.15. Izgled ekrana za dodavanje nove namirnice

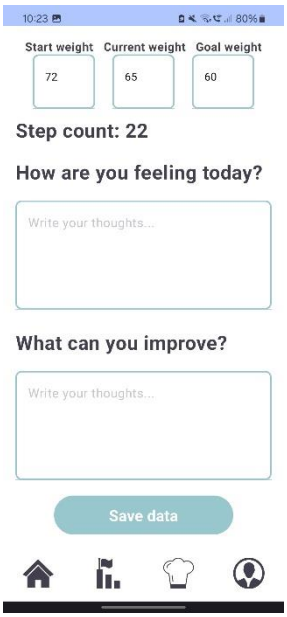
5.3. Praćenje unosa tekućine

Aplikacija omogućuje praćenje unosa tekućine tijekom dana. Na ekranu se prikazuju čaše vode koje mijenjaju boju na klik, što označava da je određena količina vode popijena. U slučaju nedovoljnog unosa tekućine, korisnicima se šalju notifikacije kako bi im se pomoglo u dostizanju dnevnog cilja unosa tekućine. Slika 5.16. prikazuje izgled ekrana nakon što je korisnik unio određenu količinu tekućine.

Sl. 5.16. Izgled ekrana kada se popije određena količina tekućine

5.4. Praćenje tjelesne aktivnosti kroz dan

Aplikacija prati korisnikovu tjelesnu aktivnost brojanjem koraka postignutih tijekom dana. Brojač koraka implementiran je pomoću senzora za aktivnost, što omogućava precizno praćenje korisnikovog kretanja. Na slici 5.17. prikazan je izgled ekrana kada je korisnik prešao određeni broj koraka.



Sl. 5.17. Izgled ekrana za praćenje aktivnosti korisnika

5.5. Praćenje napretka

Korisnici imaju mogućnost praćenja napretka u težini unosom početne, trenutne i željene težine. Također je implementirana funkcionalnost praćenja emocionalnog napretka putem unosa bilješki i korisnikovih osjećaja u određenim trenucima. Postoji i polje u koje korisnici mogu upisivati svoja razmišljanja o načinima unapređenja svog procesa i napretka. Slika 5.18. prikazuje izgled ekrana za praćenje napretka. Na ekranu se nalaze tri polja za unos težine (početna, trenutna i ciljana), koja omogućuju korisniku praćenje promjena u težini i napretka prema postavljenom cilju. Ispod ovih polja nalaze se još dva polja za unos. Jedno polje je namijenjeno unosu i praćenju emocionalnog stanja tijekom procesa, dok je drugo polje predviđeno za unos prijedloga i razmišljanja o metodama koje korisnik smatra korisnima za daljnji napredak i poboljšanje rezultata.

Sl. 5.18. Izgled ekrana za praćenje napretka

5.6. Kutak s receptima

Aplikacija nudi kutak s receptima koji služe kao inspiracija korisnicima u kuhanju. Na ekranu s receptima generira se nasumično deset recepata svaki put kada korisnik otvori ekran ili pri osvježavanju ekrana, omogućujući korisnicima da svaki put otkriju nove ideje za pripremu obroka. Slika 5.19. prikazuje izgled ekrana sa receptima.

Sl. 5.19. Izgled ekrana sa receptima

6. ZAKLJUČAK

Ovaj završni rad je posvećen razvoju Android aplikacije za praćenje unosa kalorija i tjelesnih aktivnosti, čiji je cilj pomoći korisnicima u postizanju njihovih fitnes i zdravstvenih ciljeva. Aplikacija uspješno integrira različite funkcionalnosti kao što su praćenje kalorija, pregled recepata, prijava i registracija korisnika, praćenje unosa tekućine te mogućnost ručnog dodavanja hrane te praćenje napretka i aktivnosti. Implementacija ovih funkcionalnosti predstavlja značajan korak u stvaranju sveobuhvatnog alata za podršku korisnicima u njihovom fitnes putovanju. Ova aplikacija se ističe svojom funkcionalnošću i jednostavnošću korištenja, pružajući korisnicima intuitivan način za praćenje i analizu njihovog dnevnog zdravstvenog i prehrambenog režima.

U procesu razvoja, izazov je bio diferencirati se od već prisutnih rješenja na tržištu, koje često uključuju funkcionalnosti koje se moraju nadoplatiti kako bi se koristile. Ova aplikacija nastoji ponuditi sveobuhvatan i koristan alat bez dodatnih troškova za korisnike, čime se ističe u konkurentnom okruženju aplikacija za zdravlje i fitnes.

Kako bi se dalje unaprijedila funkcionalnost aplikacije, može se razmotriti dodavanje naprednih značajki poput skeniranja proizvoda putem barkoda za brži unos hrane ili integraciju dodatnih senzora za preciznije praćenje aktivnosti i brojanje koraka tokom dana.

Upotreba alata kao što su Android Studio, Kotlin programski jezik, Jetpack biblioteke, Firebase i Spoonacular API bila je ključna za uspješan razvoj aplikacije. Ovi alati su omogućili brzo prototipiranje, efikasno upravljanje podacima te integraciju naprednih funkcionalnosti, pružajući tako bogato iskustvo u učenju i primjeni novih tehnologija u praksi.

Android aplikacija za praćenje unosa kalorija i tjelesnih aktivnosti predstavlja korisnički orijentiranu aplikaciju koja će se nastaviti kontinuirano razvijati kako bi zadovoljila potrebe korisnika i pružila im sveobuhvatan alat za upravljanje njihovim zdravljem i fitnessom.

LITERATURA

[1] Google play, Lifesum, 2024, dostupno na: https://play.google.com/store/apps/details?id=com.sillens.shapeupclub&hl=hr, [17.06.2024.]

[2] Google play, MyFitnessPal, 2024, dostupno na: https://play.google.com/store/apps/details?id=com.myfitnesspal.android&hl=hr, [17.06.2024.]

[3] Google play, Calorie Counter & Food tracker, 2024, dostupno na: https://play.google.com/store/apps/details?id=nutrition.healthy.diet.dietplan.caloriecounter&hl=en_US, [17.06.2024.]

[4] Google play, Fitia, 2024, dostupno na: https://play.google.com/store/apps/details?id=com.nutrition.technologies.Fitia&hl=hr, [17.06.2024.]

[5] Google play, Yazio, 2024, dostupno na: https://play.google.com/store/apps/details?id=com.yazio.android&hl=en, [17.06.2024.]

[6] A. Bean, The Complete Guide to Sports Nutrition, A&C Black, London, 2009, [20.05.2024.]

[7] Predavanje 1 iz kolegija Razvoj mobilnih aplikacija, Mobilne platforme [Moodle], dostupno na: https://moodle.srce.hr/2023-2024/course/view.php?id=163522#section-3, [19.06.2024.]

[8] Predložak laboratorijskih vježbi kolegija Razvoj mobilnih aplikacija, Uvod u Objektno orjentirano programiranje i kotlin [Moodle], dostupno na: https://moodle.srce.hr/2023-2024/mod/resource/view.php?id=3919041, [19.06.2024.]

[9] Predložak laboratorijskih vježbi kolegija Razvoj mobilnih aplikacija, JETPACK COMPOSE [Moodle], dostupno na: https://moodle.srce.hr/2023-2024/mod/resource/view.php?id=3928019, [20.06.2024.]

[10] „Preview your UI with Composable previews | Jetpack Compose“, Android Developers, 2024, dostupno na: https://developer.android.com/jetpack/compose/tooling/previews, [20.06.2024.]

[11] "Firebase, Google", Firebase, 2024, dostupno na: https://firebase.google.com/, [21.06.2024.]

[12] "What is Firebase? All secrets unlocked", G. Batschinski, Back4App, 2022, dostupno na: https://blog.back4app.com/firebase/, [21.06.2024.]

[13] „What is postman?“, Postman, 2024, dostupno na: https://www.postman.com/product/what-is-postman/, [17.06.2024.]

[14] „Our mission“, Spoonacular, 2024, dostupno na: https://spoonacular.com/about, [17.06.2024.]

[15] „What is figma?“, Figma, 2024, https://help.figma.com/hc/en-us/articles/14563969806359-What-is-Figma, [20.06.2024.]

[16] „Why you need ViewModels and why you don’t“, Composables – Jetpack Compose components to build your next idea to life blazing fast., 2023, dostupno na:

https://www.composables.com/tutorials/viewmodels-in-jetpack-compose, [20.06.2024.]

[17] „Kotlin coroutines on Android“, Android Developers, 2024, dostupno na: https://developer.android.com/kotlin/coroutines, [21.06.2024.]

[18] “Side-effects in Compose“ , Android Developers, 2024, dostupno na: https://developer.android.com/develop/ui/compose/side-effects, [21.06.2024.]

[19] “Retrofit with Kotlin Coroutine in Android“, GeeksforGeeks, 2022, dostupno na: https://www.geeksforgeeks.org/retrofit-with-kotlin-coroutine-in-android/, [17.06.2024.]

SAŽETAK

Glavni izazov ovog završnog rada bio je razvoj Android aplikacije za praćenje unosa kalorija i tjelesnih aktivnosti, koja korisnicima omogućuje sustavno praćenje fitness ciljeva i zdravstvenih parametara. Cilj je bio uspješno implementirati funkcionalnosti kao što su brojanje kalorija, pregled recepata, prijava i registracija korisnika, praćenje unosa tekućine, samostalno dodavanje hrane te praćenje osobnog napretka. Brojanje kalorija ostvareno je putem pretrage namirnica iz baze podataka i njihovog unosa u gramima, dok je pregled recepata omogućen integracijom Spoonacular API-ja. Prijava i registracija korisnika realizirani su pomoću Firebase Authentication usluge, a praćenje unosa tekućine vizualizirano je kroz interaktivno bojanje čaša vode. Napredak korisnika evidentiran je kroz polja za unos njihovih misli i promjena u tjelesnoj težini. U procesu razvoja, Android Studio i Jetpack Compose su igrali ključnu ulogu pružajući moćne alate za razvoj korisničkog sučelja i upravljanje aplikacijskom logikom. Korištenje Postman alata za testiranje API-ja znatno je olakšalo ispitivanje i integraciju različitih funkcionalnosti. Firebase je poslužio kao pouzdana platforma za pohranu svih podataka. Korištenjem navedenih tehnologija i alata uspješno su ostvareni zadani ciljevi projekta. U budućnosti se planira dalji razvoj i implementacija naprednih funkcionalnosti.

Ključne riječi: Andorid, aplikacija, Firebase, fitnes, kalorije,

ABSTRACT

Android application for monitoring calorie intake and physical activity

The main challenge of this thesis was to develop the Android application for monitoring calorie intake and physical activity, which enables users to systematically track fitness goals and health parameters. The goal was to successfully implement functionalities such as calorie counting, recipe browsing, user login and registration, water intake tracking, manual food entry, and personal progress tracking. Calorie counting was achieved through searching food items in the database and entering them in grams, while recipe browsing was enabled through integration with the Spoonacular API. User login and registration were handled using Firebase Authentication, and water intake tracking was visualized through interactive water glass coloring. User progress was recorded through fields for inputting thoughts and weight changes. During development, Android Studio and Jetpack Compose played a crucial role by providing powerful tools for UI development and application logic management. The use of the Postman tool for API testing significantly facilitated testing and integration of various functionalities. Firebase served as a reliable platform for storing all data. By utilizing these technologies and tools, the project's objectives were successfully achieved. Future plans include further development and implementation of advanced functionalities.

Keywords: Android, application, calories, Firebase, fitness

ŽIVOTOPIS

Jovana Paprić rođena je u 29.01.2003. u Vukovaru. Pohađala je Osnovnu školu Bobota. Nakon završenog osnovnoškolskog obrazovanja, upisala je opću gimnaziju u Vukovaru gdje je maturirala 2021. godine. Iste godine upisala je Fakultet za elektrotehniku, računarstvo i informacijske tehnologije Osijek na kojem studira i danas na smjeru stručni studij računarstvo. Jovana je tijekom obrazovanja u srednjoj školi pohađala školu stranih jezika Linguapax i aktivno se bavila plesom 10 godina. Za vrijeme studija, Jovana je dobila nagradu za uspješnost u studiranju 2024. godine. Jovana je tijekom studija naučila tehnologije: za izradu web apliakcija, mobilnih aplikacija, strojno učenje te programske jezike C, C#, HTML, CSS, Java, Python, PHP, SQL, JavaScript, Kotlin. Također je imala priliku raditi u Blenderu i Figmi (3d modeliranje i dizajn korisničkog sučelja). Pohađala je stručnu praksu u tvrtki Mono na poziciji software developer gdje je iskusila rad u timu na projektu i također stekla puno znanja u području.

_____

Potpis autora