Android aplikacija za usluge povezivanja vlasnika i šetača pasa

Toprek, Filip

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

Stručni prijediplomski studij Računarstvo

ANDROID APLIKACIJA ZA USLUGE POVEZIVANJA VLASNIKA I ŠETAČA PASA

Završni rad

Filip Toprek

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:	Filip Toprek
Studij, smjer:	Stručni prijediplomski studij Računarstvo
Mat. br. pristupnika, god.	AR4885, 27.07.2021.
JMBAG:	0165091088
Mentor:	prof. dr. sc. Krešimir Nenadić
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	doc. dr. sc. Hrvoje Leventić
Član Povjerenstva 1:	prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 2:	doc. dr. sc. Krešimir Romić
Naslov završnog rada:	%naziv_rada%
Znanstvena grana završnog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada:	Datki kratki opis zahtjeva i funkcionalnosti Android aplikacije koja spaja vlasnike pasa i osoba koje mogu pružiti uslugu vođenja pasa u šetnju. Usporediti izrađenu Android aplikaciju s postojećim rješenjima u obliku aplikacija za Android ili druge platforme (web, iOS). Za potrebe aplikacije potrebno je dizajnirati i izraditi bazu podataka te opisati postupak izrade baze. Aplikacije treba podržavati najmanje dva korisnička profila: vlasnik i šetač. Potrebno je opisati postupak izrade aplikacije kao i funkcioniranje izrađene aplikacije. Tema rezervirana: Filip Toprek

Datum ocjene pismenog dijela završnog rada od strane mentora:	09.07.2024.
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:	Filip Toprek
Studij:	Stručni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	AR4885, 27.07.2021.
Turnitin podudaranje [%]:	8

Ovom izjavom izjavljujem da je rad pod nazivom**: Android aplikacija za usluge povezivanja vlasnika i šetača pasa**

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 užurbanom društvu ljudi jedva pronalaze vremena za sebe i svoje bližnje, a samim time čovjekov najbolji prijatelj ostaje zapostavljen. Svake godine u Hrvatskoj, prema [1], napusti se više od 10 tisuća životinja, uglavnom pasa. Prema [2], zdravi odrasli psi trebali bi biti izvedeni u šetnju barem tri puta dnevno, s minimalnim ukupnim trajanjem od jednog sata provedenog vani. Kao što su šetnje korisne za ljubimce, tako su i korisne za ljude. Sve veći broj ljudi provodi svoje vrijeme unutar zatvorenih prostora ili pred ekranima. Takav način života može rezultirati brojnim zdravstvenim problemima, kako fizičkim, tako i mentalnim. Stoga, ciljna skupina ove aplikacije obuhvaća ljude koji vole pse i često provode vrijeme u zatvorenim prostorima, kao i one koji žive užurbani životni stil te imaju ograničeno vrijeme za posvetiti se svojim krznenim prijateljima.

Aplikacija koju ovaj završni rad opisuje nastoji riješiti navedene probleme. Aplikacija omogućuje korisnicima rezervaciju određene vrste šetnje za njihovog kućnog ljubimca. Također aplikacija omogućuje drugom tipu korisnika mogućnost šetanja kućnih ljubimaca.

Motivacija za izradu aplikacije je pružiti uslugu šetanja pasa ljudima koji nemaju dovoljno vremena za to. Aplikacija omogućuje rezervaciju šetnje za određeni datum i vrijeme kao i praćenje šetača u stvarnom vremenu.

U drugom poglavlju završnog rada navedeni su neki od postojećih programskih rješenja navedenog problema. Većina programskih rješenja su u obliku mobilne aplikacije. Treće poglavlje objašnjava korištene tehnologije u izradi programskog rješenja. U četvrtom poglavlju opisan je postupak izrade aplikacije te različite funkcionalnosti aplikacije. U petom poglavlju predstavljene su funkcionalnosti i korištenje same aplikacije.

1.1. Zadatak završnog rada

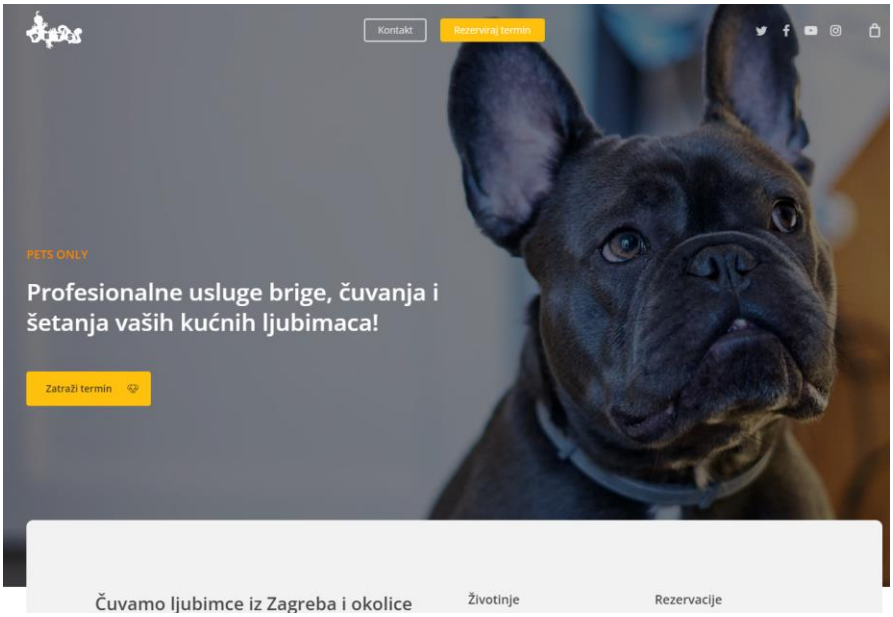
Dati kratki opis zahtjeva i funkcionalnosti Android aplikacije koja spaja vlasnike pasa i osoba koje mogu pružiti uslugu vođenja pasa u šetnju. Usporediti izrađenu Android aplikaciju s postojećim rješenjima u obliku aplikacija za Android ili druge platforme (web, iOS). Za potrebe aplikacije potrebno je dizajnirati i izraditi bazu podataka te opisati postupak izrade baze. Aplikacije treba podržavati najmanje dva korisnička profila: vlasnik i šetač. Potrebno je opisati postupak izrade aplikacije kao i funkcioniranje izrađene aplikacije.

2. POSTOJEĆA PROGRAMSKA RJEŠENJA

U ovom poglavlju je dan pregled nekih od postojećih programskih rješenja. To su najčešće programska rješenja u obliku mobilne aplikacije. Neke od trenutno dostupnih aplikacija koje rješavaju opisani problem ograničene su samo na velike države

2.1. Web aplikacija PetsOnly

PetsOnly nudi usluge šetnje i čuvanja kućnih ljubimaca. Usluge nisu samo fokusirane na pse nego i na mačke, ptice i ostale. Aplikacija je dostupna samo unutar grada Zagreba i okolice. Jedna od loših strana aplikacije je što prilikom otkazivanja rezervacije ne vraćaju kupcu novce. Prilikom korištenja potrebno je rezervirati termin jedan do dva tjedna unaprijed [3].



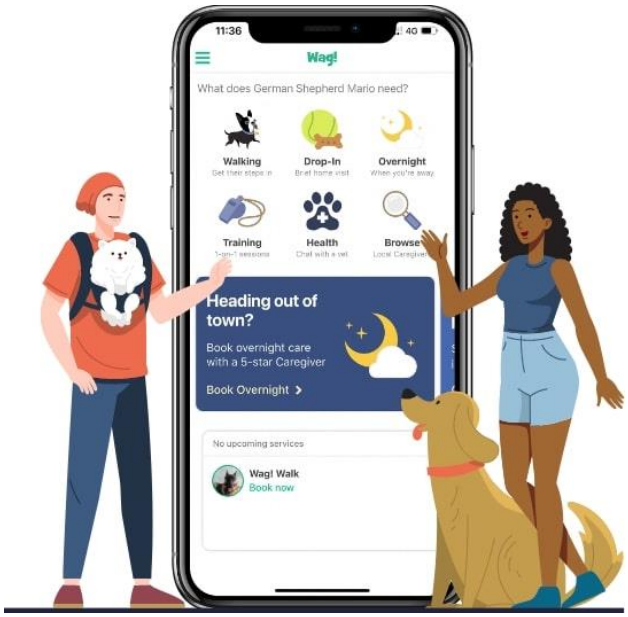
Sl. 2.1. PetsOnly web aplikacija [3]

2.2. Mobilna aplikacija Wag!

Jedna od najpoznatijih aplikacija je Wag, koja je dostupna isključivo korisnicima unutar Sjedinjenih Američkih Država. Wag nudi usluge šetnje pasa, provjere samih ljubimaca, čuvanja i treniranja kućnih ljubimaca [4]. Na slici 2.2 vidljiv je početni zaslon Wag aplikacije.

Sigurnost kućnih ljubimaca temelji se na temeljitim pozadinskim provjerama osoba koje žele pružati usluge putem aplikacije. Te provjere uključuju pregled kaznene evidencije i terorističkih popisa. Osim toga, za registraciju na aplikaciju potrebno je platiti naknadu u iznosu od 60 USD.

Osim Wag aplikacije postoje i druge aplikacije kao što su Rover i Barkly Pets, koje pružaju slične usluge, ali su također ograničene na određene zemlje ili velike gradove.



Sl. 2.2. Wag! mobilna aplikacija [4]

2.3. Mobilna aplikacija Barkly Pets

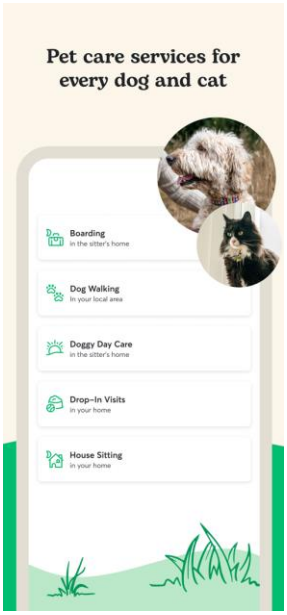
Barkly Pets je usluga za šetnje pasa dostupna samo u određenim velikim gradovima Sjedinjenih Američkih Država. Svaki šetač prolazi kroz temeljit proces koji uključuje sate obuke i provjere što rezultira zajednicom najpouzdanijih stručnjaka usmjerenih na sigurnost. Na slici 2.3 vidljiv je šetač i pas u uniformama Barkly Pets.

Sl. 2.3.Slika šetača i psa usluge Barkly Pets [5]

2.4. Mobilna aplikacija Rover

Ova mobilna aplikacija je vrlo slična prethodno navedenoj. Namijenjena je povezivanju vlasnika pasa s lokalnim pružateljima usluga šetanja, čuvanja i treniranja pasa. Rover također provodi temeljite pozadinske provjere svojih korisnika kako bi osigurali određenu sigurnost usluge.

Rover je široko rasprostranjena aplikacija koja je dostupna u Ujedinjenom Kraljevstvu, Europi i Sjedinjenim Američkim Državama. Prema [6] Rover ima više od 200 tisuća prijavljenih čuvara i šetača pasa. Rover aplikacija vidljiva je na slici 2.4.



Sl. 2.4.Rover mobilna aplikacija [6]

2.5. Mobilna aplikacija Gowalkies

Aplikacija Gowalkies nudi usluge šetnje unutar Ujedinjenog Kraljevstva. Nude praćenje u stvarnom vremenu te video i fotografije ljubimaca u šetnji. Također njihova podrška je dostupna 24/7 putem e-mail usluge. Osim šetnje nude usluge šišanja, treniranja i čuvanja [7].

Sl. 2.5.Gowalkies mobilna aplikacija [7]

3. KORIŠTENE TEHNOLOGIJE

U ovom poglavlju su navedene tehnologije koje su korištene pri izradi aplikacije. Prije nego što se krene s izradom aplikacije, bitno je odabrati odgovarajuće tehnologije iz nekoliko ključnih razloga. Odabir tehnologija prije početka razvoja omogućuje precizno planiranje, bolje performanse, skalabilnost i sigurnost aplikacije.

3.1. Android operacijski sustav

Android je operacijski sustav otvorenog koda koji je razvijen za široku uporabu na različitim uređajima. U početku je Android razvijala tvrtka pod nazivom Android Inc., koju je 2005. godine kupio Google Inc [8]. Temelji se na Linux jezgri, što omogućuje Android sustavu iskorištavanje ključnih sigurnosnih značajki i omogućuje proizvođačima razvoj hardverskih upravljačkih programa za poznatu platformu [9].

Osim što je razvijen za mobilne uređaje, Android sustav našao je primjenu u raznim drugim platformama. Google je razvio sustav za pametne televizore pod nazivom Android TV. Android Auto je još jedan prilagođeni sustav temeljen na Androidu, osmišljen za upotrebu u automobilima. Wear OS je operacijski sustav koji je razvijen za uređaje poput pametnih satova.

Sve verzije Android sustava imenovane su po desertima, a zasluge za ovu konvenciju imenovanja tradicionalno pripadaju voditelju projekta Ryanu Gibsonu [9]. Povijest Androida pokazuje da je mobilni operativni sustav prešao dug put od svojih skromnih početaka. Danas je Android vodeći mobilni operativni sustav u svijetu, s preko 70% udjela na tržištu [9].

3.2. Kotlin

Kotlin je višeplatformski programski jezik visoke razine i opće namjene. U 2019. godini, Google je objavio da Kotlin postaje preferirani jezik za razvoj Android aplikacija. Andrey Breslav, bivši glavni dizajner Kotlina, spomenuo je da je tim odlučio nazvati ga po otoku, baš kao što je programski jezik Java nazvan po indonezijskom otoku [10].

Kotlin je dizajniran kao objektno-orijentirani jezik industrijske snage i „bolji jezik“ od Jave, ali i dalje u potpunosti kompatibilan s Java kôdom, što omogućuje tvrtkama postupnu migraciju s Jave na Kotlin. Znak ; se može koristiti kao opcionalni terminator naredbi; u pravilu je dovoljan novi redak kako bi kompajler zaključio da je naredba završena [10].

3.3. Jetpack Compose

Jetpack Compose je preporučeni alat za stvaranje izvornih korisničkih sučelja na Android platformi. Njegova svrha je olakšati i ubrzati proces razvoja korisničkog sučelja.

Compose je deklarativni alat za oblikovanje korisničkog sučelja, što znači da programer opisuje željeno stanje sučelja, a alat se brine o detaljima implementacije. Jedan od izazova s deklarativnim alatima za oblikovanje korisničkog sučelja je to što ponovno iscrtavanje cijelog zaslona može biti računalno zahtjevno u smislu vremena, računalne snage i potrošnje baterije. Kako bi smanjio taj trošak, Compose pametno odabire koje dijelove korisničkog sučelja treba ponovno iscrtati u bilo kojem trenutku [11]. Jednostavni primjer Compose funkcije je vidljiv na slici 3.1.

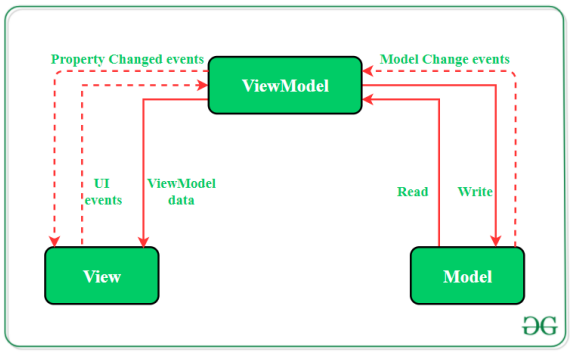
Sl. 3.1. Primjer jednostavne Compose funkcije [11]

3.4. MVVM arhitektura

Arhitektura *Model* - *View* - *ViewModel* (MVVM) je industrijski priznat obrazac programske arhitekture koji nadilazi sve nedostatke dizajnerskih obrazaca *Model-View-Presenter* (MVP) i *Model-View-Controller* (MVC). MVVM predlaže odvajanje logike prikaza od poslovne logike aplikacije. Odvojeni slojevi koda su:

- ***Model*** – zadaća ovog sloja je apstrakcija izvora podataka. *Model* i *ViewModel* surađuju kako bi dohvatili i pohranili podatke,
- ***View*** – ovaj sloj ne sadrži nikakvu logiku aplikacije. Njegova zadaća je informirati *ViewModel* o akcijama korisnika,
- ***ViewModel*** – djeluje kao poveznica između *Model* i *View* sloja. Sadrži poslovnu logiku aplikacije [12].

Shema MVVM arhitekture je vidljiva na slici 3.2.



Sl. 3.2. MVVM shema [12]

3.5. Umetanje ovisnosti (*engl. Dependency injection*)

Kada klasa A koristi neku funkcionalnost klase B, kaže se da klasa A ovisi o klasi B. Prije nego što se mogu koristiti metode drugih klasa, prvo je potrebno stvoriti objekt te klase. Prepuštanje zadatka stvaranja objekta nekom drugom te izravno korištenje ovisnosti naziva se umetanje ovisnosti [13].

Prednosti korištenja umetanja ovisnosti:

1. Pomaže pri testiranju,
2. Smanjuje količinu redundantnog koda,
3. Olakšava proširenje aplikacije,
4. Omogućuje slabo povezivanje, što je važno u programiranju aplikacija. [13]

3.6. Hilt

Hilt je biblioteka za umetanje ovisnosti za Android. Hilt pruža standardiziran način korištenja umetanja ovisnosti u aplikaciji, nudeći spremnike za svaku Android klasu u projektu i automatski upravlja njihovim životnim ciklusima. Hilt je izgrađen na popularnoj biblioteci Dagger što omogućuje bolju skalabilnost, bolje performanse i bolju podršku [14].

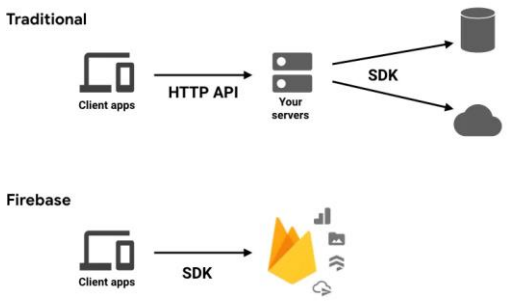
3.7. Firebase

Firebase je skup usluga i platformi za podršku razvoja mobilnih aplikacija koju pruža Google. Proizvodi su smješteni u oblaku. Neke od usluga koje pružaju su: autentikacija, usluge u oblaku, baze podataka, analitika, testiranje, *hosting*.

Kada se kaže "smješteno u oblaku", misli se na proizvode čiji su poslužiteljski (engl. *Backend*, dio softverske aplikacije koji upravlja obradom podataka, poslovnom logikom i komunikacijom s bazama podataka te vanjskim servisima) dijelovi potpuno održavani i upravljani tvrtkom Google. Klijentski SDK-ovi (engl. *Software development kit*, skup alata za razvojne inženjere) koje pruža

Firebase izravno komuniciraju s tim *backend* uslugama, bez potrebe za uspostavljanjem posrednika između aplikacije i usluge. Ako se koristi jedna od Firebase opcija baze podataka, obično se piše kôd za upite bazi podataka u klijentskoj aplikaciji [15].

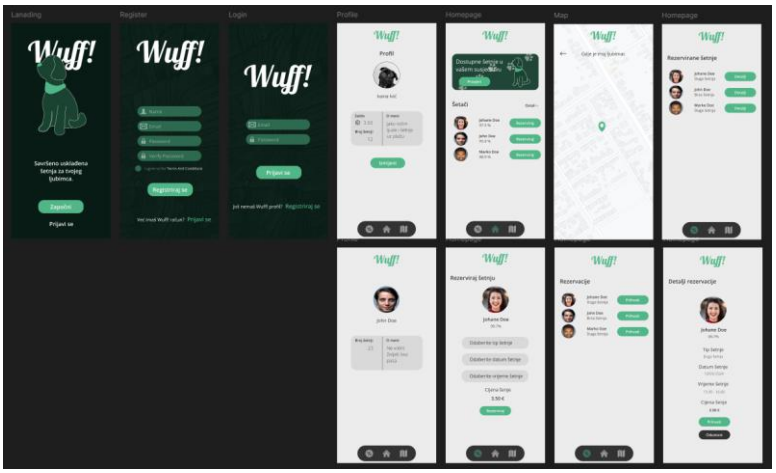
Navedeni način rada je drugačiji od tradicionalnog razvoja aplikacija, gdje se kôd za *frontend* i *backend* piše odvojeno. *Frontend* kôd samo poziva API (engl. *Application programming interface*, sučelje koje omogućuje različitim softverskim aplikacijama da međusobno komuniciraju) krajnje točke, dok *backend* kôd zapravo obavlja posao [15]. Na slici 3.3 vidljiva je komunikacija između aplikacije i Firebase usluga.



Sl. 3.3. Ilustracija komunikacije između aplikacije i Firebase usluge [15]

3.8. Figma

Figma je aplikacija za dizajn sučelja koja se izvršava u pregledniku, ali postoji i desktop verzija za Windows i Mac OS. Figma korisnicima nudi spremanje dizajna u oblaku. Omogućuje suradnju u realnom vremenu, što omogućava istovremeno uređivanje dizajna više članova tima. Najnovije promjene uvijek su odmah dostupne u datoteci i nije potrebno brinuti se oko prijenosa datoteka među članovima tima [16]. Dizajn korisničkog sučelja vidljiv je na slici 3.4.



Sl. 3.4. Dizajn korisničkog sučelja aplikacije

3.9. Stripe

Stripe je pružatelj usluga plaćanja koji omogućuje trgovcima prihvaćanje kreditnih i debitnih kartica te drugih oblika plaćanja. Njegovo rješenje za procesiranje plaćanja, Stripe Payments, najbolje odgovara poslovnim subjektima koji većinu svojih prodaja obavljaju online, budući da su njegove jedinstvene značajke uglavnom usmjerene na online prodaju [17].

Stripe pohranjuje sve informacije kreditnih kartica na siguran način, što znači da Stripe ne može vidjeti brojeve kreditnih kartica bez dodatnih koraka. Također, Stripe zahtijeva da se sve online transakcije odvijaju preko sigurnijeg HTTPS (engl. Hypertext transfer protocol secure, protokol koji omogućuje sigurnu komunikaciju između korisnika i web stranice) protokola [17].

3.10. Vercel

Vercel nudi proizvod *frontend-as-a-service*, olakšavajući inženjerima implementaciju i pokretanje njihovih aplikacija. Aplikacije se sastoje od dva dijela: klijentskog i poslužiteljskog. Klijentski dio se obično implementira s poslužiteljskim dijelom na jednom serveru što zahtijeva infrastrukturu za postavljanje i upravljanje. Vercel omogućava timovima i pojedincima jednostavno implementiranje njihovog klijentskog dijela, odvojeno od poslužiteljskog. Korištenje Vercela pruža brojne prednosti kao što su pregledi implementacija, funkcije kao usluga, analitika i više. Vercel također nudi i značajke poput pohrane i baza podataka [18].

3.11. Node.js

Node.js je okruženje za izvođenje JavaScript koda koje radi na više platformi. Node.js je otvorenog koda, što znači da je izvorni kôd za Node.js javno dostupan. Održavaju ga suradnici iz cijelog svijeta. Node.js ne ovisi o softveru operativnog sustava i može raditi na Linuxu, macOS-u ili Windowsu [19].

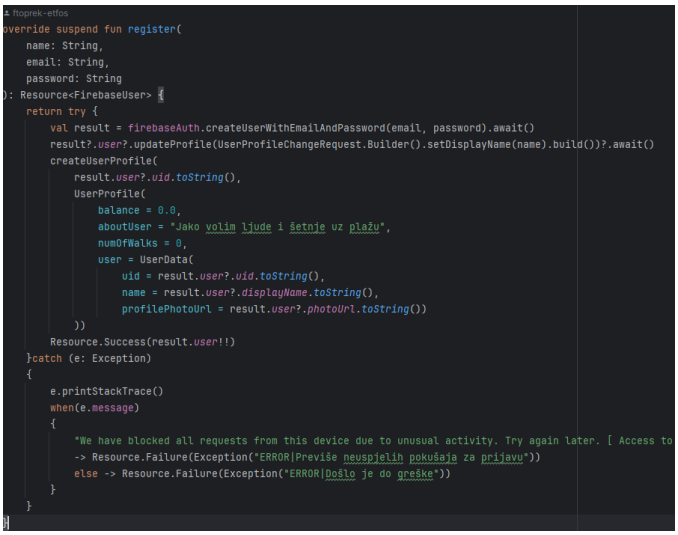
Preglednici poput Chromea i Firefoxa imaju okruženja za izvođenje JavaScript koda. Prije nego što je stvoren Node.js, JavaScript je mogao raditi samo u pregledniku i koristio se za izgradnju samo korisničkog dijela aplikacija. Node.js pruža okruženje za izvođenje izvan preglednika. To omogućuje izgradnju aplikacija na serverskoj strani korištenjem istog programskog jezika [19].

4. IMPLEMENTACIJA RJEŠENJA APLIKACIJE

U ovom poglavlju je opisano programsko rješenje aplikacije te uvid u bazu podataka. Ovdje je objašnjeno kako su glavne komponente aplikacije dizajnirane i kako međusobno komuniciraju kako bi pružile željene funkcionalnosti.

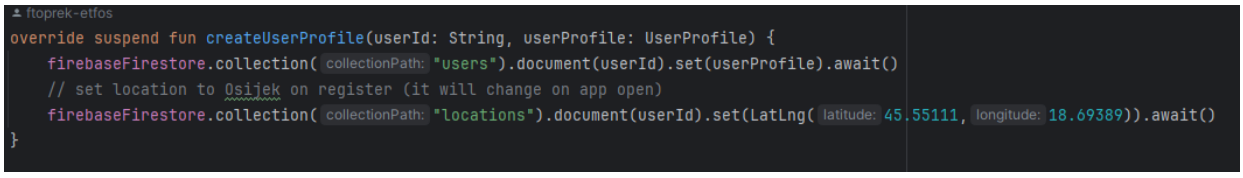
4.1. Autentikacija korisnika

Prilikom registracije korisnik unosi svoje ime i prezime, svoju adresu e-pošte i zaporku pomoću koje će se prijavljivati. Zaporka se ne sprema u bazi podataka u tekst formatu već je kodirana scrypt algoritmom. Kôd koji omogućuje registraciju korisnika prikazan je na slici 4.1.



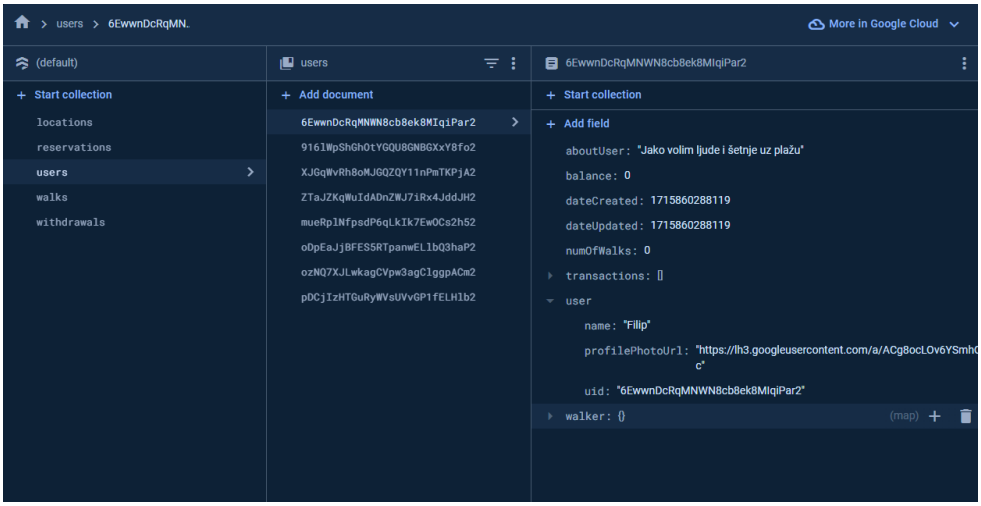
Sl. 4.1. Autentikacijski kôd

Ukoliko dođe do greške prilikom registracije korisnik dobiva odgovarajuću poruku na zaslonu.



Sl. 4.2. Kreiranje korisničkog profila

Svaki korisnik posjeduje profil na kojem su prikazane informacije kao što su ime, prezime, broj šetnji i opis profila. Prilikom kreiranja korisnikovog profila podatci se spremaju u bazu podataka. Na slici 4.2 prikazan je kôd za kreiranje korisničkog profila kao i postavljanje početne lokacije korisnika. U bazi podataka profil je spremljen unutar kolekcije *users*. Baza podataka se sastoji od pet kolekcija koje sadrže različite podatke o korisnicima.



Sl. 4.3. Korisnički profil u bazi podataka

Jedinstvena oznaka kao i podatci profila su vidljivi na slici 4.3. Na slici 4.4.prikazan je kod koji se izvršava prilikom prijave. Ukoliko dođe do greške prilikom prijave korisniku je prikazana odgovarajuća poruka.

```
override suspend fun login(email: String, password: String): Resource<FirebaseUser> {
    return try {
        val result = firebaseAuth.signInWithEmailAndPassword(email, password).await()
        Resource.Success(result.user!!)
    }catch (e: Exception)
    {
        e.printStackTrace()
        when(e.message)
        {
            "The supplied auth credential is incorrect, malformed or has expired." -> Resource.Failure(Exception("ERROR|Krivo uneseni podatci ili račun ne postoji"))
            "We have blocked all requests from this device due to unusual activity. Try again later. [ Access to this account has been temporarily disabled due to many f
            -> Resource.Failure(Exception("ERROR|Previše neuspjelih pokušaja za prijavu"))
            "The user account has been disabled by an administrator." -> Resource.Failure(Exception("ERROR|Administrator je onemogućio korisnički račun"))
            else -> Resource.Failure(Exception("ERROR|Došlo je do greške"))
        }
    }
}
```

Sl. 4.4. Prijava korisnika

```
ftoprek-etfos
override suspend fun firebaseSignInWithGoogle(googleAuthCredential: AuthCredential): Resource<FirebaseUser> {

    return try {
        val result = firebaseAuth.signInWithCredential(googleAuthCredential).await()
        if(result?.additionalUserInfo?.isNewUser == true)
        {
            createUserProfile(
                result.user?.uid.toString(),
                UserProfile(
                    balance = 0.0,
                    aboutUser = "Jako volim ljude i šetnje uz plažu",
                    numOfWalks = 0,
                    user = UserData(
                        uid = result.user?.uid.toString(),
                        name = result.user?.displayName.toString(),
                        profilePhotoUrl = result.user?.photoUrl.toString())
                ))
        }
        Resource.Success(result?.user!!)
    }catch (e: Exception)
    {
        Resource.Failure(e)
    }
}
```

Sl. 4.5. Autentikacija korisnika putem Google računa

Ukoliko korisnik želi koristiti prijavu ili registraciju putem Google računa onda se izvršava kôd na slici 4.5. Prilikom izvršavanja provjerava se je li korisnik novi ili postojeći. Ukoliko korisnik ne postoji kreira se novi profil i sprema se u bazu podataka. Nakon spremanja novog profila korisnik se automatski prijavljuje.

4.2. Dohvaćanje dostupnih šetača

Kada prijavljeni korisnik otvori aplikaciju tada se izvršava kôd na slici 4.6 za dohvaćanje šetača i korisniku se prikazuje lista dostupnih šetača. Dohvaća se lista šetača koji se nalaze samo unutar korisnikovog mjesta.

```
ftoprek-etfos
private fun getWalkers() {
    viewModelScope.launch { this: CoroutineScope
        val currentUserProfile = profileRepository.getUserProfile(authRepository.currentUser?.uid.toString())
        _homeFlow.value = Resource.Loading
        val result = homeRepository.getWalkerList(currentUserProfile)

        if(result.isEmpty())
        {
            _homeFlow.value = Resource.Failure(Exception("Error home"))
        }else
        {
            _walkerList.value = result
            _homeFlow.value = Resource.Success(result)
        }
    }
}
```

Sl. 4.6. Kôd za dohvaćanje liste šetača

Prvo se dohvaća kolekcija *users* potom se filtriraju svi dokumenti unutar te kolekcije. Filtriraju se na način da ostanu samo oni dokumenti u kojima su odobreni šetači i čija je lokacija ista kao i korisnikova. Nakon filtriranja dokumenti se pretvaraju u objekt *UserProfile* i vraća se lista profila. Ukoliko dođe do greške vraća se prazna lista.

```
ftoprek-etfos *
override suspend fun getWalkerList(user: UserProfile?): List<UserProfile?> {
    return try {
        val snapshot = firebaseFirestore.collection( collectionPath: "users") CollectionReference
            .get() Task<QuerySnapshot!>
            .await()

        snapshot.documents (Mutable)List<DocumentSnapshot!>
            .filter { document ->
                val walker = document["walker"] as? Map<*, *>
                val isApproved = walker?.get("approved") as? Boolean ?: false
                val city = document["city"] as? String ?: ""

                walker != null && isApproved && (user == null || city == user.city) ^filter
            } List<DocumentSnapshot!>
            .map { document ->
                document.toObject(UserProfile::class.java)
            }
    } catch (e: Exception) {
        emptyList()
    }
}
```

Sl. 4.7. Kôd za dohvaćanje šetača iz baze podataka

Na slici 4.7 vidljiv je kôd za dohvaćanje i filtriranje šetača.

4.3. Dohvaćanje profila korisnika

Kako bi prikazali podatke korisnika na vlastitom profilu potrebno je dohvatiti ih iz baze te spremiti u poznati format. Na slici 4.8 prikazana je funkcija koja dohvaća korisnika prema njegovom jednoznačnom identifikatoru te ukoliko je uspješno dohvaćen pretvara ga i vraća objekt *UserProfile*, ako nije uspješno dohvaćen vratiti će *null* vrijednost.

```
ftoprek-etfos *
override suspend fun getUserProfile(userId: String): UserProfile? {
    return try {
        val result = firebaseFirestore.collection( collectionPath: "users").document(userId).get().await()
        result?.toObject<UserProfile>()
    } catch (e: Exception) {
        e.printStackTrace()
        null
    }
}
```

Sl. 4.8. Kôd za dohvaćanje profila korisnika

Ukoliko korisnik popuni tražene podatke tada će njegovi podatci biti spremljeni u bazu podataka te se čeka administrator kako bi provjerio i odobrio njegov profil. Funkcija na slici 4.9 omogućuje spremanje tih podataka te sprema vrijednost kada je zadnji put korisnikov profil ažuriran.

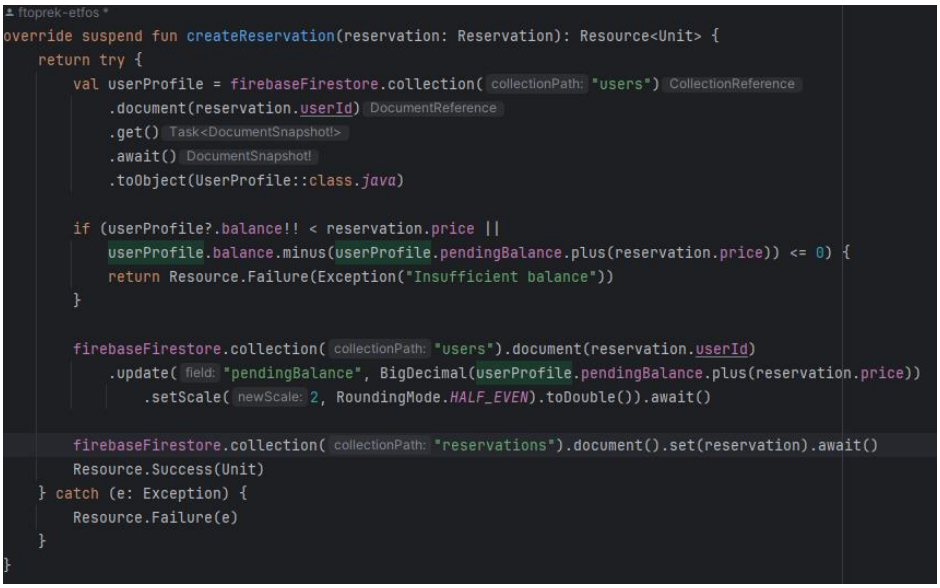
```
ftoprek-etfos *
override suspend fun becomeWalker(userProfile: UserProfile, userId: String): Resource<UserProfile>? {
    return try {
        firebaseFirestore.collection( collectionPath: "users").document(userId).update( field: "walker", userProfile.walker).await()
        firebaseFirestore.collection( collectionPath: "users").document(userId).update( field: "dateUpdated", System.currentTimeMillis()).await()
        Resource.Success(userProfile)
    } catch (e: Exception) {
        Resource.Failure(e)
    }
}
```

Sl. 4.9. Kôd za slanje zahtijeva za šetača

4.4. Rezervacija šetnje

Prilikom rezervacije šetnje kada korisnik unese sve tražene podatke provjerava se sadrži li korisnikov virtualni novčanik dovoljan iznos za odabranu rezervaciju. Ukoliko korisnik nema dovoljan iznos tada se prikazuje odgovarajuća poruka na zaslonu i korisnik nema mogućnost

napraviti novu rezervaciju. Kada korisnik ima dovoljan iznos tada se izvršava kod na slici 4.10 koji dodaje novo polje u bazu podataka na korisnički profil nazvan *pendingBalance*.

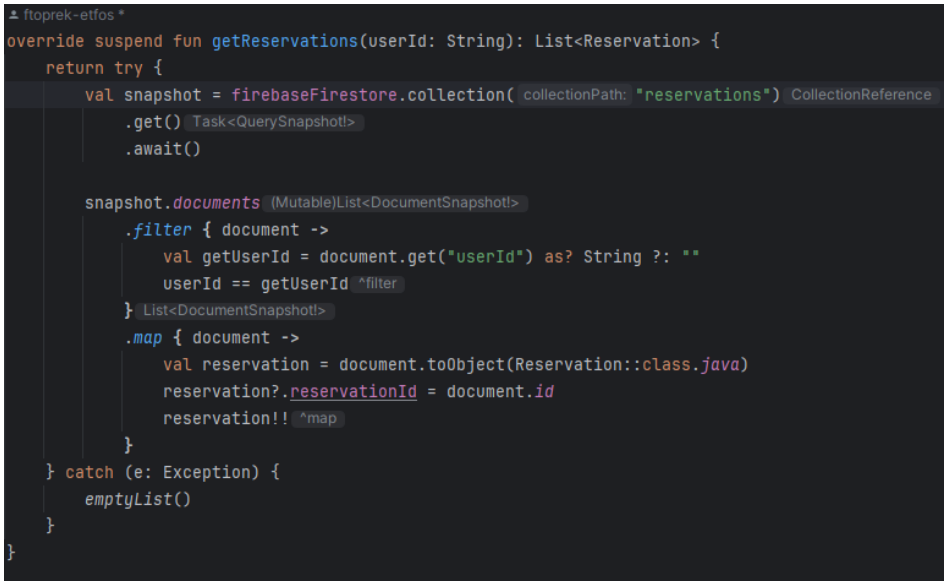


Sl. 4.10. Funkcija za kreiranje nove rezervacije

To polje sadrži ukupni iznos trenutno napravljenih ili prihvaćenih rezervacija.

4.5. Lista rezervacija

Kada se korisnik nalazi na zaslonu za prikaz rezervacija izvršava funkcija vidljiva na slici 4.11 koja dohvaća listu rezervacija. Prikazana funkcija radi na način da dohvaća kolekciju *reservations* te traži sve rezervacije koje sadrže korisnikov jedinstveni identifikator potom ih pretvara u objekt *Reservation* i vraća ga. Ukoliko dođe do greške funkcija vraća praznu listu.



Sl. 4.11. Funkcija za dohvaćanje rezervacija

4.6. Nadoplata virtualnog novčanika

Kada korisnik unese vrijednost nadoplate za virtualni novčanik šalje se vrijednost nadoplate putem HTTP zahtijeva na API krajnju točku. Kod za komunikaciju između aplikacije i Stripe poslužitelja prikazan je na slici 4.12, a kod za HTTP zahtjev prikazan je na slici 4.13. Kreira se novi korisnik za plaćanje, privremeni ključ i namjera plaćanja.

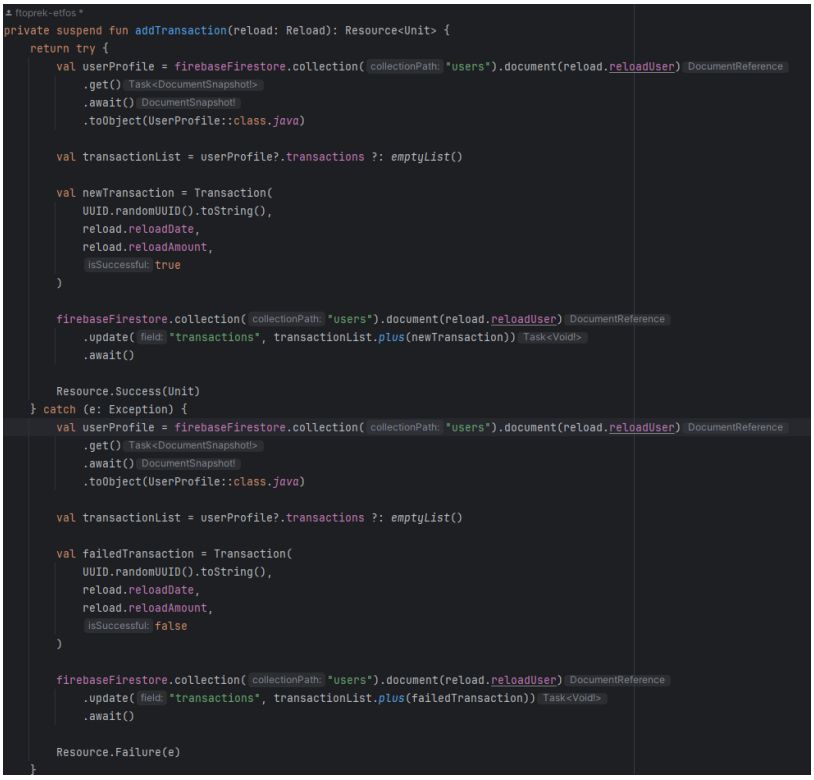
```
import Stripe from 'stripe';

const stripe = new Stripe(process.env.STRIPE_SECRET_KEY);

export default async function handler(req, res) {
  if (req.method === 'POST') {
    try {
      const customer = await stripe.customers.create();

      const ephemeralKey = await stripe.ephemeralKeys.create(
        { customer: customer.id },
        { apiVersion: '2022-11-15' }
      );

      const { amount } = req.body;

      const paymentIntent = await stripe.paymentIntents.create({
        amount: amount,
        currency: 'eur',
        customer: customer.id,
        payment_method_types: ['card'],
      });

      res.status(200).json({
        paymentIntentClientSecret: paymentIntent.client_secret,
        ephemeralKeySecret: ephemeralKey.secret,
        customerId: customer.id,
        publishableKey: "pk_test_                                                    "
});
    } catch (error) {
      res.status(500).json({ error: error.message });
    }
  } else {
    res.setHeader('Allow', 'POST');
    res.status(405).end('Method Not Allowed');
  }
}
```

Sl. 4.12. Kod za komunikaciju sa Stripe poslužiteljem

```
CoroutineScope(Dispatchers.IO).launch { this: CoroutineScope
    val amount = reloadAmount.toDouble() * 100.0
    "https://stripe-api-eight.vercel.app/api/create-payment-intent".httpPost()
        .header("Content-Type", value: "application/json")
        .body("{ \"amount\": $amount }")
        .responseJson { _, _, result ->
            if (result is Result.Success) {
                val responseJson = result.get().obj()
                paymentIntentClientSecret =
                    responseJson.getString( name: "paymentIntentClientSecret")
                customerConfig = PaymentSheet.CustomerConfiguration(
                    responseJson.getString( name: "customerId"),
                    responseJson.getString( name: "ephemeralKeySecret")
                )
                val publishableKey = responseJson.getString( name: "publishableKey")
                PaymentConfiguration.init(context, publishableKey)
                presentPaymentSheet(
                    paymentSheet,
                    customerConfig!!, paymentIntentClientSecret!!
                )
            }
        }
}
```

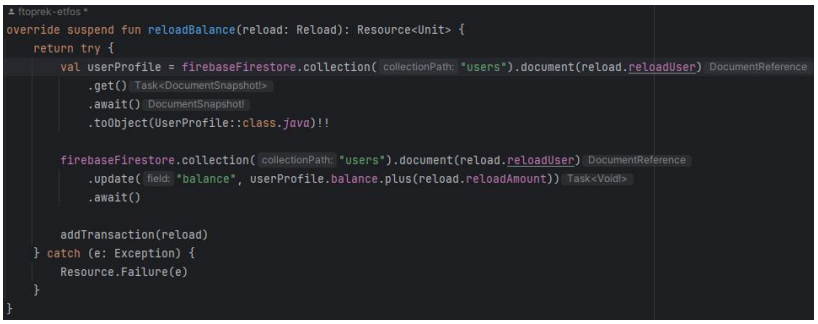
Sl. 4.13. Slanje vrijednosti nadoplate na API krajnju točku

Nakon što je nadoplata uspješna dodaje se nova transakcija u bazu podataka što je prikazano na slici 4.14. Zatim se povećava vrijednost korisnikovog novčanika za nadoplaćenu vrijednost što je prikazano na slici 4.15. Nova transakcija se dodaje na način da se prvo pročita trenutna lista transakcija iz baze podataka, nakon toga se kreira nova transakcija te ju nadodajemo na postojeću listu transakcija i zapisujemo u bazu podataka.



Sl. 4.14. Dodavanje nove transakcije

Ukoliko dođe do greške zapisujemo transakciju sa svojstvom *isSuccessful* u vrijednosti *false*.



Sl. 4.15. Nadopuna novčanika za nadoplaćenu vrijednost

4.7. Isplata novaca

Kada korisnik uđe na zaslon za prikaz detalji rezervacije tada se dohvaća lista svih isplata koje je korisnik napravio. Kod zadužen za tu funkcionalnost je vidljiv na slici 4.16. Dohvaćaju se sve isplate unutar kolekcije *withdrawals* koje pripadaju korisniku.

Nakon dohvaćanja mapiraju se i vraćaju objekt *Withdrawals*. Ukoliko dođe do greške vraća se prazna lista odnosno prazan objekt *Withdrawals*.

```
ftoprek-etfos *
override suspend fun getWithdrawals(userProfile: UserProfile): Withdrawals {
    return try {
        firebaseFirestore.collection( collectionPath: "withdrawals").document(userProfile.user.uid) DocumentReference
            .get() Task<DocumentSnapshot!>
            .await() DocumentSnapshot!
            .toObject(Withdrawals::class.java) ?: Withdrawals()
    } catch (e: Exception) {
        Log.w( tag: "ERROR",  msg: "Failed to get withdrawals: ${e.message}")
        Withdrawals()
    }
}
```

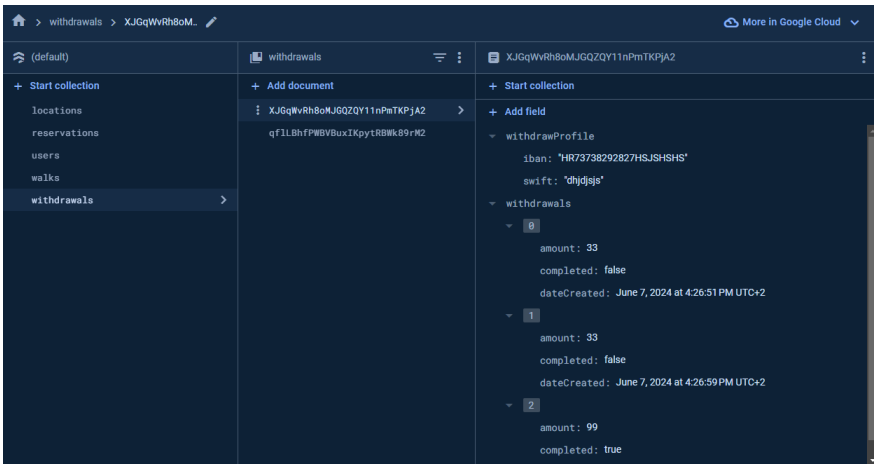
Sl. 4.16. Dohvaćanje liste isplata

Funkcionalnost kreiranja nove isplate prikazana je na slici 4.17 gdje se prvo dohvaća korisnikov profil nakon toga se dohvaća lista svih prethodnih isplata. Ukoliko korisnik nema prethodne isplate kreira se nova lista isplate te se nadodaje novo kreirana isplata. Ukoliko korisnik ima prethodne isplate tada se samo nadodaje novo kreirana isplata na prethodnu listu te se ažurira lista u bazi podataka. Na kraju iznos korisnikovog računa se umanjuje za vrijednost isplate.

```
ftoprek-etfos *
override suspend fun createWithdrawalRequest(withdraw: Withdraw, withdrawProfile: WithdrawProfile, userProfile: UserProfile) {
    try {
        val user = firebaseFirestore.collection( collectionPath: "users") CollectionReference
            .document(userProfile.user.uid) DocumentReference
            .get() Task<DocumentSnapshot!>
            .await() DocumentSnapshot!
            .toObject(UserProfile::class.java)

        val withdrawals = firebaseFirestore.collection( collectionPath: "withdrawals") CollectionReference
            .document(userProfile.user.uid) DocumentReference
            .get() Task<DocumentSnapshot!>
            .await() DocumentSnapshot!
            .toObject(Withdrawals::class.java)

        if (withdrawals == null) {
            firebaseFirestore.collection( collectionPath: "withdrawals").document(userProfile.user.uid).set(
                Withdrawals(
                    withdrawals = listOf(withdraw),
                    withdrawProfile = withdrawProfile
                )
            ).await()
        } else {
            firebaseFirestore.collection( collectionPath: "withdrawals").document(userProfile.user.uid).update(
                 field: "withdrawals", withdrawals.withdrawals.plus(withdraw)
            ).await()
        }

        firebaseFirestore.collection( collectionPath: "users").document(userProfile.user.uid).update(
             field: "balance", BigDecimal( val: user?.balance!! - withdraw.amount)
                .setScale( newScale: 2, RoundingMode.HALF_EVEN).toDouble()
        ).await()
    } catch (e: Exception) {
        Log.w( tag: "ERROR",  msg: "Failed to create withdrawal request: ${e.message}")
    }
}
```

Sl. 4.17. Kreiranje nove isplate

Na slici 4.18 prikazana je kolekcija *withdrawals* te primjer liste isplata koje je korisnik napravio.



Sl. 4.18. Spremanje isplate u bazu podataka

4.8. Upravljanje rezervacijama

Kada korisnik pošalje rezervaciju šetaču tada postoje opcije prihvaćanja ili odbijanja šetnje. Ukoliko šetač odluči odbiti rezervaciju izvršava se kod na slici 4.19 i tada se u bazi podataka svojstvo *declined* postavlja na vrijednost *true* te korisnikov zamrznut novac se briše.

```
ftoprek-etfos *
override suspend fun declineReservation(reservation: Reservation): Resource<Unit> {
    return try {
        firebaseFirestore.collection( collectionPath: "reservations").document(reservation.reservationId)
            .update( field: "declined",  value: true).await()

        val userProfile = firebaseFirestore.collection( collectionPath: "users") CollectionReference
            .document(reservation.userId) DocumentReference
            .get() Task<DocumentSnapshot!>
            .await() DocumentSnapshot!
            .toObject(UserProfile::class.java)!!

        firebaseFirestore.collection( collectionPath: "users").document(reservation.userId)
            .update( field: "pendingBalance", userProfile.pendingBalance.minus(reservation.price)).await()

        Resource.Success(Unit)
    } catch (e: Exception) {
        Resource.Failure(e)
    }
}
```

Sl. 4.19. Funkcija za odbijanje rezervacije

```
ftoprek-etfos *
override suspend fun acceptReservation(reservationId: String): Resource<Unit> {
    return try {
        firebaseFirestore.collection( collectionPath: "reservations").document(reservationId)
            .update( field: "accepted",  value: true).await()
        Resource.Success(Unit)
    } catch (e: Exception) {
        Resource.Failure(e)
    }
}
```

Sl. 4.20. Funkcija za prihvaćanje rezervacije

Ukoliko šetač prihvati rezervaciju izvršava se kod vidljiv na slici 4.20 te se tada u bazi podataka svojstvo *accepted* postavlja na vrijednost *true*. Kada šetač odluči započeti šetnju izvršava se kod

na slici 4.21 i tada se svojstvo *started* postavlja na vrijednost *true* te svojstvo koje označava vrijeme početka šetnje *timeWalkStarted* postavlja na trenutno vrijeme.

```
ftoprek-etfos *
override suspend fun startWalk(reservationId: String): Resource<Unit> {
    return try {
        firebaseFirestore.collection( collectionPath: "reservations").document(reservationId)
            .update( field: "started",  value: true).await()

        firebaseFirestore.collection( collectionPath: "reservations").document(reservationId)
            .update( field: "timeWalkStarted", Timestamp.now()).await()
        Resource.Success(Unit)
    } catch (e: Exception) {
        Resource.Failure(e)
    }
}
```

Sl. 4.21. Početak šetnje

Kada se šetnja završi tada se svojstvo *completed* postavlja na vrijednost *true* i svojstvo koje označava vrijeme završetka šetnje se postavlja na trenutno vrijeme. Nakon završetka šetnje poziva se funkcija na slici 4.22 koja je zadužena za prebacivanje iznosa šetnje iz korisnikovog novčanika u novčanik šetača. Također se zamrznuti iznos u bazi podataka oduzima za vrijednost šetnje.

```
ftoprek-etfos *
private suspend fun transferCoins(reservation: Reservation) {
    try {
        val userProfile = firebaseFirestore.collection( collectionPath: "users") CollectionReference
            .document(reservation.userId) DocumentReference
            .get() Task<DocumentSnapshot!>
            .await() DocumentSnapshot!
            .toObject(UserProfile::class.java)

        val walkerProfile = firebaseFirestore.collection( collectionPath: "users") CollectionReference
            .document(reservation.walkerUserId) DocumentReference
            .get() Task<DocumentSnapshot!>
            .await() DocumentSnapshot!
            .toObject(UserProfile::class.java)

        firebaseFirestore.collection( collectionPath: "users").document(reservation.userId).update(
            field: "balance", BigDecimal(userProfile?.balance?.minus(reservation.price)!!)
                .setScale( newScale: 2, RoundingMode.HALF_EVEN).toDouble()
        ).await()

        firebaseFirestore.collection( collectionPath: "users").document(reservation.walkerUserId).update(
            field: "balance", BigDecimal(walkerProfile?.balance?.plus(reservation.price)!!)
                .setScale( newScale: 2, RoundingMode.HALF_EVEN).toDouble()
        ).await()

        firebaseFirestore.collection( collectionPath: "users").document(reservation.userId).update(
            field: "pendingBalance", userProfile.pendingBalance.minus(reservation.price)
        ).await()

        firebaseFirestore.collection( collectionPath: "users").document(reservation.walkerUserId).update(
            field: "numOfWalks", walkerProfile.numOfWalks.plus( other: 1)
        ).await()

        firebaseFirestore.collection( collectionPath: "users").document(reservation.userId).update(
            field: "numOfWalks", userProfile.numOfWalks.plus( other: 1)
        ).await()

    } catch (e: Exception) {
        e.printStackTrace()
    }
}
```

Sl. 4.22. Funkcija koja prebacuje iznos šetnje u šetačev novčanik

Na slici 4.23 prikazana je funkcija koja se poziva u trenutku završetka šetnje.

```
ftoprek-etfos *
override suspend fun endWalk(reservation: Reservation): Resource<Unit> {
    return try {
        firebaseFirestore.collection( collectionPath: "reservations").document(reservation.reservationId)
            .update( field: "timeWalkEnded", Timestamp.now()).await()

        firebaseFirestore.collection( collectionPath: "reservations").document(reservation.reservationId)
            .update( field: "completed",  value: true).await()

        transferCoins(reservation)
        Resource.Success(Unit)
    } catch (e: Exception) {
        Resource.Failure(e)
    }
}
```

Sl. 4.23. Kraj šetnje

4.9. Praćenje lokacije šetača

Kada šetač započne šetnju tada se njegova lokacija sprema u bazu podataka svakih 10 sekundi. Kod koji sprema lokaciju u bazu podataka je vidljiv na slici 4.24.

```
ftoprek-etfos *
override suspend fun sendLocation(location: Location) {
    firebaseFirestore.collection( collectionPath: "locations").document(location.userId).update(
        field: "longitude", location.longitude,
        ...moreFieldsAndValues: "latitude", location.latitude
    ).await()
}
```

Sl. 4.24. Spremanje lokacije u bazu podataka

Podatci o lokaciji se spremaju unutar *locations* kolekcije te se spremaju podatci zemljopisne dužine i širine.

```
ftoprek-etfos *
override suspend fun getLocation(userId: String): Location {
    return try {
        firebaseFirestore.collection( collectionPath: "locations").document(userId).get().await().toObject(Location::class.java)!!
    } catch (e: Exception) {
        e.printStackTrace()
        Location()
    }
}
```

Sl. 4.25. Dohvaćanje lokacije iz baze podataka

Kada korisnik pristupi zaslonu na kojem se prikazuje lokacija šetača tada se izvršava funkcija na slici 4.25. Navedena funkcija dohvaća zemljopisnu dužinu i širinu iz baze podataka te ju prikazuje na mapi.

```
ftoprek-etfos *
override suspend fun sendWalkLocationPoint(locationPoint: LocationPoint) {
    val userDocument = firebaseFirestore.collection( collectionPath: "locations").document(locationPoint.location.userId)
    try {
        firebaseFirestore.runTransaction { transaction ->
            val documentSnapshot = transaction.get(userDocument)
            val array = documentSnapshot.get(locationPoint.reservationId) as? List<Location>
            val updatedArray = array?.plus(locationPoint.location) ?: listOf(locationPoint.location)
            transaction.update(userDocument, locationPoint.reservationId, updatedArray) *runTransaction
        }.await()
    } catch (e: Exception) {
        e.printStackTrace()
    }
}
```

Sl. 4.26. Spremanje lokacijskih podataka

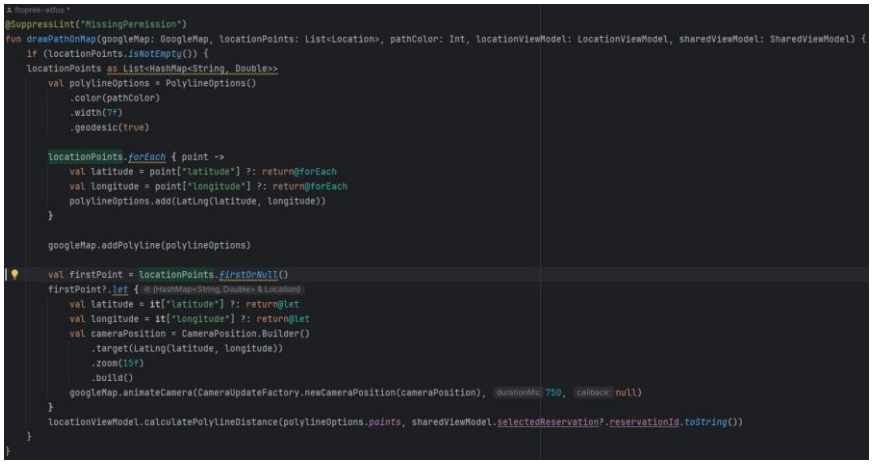
Prilikom šetnje zapisuje se svaka lokacija unutar liste koja je jednoznačno definirana rezervacijskim identifikatorom, a funkcija za spremanje prikazana je na slici 4.26.

4.10. Pregled rute šetnje

Prilikom pregleda rute šetnje dohvaća se lista lokacijskih podataka koja je kreirana prilikom šetnje te je ta funkcionalnost vidljiva na slici 4.27.

Sl. 4.27. Dohvaćanje lokacijskih podataka

Lista lokacijskih podataka upotrebljava se za iščitavanje vrijednosti zemljopisne dužine i širine potom se izračunava ruta šetnje i prikazuje se na mapi.



Sl. 4.28. Iscrtavanje rute šetnje

Kod za iscrtavanje rute šetnje je vidljiv na slici 4.28. Tijekom crtanja rute šetnje izračunava se i ukupna prijeđena udaljenost u kilometrima.

4.11. Lokacijski servis

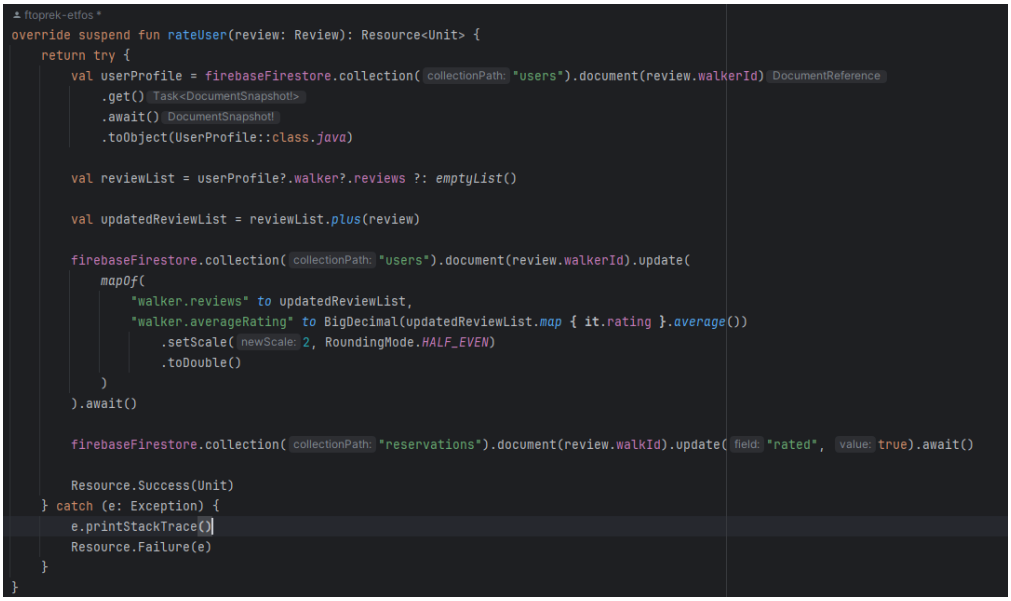
Lokacijski servis provjerava je li korisnik autentificiran i je li proces praćenja već pokrenut. Ukoliko nije, pokreće praćenje, prikazuje trajnu notifikaciju i započinje dobivanje ažuriranja lokacije svakih 10 sekundi. Svaki korisnik koji koristi aplikaciju mora dati odgovarajuće dopuštenje kako bi se mogao koristiti servis praćenja lokacije. Na taj način omogućuje aplikaciji

pristup GPS podacima korisnika i pruža precizne informacije o korisnikovoj trenutnoj lokaciji tijekom korištenja aplikacije. Kod opisane funkcionalnosti vidljiv je na slici 4.29.

Sl. 4.29. Lokacijski servis

4.12. Ocjenjivanje šetača

Funkcija za ocjenjivanje šetača vidljiva je na slici 4.30. Prikazana funkcija dohvaća korisnički profil šetača iz baze podataka i pretvara ga u objekt *UserProfile*. Ukoliko ga uspješno pretvori tada dodaje novu recenziju postojećem popisu recenzija šetača i ažurira prosječnu ocjenu šetača. Zatim se ažurira odgovarajući dokument rezervacije kako bi se označilo da je šetnja ocijenjena. Prosječna ocjena šetača se zaokružuje na dvije decimale. Ocjena šetača se prikazuje na profilu na način da se iščita iz baze podataka svojstvo *averageRating*.



Sl. 4.30. Funkcija za ocjenjivanje šetača

5. FUNKCIONALNOSTI APLIKACIJE

Funkcionalnost aplikacije detaljno je prikazana u ovom poglavlju, obuhvaćajući sve mogućnosti i značajke koje ona pruža korisnicima. Poglavlje započinje predstavljanjem grafičkog sučelja aplikacije. Zatim se fokusira na objašnjenje korištenja aplikacije, vodeći kroz osnovne i napredne funkcije.

5.1. Grafičko sučelje aplikacije

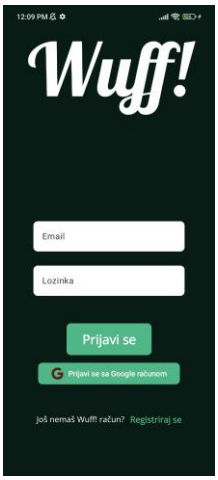
Početni zaslon aplikacije karakterizira jednostavan dizajn s ograničenom paletom boja. Pozadina je tamno zelene boje, što stvara snažan kontrast i dubinu. Bijela boja koristi se za tekst, ikone i ostale elemente sučelja, osiguravajući jasnu vidljivost i čitljivost. Dodatne nijanse zelene mogu se koristiti za naglašavanje određenih dijelova sučelja, čime se postiže vizualno ugodan izgled aplikacije.

Sl. 5.1. Početni zaslon aplikacije

5.2. Autentikacija korisnika

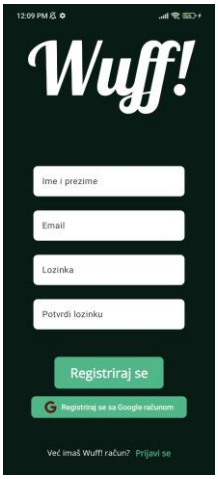
Kako bi korisnici mogli pristupiti aplikaciji potrebno je napraviti račun ili prijaviti se pomoću Google računa. Pritiskom na gumb „Započni“ vidljiv na slici 5.1. korisnik je preusmjeren na zaslon za registraciju. Zaslon za registraciju je vidljiv na slici 5.2. Prilikom registracije korisnik unosi ime i prezime, adresu e-pošte i zaporku. Ukoliko korisnik želi iskoristiti svoj Google račun za registraciju potrebno je pritisnuti gumb „Registriraj se sa Google računom“.

Ukoliko je korisnik već prošao registraciju ili posjeduje račun tada je potrebno prijaviti se. Pritiskom na gumb „Prijavi se“ vidljiv na slici 5.1. korisnik se preusmjerava na zaslon za prijavu. Prilikom prijave korisnik unosi adresu e-pošte i zaporku koju je koristio prilikom registracije.



Sl. 5.2. Zaslon prijave

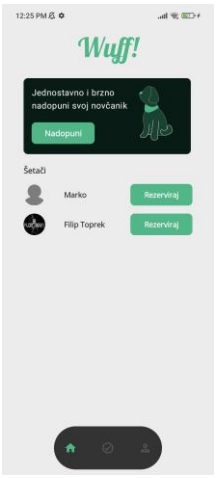
Zaslon registracije je vidljiv na slici 5.3, a zaslon prijave je vidljiv na slici 5.2.



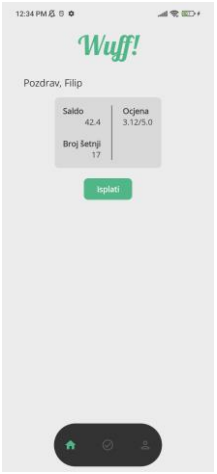
Sl. 5.3. Zaslon registracije

5.3. Početni zaslon

Nakon što je korisnik prijavljen prikazuje mu se početni zaslon aplikacije. Početni zaslon se razlikuje za korisnika koji prima uslugu i korisnika koji nudi uslugu šetnje. Zaslon korisnika koji prima uslugu vidljiv je na slici 5.4. Na početnom zaslonu se nalazi gumb za nadoplatu i lista dostupnih šetača u mjestu korisnika. Početni zaslon šetača je vidljiv na slici 5.5 gdje je vidljiv gumb za isplatu novaca kao i detaljni podatci šetača.



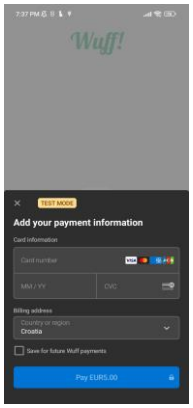
Sl. 5.4. Početni zaslon korisnika



Sl. 5.5. Početni zaslon šetača

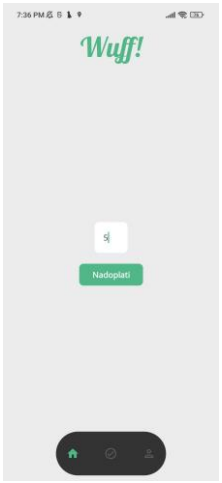
5.4. Nadopuna virtualnog novčanika

Ukoliko korisnik klikne na gumb „Nadopuni“ prikazati će se zaslon gdje je moguće unijeti vrijednost nadoplate. Ukoliko korisnik unese vrijednost te pritisne gumb „Nadoplati“ prikazati će se obrazac za unos podataka kartice koji je vidljiv na slici 5.6. Nakon unosa točnih podataka te pritiskom na gumb „Pay“ će se izvršiti terećenje kartice u odabranom iznosu.



Sl. 5.6. Unos podataka kartice

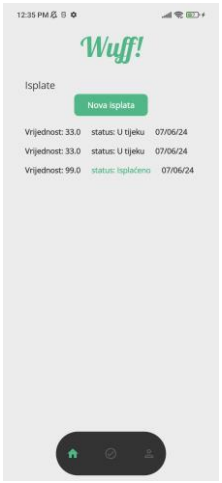
Na slici 5.7 prikazano je polje za unos željenog iznosa nadoplate.



Sl. 5.7. Zaslon nadopune

5.5. Isplata virtualnog novčanika

Kako bi korisnik mogao izvršiti isplatu virtualnog novčanika potrebno je pritisnuti gumb „Isplati“ na početnom zaslonu vidljivom na slici 5.5.



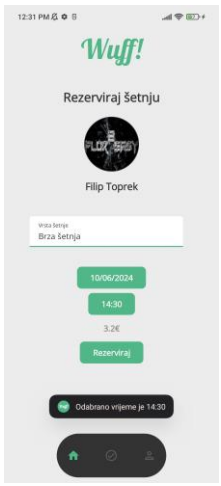
Sl. 5.8. Lista isplata

Na zaslonu isplata vidljivi su detalji zatraženih i završenih isplata. Pritiskom na gumb „Nova isplata“ na slici 5.8 korisniku će biti prikazan obrazac za unos podataka bankovnog računa kao i iznosa koji želi isplatiti. Slika 5.9 prikazuje zaslon nove isplate.

Sl. 5.9. Zaslon nove isplate

5.6. Zaslon nove rezervacije

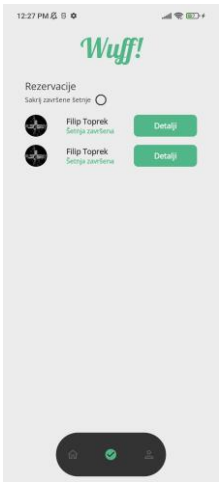
Kako bi korisnik kreirao novu rezervaciju potrebno je odabrati odgovarajućeg šetača te potom kliknuti na gumb „Rezerviraj“ koji se nalazi pored imena šetača na slici 5.4. Nakon pritiska na gumb prikazati će se obrazac za unos detalja šetnje. Detalji uključuju vrijeme i datum šetnje kao i vrstu šetnje. Vrsta šetnje se odnosi na trajanje. Svi detalji šetnje su vidljivi na slici 5.10.



Sl. 5.10. Zaslon nove rezervacije

5.7. Zaslon svih rezervacija

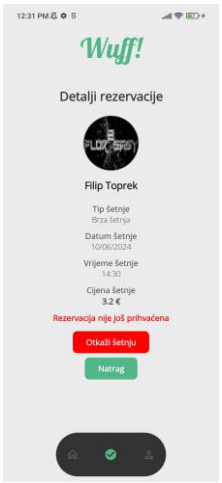
Zaslon svih rezervacija omogućuje korisnicima uvid u prijašnje kao i nadolazeće šetnje. Zaslon liste rezervacija vidljiv je na slici 5.11.



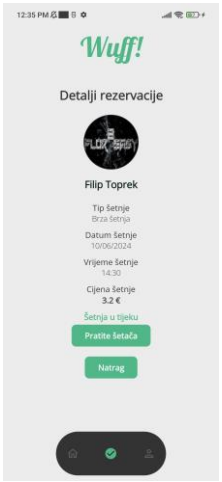
Sl. 5.11. Zaslon liste rezervacije

5.8. Zaslon detalja rezervacija

Prilikom pritiska na gumb „Detalji“ na slici 5.11 korisniku su prikazani detalji odabrane rezervacije. Ukoliko rezervacija još nije prihvaćena korisniku je prikazan zaslon na slici 5.12



Sl. 5.12. Detalji trenutno napravljene rezervacije



Sl. 5.13. Detalji prihvaćene i započete rezervacije

Pritiskom na gumb „Otkaži šetnju“ rezervacija se briše te neće više biti vidljiva. Ukoliko je rezervacija prihvaćena i šetnja je započela onda se korisniku nudi opcija praćenja šetača koja je vidljiva na slici 5.13.Ukoliko je korisnik šetač onda ima opciju za prihvatiti ili odbiti rezervaciju.

Pritiskom na „Prihvati šetnju“ na slici 5.14. šetnja više ne može biti obrisana.

Sl. 5.14. Detalji nove rezervacije za korisnika šetača

Sl. 5.15. Detalji prihvaćene rezervacije za korisnika šetača

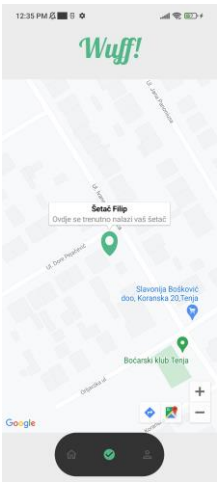
Nakon što je šetnja prihvaćena moguće ju je započeti. Sučelje koje prikazuje detalje prihvaćene rezervacije vidljivo je na slici 5.15.

Kada je šetnja završila korisniku je vidljiv zaslon na slici 5.16. Nakon završetka šetnje korisnik može vidjeti koliko je šetnja trajala i koliko je bila dugačka.

Sl. 5.16. Detalji završene šetnje

5.9. Praćenje šetača

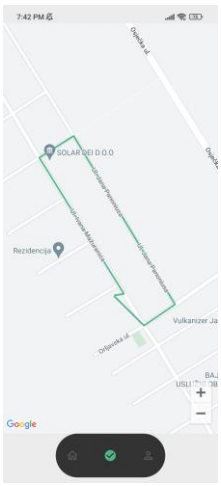
Pritiskom na gumb „Prati šetača“ korisniku je prikazan zaslon sa mapom te trenutnom lokacijom šetača. Pritiskom na zelenu oznaku na karti otvara se prozor sa imenom šetača. Na slici 5.17 vidljiv je zaslon koji prikazuje trenutnu lokaciju šetača.



Sl. 5.17. Trenutna lokacija šetača

5.10. Pregled rute šetnje

Kada je šetnja završena i korisnik pritisne gumb „Pregled šetnje“ vidljiv na slici 5.16 prikazuje se zaslon sa mapom te rutom šetnje iscrtanom na toj mapi. Zaslon rute šetnje je vidljiv na slici 5.18.



Sl. 5.18. Ruta šetnje

5.11. Profil šetača

Pritiskom na sliku određenog šetača na slici 5.4 prikazuje se profil šetača koji se sastoji od opisa, broja šetnji i prosječne ocijene. Zaslon profila šetača je vidljiv na slici 5.19.

Sl. 5.19. Profil šetača

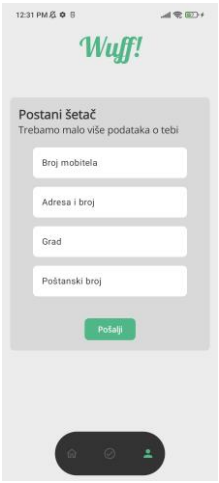
5.12. Zaslon vlastitog profila

Pritiskom na zadnju opciju u navigaciji koja se nalazi na dnu aplikacije prikazuje se vlastiti profil korisnika. Unutar vlastitog profila korisnik može odabrati gumb „Izmijeni“ kako bi izmijenio podatke koji se nalaze pod „O meni“.

Sl. 5.20. Vlastiti profil

5.13. Zaslon postani šetač

Ukoliko korisnik želi postati šetač tada je potrebno pritisnuti gumb „Postani šetač“ koji se nalazi na vlastitom profilu i vidljiv je na slici 5.20. Pritiskom na navedeni gumb korisniku će biti prikazan zaslon koji je vidljiv na slici 5.21. Korisnik mora ispuniti određene detalje o sebi koji služe kao sigurnosni sistem. Kako bi korisnik postao šetač administrator mora odobriti prijavu.



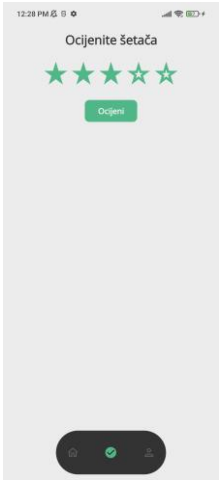
Sl. 5.21. Zaslon postani šetač

5.14. Zaslon ocjene šetača

Kako bi korisnik ocijenio šetača potrebno je odabrati šetnju koja je završila i odabrati „Ocijenite šetnju“. Sučelje koje prikazuje detalje šetnje i gumb za ocjenjivanje vidljiv je na slici 5.22.

Sl. 5.22.Detalji šetnje

Prilikom ocjenjivanja šetača, potrebno je odabrati odgovarajući broj zvjezdica i zatim pritisnuti gumb „Ocijeni“ koji je prikazan na slici 5.23.



Sl. 5.23. Zaslon ocijene šetača

6. ZAKLJUČAK

U ovom završnom radu opisana je implementaciju Android mobilne aplikacije koja spaja ljude koji nemaju vremena za šetanje kućnih ljubimaca i one koji vole šetnje. Mobilne aplikacije danas su ključne za olakšavanje svakodnevnih aktivnosti, a ova aplikacija nudi korisnicima praktično rješenje za zbrinjavanje kućnih ljubimaca dok istovremeno omogućuje šetačima da ostvare dodatni prihod.

Aplikacija koja je opisana u ovom završnom radu pomaže ljudima koji vole pse i često provode vrijeme u zatvorenim prostorima na način da im pruža mogućnost šetnje pasa. Također pomaže ljudima koji žive užurbani životni stil te imaju ograničeno vrijeme za posvetiti se svojim krznenim prijateljima. Aplikacija omogućuje korisnicima rezervaciju odgovarajuće vrste šetnje za njihovog krznenog prijatelja.

U usporedbi sa već dostupnim aplikacijama, koje su slične onoj koja je opisana u ovom završnom radu, postoji puno proširenja i poboljšanja koja bi se mogla realizirati. Neka od proširenja uključuju kontakt šetača putem poruke ili poziva unutar aplikacije, čime se korisnicima olakšava komunikacija sa šetačem i njihovim krznenim prijateljem. Također će biti unaprijeđeni sigurnosni mehanizmi prilikom registracije novih šetača kako bi se osigurala viša razina sigurnosti. Konačno, korisničko sučelje će biti poboljšano i učinjeno estetski privlačnijim i intuitivnijim za korištenje.

LITERATURA

[1] A. Štrelov, Nova i stroža pravila za napuštanje, mučenje i ubijanje životinja [online], Hrvatska radiotelevizija, 2024., dostupno na: https://magazin.hrt.hr/price-iz-hrvatske/nova-i-stroza-pravila-za-napustanje-mucenje-i-ubijanje-zivotinja-11449978. [20.6.2024.].

[2] I. Petak, Svaki pas treba izaći u šetnju svaki dan [online], Argos, 2020., dostupno na: https://argos.hr/kucni-ljubimci/svaki-pas-treba-izaci-u-setnju-svaki-dan/. [20.6.2024.].

[3] Pets Only, Profesionalne usluge brige, čuvanja i šetanja vaših kućnih ljubimaca! [online], PetsOnly, 2024, dostupno na: https://petsonly.hr/ [1.7.2024.]

[4] EXCELLENT WEBWORLD, Wag App [online], EXCELLENT WEBWORLD, 2024., dostupno na: https://www.excellentwebworld.com/best-app-of-the-week/wag-app/. [21.6.2024.].

[5] Barkly Pets, Barkly Pets [online], Pet Care Innovation, 2024, dostupno na: https://petcareinnovation.net/portfolio/barkly-pets/ [1.7.2024.]

[6] Apple App Store, Rover—Dog Sitters & Walkers [online], Rover, 2024., dostupno na: https://apps.apple.com/gb/app/rover-dog-sitters-walkers/id547320928. [21.6.2024.].

[7] Gowalkies, The app that connects dog walkers with dog owners [online], gowalkies, 2024, dostupno na: https://gowalkies.co.uk/

[8] J. Callaham, The history of Android: The evolution of the biggest mobile OS in the world [online], Authority Media, 2024., dostupno na: https://www.androidauthority.com/history-android-os-name-789433/. [21.6.2024.].

[9] Google For Developers, Platform architecture [online], Google, 2024., dostupno na: https://developer.android.com/guide/platform. [21.6.2024.].

[10] KotlinLang, Kotlin Multiplatform [online], JetBrains, 2024., dostupno na: https://kotlinlang.org/docs/multiplatform.html [21.6.2024.].

[11] G. F. Developers, Thinking in Compose [online], Google, 2024., dostupno na: https://developer.android.com/develop/ui/compose/mental-model. [21.6.2024.].

[12] GeeksForGeeks, MVVM (Model View ViewModel) Architecture Pattern in Android [online], GeeksForGeeks, 2022., dostupno na: https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/. [23.6.2024.].

[13] B. Karia, A quick intro to Dependency Injection: what it is, and when to use it [online], freeCodeCamp, 2018., dostupno na: https://www.freecodecamp.org/news/a-quick-intro-to-dependency-injection-what-it-is-and-when-to-use-it-7578c84fa88f/. [23.6.2024.].

[14] G. F. Developers, Dependency injection with Hilt [online], Google, 2024., dostupno na: https://developer.android.com/training/dependency-injection/hilt-android. [23.6.2024.].

[15] D. Stevenson, What is Firebase? The complete story, abridged. [online], Medium, 2018., dostupno na: https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0. [23.6.2024.].

[16] K. Bracey, What is Figma? [online], envatotuts+, 2024., dostpuno na: https://webdesign.tutsplus.com/what-is-figma--cms-32272a. [23.6.2024.].

[17] R. Murphy, What Is Stripe, and How Does It Work to Accept Payments? [online], nerdwallet, 2023., dostupno na: https://www.nerdwallet.com/article/small-business/what-is-stripe . [23.6.2024.].

[18] J. Gage, What does Vercel do? [online], Vercel, 2023., dostupno na: https://vercel.com/blog/what-is-vercel. [23.6.2024.].

[19] B. Semah, What Exactly is Node.js? Explained for Beginners [online], freeCodeCamp, 2022., dostupno na: https://www.freecodecamp.org/news/what-is-node-js/. [23.6.2024.].

SAŽETAK

U modernom svijetu obilježenom užurbanošću, ljudi često imaju malo vremena za sebe i svoje bližnje, što može dovesti do zapostavljanja njihovih najboljih krznenih prijatelja. U ovom završnom radu opisana je implementacija rješenja takvog problema.

Aplikacija je razvijena korištenjem Kotlin programskog jezika, Jetpack Compose alata za izgled korisničkog sučelja i Stripe poslužitelja za naplaćivanje. Aplikacija omogućuje prijavu i registraciju putem adrese e-pošte i zaporke ili korištenjem Google računa, pronalazak šetača koji se nalaze u istom mjestu kao i korisnik, rezervaciju i plaćanje usluga šetnje. Svaki korisnik ima mogućnost pregleda rute održanih šetnji kao i praćenje lokacije šetača u stvarnom vremenu. Ocjenjivanje šetača je moguće putem aplikacije kako bi svi vlasnici pronašli odgovarajuću osobu za svog kućnog ljubimca.

Uspoređujući aplikaciju opisanu u ovom završnom radu s postojećim sličnim aplikacijama, identificiraju se brojna proširenja i poboljšanja. Ta proširenja uključuju mogućnost kontakta šetača putem poruke ili poziva unutar same aplikacije radi olakšane komunikacije između korisnika i šetača, kao i njihovih kućnih ljubimaca. Također, predloženo je implementiranje naprednijeg sigurnosnog mehanizma pri registraciji novih šetača kako bi se osigurala veća razina sigurnosti. Nadalje, planira se poboljšanje korisničkog sučelja kako bi bilo intuitivnije i estetski privlačnije korisnicima.

Ključne riječi: Android, Firebase, Jetpack Compose, Kotlin, MVVM, Stripe

ABSTRACT

Android application for connecting dog owners and dog walkers

In the modern world characterized by fast-paced living, people often have little time for themselves and their loved ones, which can lead to the neglect of their best furry friends. This thesis describes the implementation of a solution to such a problem.

The application was developed using Kotlin programming language, Jetpack Compose tool for user interface layout and Stripe payment provider. The application enables login and registration via email address and password or using a Google account, finding walkers who are in the same place as the user, booking and paying for walking services. Each user has the option of viewing the route of the walks as well as tracking the location of the walkers in real time. Reviewing of walkers is possible through the app so all owners can find the right person for their pet.

By comparing the application described in this thesis with existing similar applications, a number of improvements are identified. These improvements include the ability to contact walkers via message or call within the app itself to facilitate communication between users and walkers, as well as their pets. It was also proposed to implement a more advanced security mechanism when registering new walkers to ensure a higher level of security. Furthermore, it is planned to improve the user interface to make it more intuitive and aesthetically appealing to users.

Keywords: Android, Firebase, Jetpack Compose, Kotlin, MVVM, Stripe

ŽIVOTOPIS

Filip Toprek rođen je u Osijeku, 21. svibnja 2002. godine. Završio je Osnovnu školu Tenja, nakon koje upisuje Elektrotehničku i prometnu školu u Osijeku, smjer tehničar za računarstvo. Akademske godine 2021./2022. upisao je stručni prijediplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek. Uz studij je odradio praksu kao web developer.

_____

Potpis autora