SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6559

Mobilna aplikacija za dopisivanje zasnovana na radnom okviru Flutter

Dominik Mesek

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6559

Mobilna aplikacija za dopisivanje zasnovana na radnom okviru Flutter

Dominik Mesek

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 13. ožujka 2020.

ZAVRŠNI ZADATAK br. 6559

Pristupnik:	**Dominik Mesek (0036501330)**
Studij:	Računarstvo
Modul:	Programsko inženjerstvo i informacijski sustavi
Mentor:	izv. prof. dr. sc. Alan Jović

Zadatak: **Mobilna aplikacija za dopisivanje zasnovana na radnom okviru Flutter**

Opis zadatka:

U ovom završnom radu potrebno je proučiti i opisati značajke radnog okvira Flutter od tvrtke Google koji se koristi za razvoj mobilnih aplikacija na klijentskoj strani. Radni okvir Flutter potrebno je iskoristiti za razvoj mobilne aplikacije koja će podržati udaljeno dopisivanje. Računalna arhitektura sastojat će se od klijentske strane i baze podataka tipa NoSQL (npr. Firebase). U radu je potrebno kvalitativno i kvantitativno po pitanju responzivnosti usporediti radni okvir Flutter s drugim sličnim rješenjima (npr. React Native, Xamarin) te se osvrnuti na prednosti i nedostatke predloženog računalnog arhitekturnog rješenja.

Rok za predaju rada: 12. lipnja 2020.

Sadržaj

Uvod

Svijet moblnih aplikacija razvija se iz dana u dan. Zajedno s razvojem najpopularnijih mobilnih platformi, Androida i iOS-a, rastu i korisnički zahtjevi. Što su korisnički zahtjevi složeniji, tako razvoj i održavanje mobilnih aplikacija postaju sve teži i skuplji. Klijenti većinom traže razvoj mobilne aplikacije za obje platforme, tako da razvojni inžinjeri u suštini prevode cjelokupnu logiku aplikacije pisane u jeziku za jednu platformu u jezik za drugu platformu. Taj cjelokupan proces uzima puno vremena i novaca. Iz tog razloga, javila se potreba za višeplatformskim razvojem. Velike tvrtke kao što su Google, Microsoft i Facebook su izdale svoje rješenje koje je krenuo koristiti sve veći broj ljudi. Cilj ovog rada je upoznavanje s radnim okvirom Flutter razvijenim od tvrtke Google. Razlog odabira upravo ove tehnologije je njen nagli rast u popularnosti, odlična službena dokumentacija i sve veća zajednica razvojnih inžinjera.

1. Radni okvir Flutter

Flutter je besplatan Googleov radni okvir otvorenog koda koji je nastao 2017. godine. Pomoću Fluttera moguće je razvijati mobilne aplikacije za Android i iOS koristeći samo jedan *codebase*. Flutter između ostaloga dolazi uz vlastiti SDK (eng. *Software Development Kit*) koji uključuje razne alate, kao što je prevodilac koda pisanog u programskom jeziku Dart u nativni strojni jezik za obje platforme. Sam radni okvir sastoji se od kolekcije ponovno iskoristivih elemenata korisničkog sučelja zvanih *widgeti*.

1.1. *Widgeti*

Svaki *widget* je sam po sebi jednostavan i ima samo jednu funkcionalnost. Ukoliko se želi izgraditi kompleksniji *widget*, isti se mora omotati u drugi sa željenim svojstvima i to se tako može dalje nastaviti. Taj način gradnje posljedica je dobre programerske prakse prednosti sastava ispred nasljeđivanja (eng. *composition over inheritence*). Svaki *widget* ima funkciju *build* koja vraća stablo ili hijerarhiju *widgeta*. Jednom kada su svi *widgeti* dobro izdefinirani, radni okvir rekurzivno poziva funkciju *build* svakog *widgeta* te naposlijetku nastaje jedno stablo elemenata korisničkog sučelja.

1.2. Vrste *widgeta*

Postoje dvije vrste *widgeta*: oni sa stanjem (eng. *statefull widget*) i oni bez stanja (eng. *stateless widget*). *Widgeti* sa stanjem u popratnom objektu prate svoje stanje. Ukoliko se stanje u nekom trenutno promijeni, taj dio stabla *widgeta* će se ponovno izgraditi. Razvojni inžinjer ima mogućnost promijeniti stanje *widgeta* pozivom funkcije *set state.*

1.3. Postupak učitavanja *Hot reload*

Zasigurno najomiljenija funkcionalnost i osobitost Fluttera je takozvani *hot reload*. Jednom kada se Dartov kod prevede i aplikacija se pokrene, bilo na emulatoru, simulatoru ili fizičkom uređaju, razvojni inžinjer može raditi promjene u kodu, sačuvati te promjene i automatski vidjeti rezultat istih.

2. Programski jezik Dart

Dart je objektno-orijentiran programski jezik prestavljen u listopadu 2011. godine. Jezik ima podržan sakupljač smeća. Po mnogima podsjeća na JavaScript, što nije čudno s obzirom da je jedan od osnivača Darta Lars Bak upravo poznat po tome što je stručnjak JavaScripta. Dart se može prevesti u strojni kod ili u JavaScript.

2.1. Zašto Dart?

Pitanje koje se svakako može postaviti jest zašto je Google odabrao baš Dart kao jezik za Flutter. To pitanje je osobito zanimljivo, jer Google ima već dobro razvijen jezik koji se koristi za razvoj Android aplikacija – Kotlin. Jedan od glavnih razloga je taj što uz Dart nisu potrebna dodatna pomagala i jezici za izradu grafičkog sučelja. Pomoću Dartovog alata obavlja se automatsko formatiranje kojim kod postaje iznimno čitak i jasan prilikom izrade grafičkog sučelja. Još jedan presudan razlog odabira upravo Darta je mogućnost prevodioca da prevodi program "ispred vremena" (eng. *Ahead of Time*) kao i "na vrijeme" (eng. *Just in Time*). Na taj način se tijekom razvoja koristi prevođenje "na vrijeme" kako bi se značajno ubrzao proces prevođenja. Jednom kada je aplikacija spremna za izdavanje ona se prevodi "ispred vremena". Ta iznimna moć prevođenja programa "na vrijeme" iskorištena je u Flutteru za razvoj funkcionalnosti *hot reload* [1].

3. Usporedba radnih okvira Flutter i React Native

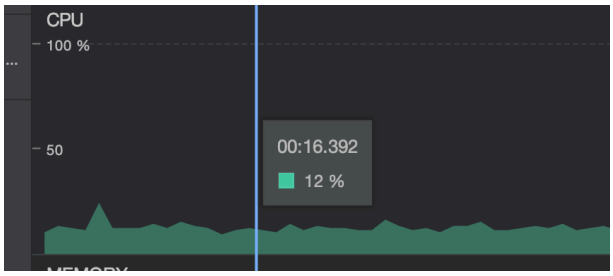
Flutter nije jedino rješenje za stvaranje višeplatformskih aplikacija. Postoje mnogi radni okviri koji konkuriraju Flutteru, kao što su Ionic, Native Script, Microsoftov Xamarin i React Native. U svrhu usporedbe sa Flutterom odabran je Facebookov React Native.

3.1. Aplikacija za testiranje

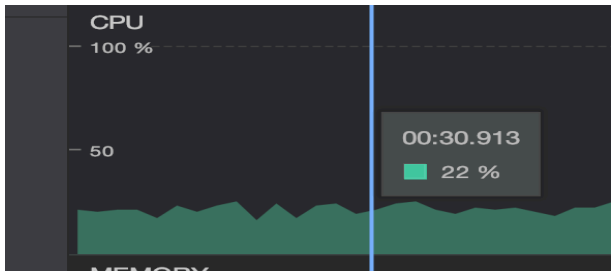
Kako bi uspješno usporedili Flutter i React Native potrebno je izgraditi identičnu aplikaciju za oba radna okvira. U tu svrhu odabrana je vrlo jednostavna aplikacija s tekstom na sredini zaslona. Taj tekst prikazuje proteklo vrijeme u sekundama i milisekundama od pokretanja aplikacije. Ta informacija ažurira se svakih 10 milisekundi kako bi simulirala konstantno opterećenje grafičkog sučelja. Izvorni kod za React Native projekt pruzet je s portala Thoughbot [2]. Sva mjerenja obavljena su u alatu za profiliranje Android Studia. Bitno je naglasiti kako je svaka inačica aplikacije pokrenuta u načinu rada *debug* na emulatoru Android uređaja s istog računala.

3.1.1. Opterećenje procesora

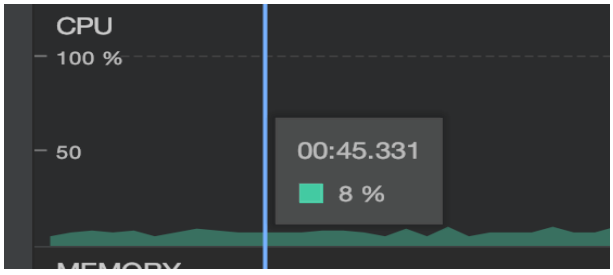
Prvi parametar koji je uzet u obzir ijekom ovog testiranja je opterećenje procesora. Prilikom ovog testiranja, Flutter je pokazao zavidne performanse (sl. 3.1), koje nisu daleko niti od nativne Androidove aplikacije (sl. 3.3), dok je React Native na začelju sa najlošijim rezultatom (sl 3.2).



Sl 3.1 Opterećenje procesora kod Fluttera



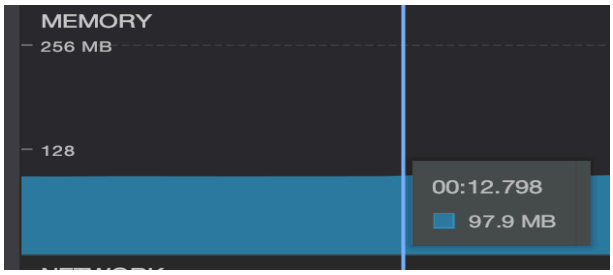
Sl 3.2 Opterećenje procesora kod React Nativea



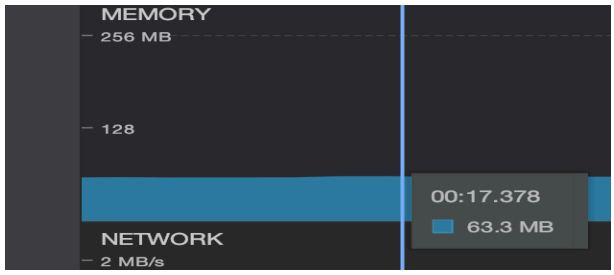
Sl 3.3 Opterećenje procesora kod Androida

3.1.2. Opterećenje memorije

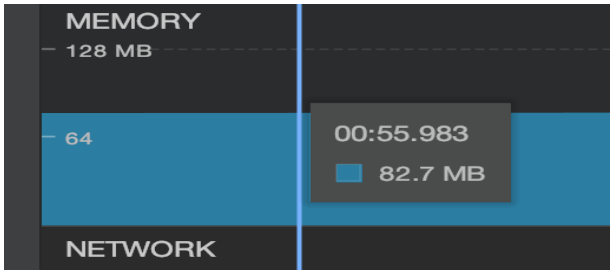
Drugi razmatrani parametar je opterećenje memorije. Rezultati ovog testiranja su poprilično zanimljivi. Kao što se da vidjeti iz slika 3.4, 3.5 i 3.6, React Native ima manje opterećenje memorije čak i od samog Androida. Razlog tomu jest već gore spomenut način rada *debug*. Usprkos tome, iz dobivenih podataka možemo zaključiti kako će opterećenje memorije tijekom razvoja u React Nativeu biti najmanje, što donosi bolje performanse.



Sl 3.4 Opterećenje memorije kod Flutteru



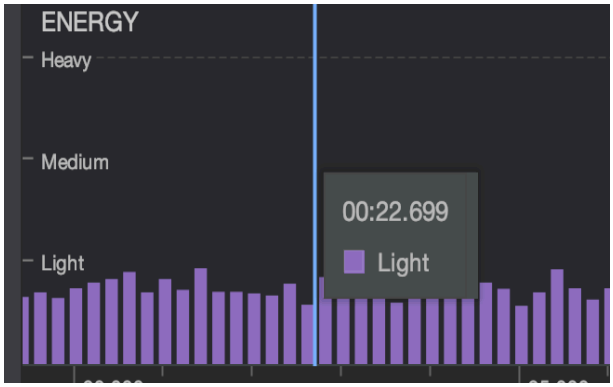
Sl 3.5 Opterećenje memorije kod React Nativea



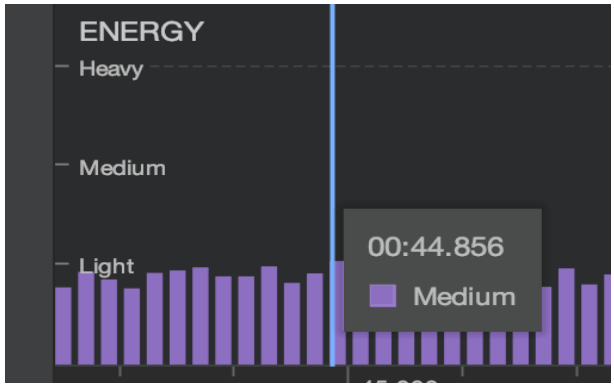
Sl 3.6 Opterećenje memorije kod Androida

3.1.3. Potrošnja baterije

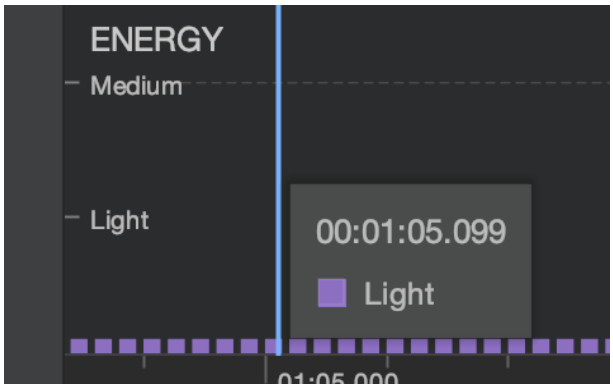
Također značajan parametar koji je proučavan u ovom ispitivanju je potrošnja baterije. Iz priloženih grafova na slikama 3.7, 3.8 i 3.9 možemo vidjeti značajnu razliku u potrošnji baterije između nativnog i višeplatformskog rješenja. Također je vidljiva nešto manja potrošnja kod Fluttera u odnosu na React Native.



Sl 3.7 Potrošnja energije kod Fluttera



Sl 3.8 Potrošnja energije kod React Nativea



Sl 3.9 Potrošnja energije kod Androida

3.1.4. Vrijeme izgradnje projekta

Posljednji parametar koji je uzet u obzir tijekom ovog testiranja je vrijeme izgradnje projekta. Ova faza testiranja ispitivala je dva slučaja: kada projekt prethodno nije izgrađen te kada projekt jest nekada prije izgrađen. Kod Fluttera, potpuna izgradnja projekta trajala je 10 sekundi, što je za sekundu brže od konkurentnog React Nativea. Što se tiče djelomične izgradnje projekta, Flutter je također bio brži sa impresivnih 1.5 sekundi, dok je izgradnja kod React Nativea trajala 4 sekunde.

3.1.5. Zaključak testiranja

Provedeno ispitivanje nije uspjelo prikazati značajnu razliku između ovih radnih okvira. Dok se Flutter pokazao boljim u procesorskom opterećenju, React Native je bio značajno ekonomičniji kod opterećenja memorije. Iako je Flutter također bio nešto bolji kod potrošnje baterije i izgradnje projekta, te razlike nikako nisu presudne u odabiru radnog okvira. Svaki razvojni inžinjer bi mogao imati drugačije mišljenje ovisno o svojem prethodnom iskustvu. Razvojni inžinjer za web bi najvjerojatnije kao prvi izbor odabrao neki radni okvir baziran na JavaScriptu, kao što su React Native i Native Script, dok bi s druge strane razvojni inženjer za platformu .NET vjerojatno odabrao Xamarin. Također će odabir radnog okvira ovisiti o potrebama projekta, jer svaki radni okvir ima neke prednosti i nedostatke.

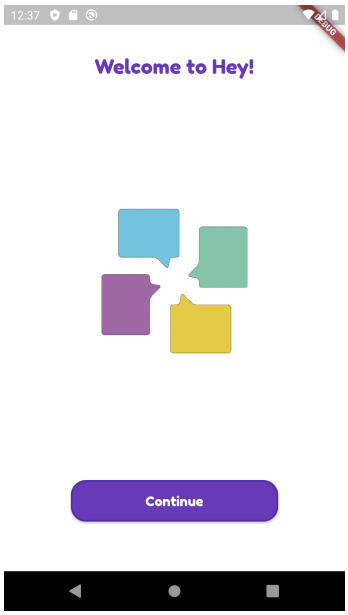
4. Korisnički opis aplikacije Hey!

U okviru završnog rada napravljena je višeplatformska mobilna aplikacija za udaljeno dopisivanje po imenu Hey! Aplikacija je napravljena kako bi se pokazalo da su Flutter mobilne aplikacije učinkovite i brze poput nativnih mobilnih aplikacija.

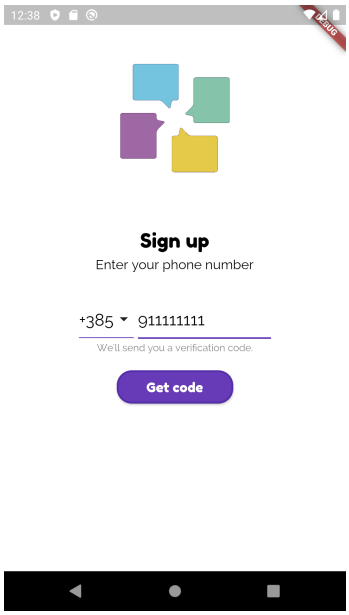
4.1. Podržane funkcionalnosti

4.1.1. Registracija

Prilikom prvog pokretanja aplikacije korisnik dolazi na početni zaslon (sl. 4.1.), te nakon pritiska na gumb s oznakom *Continue* biva proslijeđen na zaslon za unos telefonskog broja (sl. 4.2.). Pritiskom na inicijalni pozivni broj otvara se dialog za odabir pozivnog broja (sl. 4.3.). Zatim korisnik upisuje svoj telefonski broj te se pritiskom na gumb *Get code* otvara novi dialog (sl. 4.4) gdje korisnik mora potvrditi kako je unesen pravilan telefonski broj. U slučaju pogreške, poruka o pogrešci ispisuje se na zaslon.

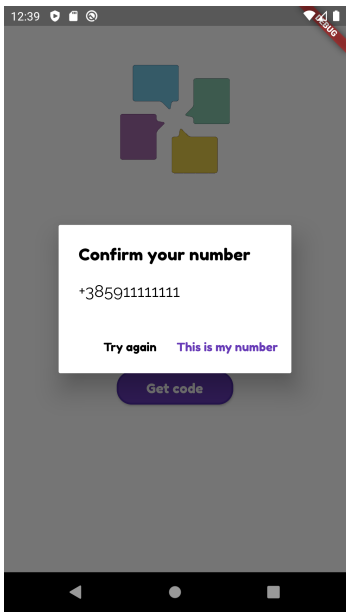


Sl. 4.1 Početni zaslon



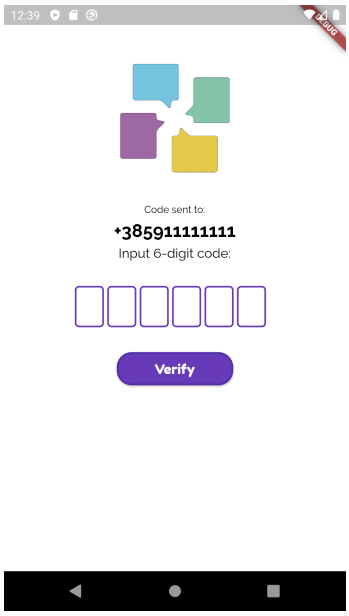
Sl. 4.2 Unos telefonskog broja

Sl. 4.3 Odabir pozivnog broja

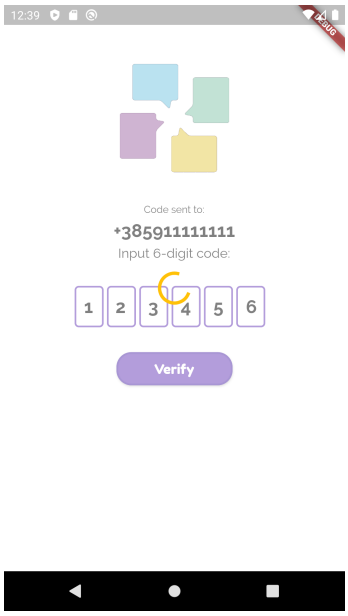


Sl. 4.4 Potvrda unesenog broja

Nakon pravilnog unosa telefonskog broja, korisnik dobiva SMS poruku s aktivacijskim kodom kojeg je potrebno upisati u odgovarajuće polje (sl. 4.5). U slučaju unosa pogrešnog koda, ispisuje se poruka o pogrešci korisniku. Nakon unosa odgovarajućeg šesteroznamenkastog aktivacijskog koda, korisnik je automatski registriran (sl 4.6).

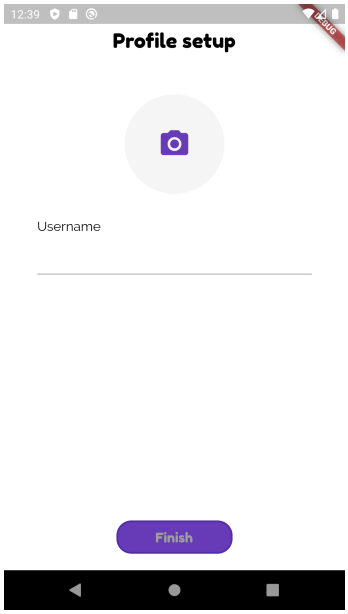


Sl. 4.5 Unos aktivacijskog koda

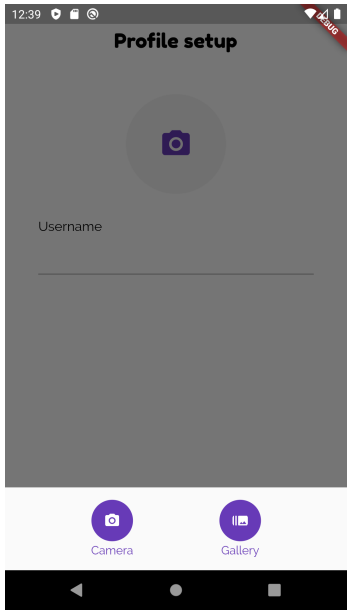


Sl. 4.6 Ispravan unos aktivacijskog koda

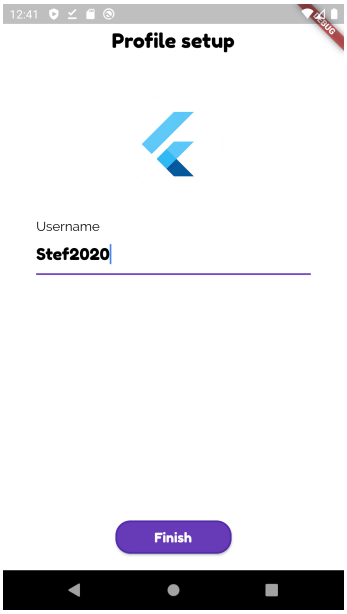
Na kraju registracijskog toka, korisnik je dužan unijeti korisničko ime te po želji korisničku fotografiju. Fotografija se može učitati iz galerije ili se može fotografirati.



Sl. 4.7 Zaslon postavljanja profila



Sl. 4.8 Odabir izvora fotografije

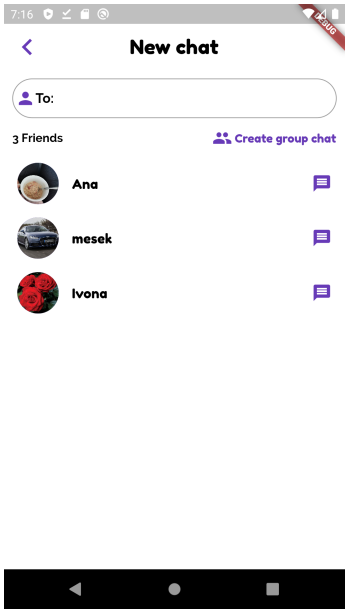


Sl. 4.9 Uspješno postavljanje profila

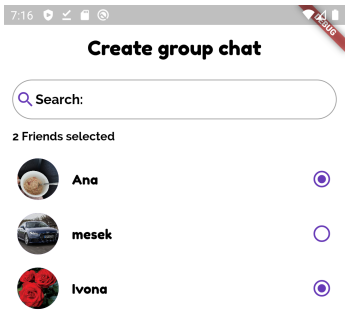
4.1.2. Udaljeno dopisivanje

Jednom kada je registracija uspješno obavljena, korisnik je u mogućnosti obaviti udaljeno dopisivanje s nekim kontaktom iz svog imenika koji također ima instaliranu aplikaciju. Pritiskom na gumb + korisnik dolazi na zaslon gdje odabire s kojim kontaktom želi započeti razgovor ili je u mogućnosti kreirati grupni razgovor (sl. 4.11). Korisnik je također u mogućnosti pretraživati kontakte.

Sl. 4.10 Glavni zaslon



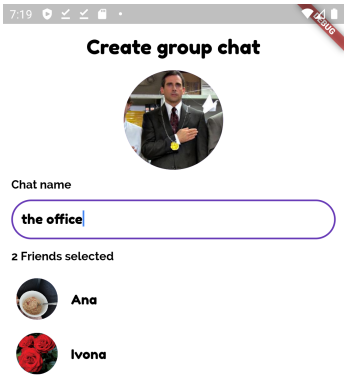
Sl. 4.11 Stvaranje novog razgovora

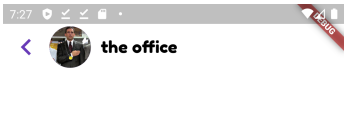


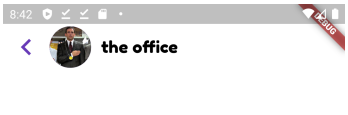
Ukoliko se korisnik odluči na stvaranje grupnog razgovora, mora pritisnuti gumb *Create group chat* te na novom zaslonu odabrati barem jednu osobu s kojom želi stupiti u kontakt (sl. 4.12). Nakon odabira kontakata za grupni razgovor, korisnik je dužan unijeti naziv grupe te sliku razgovora po želji (sl. 4.13).

Sl. 4.12 Odabir kontakata za grupni razgovor

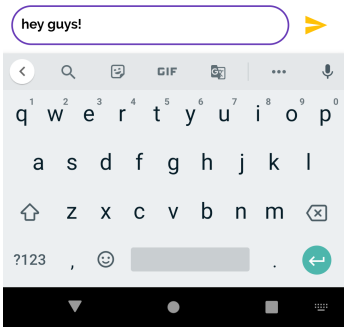
Jednom kada je stvoren novi razgovor, može započeti udaljeno dopisivanje. Slanje poruke obavlja se preko polja za unos teksta na dnu zaslona (sl. 4.14). Aplikacija također podržava bezmrežni način rada, tako da će poruka koja je poslana bez aktivne veze biti dostavljena u trenutku ponovne uspostave veze. Udaljeno dopisivanje izvršava se u stvarnom vremenu (sl. 4.15), tako da je korisnik u mogućnosti vidjeti poruku već nekoliko trenutaka nakon njenog slanja.



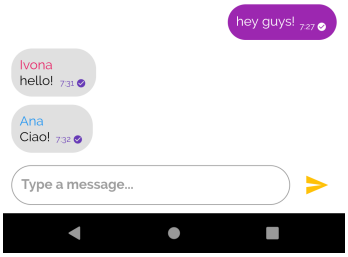




Sl. 4.13 Postavljanje naziva i grupne fotogrrafije

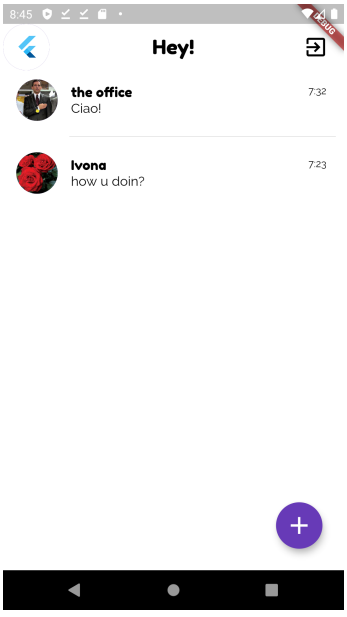


Sl. 4.14 Slanje nove poruke u grupni razgovor



Sl. 4.15 Primjer grupnog razgovora u stvarnom vremenu

Pregled i odabir već postojećih razgovora moguć je na glavnom zaslonu, gdje se uz ime razgovora prikazuje i posljednja poruka te vrijeme posljednje poruke (sl. 4.16). Prilikom ponovnog ulaska u aplikaciju korisnik je, ukoliko se prethodno nije odjavio pritiskom na gumb za odjavu, automatski proslijeđen na glavni zaslon aplikacije.



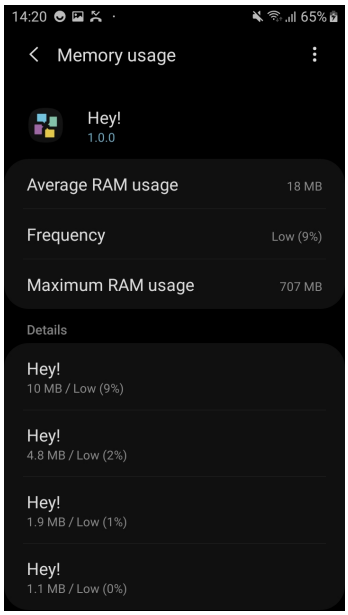
Sl. 4.16 Pregled postojećih razgovora

4.2. Ispitivanje performansi aplikacije Hey!

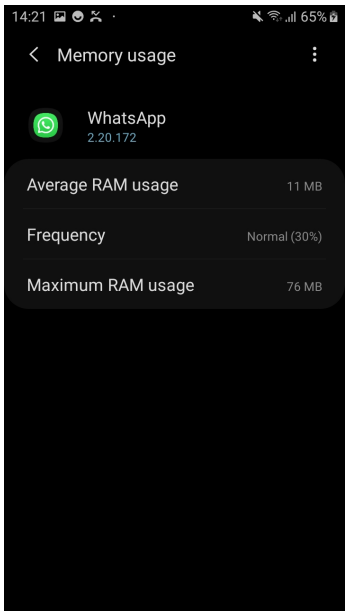
Kako bi se pokazala učinkovitost mobilne aplikacije ispitana su tri ključna parametra: zauzeće memorije, opterećenje procesora i potrošnja baterije.

4.2.1. Zauzeće memorije

Ispitivanje prosječnog zauzeća memorije napravljeno je tako da je aplikacija korištena neko vrijeme, zatim je provjereno prosječno zauzeće memorije odlaskom u postavke mobitela. Iz slike 4.17 vidljivo je kako je prosječno zauzeće aplikacije Hey! 18 MB, što je usporedbe radi 7 MB više od aplikacije WhatsApp (sl. 4.18). Razlika u zauzeću memorije nije značajna, no naravno treba uzeti u obzir kako je WhatsApp daleko složenija aplikacija ali je također i više poboljšana. Ekstremno visok iznos maksimalnog zauzeća memorije aplikacije Hey! je posljedica pokretanja alata za pronalazak pogrešaka (eng. *debugger*).



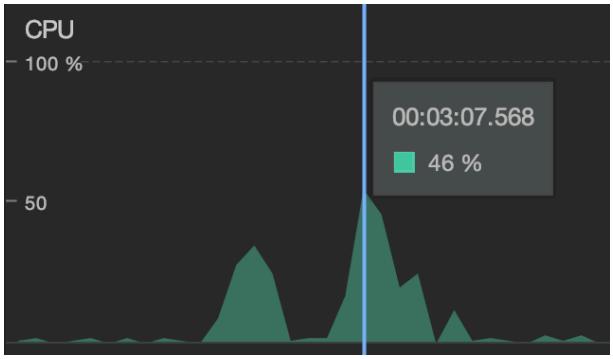
Sl. 4.17 Prosječno zauzeće memorije aplikacije Hey!



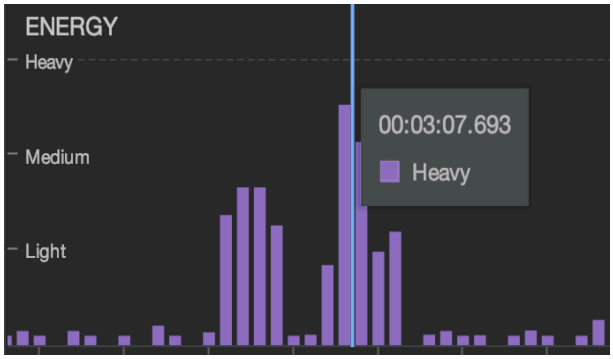
Sl. 4.18 Prosječno zauzeće memorije aplikacije WhatsApp

4.2.2. Potrošnja baterije i opterećenje procesora

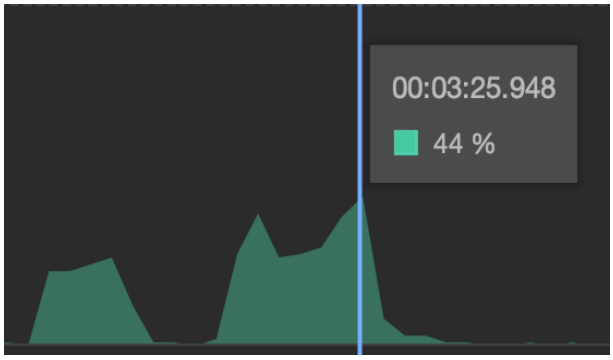
S obzirom da je potrošnja baterije proporcionalna opterećenju procesora, oba parametra će biti razmatrana istovremeno. Mjerenja sa slika 4.19, 4.20, 4.21 i 4.22 uzeta su tijekom izvršavanja dvije najčešće radnje aplikacije za udaljeno dopisivanje: pregled i slanje poruka. Rezultati mjerenja pokazuju relativno visoku potrošnju baterije tijekom pregleda i slanja poruka. Iako je potrošnja prilikom ove dvije operacije visoka, treba uzeti u obzir kako se ne radi o konstantnom, već trenutačnom opterećenju, tako da je prosjek potrošnje baterije puno manji.



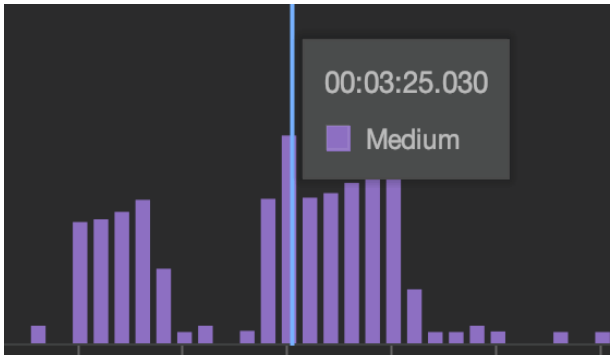
4.19 Opterećenje procesora za vrijeme slanja poruke



4.20 Potrošnja baterije za vrijeme slanja poruke



4.21 Opterećenje procesora za vrijeme pregleda poruka

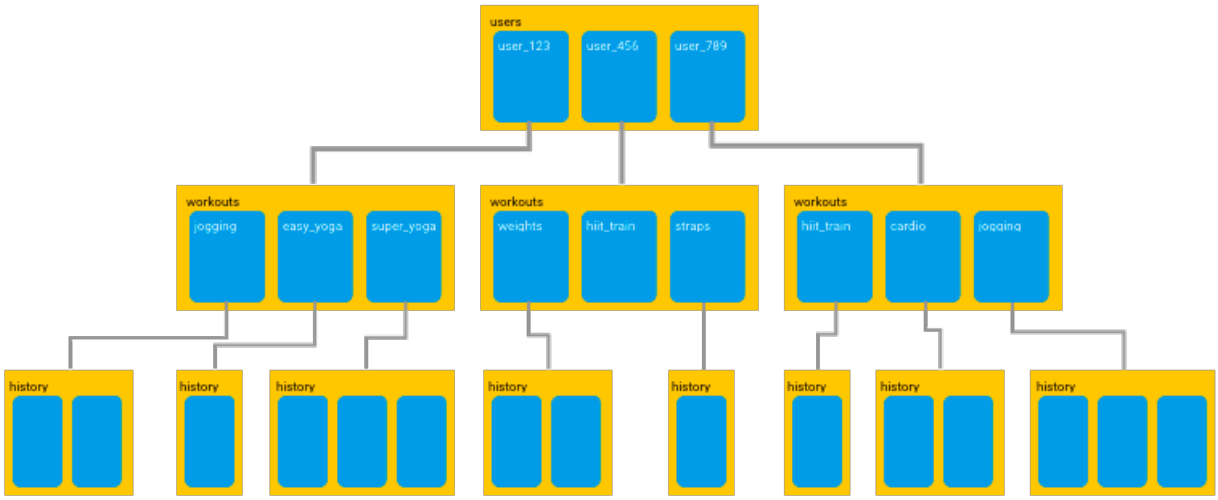


4.22 Potrošnja baterije za vrijeme pregleda poruka

5. Opis baze podataka

5.1. Nerelacijska baza podataka *Firebase Firestore*

Firestore je nerelacijska baza podataka u oblaku od tvrtke Google koja se može koristiti prilikom izrade Androidovih, iOS-ovih i web aplikacija [3]. Podaci se u *Firestoreu* spremaju u dokumente koji sadrže polja mapirana u konkretne vrijednosti. Dokumenti su pohranjeni u kolekcije koje se koriste za organizaciju podataka nad kojima se izvršavaju upiti. Dokumenti podržavaju razne tipove podataka, od najjednostavnijih *stringova* do kompleksnijih ugniježdenih objekata. Također je moguće stvaranje podkolekcija unutar dokumenata kako bi se stvorila skalabilna hijerarhijska struktura podataka vidljiva na slici 5.1 [4].

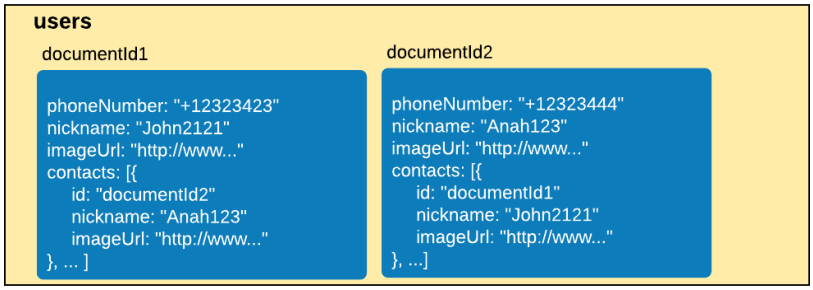


Sl. 5.1 Primjer hijerarhijske struture podataka

Upiti u *Cloud Firestoreu* su plitki, što znači da je moguće dohvatiti podatke na razini dokumenta, bez potrebe za dohvaćanjem bilo kakve kolekcije ili podkolekcije. *Firestore* ostvaruje dobre performanse upita tako što osigurava indeks za svaki upit. Posljedica toga je da performanse upita ne ovise o količini podataka u bazi, već o veličini rezultata upita. Automatski se za korisnika stvaraju indeksi za svako polje u dokumentu te za svako polje mape unutar dokumenta. U slučaju da indeks za neki složeni upit nije kreiran, *Firebase* korisniku javlja i predlaže kreiranje složenog indeksa.

5.2. Kolekcija korisnika

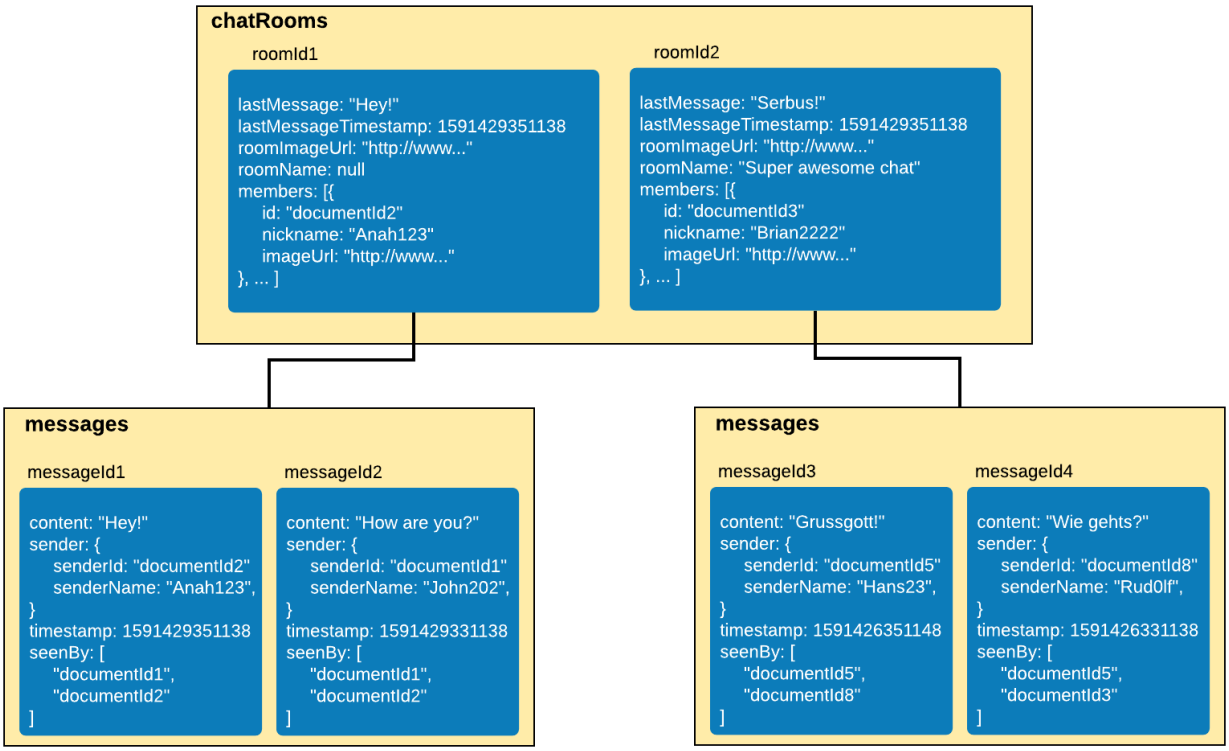
Mobilna aplikacija za udaljeno dopisivanje Hey! vodi evidenciju o registriranim korisnicima kao što je prikazano na slici 5.2. Svaki korisnik ima polje za nadimak (eng. *nickname*), putanje do slike profila (eng. *image url*), broj mobitela (eng. *phone number*) te listu kontakata (eng. *contacts*), točnije mapu s relevantnim informacijama o svakom kontaktu korisnika. Identifikator svakog korisnika ujedno je i identifikator pripadnog dokumenta.



Sl 5.2 Primjer kolekcije korisnika

5.3. Kolekcija razgovora

Kako bi se omogućilo udaljeno dopisivanje korisnika pohranjenih u istoimenoj kolekciji, stvorena je kolekcija razgovora. Svaki razgovor sadrži listu članova (eng. *member*), naziv sobe (eng. *room name*), poveznice do slike sobe (eng. *room image url*), zadnje poslane poruke (eng. *last message*), te vremenske oznake posljednje poslane poruke (eng. *last message timestamp*). Svaki razgovor sadrži podkolekciju poruka (eng. *messages*). Svaka poruka sastoji se od svog sadržaja (eng. *content*), vremenske oznake, identifikacije pošiljatelja (eng. *sender id*), imena pošiljatelja (eng. *sender name*) te liste identifikatora korisnika koji su pročitali pripadnu poruku (eng. *seen by*). Primjer hijerarhije kolekcije razgovora prikazan je slikom 5.3.



Sl 5.3 Primjer kolekcije razgovora i podkolekcije poruka

6. Opis izrade mobilne aplikacije Hey!

Prilikom izrade mobilne aplikacije za udaljeno dopisivanje treba voditi računa o poprilično broju funkcionalnosti. U nastavku ovog odlomka bit će opisane sve funkcionalnosti koje podržava ova aplikacija, te neke implementacijske pojedinosti. Jedna od navedenih funkcionalnosti bit će detaljnije objašnjena u nastavku rada.

6.1. Funkcionalnosti mobilne aplikacije

6.1.1. Registracija

Registracija korisnika obavlja se autentikacijom telefonskog broja putem SMS poruke. Navedena funkcionalnost ostvarena je pomoću paketa *Firebase Authentication*. Kako bi se koristila navedena funkcionalnost Googleovog *Firebasea*, potrebno je omogućiti prijavu putem telefonskog broja u konzoli *Firebasea*.

6.1.2. Postavljanje i uređivanje profila

Prilikom postavljanja profila, korisnik ima mogućnost učitati fotografiju koju želi koristiti. Ta fotografija pohranjena je pomoću *Firebase Storagea*, a putanja do te fotografije se zajedno s telefonskim brojem i imenom sprema u *Cloud Firestore*.

6.1.3. Sinkronizacija kontakata

Kako bi korisnik mogao kontaktirati poznanike iz svog telefonskog imenika, potrebno je za svakog korisnika pratiti kontakte sa aktivnom aplikacijom. U tu svrhu koristi se gotova biblioteka *contacts_service* za čitanje imenika. Na Androidu pa tako i na uređajima iOS, potrebno je zatražiti dopuštenje za čitanje imenika, pa se u tu svrhu koristi biblioteka *permission_handler*. Nakon što su kontakti iz imenika pročitani, na *Cloud Firestore* se šalje upit prema kolekciji korisnika koji provjerava postoji li korisnik s određenim telefonskim brojem. Ukoliko je odgovor potvrdan, kontakt biva pohranjen u listu dostupnih kontakata za korisnika.

6.1.4. Stvaranje grupnog razgovora

Stvaranje grupnog razgovora implementirano je na jednostavan način. Nakon što korisnik odabere željene članove grupe te postavi sliku i naziv grupe, u kolekciji *chatRooms* na *Cloud Firestoreu* stvara se novi dokument s postavljenim vrijednostima i automatski generiranim identifikatorom.

6.1.5. Stvaranje privatnog razgovora

Prilikom odabira željenog kontakta za privatan razgovor, u kolekciji *chatRooms* na *Cloud Firestoreu* stvara se novi dokument u kojem su članovi trenutni korisnik i odabrani kontakt. Kako bi se izbjeglo višestruko stvaranje privatnih razgovora s istim članovima, identifikator sobe stvara se kombinacijom identifikatora članova tog privatnog razgovora,. Ostale vrijednosti dokumenta postavljene su na *null*, jer slika i naziv sobe ovise o trenutnom korisniku.

6.1.6. Pregled postojećih soba za dopisivanje

Glavni zaslon aplikacije zadužen je za prikaz postojećih soba za dopisivanje. Te sobe se dohvaćaju slanjem upita na *Cloud Firestore*, gdje se iz kolekcije *chatRooms* povlače svi razgovori u kojima je sudionik s jednakim identifikatorom kao što je identifikator trenutnog korisnika. Ukoliko je neka soba za razgovor, bila ona privatna ili grupna, stvorena, ali nije poslana niti jedna poruka, ista se neće prikazati na glavnom zaslonu.

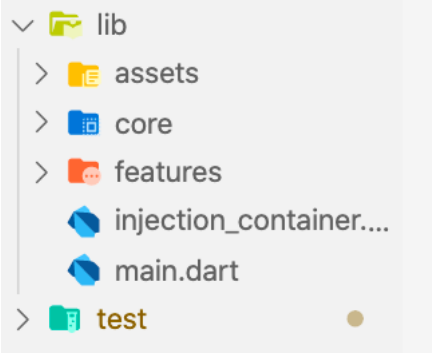
6.1.7. Udaljeno dopisivanje

Aplikacija Hey! podržava isključivo tekstni sadržaj poruke. Poruka se upisuje u polje za unos teksta te se pritiskom na gumb za slanje stvara novi dokument u podkolekciji *messages* unutar odgovarajućeg dokumenta kolekcije *chatRooms*. Uz sam sadržaj poruke šalje se i vremenska oznaka u milisekundama, u polje *sender* upisuje se identifikator i ime korisnika te se u listu *seenBy* također identifikator korisnika. Valja napomenuti kako se prilikom ulaska u sobu za razgovor ažurira vrijednost liste *seenBy* u dokumentu svake poruke gdje identifikator korisnika nije prisutan kako bi se označilo da je korisnik pročitao sve poruke tog razgovora.

6.2. Struktura projekta

Vršni direktorij svakog Flutter projekta je direktorij *lib* (sl. 6.1). Uz njega automatski je kreiran i direktorij *test* u kojemu su smješteni testovi. Unutar direktorija *lib* smještena su tri glavna direktorija: *assets*, *core* i *features*. U direktoriju *assets* smještene su sve slike i fontovi korišteni u aplikaciji, kao i svi *stringovi* korišteni kao oznake radi eventualne lakše lokalizacije. Direktorij *core* sadrži sav programski kod koji se koristi na više odvojenih mjesta i konfiguracijske datoteke, kao

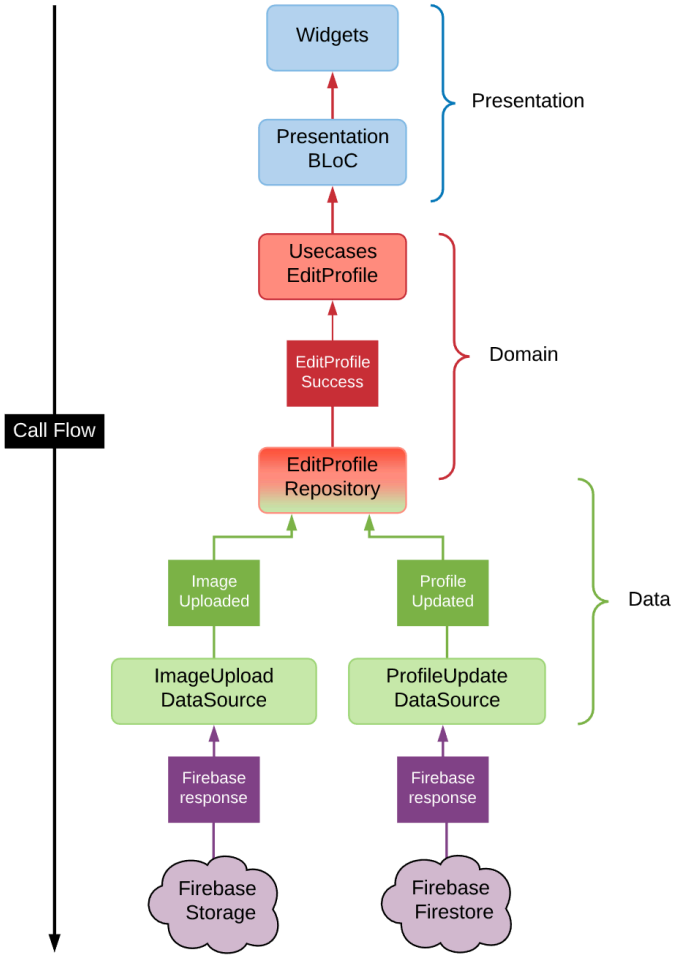
što je *app_theme.dart* gdje su definirane korištene boje, fontovi i veličine raznih stilova. Direktorij *Features* sadrži posebne direktorije za svaku od navedenih funkcionalnosti.



Sl. 6.1 Vršni direktoriji

6.3. Arhitektura

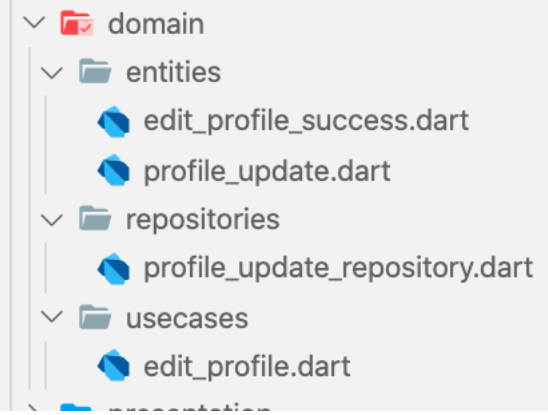
Projekt je pisan koristeći čistu arhitekturu koja teži odvajanju koda u nezavisne slojeve, koje ne ovise o konkretnim implementacijama, već o njihovim apstrakcijama. Tvorac ove arhitekture je Robert C. Martin, koji je izdao mnoge knjige o pisanju dobrog, ekspresivnog, čistog i proširivog koda. Arhitektura korištena u ovom projektu ne slijedi u potpunosti sve smjernice čiste arhitekture, već je ona izvedenica primjenjena na radni okvir Flutter. Prateći takvu slojevitu arhitekturu, svaka funkcionalnost podijeljena je u tri sloja: prezentacijski sloj, podatkovni sloj i sloj domene. Na primjeru dijagrama međuzavisnosti sa slike 6.2 [5] funkcionalnosti postavljanja/ažuriranja profila može se uočiti slojevitost arhitekture.



Sl 6.2 Dijagram međuzavisnosti funkcionalnosti ažuriranja profila

6.3.1. Sloj domene

Sloj domene je centralni sloj ovakve arhitekture. Struktura direktorija prikazana je slikom 6.3. Njegova je zadaća povezati prezentacijski i podatkovni sloj, ali u isto vrijeme biti nezavisan od oba sloja. To je postignuto obrtanjem zavisnosti (eng. *dependency inversion)*. Domenski sloj definira ugovor koji repozitorij podatkovnog sloja mora zadovoljiti. Konkretno kod uređivanja profila, repozitorij podatkovnog sloja mora implementirati funkciju *updateProfile* koja će ažurirati profil te ovisno o rezultatu vratiti pogrešku ili model *EditProfileSuccess* koji označava da je sve prošlo u redu (sl. 6.4.). Također, sloj domene sadrži obrasce uporabe (eng. *usecase*) i entitete koji se koriste. Jedini obrazac uporabe u ovoj funkcionalnosti je uređivanje profila (eng. *edit profile*), kojeg poziva prezentacijski sloj.



Sl. 6.3. Struktura direktorija domenskog sloja

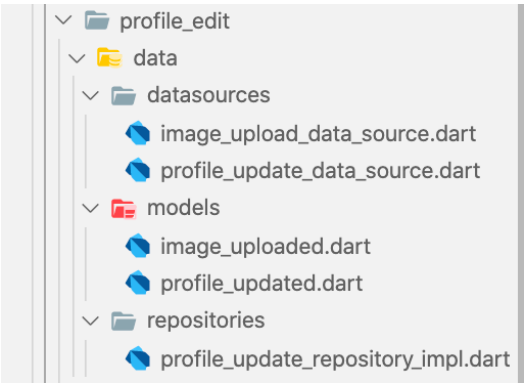
```dart
abstract class ProfileUpdateRepository {
  Future<Either<Failure, EditProfileSuccess>> updateProfile(ProfileUpdate profileUpdate);
}
```

Sl. 6.4. Isječak koda ugovora domenskog sloja

6.3.2. Podatkovni sloj

Uloga podatkovnog sloja je izravna komunikacija s vanjskim svijetom. To može biti lokalna baza podataka, neka udaljena usluga ili poslužitelj. Podatkovni sloj sastoji se od jednog ili više izvora

podataka (eng. *data source*), repozitorija i modela koje vraćaju izvori podataka, kao što je vidljivo na slici 6.5. Zadaća repozitorija podatkovnog sloja je da dobivene vrijednosti iz izvora podataka zapakira u entitete iz sloja domene. Druga zadaća repozitorija je da hvata sve eventualno bačene iznimke te umjesto njih vraća pogreške u sloj više razine (sl. 6.6).



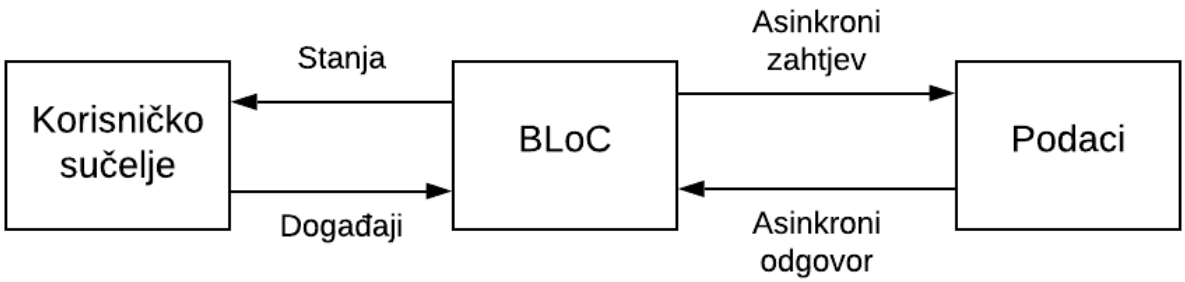
Sl. 6.5 Struktura direktorija podatkovnog sloja

```dart
@override
Future<Either<Failure, EditProfileSuccess>> updateProfile(
    ProfileUpdate profileUpdate) async {
  final isConnected = await networkInfo.isConnected;
  if (!isConnected) return Left(NetworkFailure());
  try {
    final imageUploaded = await imageUploadDataSource.uploadImage(profileUpdate.image);
    await userStorageDataSource.updateUser(userImageUrl: imageUploaded.imageUrl, userNickname: profileUpdate.nickname);
    return Right(EditProfileSuccess());
  } on ImageUploadException {
    return Left(ImageUploadFailure());
  } on ProfileUpdateException {
    return Left(ProfileUpdateFailure());
  }
}
```

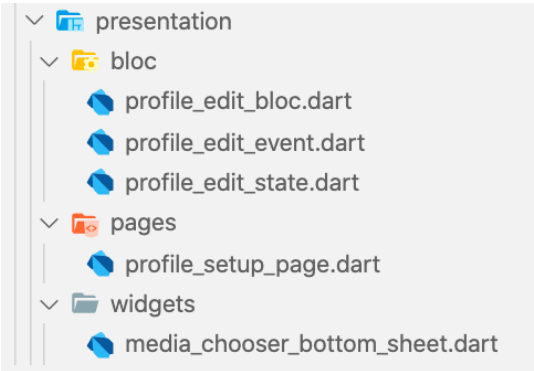
Sl. 6.6 Isječak koda repozitorija podatkovnog sloja

6.3.3. Prezentacijski sloj

Prezentacijski sloj je u izravnoj interakciji s korisnikom. Zadaća ovog sloja je pravilno prikazivanje dobivenih podataka s nižih slojeva. Kao način upravljanja stanja za ovaj projekt odabrana je biblioteka BLoC (eng. *Business Logic Component*). Princip njenog rada prikazan je na slici 6.7. Programski jezik Dart podržava asinkrone tokove podataka. Ta činjenica iskorištena je u biblioteci BLoC. *Widgeti* koji se prikazuju korisniku na zaslonu u stanju su slati događaje u asinkroni tok podataka, taj događaj okida određeni obrazac uporabe iz sloja domene, eventualno formatira podatke na adekvatan način za prikaz u korisničkom sučelju, te naposlijetku podatke vraća unutar stanja pozivajućem *widgetu*. Na taj način je efikasno odvojeno upravljanje stanja grafičkog sučelja od njegovog prikazivanja na zaslonu. Datotečna struktura prikazana je na slici 6.8.



Sl 6.7. Princip rada BLoC upravitelja stanja



Sl 6.8. Datotečna struktura prezentacijskog sloja

7. Razvoj programske potpore temeljen na ispitivanju

Aplikacija je razvijena primjenom razvoja programske potpore temeljenog na ispitivanju (eng. *Test Driven Development*), što znači da su ispitni slučajevi pisani prije konkretne implementacije. Ispitni slučajevi ispituju jednu najmanju jedinicu rješenja (eng. *Unit testing)*. Svaki ispitni slučaj koji je napisan inicijalno padne, a zatim se napiše minimalna količina koda kako bi isti prošao.

7.1. Pozitivne strane razvoja programske potpore temeljenog na ispitivanju

- Dokumentacija
  - Dobro napisani ispitni slučajevi služe kao svojevrsna dokumentacija koda. Iz samog sadržaja ispitnog slučaja treba biti jasno što se ispituje, koji su parametri te što se očekuje kao rezultat.
- Osiguravanje integriteta rješenja
  - Razvoj programske potpore je dinamičan proces. Zahtjevi se često mijenjaju i nadograđuju. Jednostavnim pokretanjem ispitivanja nakon većeg ili manjeg mijenjanja koda može se provjeriti je li neki dio prestao funkcionirati kao posljedica promjena koda.
- Opisivanje korisničkih zahtjeva
- Sigurnost
  - Nakon pisanja dobro strukturiranog ispitnog slučaja i uspješnog prolaska istog, razvojni inžinjer može biti siguran kako napisani kod neće prouzročiti neku vrstu pogreške kasnije tijekom razvoja.

7.2. Negativne strane razvoja programske potpore temeljenog na ispitivanju

- Dodatan kod za održavanje

- Prilikom preimenovanja nekog razreda, atributa ili parametra koji se ispituje u nekom ispitnom slučaju, potrebno je ručno preimenovati naziv ispitnog slučaja kako bi odgovarao novom nazivlju.

• Vrijeme

- Sigurno najveći razlog slabog korištenja ove prakse je vrijeme. Razvojni inžinjer provodi znatnu količinu vremena razmišljajući o adekvatnim ispitnim slučajevima, a zatim pisajući ih.

Zaključak

U programskom inžinjerstvu ne postoji najbolje riješenje. Prilikom izrade svakog projekta razvojni inžinjer mora razmisliti kakav programski proizvod želi isporučiti i s kakvim karakteristikama. Višeplatformske aplikacije nisu iznimka. Ne može se jednoznačno odrediti koji radni okvir je najbolji. Također će uvijek postojati preferenca svakog pojedinog razvojnog inžinjera prema nekom radnom okviru ili programskom jeziku. Višeplatformski radni okviri još zasigurno nisu dosegli svoj vrhunac, na temelju saznanja prikupljenih tijekom izrade ovog rada, vjerujem kako njihovo vrijeme tek dolazi.

Literatura

1. Wm Leler, *Why Flutter uses Dart*, Hackernoon (2017, prosinac). Poveznica: https://hackernoon.com/why-flutter-uses-dart-dd635a054ebf; pristupljeno 8. lipnja 2020.
2. Alex Sullivan, *Examining performance differences between Native, Flutter, and React Native mobile development*, Thoughbot, (2018, svibanj). Poveznica: https://thoughtbot.com/blog/examining-performance-differences-between-native-flutter-and-react-native-mobile-development; pristupljeno 8. lipnja 2020.
3. *Cloud Firestore*, Google Developers, (2020, travanj). Poveznica: https://firebase.google.com/docs/firestore; pristupljeno 6. lipnja 2020.

4. Francisco Garcia Sierra, *Working with Firestore: Building a simple database model,* (2018, ožujak), Poveznica: https://proandroiddev.com/working-with-firestore-building-a-simple-database-model-79a5ce2692cb*;* pristupljeno 6. lipnja 2020.
5. Matej Rešetar, *Clean Architecture & Flutter*, (2019, kolovoz), Poveznica: https://resocoder.com/2019/08/27/flutter-tdd-clean-architecture-course-1-explanation-project-structure/; pristupljeno 20. srpnja 2020.

Sažetak

Tema ovog završnog rada bila je proučiti i opisati značajke radnog okvira Flutter od tvrtke Google koji se koristi za razvoj mobilnih aplikacija na klijentskoj strani. U radu je dan kratak pregled radnog okvira kao i programskog jezika Dart. Radni okvir Flutter uspoređen je s radnim okvirom identične namjene React Nativeom. Kao rezultat rada razvijena je aplikacija za udaljeno dopisivanje po imenu “Hey!” u radnom okviru Flutter. U radu je detaljno opisana izrađena aplikacija kao i korištena arhitektura.

Summary

The theme of this thesis was to explore and describe the features of the Flutter framework made by Google, which is used to develop cross-platform mobile applications. The Flutter framework was compared with a similar framework for cross-platform development, React Native. As a result of this thesis, a chat application was developed in Flutter, named “Hey!”. The features of the developed app are throughly explained, as well as the used architecture.