SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

**Lucija Bagić,**

**Daria Alilović**

DOSTAVI FRENDE

**PROJEKT IZ KOLEGIJA ANALIZA I RAZVOJ PROGRAMA**

TEHNIČKA DOKUMENTACIJA

Varaždin, 2019.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

**Oznaka tima: AIR1822**

**Lucija Bagić**

**Daria Alilović**

**Link na github stranicu:** <https://github.com/lucbagic/Dostavi-frende>

**Trello link:** <https://trello.com/zastupanjesabora/home>

DOSTAVI FRENDE

**PROJEKT IZ KOLEGIJA ANALIZA I RAZVOJ PROGRAMA**

TEHNIČKA DOKUMENTACIJA

Mentor:

Doc. Dr. Sc. Zlatko Stapić

Varaždin, kolovoz 2019.

Sadržaj

[1. Uvod 1](#_Toc17644094)

[2. Specifikacija zahtjeva (eng. Software Requirements Specification) 2](#_Toc17644095)

[2.1. Svrha 2](#_Toc17644098)

[2.2. Kome je dokument namijenjen i kako ga koristiti 2](#_Toc17644099)

[2.3. Obujam aplikacije 2](#_Toc17644100)

[2.4. Perspektiva proizvoda 2](#_Toc17644101)

[2.5. Funkcionalnosti aplikacije 3](#_Toc17644102)

[2.6. Klase korisnika i njihove karakteristike 5](#_Toc17644103)

[2.7. Razvojno okruženje 5](#_Toc17644104)

[2.8. Ostali nefunkcionalni zahtjevi 6](#_Toc17644105)

[3. Specifikacija dizajna (eng. Software design specification) 7](#_Toc17644106)

[3.1. Arhitektura sustava 7](#_Toc17644108)

[3.2. Konceptualno i logičko oblikovanje 8](#_Toc17644109)

[3.2.1. Dijagram aktivnosti Pregled aktivnih korisnika 8](#_Toc17644110)

[3.2.2. Dijagram aktivnosti za Registraciju korisnika 10](#_Toc17644111)

[3.2.3. Dijagram aktivnosti za Prijavu korisnika 12](#_Toc17644112)

[3.2.4. Aktivnosti Profil korisnik i Odjava 14](#_Toc17644113)

[3.2.5. Aktivnost Chat 15](#_Toc17644114)

[3.2.6. Aktivnost Moji razgovori 16](#_Toc17644115)

[3.2.7. Dijagram aktivnosti Sklopljeni dogovori 18](#_Toc17644116)

[3.2.8. Dijagram aktivnosti Ponude usluge dostavljanja 19](#_Toc17644117)

[3.2.9. Aktivnost Pregled vlastitih usluga 21](#_Toc17644118)

[3.3. Dizajn modula 23](#_Toc17644119)

[4. Implementacija 24](#_Toc17644120)

[4.1. Implementacija Registracije 24](#_Toc17644122)

[4.2. Implementacija Prijave 26](#_Toc17644123)

[4.3. Implementacija Profil korisnika 28](#_Toc17644124)

[4.4. Implementacija Moji dogovori 30](#_Toc17644125)

[4.5. Implementacija Moje usluge 31](#_Toc17644126)

[4.6. Implementacija Pregled aktivnih/slobodnih korisnika 33](#_Toc17644127)

[4.7. Implementacija Chata 33](#_Toc17644128)

[4.8. Implementacija Ponuda usluge dostavljanja 33](#_Toc17644129)

1. Uvod

Kroz dokumentaciju se prikazuju sve važne informacije o ovom projektu, sadrži opis funkcionalnosti aplikacije, tehnologija i alati koji su potrebni za izradu aplikacije. Android aplikacija Dostavi frende namjenjena je slobodnoj upotrebi i bazirana je na konceptu ponude i potražnje. Aplikacija je pojednostavljeni oblik već postojećih aplikacija na tržištu koja omogućava dogovor oko dostavljanja namirnica ili nekih drugih potrepština na kućna vrata. Ovu aplikaciju korisnici mogu slobodno koristiti na svoju odgovornost kroz oslanjanje na pozitivne ocjene korisnika (npr. BlaBlaCar).

Kod izrade mobilne aplikacije odlučili smo se na korištenje najpopularnijeg pristupa koja je agilna metodologija razvoja ''SCRUM''. Proces je podjeljen na sprintove, a izrada ove aplikacije će trajati tri sprinta. Svaki sprint će biti razrađen u dogovoru između članova tima i podjednako će se podijeliti funkcionalnosti po svakom sprintu. Za provođenje navedene metodologije koristi se aplikacija Trello, slobodna za besplatno korištenje.

1. Specifikacija zahtjeva   
   (eng. Software Requirements Specification)
3. 1. Svrha

U ovom poglavlju specificiramo zahtjeve za aplikaciju Dostavi frende. Aplikacija je prva takva aplikacija koja dopušta slobodno korištenje na odgovornost korisnika na tržištu. Specifikacija je za kompletnu aplikaciju.

* 1. Kome je dokument namijenjen i kako ga koristiti

Dokument je prvenstveno namijenjen razvojnom timu, a i osobama koje imaju želju za dublje razumijevanje funkcioniranja aplikacije Dostavi frende, njezinog dizajna i arhitekture.

* 1. Obujam aplikacije

Aplikacija Dostavi frende dopušta pregled korisnika koji se ponude za izvršavanje usluge dostavljanja. Korisnik se može javiti svim dostupnim korisnicima i dogovoriti se putem chat-a za uslugu. Jednom kad se dogovor izvrši obje strane mogu potvrditi taj dogovor i pregledavati sve aktivne dogovore. Aktivni dogovor se završava ocijenjivanjem korisnika. Svaka osoba koja je prijavljena također može ponuditi dostavljanje i u bilo kojem trenutku se maknuti s te liste. Detaljniji opis svake funkcionalnosti se može vidjeti iz Tablica 1.

* 1. Perspektiva proizvoda

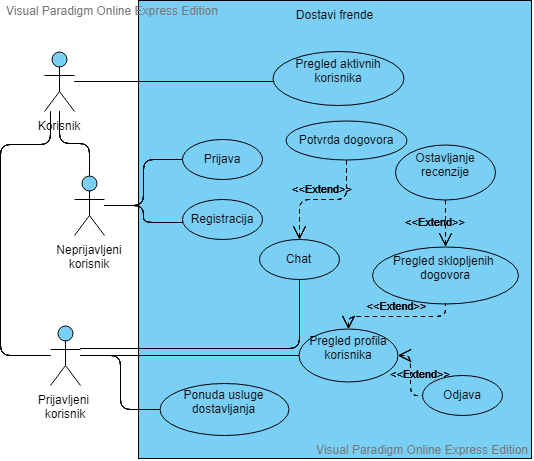
Na domaćem tržištu trenutno ne postoji masovno poznata aplikacija koja nudi opis usluga aplikacije Dostavi frende što je i razlog njenog razvoja. Ideja je bazirana na već postojećoj aplikaciji Postmates koja je dostupna za američko tržište. Aplikacija je dokazano uspješna i zato je dobra ideja kreirat vlastitu verziju koja bi mogla zaživjet na našem tržištu. Od naše strane aplikacija je potpuno novi proizvod i za razliku od američke verzije nije u vlasništvu nekog poduzeća već se čitav sustav temelji na međusobnom povjerenju i pozitivnim ocjenama. U ovoj verziji nitko nije zaposlenik već svaka osoba ima slobodu nuditi uslugu na temelju svog slobodnog vremena.

* 1. Funkcionalnosti aplikacije

U tablici su navedene sve funkcionalnosti koje su zahtjevane od korisničke strane. Svaka funkcionalnost sadrži opis kako bi trebala funkcionirati u konačnici. Ispod tablice funkcionalnosti je prikaz Use case dijagrama.

|  |  |  |
| --- | --- | --- |
|  | Funkcionalnosti | Opis |
| 1. | Prijava korisnika | Za korištenje aplikacije najprije potrebna je prijava korisnika. Ukoliko korisnik nije registriran može to obaviti klikom na registraciju. |
| 2. | Registracija korisnika | Registracijom se kreira novi korisnički račun spremanjem potrebnih informacija u bazu. |
| 3. | Odjava korisnika | Brisanje sesije korisnika. |
| 4. | Pregled aktivnih/slobodnih korisnika | Kada se korisnik prijavi omogućeno mu je pretraživanje slobodnih korisnika za određeni grad. Može vidjeti kada koji korisnik može obaviti mu željenu uslugu. Korisnik se može javiti u razgovor odabranoj osobi. Od korisnika još vidi i njegovu profilnu sliku i ostavljene ocijene. |
| 5. | Ponuda usluge dostavljanja | Također korisnik može ponuditi da obavi neku uslugu za određeni grad. |
| 6. | Chat | Chat razgovori služe za uspostavu komunikacije između korisnika za detaljne upute izvršavanja usluge i iz istog potvrđuje dogovorenu uslugu. |
| 7. | Recenzija korisnika | Prilikom uspješno ili neuspješno obavljene usluge korisnici međusobno moraju ostaviti ocijenu za obavljenu uslugu. |
| 8. | Profil korisnika | Klikom na profil korisnik može vidjet svoje informacije, profilnu sliku koju može mijenjati i gumb koji vodi na aktivnost gdje se vide sklopljeni dogovori. |
| 9. | Sklopljeni dogovori | Korisnik vidi ispis svih sklopljenih dogovora i od tamo za micanje istog mora ocijeniti korisnika po završetku dostave. |
| 10. | Potvrda dogovora obje strane putem chata | Obje strane nakon što se kroz razgovor dogovore o dostavi potvrđuju u chatu preko gumba „Dogovoreno“ dogovor. Jednom kad obje strane potvrde dogovor se prikaže u sklopljenim dogovorima do kojih se može doć preko profila. |
| 11. | Pregled vlastith usluga | Odabirom Moje usluge korisnik može pregledati sve usluge koje je ponudio. Samim tim može ih i obrisati ukoliko ne želi ili nije u mogućnosti da izvrši uslugu. |

Tablica 1. Popis funckionalnosti



Slika 1 Use case dijagram glavnih funkcionalnosti

* 1. Klase korisnika i njihove karakteristike

Korisnike dijelimo na registrirane korisnike i neregistrirane. Neregistriranom korisniku je vidljiv samo prikaz aktivnih korisnika za dostavu i ne može koristiti ostale dijelove aplikacije. Registrirani korisnik može obavljati sve funkcionalnosti aplikacije koje su prethodno navedene. Neregistrirani korisnik postaje registrirani nakon uspješne registracije putem obrasca.

* 1. Razvojno okruženje

Programski proizvod radi se u razvojnom okruženju Android Studio. Koriste se jezici Java, XML i Gradle Script te operacijski sustav Android za koji je namijenjea aplikacija. Aplikacija komunicira s bazom podataka. Za bazu podataka služit ćemo se Firebaseom. Za samo tesiranje aplikacije koristit ćemo Android Virtual Device sa verzijom Androida 8.0.0. i izravno testiranje preko Samsung Galaxy S9 uređaja.

* 1. Ostali nefunkcionalni zahtjevi

Nefukcionalni zahtjevi koji se zahtjevaju u aplikaciji su da nema kašnjenja s porukama, poruke se moraju isporučivati u brzini od par sekundi. Također svi ostali zahtjevi prema bazi i prikaz podataka se mora također odvijati u roku od nekoliko sekundi, tj. da nema velikih kašnjenja.

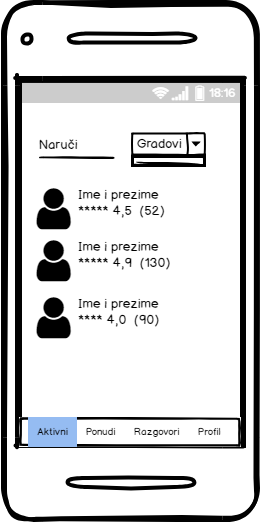
1. Specifikacija dizajna  
   (eng. Software design specification)
2. 1. Arhitektura sustava

Čitava arhitektura sustava se bazira na dvije sastavnice koje su mobilni uređaj i Firebase sustav. Dostavi frende za spremanje i dohvat podataka koristi Firebase. Firebase nudi jednostavno rješavanje zadataka kao što su prijava i registracija. U svojem sustavu pamti registrirane korisnike i također obavlja prijavu tih korisnika kroz svoju pripremljenu funkciju. Također pamti sesije prijavljenih korisnika i briše ju kod odjave. Nudi realtime bazu podataka za spremanje podataka i storage za spremanje slika.



Slika 2 Arhitektura sustava

* 1. Konceptualno i logičko oblikovanje

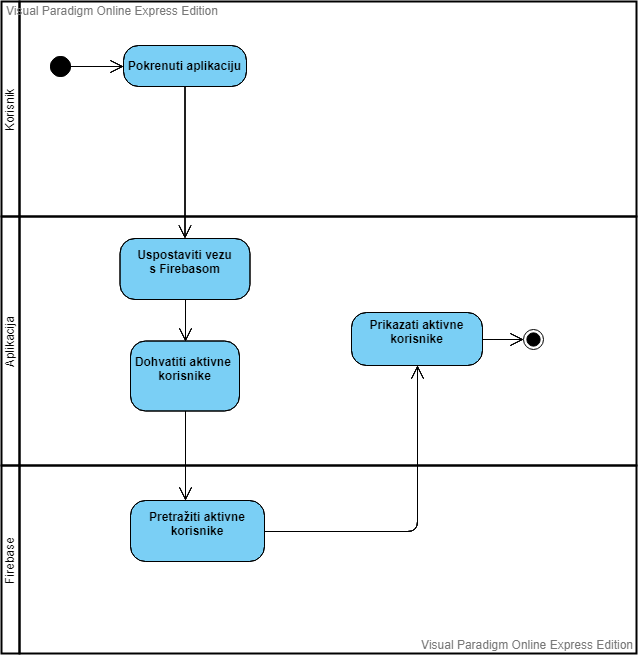
**Početni zaslon**

Slika 3 Mockup Početne stranice

Prilikom pokretanja aplikacija korisniku se prikazuje početni zaslon(Home Page) na kojem su vidljivi korisnici za grad za koji odabire sam korisnik. Prikazuju se profilne slike korisnika njihovo ime i prezime, te prosječna ocjena za sve obavljene zadatke. Korisniki su vidljivi i ostali dijelovi navigacije poput Ponudi, Razgovori te Profil. No ukoliko korisniki nije registriran klikom na jedan od tih tabova korisnika se preusmjerava ga na Prijavu ili Registraciju. Ukoliko korisnik nema kreiran korisnički račun najprije je potrebno obaviti registraciju, a potom prijavu za samo korištenje aplikacije.

* + 1. Dijagram aktivnosti Pregled aktivnih korisnika

Na sljedećoj slici prikazan je dijagram aktivnosti pregled aktivnih korisnika. Pregled aktivnih korisnika kao funkiconalnost aplikacije dostupna je svim korisnicma bez obzira da li imaju kreiran korisnički račun ili ne. Kada korisnik pokrene aplikaciju najprije u aplikaciji se uspostavlja veza s web servisom a u ovom slučaju to je Firebase. U aplikaciji se vrši dohvaćanje aktivnih korisnika. Dok na Firebase se pretražuju za dan kad se aplikacija koristi korisnici koji su stavili svoje ponude usluga. Nakon što se izvrši upit u aplikaciji se prikažu aktivni korisnici.



Slika 4 Dijagram aktivnosti Pregled aktivnih korisnika

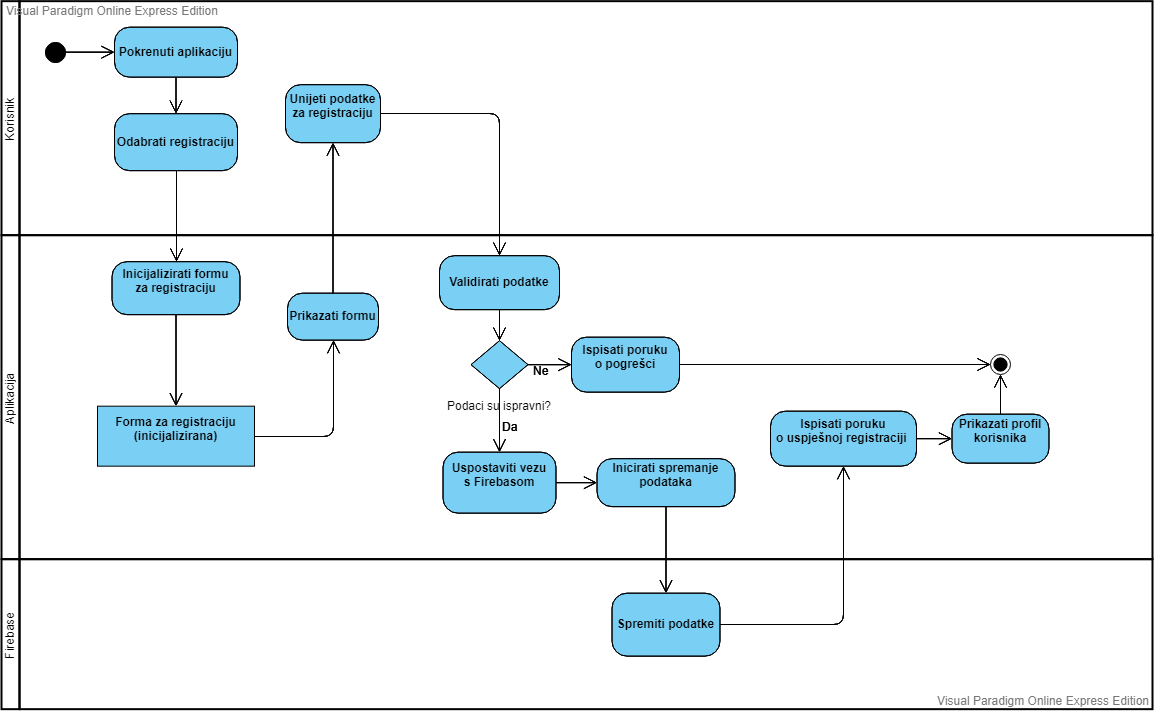
**Registracija korisnika**

Slika 5 Mockup Registracije

Registracija na aplikaciju Dostavi frende krajnje je jednostavna. Sve što je potrebno je popuniti osnovne podatke o korisniku Ime, Prezime, Godina rođenja, Email, Lozinka te Potvrda lozinke. Od korisnika se zahtjeva da ispuni sva polja registracije. Također registracijom korisnik potvrđuje da se slaže sa uvjetima korištenja aplikacije Dostavi frende. Klikom na gumb Registriraj se podaci se spremaju u Firebase bazu podataka, te korisniku omogućava automatsku prijavu u aplikaciju. Korisniku su vidljivi preostali zasloni koji su neregistriranim korisnicima nedostupni.

* + 1. Dijagram aktivnosti za Registraciju korisnika

Na slici je prikazan dijagram aktivnosti Registracije. Komunikacija se odvija između Korisnika, Aplikacije te web servisa Firebasea. Korisnik započinje aktivnosti pokretanjem same aplikacije. Zatim odabire registraciju. U aplikaciji se vrši inicijalizacija forme za registraciju. Objekt aktivnosti predstavlja da je forma inicijalizirana, te se prikaže forma korisniku. Nakon što je korisniku vidljiva forma za registraciju on unosi svoje podatke potrebne (ime, prezime, lozinka...). U aplikaciji se izvršava validacija unesenih podataka. U slučaju da su podaci ispravni sljedeći korak je uspostava veze s Firebaseom, no ukoliko podaci nisu ispravni ispisuje se poruka o pogrešci i tu aktivnost završava. Nakon uspostave veze s Firebaseom inicira se spremanje podataka u aplikaciji. Na Firebase se spremaju podaci, te se u aplikaciji korisniku ispisuje poruka o uspješnoj registraciji. Zatim se prikazuje korisnikov profil, gdje završava ta aktivnosti.



Slika 6 Dijagram aktivnosti za registraciju

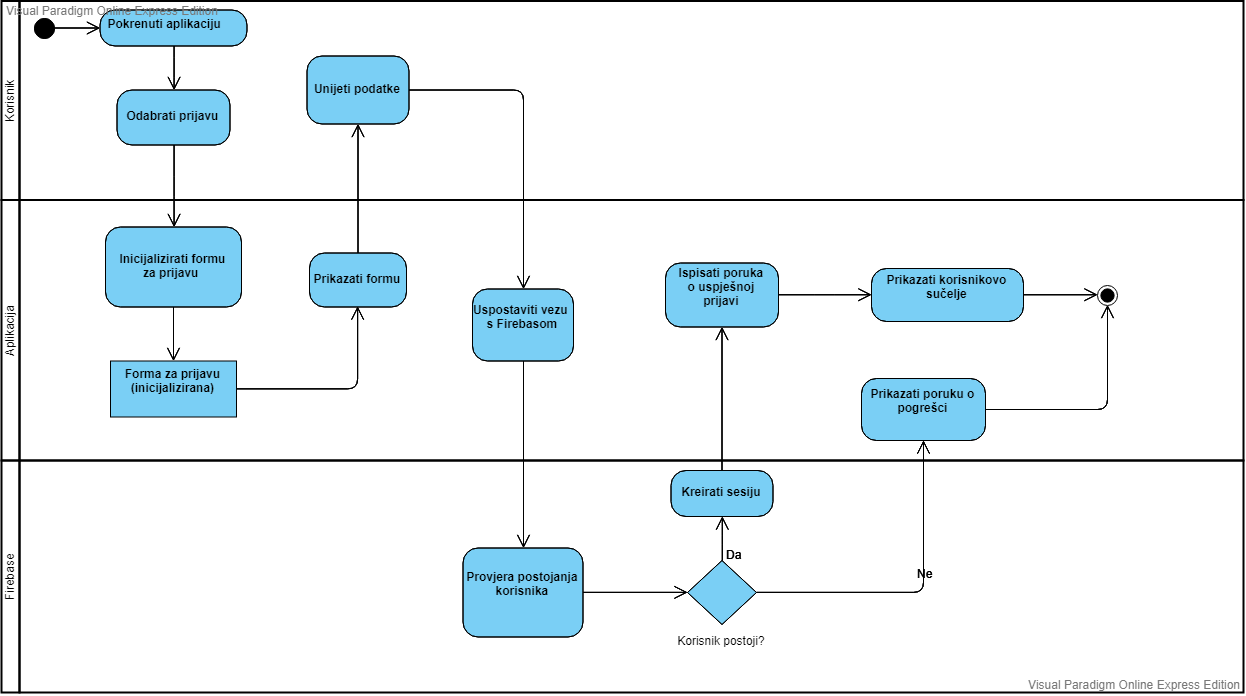
**Prijava korisnika**

Slika 7 Mocukp Prijave

Za prijavu korisnika potrebno je ispuniti polja email i lozinku. Firebase funkcija za prijavu provodi samu validaciju, te nije moguće se prijaviti ukoliko korisnik nema korisnički račun.Klikom na gumb korisnika se preusmjerava se na početnu stranicu za registriranog korisnika (omogućeni su mu razgovori, mogućnost prikaza vlastitog profila te da napravi ponudu.)

* + 1. Dijagram aktivnosti za Prijavu korisnika

Aktivnost prijava započinje korisnikovim pokretanjem aplikacije, zatim odabirom same prijave. Također sudionici ove aktivnosti su korisnik, aplikacija te Firebase. U aplikaciji se odvija inicijalizacija forme, te nakon što je inicijalizirana prikazuje se korisniku forma za registraciju. Korisnik unosi osobne podatke. Upostavlja se veza s Firebaseom u aplikaciji, te na strani Firebase se provjerava da li korisnik već postoji u bazi. Ukoliko korisnik postoji ispisuje se poruka o pogrešci, zapravo da već postoji takav korisnik u bazi, te završava aktivnost. U slučaju da ne postoji kreira se korisnička sesija i ispisuje se poruka u aplikaciji o uspješnoj prijavi. Zatim se prikazuje korisnikovo sučelje i tu završava ova aktivnost.



Slika 8 Dijagram aktivnosti prijave korisnika

**Profil korisnika i odjava**



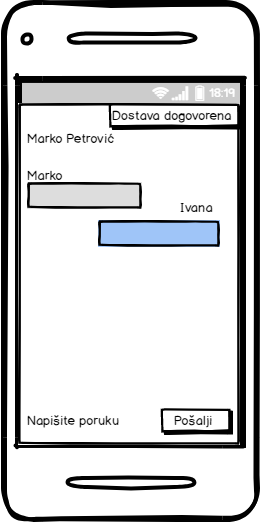
Slika 9 Mockup Profil korisnika

Svaki registirani korisnik ima mogućnost uvida u svoj profi. Na svom korisničkom profilu može dodati profilnu sliku, te je može izmjenjivati. Tu su podaci poput ime i prezime, email adresa, starosna dob, ocjena korisnika koje je dobio od drugih korisnika na temelju obavljenih dogovora. Također klikom na Moji dogovori ima mogućnost vidjet povijest dogovora s drugim korisnicima. Tu se još nalazi i gumb Odjavi se te klikom na njega prestaje aktivnost korisnika i briše se sesija.

* + 1. Aktivnosti Profil korisnik i Odjava

Aktivnost Profil korisnika omogućava prikaz korisniku njegovih podataka koji se se uspostavom veze s Firebaseom u aplikaciji. U bazi se pretražuju podaci korisnika. Aplikacija vraća rezultate upita te ih prikazuje korisniku.

Korisnik odabire odjavu, te se uspostavlja veza s Firebaseom. U njemu se briše sesija korisnika. Aplikacija vraća poruku korisniku da je odjava uspješno provedena.

**Chat**

Chat zaslon omogućava korisnicima izmjenjivanje poruka, gdje se međusobno mogu dogovoriti za detalje izvršavanja dogovorene usluge. Korisnici preko poruka dogovaraju vrstu usluge, cijenu, mjesto isporuke te ostale detalje. U gornje desnom kutu se nalazi gumb za potvrdu da je dostava dogovorena. Potrebno je da obe strane potvrde da je dostava dogovorena.

Slika 10 Mockup Chat

* + 1. Aktivnost Chat

Aktivnost Chat se odvija između dva korisnika aplikacije, same aplikacije te Firebasea. Ova aktivnost bit će naknadno detaljnije opisana.

****

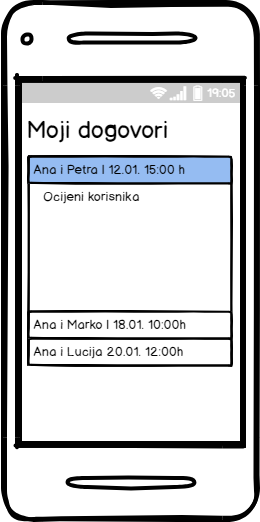
**Moji razgovori (Inbox)**

Korisniku je moguć uvid u njegove razgovore s ostalim korisnicima. Klikom na željenog korisnika prikaže se povijest razgovora. Moji razgovori predstavljaju zapravo Inbox svakog korisnika.

Slika 11 Mockup Moji razgovori

* + 1. Aktivnost Moji razgovori

Aktivnost Moji razgovori predstavlja popis svih razgovora koje je korisnik razmijenio s drugim korisnicima aplikacije. Korisnik odabire Moji razgovori te uspostavom veze s bazom podataka u aplikaciji šalje se zahtjev za dohvaćanjem korisnikovih poruka. Na Firebaseu se pretražuju poruke korisnika, te se u aplikaciji na zaslonu prikazuju korisniku. Prikazom korisnikovih poruka ova aktivnost završava.

****

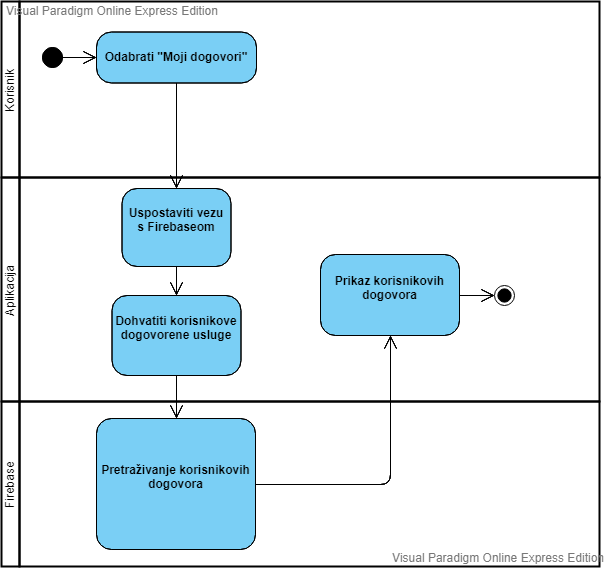
Slika 12 Mockup Moji dogovori

**Moji dogovori**

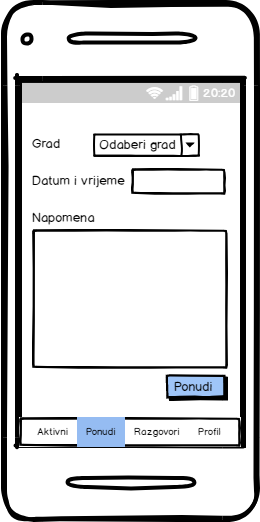
Moji dogovori su predstavlja popis svih dogovorenih usluga s drugim korisnicima. Prikazuje se s kojim korisnikom smo ostvarili suradnju te datum i vrijeme. Klikom na dogovor ponudi nam se opcija da ocijenimo korisnika ocjenom od 1 do 5. Potrebno je ocjeniti svakog korisnika kako bi se dogovor označio kao izvršena usluga. Te ocjenjivanjem dogovor se briše i uklanja s popisa Moji dogovori. Ocjenjivanje korisnika služi i tome da ostali korisnici imaju bolji uvid za ostale korisnike da li ispunjavaju odgovorno i na vrijeme zakazane usluge ili ne.

* + 1. Dijagram aktivnosti Sklopljeni dogovori

Sljedeća slika prikazuje dijagram aktivnosti za sklopljene dogovore. Korisnik odabire Moji dogovori a to je prikaz svih dogovora koje je korisnik sklopio s drugim korisnicima. Uspostavlja se veza s web servisom u aplikaciji te se dohvaćaju dogovorene usluge od korisnika koji je zatražio popis svojih dogovora. Na strani Firebasea pretražuju se korisnikovi dogovori, te se u aplikaciji nakon što se pronađu prikažu i korisniku. Ova aktivnost završava prikazom korisnikovih dogovora.



Slika 13 Dijagram aktivnosti Sklopljeni dogovori



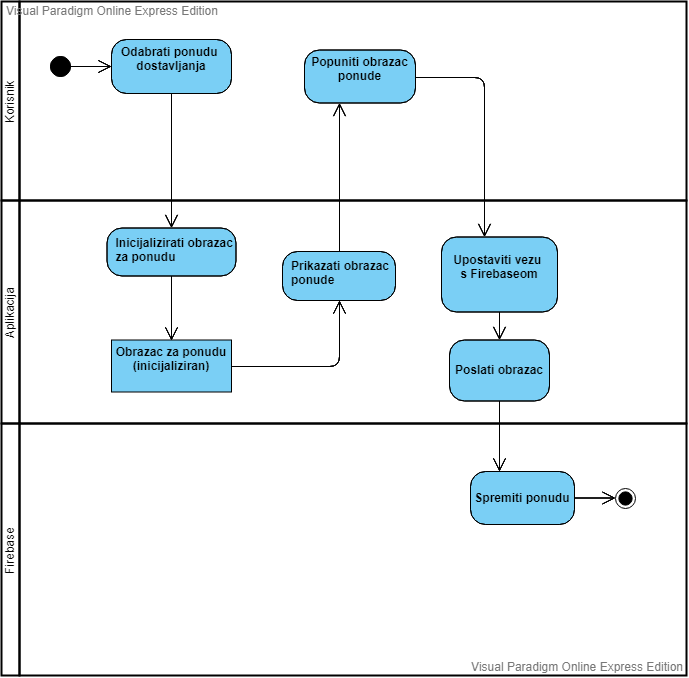
Slika 14 Mockup Ponuda usluge dostavljanja

**Ponuda usluge dostavljanja**

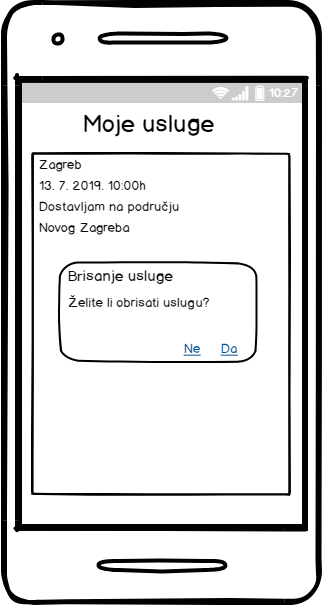
Kako bi korisnik mogao odabrati ponudu usluge dostavljanja najprije treba biti prijavljen u aplikaciju. Na ekranu se prikazuje forma koja sadrži potrebne informacije poput grada, datum i vrijeme izvršavanja usluge, te napomena u slučaju da korisnik ima neke dodatne informacije. Kada korisnik ispuni obrazac treba kliknuti gumb ponudi kako bi ponuda bila spremljena u bazu, te vidljiva ostalim korisnicima.

* + 1. Dijagram aktivnosti Ponude usluge dostavljanja

Na ovom dijagram aktivnosti prikazana je komunikacija između korisnika, aplikacije te Firebase za aktivnost ponude usluge dostavljanja. Korisnik na navigaciji odabire ponudi dostavljanje. U aplikaciji se vrši inicijalizacija obrazca za ponudu. Nakon inicijalizacije obrazca za ponudu, u aplikaciji se prikaže obrazac koji je vidljiv korisniku. Korisnik ispunjava polja kao što su grad, datum i vrijeme, napomena i sl. Uspostavlja se veza s Firebaseom, te se u aplikaciji šalje obrazac na bazu podataka. U bazi podataka koja se nalazi na Firebaseu spremaju se svi podaci i tu ova aktivnost završava.



Slika 15 Dijagram aktivnosti ponude usluge dostavljanja



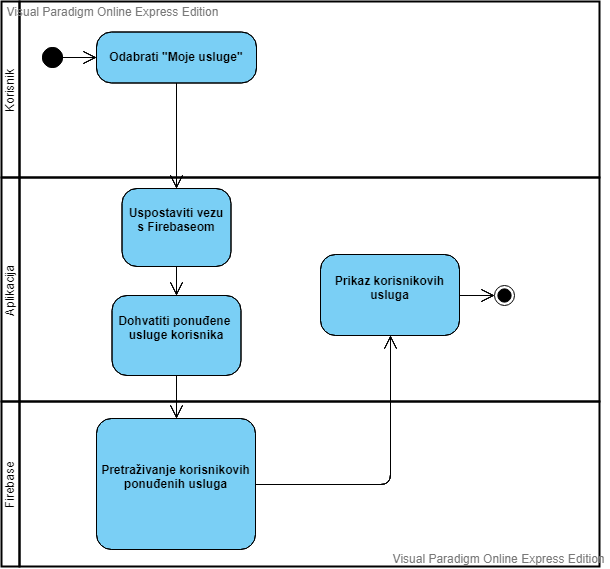
Slika 16 Mockup Moje usluge

**Pregled vlastitih usluga**

Usluge koje je korisnik ponudio može vidjeti pritiskom na gumb Moje usluge na vlastitom profilu. Nakon toga ispišu mu se sve usluge koje sadrža grad gdje se obavlja, zatim datum te napomenu. Prilikom odabira usluge moguće je obrisati iz popisa ukoliko korisnik ne želi da obavlja odabranu uslugu.

* + 1. Aktivnost Pregled vlastitih usluga

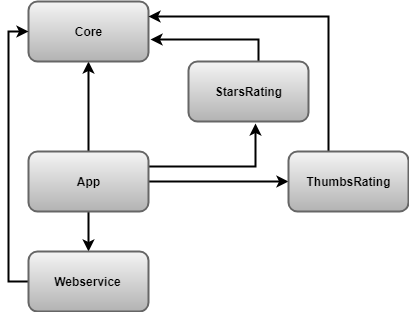
Na sljedećoj slici prikazan je dijagram aktivnosti za usluge koje je sam korisnik ponudio da obavlja. Korisnik odabire Moje usluge, te mu se prikažu sve one koje je ponudio. Uspostavlja se veza s web servisom u aplikaciji te se dohvaćaju usluge tog korisnika. Na strani Firebase se pretražuju usluge tog korisnika koji je poslao zahtjev. Prilikom pronalaska se prikažu korisnikove usluge. U slučaju da korisnik ne želi obavljati neku od usluga omogućeno mu je brisanje odabirom željene usluge.



Slika 17 Dijagram aktivnosti Moje usluge

* 1. Dizajn modula

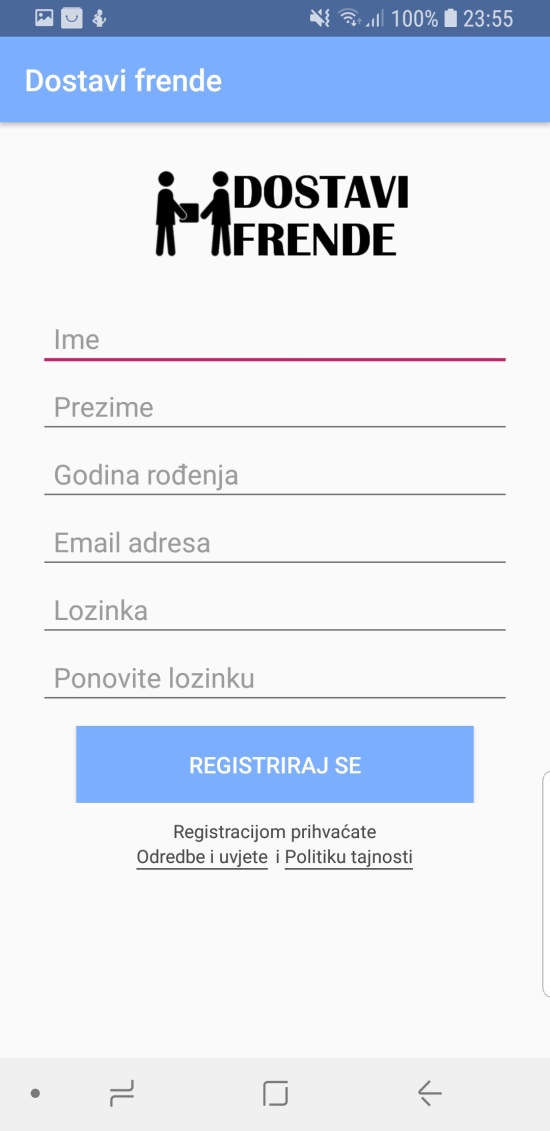
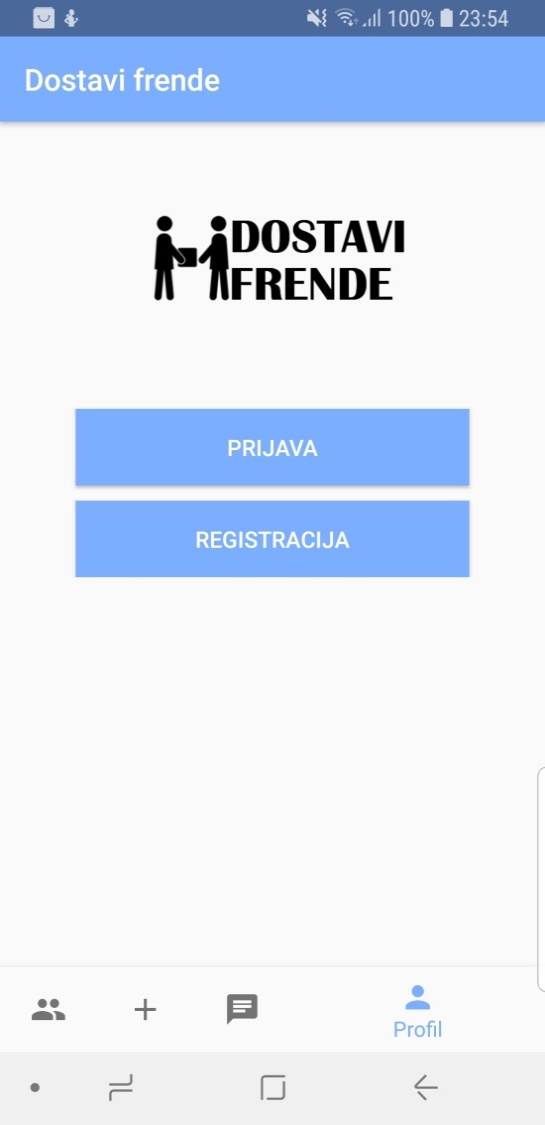
Na slici je prikazan dijagram modula kojim je prikazana modularnost sustava. Dijagramom je prikazana podjela i komunikacija među modulima te su tako prikazane i ovisnosti pojedinih modula o drugima.



Slika 18 Moduli aplikacije

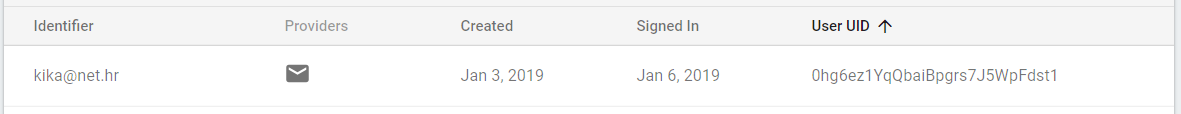
Glavni modul je app modul koji komunicira sa svima i koji je svima nadređen, unutar njega će biti implementirana funkcionalnost sustava, tj. glavnina aplikacije. Drugi modul je webservice Firebase modul unutar kojega će biti implementirane prijava korisnika te registracija, nakon uspješne prijave/registracije zapisati u bazu te će se tako rukovati s korisnicima aplikacije. Na Firebase će se spremati ostali podaci aplikacije. Također nudi se realtime baza podataka za spremanje te storage koji nudi mogućnost spremanja profilnih slika korisnika. U poglavlju Implementacija detaljnije će biti napisano o samom Firebase kako smo povezali naš projekt, te rad s podacima. U ovom trenutku imamo komunikaciju samo između ova dva modula. No daljnjim razvojem aplikacije bit će kreirano još modula, te će se ova slika naknadno ažurirati.

1. Implementacija
2. 1. Implementacija Registracije



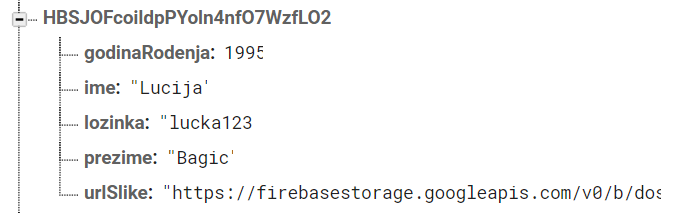
Slika 19 Zaslon registracije korisnika

Kada se korisnik odluči registrirati dolazi na aktivnost registracije. Na početku se inicijaliziraju sve potrebne varijable u onCreate i stvori se veza na firebase bazu. Sav dio oko stvaranja varijabli je trivijalan. Jedini zanimljiv dio je firebase funkcija koja prijavljuje korisnika u svoj sustav. Ono što on zapisuje tom fukcijom je prikazano na slici. Za svakog korisnika se osim njegove email adrese sprema i kreira njegoj jedinstveni user id.



Pošto nam je nedovoljno što sprema fukncija createUserWithEmailAndPassword()

dodali smo još podataka u Firebase Database. Još spremamo i podatke prikazane na slici. Za spremanje tih podataka postoji klasa User. Podaci se spremaju u obliku JSON podataka. Preko konstruktora se postavljaju podaci za objekt i taj se objekt postavlja na mjesto djeteta gdje nam to paše, a to je na granu User.



**public void** createUser(View v){  
 **progressDialog**.setMessage(**"Registracija u tijeku."**);  
 **progressDialog**.show();  
 String errorMessage = validateFields();  
 **if**(errorMessage != **""**){  
 Toast.*makeText*(getApplicationContext(), errorMessage, Toast.***LENGTH\_SHORT***).show();  
 **progressDialog**.dismiss();  
 }  
 **else** {  
 **final** String email = **editTextEmail**.getText().toString();  
 **final** String lozinka = **editTextLozinka**.getText().toString();  
 **final** String ime = **editTextIme**.getText().toString();  
 **final** String prezime = **editTextPrezime**.getText().toString();  
 **final** Integer godinaRodenja = Integer.*parseInt*(**editTextGodinaRodenja**.getText().toString());  
  
 **auth**.createUserWithEmailAndPassword(email, lozinka)  
 .addOnCompleteListener(**new** OnCompleteListener<AuthResult>() {  
 @Override  
 **public void** onComplete(@NonNull Task<AuthResult> task) {  
 **if** (task.isSuccessful()){  
 **firebaseUser** = **auth**.getCurrentUser();  
 User newUserInsertObj = **new** User(ime, prezime, lozinka, godinaRodenja);  
 **rootReference**.child(**"Users"**).child(**firebaseUser**.getUid()).setValue(newUserInsertObj)  
 .addOnCompleteListener(**new** OnCompleteListener<Void>() {  
 @Override  
 **public void** onComplete(@NonNull Task<Void> task) {  
 **if** (task.isSuccessful()){  
 **progressDialog**.dismiss();  
 Toast.*makeText*(getApplicationContext(), **"Uspješno ste se registrirali"**, Toast.***LENGTH\_SHORT***).show();  
 finish();  
 Intent intent = **new** Intent(getApplicationContext(), MainActivity.**class**);  
 startActivity(intent);  
 }  
 }  
 });  
 }**else** {  
 **progressDialog**.dismiss();  
 Toast.*makeText*(getApplicationContext(), **"Korisnik nije kreiran"**, Toast.***LENGTH\_SHORT***).show();  
 }  
 }  
 });  
  
 }  
}

Funkcija validateFields() validira polja, od validacije se provjerava jesu li sva polja unešena, jesu li lozinke jednake i jeli godina rođenja neki broj veći od 1900.

**private** String validateFields(){  
 String errorMessage = **""**;  
 String lozinka = **editTextLozinka**.getText().toString().trim();  
 String lozinka1 = **editTextLozinka1**.getText().toString().trim();  
 **if** (**editTextEmail**.getText().toString().equals(**""**) || **editTextLozinka**.getText().toString().equals(**""**)  
 || **editTextIme**.getText().toString().equals(**""**) ||

**editTextPrezime**.getText().toString().equals(**""**)  
 || **editTextLozinka1**.getText().toString().equals(**""**)){  
 errorMessage = **"Niste ispunili sva polja."**;  
 }  
 **else if** (!lozinka.equals(lozinka1)) {  
 errorMessage = **"Lozinke se ne podudaraju."**;  
 }  
 **else if** (Integer.*parseInt*(**editTextGodinaRodenja**.getText().toString())<1900){  
 errorMessage = **"Kriva godina rođenja."**;  
 }**else** {  
 errorMessage = **""**;  
 }  
 **return** errorMessage;  
}

* 1. Implementacija Prijave

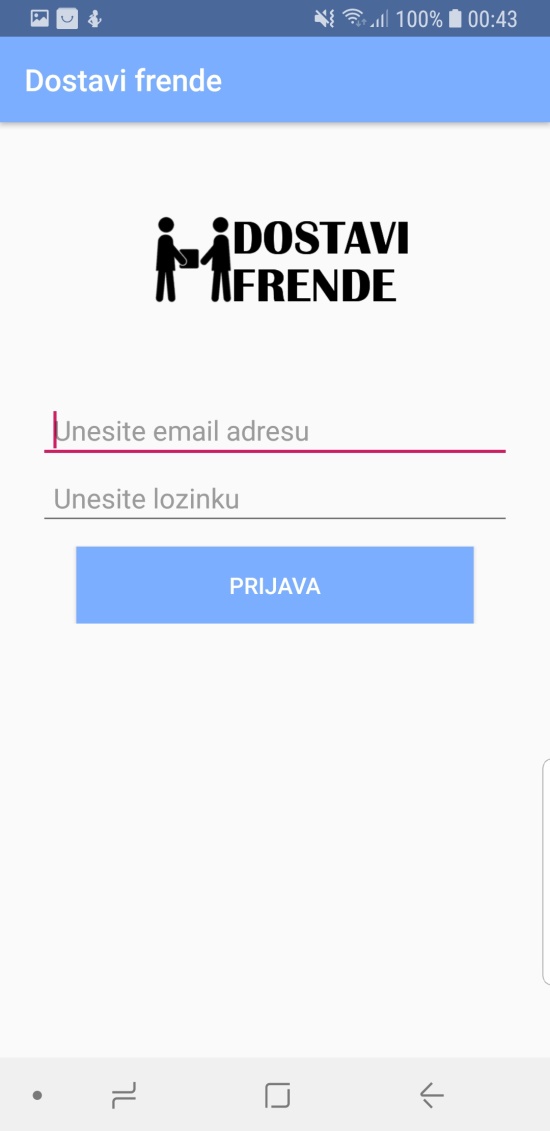
Za prijavu se isto kao i za registraciju koristi firebase već ugrađena funkcija signInWithEmailAndPassword(). Firebase provjerava jesu li točno unešeni podaci i postoji li takav korisnik u njegovom sustavu, ukoliko takav korisnik postoji task se izvršava kao uspješan. Pojavljuje se poruka o uspješnosti i korisnika se prebacuje na glavnu aktivnost.

**public void** loginUser(View v){  
 **if** (**editTextEmail**.getText().toString().equals(**""**) && **editTextLozinka**.getText().toString().equals(**""**)){  
 Toast.*makeText*(getApplicationContext(), **"Prazna polja nisu dozvoljena"**, Toast.***LENGTH\_SHORT***).show();  
 }**else** {  
 **auth**.signInWithEmailAndPassword(**editTextEmail**.getText().toString(), **editTextLozinka**.getText().toString())  
 .addOnCompleteListener(**new** OnCompleteListener<AuthResult>() {  
 @Override  
 **public void** onComplete(@NonNull Task<AuthResult> task) {  
 **if** (task.isSuccessful()){  
 Toast.*makeText*(getApplicationContext(), **"Uspješna prijava"**, Toast.***LENGTH\_SHORT***).show();  
 finish();  
 Intent i = **new** Intent(getApplicationContext(), MainActivity.**class**);  
 startActivity(i);  
 }**else** {  
 Toast.*makeText*(getApplicationContext(), **"Neuspješna prijava"**, Toast.***LENGTH\_SHORT***).show();  
 }  
 }  
 });  
  
 }  
}

Firebase čuva sesiju korisnika i ona postoji sve dok ju se ne uništi odlogiravanjem. Korisnika se odjavljuje s firebase funkcijom signOut() koja se poziva preko auth je instanca FirebaseAuth.

**auth** = FirebaseAuth.*getInstance*();

**public void** signOut(View v){  
 **auth**.signOut();  
 getSupportFragmentManager().beginTransaction().replace(R.id.***fragment\_container***,  
 **new** UserUnknownFragment()).commit();  
}



Slika 20 Zaslon prijave korisnika

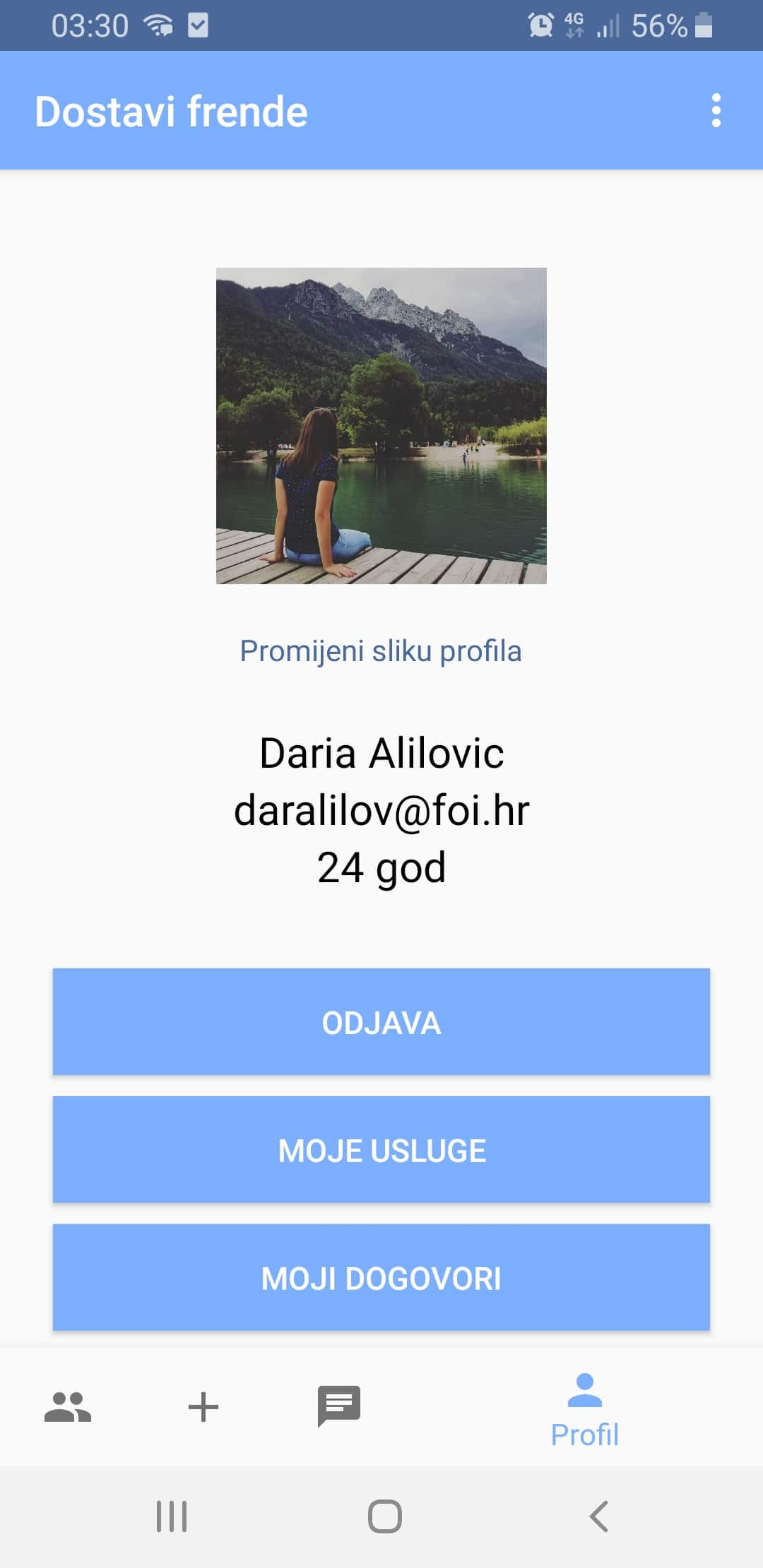
* 1. Implementacija Profil korisnika

Nakon što se korisnik prijavi u aplikaciju omogućen mu je pristup svim dijelovima aplikacije pa tako i svom vlastitom profilu. Na profilu su prikazane osobne informacije poput ime i prezime, email adresa te godine. Uz to je moguće i postaviti sliku profila te isto tako je izmjeniti. Tu se još nalazi gumb za Odjavu, za pregled vlastitih usluga i dogovora s drugim korisnicima. Podaci se dohvaćaju s Firebase, metoda getData().

**public void** getData(){  
 **reference**.addValueEventListener(**new** ValueEventListener() {  
 @Override  
 **public void** onDataChange(@NonNull DataSnapshot dataSnapshot) {  
 String ime = dataSnapshot.child(**"ime"**).getValue().toString();  
 String prezime = dataSnapshot.child(**"prezime"**).getValue().toString();  
 Integer godinaRodenja = Integer.*parseInt*(dataSnapshot.child(**"godinaRodenja"**).getValue().toString());  
 Integer godina = Calendar.*getInstance*().get(Calendar.***YEAR***) - godinaRodenja;  
 **textViewImePrezime**.setText(ime+**" "**+prezime);  
 **textViewGodina**.setText(godina.toString()+**" "**+**"god"**);  
 }  
  
 @Override  
 **public void** onCancelled(@NonNull DatabaseError databaseError) {  
  
 }  
 });  
}

Te prikaz informacija korisniku, no najprije se provjerava da li je User ulogiran, ukoliko nije proslijedi se na UserUnknownFragment.

**public** View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {  
 View view = inflater.inflate(R.layout.***fragment\_profile***, container, **false**);  
  
 **auth** = FirebaseAuth.*getInstance*();  
 **user** = **auth**.getCurrentUser();  
 **if** (**user** == **null**){  
 UserUnknownFragment userUnknownFragment = **new** UserUnknownFragment();  
 getFragmentManager().beginTransaction().replace(R.id.***fragment\_container***,  
 userUnknownFragment).commit();  
 }**else** {  
 **storageReference** = FirebaseStorage.*getInstance*().getReference().child(**"User\_images"**);  
 **reference** = FirebaseDatabase.*getInstance*().getReference().child(**"Users"**).child(**user**.getUid());  
 **ref** = FirebaseDatabase.*getInstance*().getReference().child(**"Offers"**).child(**user**.getUid());  
  
 **imageViewProfilna** = view.findViewById(R.id.***imageViewProfilna***);  
 **textViewImePrezime** = view.findViewById(R.id.***textViewImePrezime***);  
 **textViewGodina** = view.findViewById(R.id.***textViewGodina***);  
 **textViewEmail** = view.findViewById(R.id.***textViewEmail***);  
 **textViewEmail**.setText(**user**.getEmail());  
 **offerBtn** = view.findViewById(R.id.***buttonMyOffer***);  
 **dealsBtn** = view.findViewById(R.id.***buttonMyDeals***);  
 getData();  
 displayImage();  
 **offerBtn**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 getFragmentManager().beginTransaction().replace(R.id.***fragment\_container***,  
 **new** MyOffersFragment()).commit();  
 }  
 });  
 **dealsBtn**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 getFragmentManager().beginTransaction().replace(R.id.***fragment\_container***,  
 **new** MyDealsFragment()).commit();  
 }  
 });  
 }  
 **return** view;  
}



Slika 21 Profil korisnika

* 1. Implementacija Moji dogovori

Kada korisnik odabere opciju Moji dogovori prelazi se na fragment MyOffersFragment. Korisniku se prikažu svi njegovi dogovori koje je sklopio s drugim korisnicima putem Chata. Prikaže se s kojom osoba, status dogovora te vrijeme kad je dogovor sklopljen. FirebaseRecyclerAdapter mapira podatke koje smo dohvatili s Firebase lokacije a to je Deals.

**firebaseRecyclerAdapter** = **new** FirebaseRecyclerAdapter<Deal, DealViewHolder>  
 (Deal.**class**, R.layout.***list\_my\_deals***, DealViewHolder.**class**, **rootReference**.child(**"Deals"**).child(**user**.getUid()).orderByChild(**"status"**)) {

Zatim puni DealViewHolder koji sadrži metode setKorisnik(), setStatus(), setDatum(). Ukoliko ne želimo izbrisati neki od dogovora klikom na njeg pojavit će nam se AlertDialog na kojem će se ispisati poruka „Želite li obrisati dogovor?, Da ili Ne“. Odabiramo Da dogovor se briše s Firebasea.

**protected void** populateViewHolder(**final** DealViewHolder viewHolder, **final** Deal model, **final int** position) {  
 viewHolder.setKorisnik(model.getKorisnik());  
 viewHolder.setStatus(model.getStatus());  
 viewHolder.setDatum(DateFormat.*format*(**"dd-MM-yyyy (HH:mm:ss)"**, model.getDealTime()));  
  
 viewHolder.**view**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 AlertDialog.Builder alertDialog = **new** AlertDialog.Builder(getContext());  
 alertDialog.setMessage(**"Jeste li sigurni da zelite obrisati dogovor?"**).setCancelable(**false**)  
 .setPositiveButton(**"Da"**, **new** DialogInterface.OnClickListener() {  
 @Override  
 **public void** onClick(DialogInterface dialogInterface, **int** i) {  
 **firebaseRecyclerAdapter**.getRef(position).removeValue();  
 **firebaseRecyclerAdapter**.notifyDataSetChanged();  
 }  
 })  
 .setNegativeButton(**"Ne"**, **new** DialogInterface.OnClickListener() {  
 @Override  
 **public void** onClick(DialogInterface dialogInterface, **int** i) {  
 dialogInterface.cancel();  
 }  
 });  
 AlertDialog dialog = alertDialog.create();  
 dialog.setTitle(**"Brisanje dogovora"**);  
 dialog.show();  
 }  
 });  
}

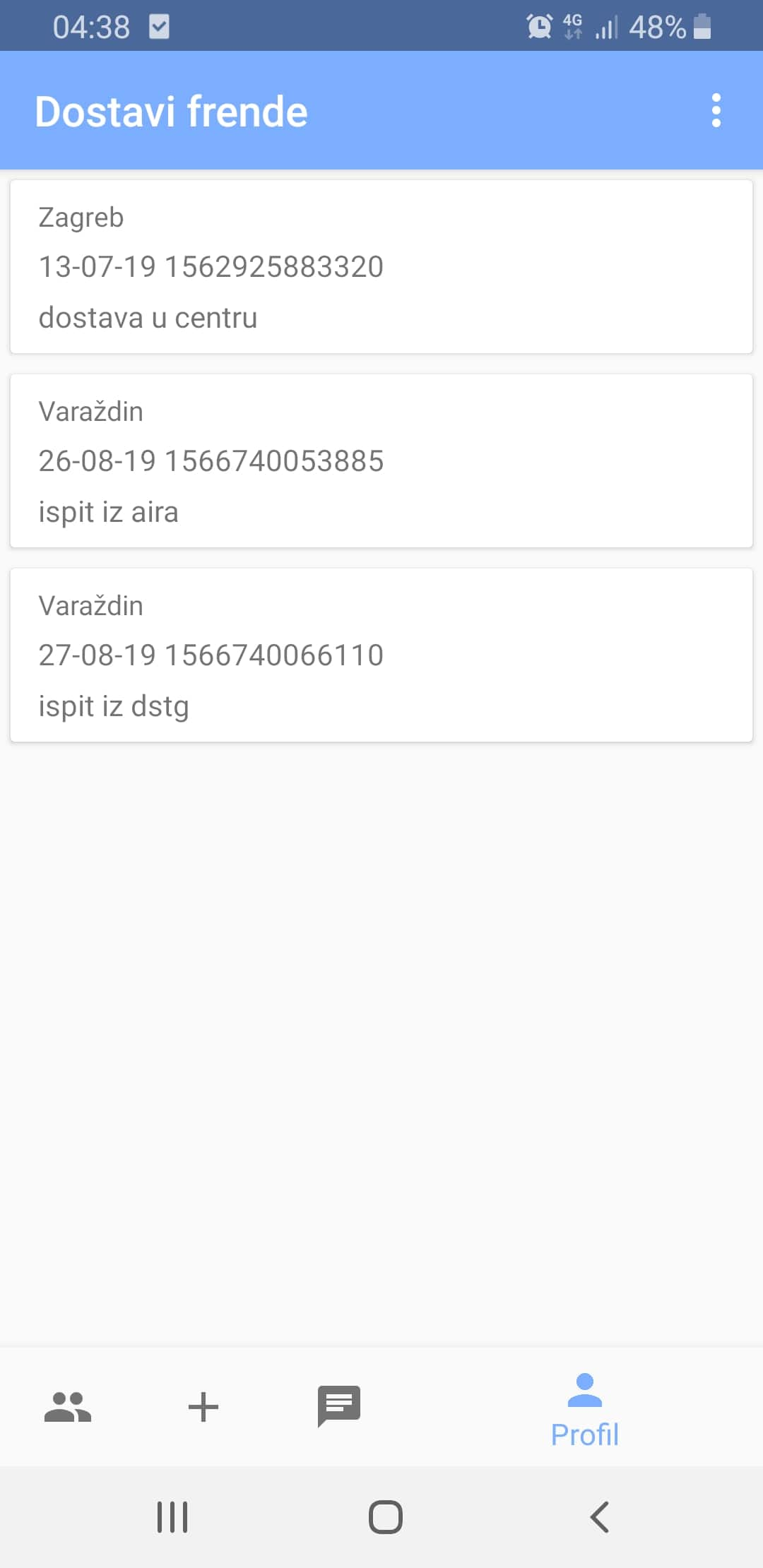
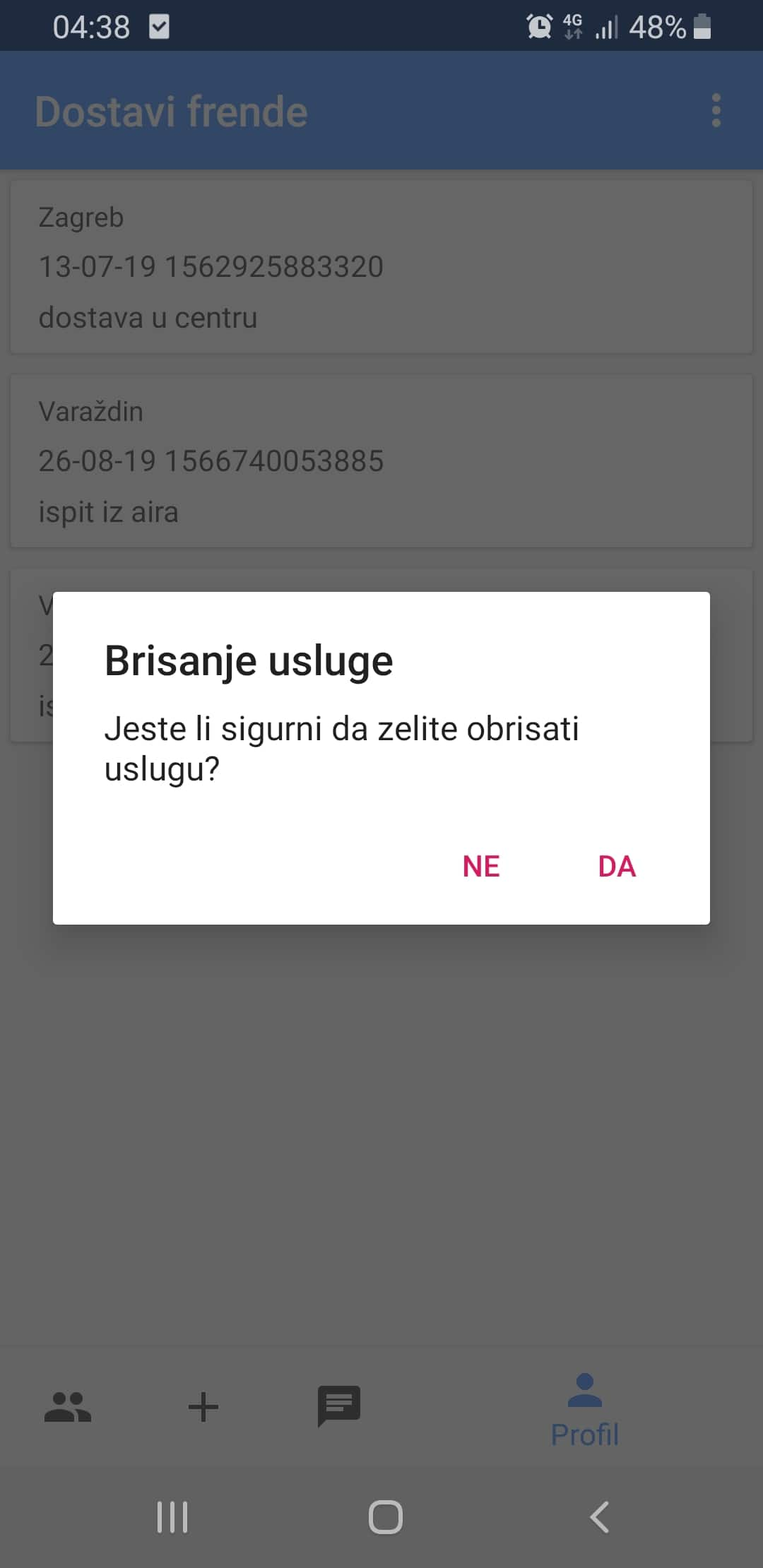
* 1. Implementacija Moje usluge

Na sličan način kao i Moji dogovori. Korisnik kada ponudi uslugu dostavljanja, vidljiva mu je u Aktivni korisnici a isto tako i na njegovom profilu kada klikne na button Moje usluge. No kada mu se prikažu usluge omogućeno mu je i brisanje ukoliko ne želi više obavljati određenu uslugu. Pomoću FirebaseRecycleAdapter se mapiraju podaci iz Firebase u OfferViewHolder prema korisniku koji je jednak trenutnom user odnosno ulogiranom korisniku.

**firebaseRecyclerAdapter** = **new** FirebaseRecyclerAdapter<Deal, DealViewHolder>  
 (Deal.**class**, R.layout.***list\_my\_deals***, DealViewHolder.**class**, **rootReference**.child(**"Deals"**).child(**user**.getUid()).orderByChild(**"status"**))

OfferViewHolder se puni s podacima koji su mapirani u FirebaseRecyclerAdapteru, a te sadrži metode setKorisnik(), setStatus() i setDatum(). Također postoji mogućnost brisanja dogovora.

**protected void** populateViewHolder(**final** DealViewHolder viewHolder, **final** Deal model, **final int** position) {  
 viewHolder.setKorisnik(model.getKorisnik());  
 viewHolder.setStatus(model.getStatus());  
 viewHolder.setDatum(DateFormat.*format*(**"dd-MM-yyyy (HH:mm:ss)"**, model.getDealTime()));  
  
 viewHolder.**view**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 AlertDialog.Builder alertDialog = **new** AlertDialog.Builder(getContext());  
 alertDialog.setMessage(**"Jeste li sigurni da zelite obrisati dogovor?"**).setCancelable(**false**)  
 .setPositiveButton(**"Da"**, **new** DialogInterface.OnClickListener() {  
 @Override  
 **public void** onClick(DialogInterface dialogInterface, **int** i) {  
 **firebaseRecyclerAdapter**.getRef(position).removeValue();  
 **firebaseRecyclerAdapter**.notifyDataSetChanged();  
 }  
 })  
 .setNegativeButton(**"Ne"**, **new** DialogInterface.OnClickListener() {  
 @Override  
 **public void** onClick(DialogInterface dialogInterface, **int** i) {  
 dialogInterface.cancel();  
 }  
 });  
 AlertDialog dialog = alertDialog.create();  
 dialog.setTitle(**"Brisanje dogovora"**);  
 dialog.show();  
 }  
 });  
}

Slika 22 Moje usluge

Slika 23 Brisanje usluge

* 1. Implementacija Pregled aktivnih/slobodnih korisnika

Pregled aktivnih korisnika koji su objavili svoje usluge dostavljanja se odvija u ActiveUsersFragment.

**public void** pretraziGrad(Query pretraziGrad){  
  
 **final** FirebaseRecyclerAdapter<Offer, OfferViewHolder> firebaseRecyclerAdapter = **new** FirebaseRecyclerAdapter<Offer, OfferViewHolder>  
 (Offer.**class**, R.layout.***list\_active\_users***, OfferViewHolder.**class**, pretraziGrad) {  
 @Override  
 **protected void** populateViewHolder(**final** OfferViewHolder viewHolder, **final** Offer model, **int** position) {  
 viewHolder.setKorisnik(model.getImePrezime());  
  
 DatabaseReference slikaKorisnika = **rootReference**.child(**"Users"**).child(model.**korisnik**.trim());  
 slikaKorisnika.addValueEventListener(**new** ValueEventListener() {  
 @Override  
 **public void** onDataChange(@NonNull DataSnapshot dataSnapshot) {  
String slika = dataSnapshot.child(**"urlSlike"**).getValue().toString();  
viewHolder.setImage(getContext(), slika);

**if** (dataSnapshot.child(**"thumbsDown"**).getValue() != **null**){  
  
 viewHolder.setThumbsUp(dataSnapshot.child(**"thumbsUp"**).getValue().toString());  
 viewHolder.setThumbsDown(dataSnapshot.child(**"thumbsDown"**).getValue().toString());  
 }  
viewHolder.setRating(dataSnapshot.child(**"ocjena"**).getValue().toString());  
 }  
 @Override  
 **public void** onCancelled(@NonNull DatabaseError databaseError) {  
  
 }  
 });  
  
 viewHolder.**view**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 **final** Dialog myDialog = **new** Dialog(getContext());  
 myDialog.setContentView(R.layout.***costumpopup***);  
 TextView textViewNapomena = myDialog.findViewById(R.id.***textViewNapomena***);  
 Button javiSe = myDialog.findViewById(R.id.***buttonChat***);  
 textViewNapomena.setText(model.getNapomena());  
 myDialog.show();  
 javiSe.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 ChatMessage newMessageInsertObj = **new** ChatMessage(**"Pozdrav!"**, **user**.getUid(), model.**korisnik**);  
 **rootReference**.child(**"Messages"**).push().setValue(newMessageInsertObj);  
 Bundle b = **new** Bundle();  
 b.putString(**"userId"**, model.**korisnik**);  
 ChatFragment f = **new** ChatFragment();  
 f.setArguments(b);  
 getFragmentManager().beginTransaction().replace(R.id.***fragment\_container***,  
 f).commit();  
 myDialog.cancel();  
 }  
 });  
 }  
 });  
 }  
 };  
 **activeUsersList**.setAdapter(firebaseRecyclerAdapter);  
}

Pregled aktivnih i slobodnih usluga ide preko metode pretraziGrad kojoj na početku prosljeđujem upit za prikaz svih gradova, a kasnije ukoliko neko pretraži neki grad za koji im je usluga potrebna prosljeđuje se novi upit. Metoda ima adapter koji popunjava ViewHolder kojem se prosljeđuje jedna ponuda. On prolazi kroz svaku i tako puni pripremljeni predmet liste. U metodi OfferViewHolder se povezuje sa predmetom liste kako bi se u populateViewHolder mogli pozivati i popunjavati odgovarajućim vrijednostima. Za sliku korisnika se mora odvojeno opet spajati s bazom kako bi se dobila trenutna učitana slika korisnika. Osim slike i imena korisnika ispisuje se i ukupna ocjena korisnika i koliko ima *thumbs up/down.* Podaci se preuzimaju s DataSnapShot što bi doslovno značilo snimak podataka. Na svakom viewHolderu je slušač na klik. Kad se pritisne otvara se dijalog s ispisanom napomenom i gumbom Javi se. Kod javljanja se stvara nova poruka, klasa ChatMessage koja reprezentira objek poruke koji se upisuje u bazu. Početna poruka koja se pošalje je „Pozdrav!“. Postavi se za argument koji se šalje u fragment korisnik kojem se javlja.

**public static class** OfferViewHolder **extends** RecyclerView.ViewHolder{  
 View **view**;  
 **public** OfferViewHolder(View itemView){  
 **super**(itemView);  
 **view** = itemView;  
 }  
 **public void** setKorisnik(String imePrezime){  
 TextView textViewImePrezime = **view**.findViewById(R.id.***textViewImePrezime***);  
 textViewImePrezime.setText(imePrezime);  
 }  
 **public void** setNapomena(String napomena){  
 TextView textViewNapomena = **view**.findViewById(R.id.***textViewNapomena***);  
 textViewNapomena.setText(napomena);  
 }  
 **public void** setImage(Context ctx, String image){  
 ImageView imageViewSlika = **view**.findViewById(R.id.***imageViewActiveUser***);  
 Picasso.*with*(ctx).load(image).into(imageViewSlika);  
 }  
 **public void** setRating(String rating){  
 TextView textViewRating = **view**.findViewById(R.id.***textViewRating***);  
 textViewRating.setText(rating);  
 }  
 **public void** setThumbsUp(String thumbsUp){  
 TextView textViewRating = **view**.findViewById(R.id.***textThumbsUp***);  
 textViewRating.setText(thumbsUp);  
 }  
 **public void** setThumbsDown(String thumbsDown){  
 TextView textViewRating = **view**.findViewById(R.id.***textThumbsDown***);  
 textViewRating.setText(thumbsDown);  
 }  
}

* 1. Implementacija Chata

Chat je implementiran u ChatFragment. Prvo se preuzima argument koji je prosljeđen fragmentu, a on je korisnik s kojim se komunicira. Najbitnija je struktura spremanja poruka u Firebase. Poruke se spremaju pod dijete „Messages“. Sve poruke imaju nasumičan ključ pod koji su spremljene. Poruka sadrži svoga pošiljatelja, primatelja, tekst poruke i vrijeme slanja.

FloatingActionButton fab = view.findViewById(R.id.***fab***);  
**input** = view.findViewById(R.id.***input***);  
fab.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 **rootReference** .child(**"Messages"**)  
 .push()  
 .setValue(**new** ChatMessage(**input**.getText().toString(),  
 **user**.getUid(),**userId**)  
 );  
 **input**.setText(**""**);  
 }  
});



Za slanje poruka se sluša mali gumb na čije okidanje se sprema poruka iz input polja u bazu podataka. Push na referenci na bazu kreira neki nasumični ključ.

**adapter** = **new** FirebaseListAdapter<ChatMessage>(getActivity(),ChatMessage.**class**,  
 R.layout.***message***, **rootReference**.child(**"Messages"**)) {  
 @Override  
 **protected void** populateView(View v, ChatMessage model, **int** position) {  
 TextView messageText = v.findViewById(R.id.***message\_text***);  
 TextView messageUser = v.findViewById(R.id.***message\_user***);  
 TextView messageTime = v.findViewById(R.id.***message\_time***);  
  
 **if** (model.getMessageReceiver().equals(**user**.getUid()) && model.getMessageSender().equals(**userId**) ||  
 model.getMessageReceiver().equals(**userId**) && model.getMessageSender().equals(**user**.getUid())){  
 messageText.setText(model.getMessageText());  
 messageUser.setText(model.getMessageSender());  
 messageTime.setText(DateFormat.*format*(**"dd-MM-yyyy (HH:mm:ss)"**,  
 model.getMessageTime()));  
 }  
  
 }  
};

Poruke se isčitavaju putem modela. Prolazi se kroz svaku poruku i svaka poruku koja odgovara uvjetu se ispisuje.

**btnDogovoreno**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 Query reference;  
 reference = **rootReference**.child(**"Deals"**).child(**user**.getUid()).orderByChild(**"status"**).equalTo(**"aktivno"**);  
 reference.addListenerForSingleValueEvent(**new** ValueEventListener() {  
  
 @Override  
 **public void** onDataChange(@NonNull DataSnapshot dataSnapshot) {  
 **if** (!dataSnapshot.exists()){  
 **rootReference**.child(**"Deals"**).child(**user**.getUid()).push().setValue(**new** Deal(**userId**, **"aktivno"**));  
 }  
 **for** (DataSnapshot ds : dataSnapshot.getChildren()) {  
 Deal deal = ds.getValue(Deal.**class**);  
 **if** (!deal.getStatus().equals(**"aktivno"**)){  
 **rootReference**.child(**"Deals"**).child(**user**.getUid()).push().setValue(**new** Deal(**userId**, **"aktivno"**));  
 }**else** {  
 Toast.*makeText*(getContext(),**"Dogovor već postoji!"**,Toast.***LENGTH\_SHORT***).show();  
 }  
 }  
 }  
 @Override  
 **public void** onCancelled(@NonNull DatabaseError databaseError) {  
  
 }  
 });  
 }  
});

Gumb dogovoreno stvara dogovor u bazi. Svaka osoba sa svoje strane dobije dogovor ispisan u svojim dogovorima. Svaki korisnik ima pravo na samo jedan aktivan dogovor i ne može stvoriti novi dok se status ne promjeni u zavrseno ostavljanjem recenzije.



* 1. Implementacija Ponuda usluge dostavljanja

**textViewGrad** = view.findViewById(R.id.***textViewGrad***);  
ArrayAdapter<String> adapter = **new** ArrayAdapter<String>(getContext(), android.R.layout.***simple\_list\_item\_1***, ***GRADOVI***);  
**textViewGrad**.setAdapter(adapter);  
  
  
**final** DatePickerDialog.OnDateSetListener date = **new** DatePickerDialog.OnDateSetListener() {  
  
 @Override  
 **public void** onDateSet(DatePicker view, **int** year, **int** monthOfYear,  
 **int** dayOfMonth) {  
 **myCalendar**.set(Calendar.***YEAR***, year);  
 **myCalendar**.set(Calendar.***MONTH***, monthOfYear);  
 **myCalendar**.set(Calendar.***DAY\_OF\_MONTH***, dayOfMonth);  
 updateLabel();  
 }  
  
};  
  
**editTextDatum**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **new** DatePickerDialog(getContext(), date, **myCalendar** .get(Calendar.***YEAR***), **myCalendar**.get(Calendar.***MONTH***),  
 **myCalendar**.get(Calendar.***DAY\_OF\_MONTH***)).show();  
 }  
});

Napravljeno je polje koje sadržava sve gradove. Gradovi se ispisuju u AutoCompleteTextView pomoću jednostavnog adaptera. Za datum se koristi postojeći DatePickerDialog.

**public void** offerService(){  
 String errorMessage = validateFields();  
 **if**(errorMessage != **""**){  
 Toast.*makeText*(getContext(), errorMessage, Toast.***LENGTH\_SHORT***).show();  
 }  
 **else** {  
 **final** String grad = **textViewGrad**.getText().toString();  
 **final** String datum = **editTextDatum**.getText().toString()+**" "**+Calendar.*getInstance*().getTimeInMillis();  
 **final** String napomena = **editTextPoruka**.getText().toString();  
  
 **firebaseUser** = **auth**.getCurrentUser();  
 Offer newOfferInsertObj = **new** Offer(**firebaseUser**.getUid(), **ime**+**" "**+**prezime**, datum, napomena, grad, **urlSlike**);  
  
 **rootReference**.child(**"Offers"**).child(datum).setValue(newOfferInsertObj)  
 .addOnCompleteListener(**new** OnCompleteListener<Void>() {  
 @Override  
 **public void** onComplete(@NonNull Task<Void> task) {  
 Toast.*makeText*(getContext(), **"Spremili ste uslugu."**, Toast.***LENGTH\_SHORT***).show();  
 }  
 });  
 }  
}

Nova usluga se upisuje metodom offerService. Spaja se na bazu i unosi se novo dijete u bazu pomoću objekta klase Offer.

* 1. Implementacija BottomNavigationManager klase

BottomNavigationManager je singleton klasa koja puni bottom navigation manager s fragmentima koji implementiraju sučelje FragmentExtension. To sučelje je implementirano u Core modulu a sadrži metode getName(), getFragment() getIcon() koje su implementirane u fragmentima npr ActiveUsersFragment, odnosno koje implementiraju to sučelje. Ukoliko želimo dodati neki novi fragment u bottom navigation on mora biti proširen s FragmentExtension sučeljem. Ova klasa sadrži samo jedan objekt, odnosno instancu klase BottomNavigationManager.

private static BottomNavigationManager *sInstance*;

U konstruktoru klase koji je private se dodaju četiri fragmenta i sva četiri implementiraju sučelje FragmentExtension.

private BottomNavigationManager() {  
  
 bottomNavigationItems = new ArrayList<>();  
 bottomNavigationItems.add(new ActiveUsersFragment());  
 bottomNavigationItems.add(new OfferServiceFragment());  
 bottomNavigationItems.add(new UsersFragment());  
 bottomNavigationItems.add(new ProfileFragment());  
  
}

Klasa također sadrži metodu koja kreira novu instancu klase BottomNavigationManager.

public static BottomNavigationManager newInstnace() {  
 if (*sInstance* == null){  
 *sInstance* = new BottomNavigationManager();  
 }  
 return *sInstance*;  
}

Metoda startModule mijenja fragmente ovisno o tome koji fragment je korisnik odabrao. Ukoliko nije ni jedan odabran metoda statMainModule postavlja fragment koji je na poziciji 0.

private void startModule(FragmentExtension module) {  
 FragmentManager mFragmentManager = activity.getSupportFragmentManager();  
 mFragmentManager.popBackStack(null, FragmentManager.*POP\_BACK\_STACK\_INCLUSIVE*);  
  
 mFragmentManager.beginTransaction()  
 .replace(R.id.*fragment\_container*, module.getFragment())  
 .setTransition(FragmentTransaction.*TRANSIT\_FRAGMENT\_OPEN*)  
 .commit();  
}  
  
public void startMainModule() {  
  
 FragmentExtension mainModule = bottomNavigationItems != null ? bottomNavigationItems.get(0) : null;  
 if (mainModule != null)  
 startModule(mainModule);  
}

Metoda setUpBottomNavigationDrawer() prolazi kroz listu item bottomNavigationItems, te sprema dohvaćeni item u FragmenExtension. Te na temelju toga dohvaća (getMenu) menu koji se nalazi u res folderu.

private void setUpBottomNavigationDrawer()  
{  
 for (int i = 0; i < bottomNavigationItems.size(); i++) {  
 FragmentExtension item = bottomNavigationItems.get(i);  
 bottomNavigationView.getMenu()  
 .add(dynamicGroupId, i, i+1, item.getName())  
 .setCheckable(true)  
 .setIcon(item.getIcon());  
 }  
}

Sučelje FragmentExtension je od velike važnosti za ovu klasu jer ono implementirano modularnost u ovoj klasi.