Aplikacija za kreiranje tržišta uslužnih djelatnosti

Lesar, Alen

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

Alen Lesar

APLIKACIJA ZA KREIRANJE TRŽIŠTA USLUŽNIH DJELATNOSTI

Diplomski rad

Pula, rujan 2020.

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

Alen Lesar

APLIKACIJA ZA KREIRANJE TRŽIŠTA USLUŽNIH DJELATNOSTI

Diplomski rad

JMBAG: 03033055721, redovni student

Studijski smjer: Informatika

Predmet: Mobilne aplikacije

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: izv. prof. dr. sc. Siniša Sovilj

Pula, rujan 2020.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani **Alen Lesar**, kandidat za **magistra informatike** ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, _____, 2020. godine

IZJAVA

o korištenju autorskog djela

Ja, **Alen Lesar** dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom **Aplikacija za kreiranje tržišta uslužnih djelatnosti** koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, _____

Potpis

_____

SVEUČILIŠTE JURJA DOBRILE U PULI
FAKULTET INFORMATIKE

Pula, 18. prosinca 2019.

DIPLOMSKI ZADATAK

Pristupnik:	**Lesar Alen (0303055721)**
Studij:	Sveučilišni diplomski studij Informatike
Naslov (hrv.):	**Aplikacija za kreiranje tržišta uslužnih djelatnosti**
Naslov (eng.):	Application for services makret creation
Opis zadatka:	Zadatak je izraditi mobilnu aplikaciju za Android uređaje. Aplikacija će se koristiti za kreiranje ponude ili pronalazak usluge koje se nude na području Hrvatske. Usluge će biti podijeljene u kategorije, primjerice usluge održavanja kuće, usluge transporta, usluge iznajmljivanja i slično. Također usluge će biti moguće pretraživati po području odnosno mjestu gdje su potrebne, primjerice pretraga po određenom gradu. Nakon odrađene usluge, korisnici će biti u mogućnosti ocijeniti uslugu. Aplikacija će biti razvijena u razvojnom okruženju Android Studio korištenjem programskog jezika Java te XML. Istražiti najpogodnije tehnologije pohrane podataka unutar web usluge (PostgreSQL, Firebase itd).

Zadatak uručen pristupniku: 18. prosinac 2019.

Rok za predaju rada: 1. rujna 2020.

Mentor:

_____

doc.dr.sc. Siniša Sovilj

Sadržaj

1. Uvod

Usluge predstavljaju nematerijalni proizvod koji jedna osoba zahtjeva od druge. Jedan od glavnih motiva za razvoj aplikacije *UslugeApp* je da se olakša pronalazak, odnosno mogućnost ponude usluga na području Hrvatske. Trenutno postoji nekoliko aplikacija koje mogu poslužiti u tu svrhu no ni jedna nije fokusirana samo na ponudu odnosno potražnju usluga.

Aplikacija *UslugeApp* nudi korisnicima mogućnost objave vlastitih oglasa za usluge, odnosno pronalazak usluga koje nude drugi korisnici. Korisnik nakon što nađe traženu uslugu ima mogućnost da ju naruči. Nakon što ta usluga bude odrađena, korisnik dobiva mogućnost ocjenjivanja te usluge te na taj način drugi korisnici dobivaju predodžbu o kvaliteti usluge ovisno o ocjeni koju ta usluga ima u tom trenutku.

Za razvoj bilo koje aplikacije poput desktop, mobilne ili web aplikacije, baza podataka je preduvjet za pohranu podataka. U svijetu tehnologije baza podataka postoje dvije glavne vrste baza podataka a to su *SQL* i *NoSQL* odnosno relacijske baze podataka i nerelacijske baze podataka. Razlika govori o tome kako su izgrađene, vrsti podataka koje pohranjuju kao i način na koji ih pohranjuju.

Ovaj diplomski rad se sastoji od sedam poglavlja. U prvom poglavlju definiran je pojam usluga, njihova klasifikacija te najvažnija obilježja. U drugom poglavlju su ukratko objašnjene relacijske i nerelacijske baze podataka te su za svaku navedene neke njihove prednosti odnosno nedostaci. Nadalje u četvrtom poglavlju je opisan *PostgreSQL* koji predstavlja objektno-relacijski sustav baze podataka otvorenog izvornog koda. Objašnjena je interakcija sa sustavom te način na koji se koristi jer standardna distribucija ne sadrži nikakve grafičke alate. Također je opisana sama arhitektura tog sustava kao i cjenik *PostgreSQL*-a na *Google Cloudu*. U petom poglavlju je opisan *Firebase* koji predstavlja *NoSQL* baze podataka. *Firebase* pruža dvije vrste baze podataka, jedna od njih je *Realtime* baza podataka koja je opisana u prvom dijelu petog poglavlja dok je u drugom dijelu opisan *Cloud Firestore*. Šesto poglavlje naziva *Aplikacija UslugeApp* sadrži

opis implementacije aplikacije te njene funkcionalnosti. Sedmo koje je ujedno i zadnje poglavlje sažima sve navedeno u prethodnim poglavljima.

2. Usluge

2.1. Definiranje usluga

Usluge su nematerijalni ekonomski proizvod koji pruža osoba na zahtjev druge osobe. To je aktivnost koja se provodi za nekog drugog. Mogu se isporučiti samo u određenom trenutku, to jest ne mogu se pohraniti za budućnost. Nedostaje im fizički identitet i one ne mogu biti posjedovane već samo korištene. Primjerice, ukoliko kupite kartu za gledanje filma u kinu, to ne znači da ste kupili kino, već ste platili cijenu korištenja usluge.

Teoretičarima i stručnjacima je veoma teško definirati usluge pošto su one brojne i raznovrsne. Postoje razne definicije koje su manje ili više slične ali nema jedinstvene. Neke od definicija usluga su:

- Fiziokrati (18. stoljeće) – Sve aktivnosti osim poljoprivrede.
- Adam Smith (1723. – 1790.) – Sve aktivnosti koje ne rezultiraju opipljivim proizvodima.
- Jean Baptiste Say (1767. – 1832.) – Sve neproizvođačke (nematerijalne) aktivnosti koje povećavaju korisnost dobrima.
- Alfred Marshall (1842. – 1924.) – Koristi koje nastaju u trenutku pružanja.
- Zapadne zemlje (1925. – 1960.) – Usluge ne stvaraju promjenu u obliku dobra.
- AMA (American Marketing Association, 1960.) – Aktivnosti, koristi ili zadovoljstva koja se nude na prodaju ili se pružaju vezano uz prodaju dobara.

Prema Philippu Kotleru, američkom ekonomistu i profesoru marketinga u *Kellog School of Managment* na Sveučilištu Northwestern, usluga je čin izvedbe koji jedna stranka može ponuditi drugoj koji je u osnovi nematerijalni i ne rezultira vlasništvom nad bilo čim. Njegova proizvodnja može ili ne mora biti vezana za fizički proizvod (Vučemilović i Blažević, 2016).

2.2. Klasifikacija usluga

Usluge je moguće podijeliti u različite klastere koji dijele određene tržišno relevantne karakteristike. Kao što je teško odrediti jedinstvenu definiciju usluge, isto tako ih je vrlo teško i klasificirati. Često se javljaju neke usluge koje je teško uvrstiti u neku određenu kategoriju. Određene usluge je moguće svrstati i u više kategorija dok neke neće biti moguće svrstati niti u jednu.

Kategorija	Primjeri
Vrsta tržišta	
Individualni korisnici	Pravni savjeti, popravci, dječja skrb
Poslovni korisnici	Usluge čuvanja imovine, usluge zaštite
Stupanj radne intezivnosti	
Radno intezivne	Obrazovanje, popravci, frizerske usluge
Kapitalno intezivne	Javni prijevoz, zdravstveni centri, telekomunikacije
Stupanj kontakta s korisnikom	
Visok	Hoteli, zračni prijevoz, zdravstvena zaštita
Nizak	Popravci, dostave u kuću, poštanske usluge
Kvalifikacija pružatelja usluga	
Profesionalna	Zdravstvena zaštita, računovodstvene usluge, pravni savjeti
Neprofesionalna	Usluge u kućanstvu, javni prijevoz, kemijsko čišćenje
Cilj pružatelja usluga	
Profitni	Osiguranje, zdravstvena zaštita, financijske usuge
Neprofitni	Zdravstvena zaštita, državna uprava, obrazovanje

Slika 1. Klasifikacija usluga

izvor: (Vučemilović i Blažević, 2016)

2.3. Obilježja usluga

2.3.1. Neopipljivost

Koristi se u marketingu za opisivanje nemogućnosti procjene vrijednosti dobivene angažmanom u aktivnosti koristeći bilo kakve opipljive dokaze. Često se koristi za opisivanje usluga kod kojih ne postoji opipljiv proizvod koji kupac može kupiti, koji može vidjeti, okusiti ili dodirnuti. Ovo je najvažnija karakteristika usluge koja ju razlikuje od proizvoda.

Kada kupac kupuje uslugu, on sagleda rizik povezan sa korištenjem te usluge. Klijentu je vrlo teško unaprijed znati što će dobiti. Kako bi uvjerili kupca i izgradili njegovo povjerenje, marketinški stratezi moraju pružiti „opipljiv“ dokaz kvalitete usluge. Zbog nematerijalnosti usluge, manja je vjerojatnost da će potrošači promijeniti pružatelja usluga.

S obzirom na nematerijalnost usluga, oglašavanje usluga postaje posebno težak zadatak ali opet i izuzetno važan zadatak. Zbog sve veće homogenosti u ponudi proizvoda, prateće usluge postaju sve bitnije za potrošače kod odabira određenog proizvoda. Primjerice, dva lanca brze hrane koja poslužuju sličan proizvod (Mc Donald's i Burger King), kvaliteta usluge je ta koja stvara razliku.

Neopipljivost usluga izazov je s kojim se suočavaju mnogi marketinški stručnjaci, te oni moraju biti što inovativniji kako bi se njihova usluga razlikovala od konkurenata i uz to podigla njezina vrijednost, a samim time privukla i što više novih korisnika (Lumen Learning, 2020).

2.3.2. Nedjeljivost

Koristi se u marketingu za opisivanje ključne kvalitete usluga koja ih razlikuje od robe. Nedjeljivost je uslužna karakteristika koja onemogućava razdvajanje ponude ili proizvodnje usluge od njene potrošnje. Drugim riječima, usluge se generiraju i konzumiraju u istom vremenskom okviru. Štoviše, vrlo je teško odvojiti uslugu od davatelja usluga. Oni su nerazdvojni.

Marketing usluga sastoji se od ljudi, procesa i fizičkih dokaza koji su svi jedinstveni za marketing usluga. Ljudi su ključni čimbenik u procesu pružanja usluge, jer je usluga neodvojiva od osobe koja je pruža. Primjerice, restoran je poznat koliko po hrani, tako i po usluzi koju pruža njegovo osoblje. Stoga, obuka službe za korisnike postala je glavni prioritet za mnoge tvrtke danas.

Postupak pružanja usluge je vrlo važan jer osigurava da se isti standard usluge isporučuje svakom kupcu. Stoga većina tvrtki ima plan za usluge koji sadrži detaljne postupke pružanja usluge. Takvi planovi se često svode na definiranje skripte usluge i pozdrave koje će službeno osoblje koristi.

Koncept nedjeljivosti ne znači da će se ista usluga isporučiti svakom kupcu, već da će se za svaku uslugu primjenjivati isti standardi kvalitete. Primjerice, kod korištenja usluga frizerskih salona, dvije frizure neće biti slične ali svaki će se klijent tretirati sa istim poštovanjem (Lumen Learning, 2020).

2.3.3. Prolaznost

Koristi se u marketingu za opisivanje načina na koji uslužni kapacitet ne može biti pohranjen za prodaju u budućnosti. Usluge se ne mogu pohraniti, spremiti, vratiti ili preprodati nakon što su korištene. Kad se jednom pruži kupcu, usluga se u potpunosti konzumira i ne može se isporučiti drugom kupcu.

Usluge su prolazne iz dva razloga:

1. Relevantni resursi, procesi i sustavi usluge dodjeljuju se za isporuku tijekom određenog vremenskog razdoblja. Na primjer, zrakoplovna kompanija može prodati sjedala u zrakoplovu samo prije polaska. Ta usluga je dostupna samo za to određeno vremensko razdoblje. Prazno sjedalo u avionu nikada se ne može iskoristiti i naplatiti nakon polaska.
2. Kada je usluga u potpunosti pružena, ta usluga nepovratno nestaje jer ju je potrošač iskoristio. Primjerice, nakon što putnik bude prevezen zrakoplovom do odredišta, u tom trenutku on više ne može biti opet prevezen na tu lokaciju.

Prolaznost može utjecati na uspješnost poduzeća jer je balansiranje ponude i potražnje vrlo teško. Potražnju je teško predvidjeti te ona može varirati ovisno o sezoni, doba dana ili poslovnom ciklusu. Kako potražnja oscilira, održavanje kvalitetne usluge može biti vrlo teško. Na primjer, da se nadoknadi velika potražnju tijekom turističke sezone, hotel može zaposliti više zaposlenika. Međutim, druga vremenska razdoblja nije tako lako predvidjeti. Tijekom sezona lošeg vremena, menadžer se može naći s previše osoblja. Suprotan problem, onaj s premalo osoblja, može biti istinit tijekom neočekivanog porasta potražnje (Lumen Learning, 2020).

2.3.4. Promjenjivost

Iako proizvodi mogu biti homogeni i masovno proizvedeni, isto ne vrijedi za usluge. Pojam heterogenost opisuje jedinstvenost ponude usluga (poznato i kao varijabilnost). Drugim riječima, usluge se generiraju, pružaju i konzumiraju odjednom. Usluga se nikada ne može točno ponoviti kao isto vrijeme, mjesto ili okolnosti, ili s istim konfiguracijama ili resursima, čak i ako isti potrošač zahtijeva istu uslugu.

Mnoge usluge koje se smatraju heterogenim obično se mijenjaju za svakog potrošača ili situaciju. Na primjer, taksi usluga koja potrošača prevozi od točke A to točke B, razlikuje se od taksi usluge koja će ga vratiti do točke A. Svako putovanje podrazumijevalo je različito vrijeme, drugi smjer, a možda i drugu rutu, taksista ili automobil.

S obzirom da su usluge raznolike, bitno je da svaki kupac dobije što bolju uslugu. Heterogenost kvalitete usluge ne znači da dva kupca ne mogu dobiti dobru uslugu već da one ne mogu biti identične. Kad se fizički proizvod ne može lako razlikovati, postoji prostor za prilagodbu usluge prema zahtjevima kupca. Takva prilagodba osigurava da stvarni susret kupca poprima poseban značaj za njih. No, previše prilagodbe moglo bi ugroziti standardno pružanje usluge i negativno utjecati na njezinu kvalitetu. Stoga se posebno treba paziti na osmišljavanju ponude usluge (Lumen Learning, 2020).

3. SQL i NoSQL baze podataka

Kad je u pitanju odabir baze podataka, jedna od najvećih odluka je da li odabrati relacijsku (SQL) ili ne-relacijsku (NoSQL) bazu podataka. Iako su i jedna i druga dobar izbor, svaka ima svoje prednosti odnosno nedostatke.

3.1. Što je SQL baza podataka?

SQL baze podataka podržavaju SQL DSL[1] za slanje upita i manipuliranje podataka u relacijskoj bazi podataka. „Relacijsko“ u relacijskoj bazi podataka se odnosi na model „relacijski model“ upravljanja podacima koji je ranih 1970-ih osmislio IBM-ov istraživač Edgar Frank Codd i popularizirao ga u velikom broju kasnijih sustava baza podataka počevši od System R[2]-a (Anderson i Nicholson, 2020).

Relacijske baze podataka koriste relacije (obično se nazivaju tablice) za pohranjivanje podataka i potom se slažu koristeći zajedničke karakteristike u skupu podataka. Iako SQL nije jedini mogući jezik za provedbu upita relacijskog modela, on je daleko najpopularniji. Neki od uobičajenih sustava za upravljanje relacijskim bazama podataka koji koriste SQL su PostgreSQL, MySQl, Oracle, CockroachDB.

Cobb-ov prolazni rad opisuje bazu podataka na kojoj se objekti mogu graditi i ispitivati koristeći nešto što je on nazvao SQL (Structured Query Language), strukturirani jezik upita. SQL je koristio za izradu podataka (u objektima zvanim tablice) i sheme za te podatke, koja opisuje polja u stupcima. Pojedini zapis u SQL bazi podataka naziva se redom (Shiff i Rowe, 2018).

U nastavku su navedeni prednosti i nedostaci SQL baza podataka.

[1] DSL (Domain Specific Language) - specijalizirani jezik koji se koristi u određenu svrhu
[2] System R - sustav baza podataka izgrađen kao istraživački projekt u IBM-ovom istraživačkom laboratoriju San Jose početkom 1974.

Prednosti:

- Smanjeni trag pohrane podataka (engl. data storage footprint) zbog normalizacije i drugih mogućnosti optimizacije
- Snažna i dobro razumljiva semantika integriteta podataka kroz ACID[3] (nedjeljivost transakcije, konzistentnost, izolacija i izdržljivost).
- Standardni pristup podacima putem SQL-a
- Općenito fleksibilnija podrška za upite koja može podnijeti veća opterećenja. SQL apstraktno pregledava temeljnu implementaciju i omogućava mehanizmu (engl. engine) da optimizira upite kako bi odgovarao njihovom prikazu na disku.

Nedostaci:

- „Kruti“ modeli podataka koji unaprijed zahtijevaju pažljiv dizajn kako bi osigurali odgovarajuću radni učinak i „oduprli“ se razvoju evolucijske promjene sheme, često uzrokuje prekid rada.
- Izazovno horizontalno skaliranje – ili je u potpunosti ne podržano, podržano na ad-hoc način ili samo na relativno nezrelim tehnologijama.
- Ne distribuirani mehanizmi su općenito „jedna točka kvara“ koja se mora umanjiti tehnikama kopiranja i tehnikama povratka (Anderson i Nicholson, 2020).

3.2. Što je NoSQL baza podataka?

NoSQL se počeo pojavljivati sredinom 2000. godine. NoSQL baze podataka su samoopisujuće, to znači da ne zahtijevaju shemu. Također ne nameću odnose između tablica u svim slučajevima. Svi dokumenti su JSON[4] dokumenti, koji su cjeloviti entiteti koje je lako pročitati i razumjeti. NoSQL se odnosi na neformalne baze podataka visokih performansi koje koriste širok raspon modela podataka. Te su baze podataka vrlo

[3] ACID (Atomicity, Consistency, Isolation, Durability) - skup svojstava transakcija baze podataka koje su najmjenjene da jamče valjanosti podataka usprkos greškama, nestancima napajanja i drugim .
[4] JSON (JavaScript Object Notation) – otvoreni standardni format datoteke i format razmjene podataka.

prepoznate po jednostavnosti upotrebe, skalabilnim performansama i širokoj dostupnosti. Primjeri nekih NoSQL baza podataka uključuju MongoDB, Cassandra, CloudDB, Firebase (Shiff i Rowe, 2018).

U nastavku su navedeni prednosti i nedostaci NoSQL baza podataka.

Prednosti:

- Skalabilnost i visoka dostupnost - mnoge NoSQL baze podataka obično su dizajnirane da podrže besprijekornu, internetsku horizontalnu skalabilnost bez značajnih točki neuspjeha.
- Fleksibilni modeli podataka - većina nerelacijskih sustava ne zahtijeva od programera da unaprijed naprave obveze prema modelima podataka.
- Visoke performanse – ograničavanjem onoga što baza podataka može učiniti (primjer, ublažavanjem garancija trajnosti), mnogi NoSQL sustavi mogu postići izuzetno visoke razine performansi.
- Visoka razina apstrakcije podataka – prelazeći sa modela podataka „vrijednost u ćeliji“, NoSQL sustavi mogu pružiti API-je visoke razine za složene strukture podataka.

Nedostaci:

- Nejasne interpretacije ACID ograničenja - unatoč široko rasprostranjenim tvrdnjama o ACID podršci za NoSQL sustave, interpretacija ACID-a često je toliko široka da se ne može mnogo toga shvatiti o semantici dotične baze podataka (primjer, što znači „izolacija“ bez transakcija).
- Nedostatak fleksibilnosti u pristupnim uzorcima - relacijska / SQL apstrakcija daje mehanizmu baze podataka široke ovlasti za optimizaciju upita za osnovne podatke, bez te apstrakcije, prikaz podataka na disku curi u aplikacijske upite i ne ostavlja mjesta za mehanizam da napravi optimizaciju (Anderson i Nicholson, 2020).

4. PostgreSQL

PostgreSQL je objektno-relacijski sustav baze podataka sa otvorenim izvornim kodom (engl. open source). Trenutno PostgreSQL nudi značajke poput složenih upita, stranih ključeva, okidača, pogleda, transakcijskog integriteta, pretraživanja cijelog teksta i ograničene replikacije podataka. Korisnici mogu proširiti PostgreSQL novim tipovima podataka, funkcijama, operatorima ili indeksnim metodama. Podržava razne programske jezike (uključujući C, C ++, Java, Perl, Tcl i Python), kao i sučelja baze podataka JDBC i ODBC.

PostgreSQL je izdan pod BSD licencom koja svima daje dozvolu za upotrebu, izmjenu i distribuciju PostgreSQL koda i dokumentacije u bilo koju svrhu bez ikakve naknade. PostgreSQL se koristi za implementaciju nekoliko različitih istraživanja i proizvodnih aplikacija (poput PostGIS sustava za zemljopisne informacije) i koristi se kao obrazovno sredstvo na nekoliko sveučilišta. PostgreSQL se i dalje razvija kroz doprinose velike zajednice programera (Alagiannis i Borovica-Gajic, 2019).

4.1. Interakcija sa PostgreSQL-om

Standardna distribucija PostgreSQL-a dolazi s alatima naredbenih linija[5] (engl. command-line) za administraciju baze podataka. Međutim postoji širok spektar komercijalnih open source grafičkih alata za administraciju i dizajniranje koji podržavaju PostgreSQL.

[5] Naredbena linija – tekstualno sučelje za računalo, tj. program koji prima naredbe i te naredbe prenosi na operativni sustav računala kako bi se izvršile

4.1.1. Interaktivna terminalna sučelja

Kao i većina baza podataka, PostgreSQL nudi command-line alate za administriranje baze podataka. *Psql* interaktivni terminal podržava izvršavanje SQL naredbi na poslužitelju i pregled rezultata.

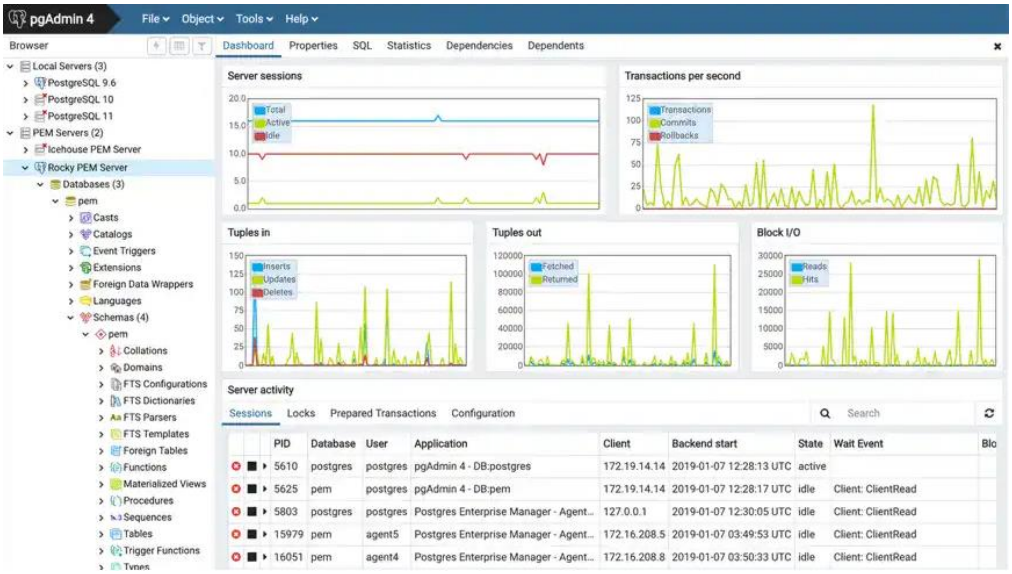
Neke od karakteristika su:

- **Varijable** – *psql* pruža varijabilne mogućnosti zamjene, slično uobičajenim Unix naredbama.
- **SQL interpolacija** - korisnik može zamijeniti ("interpolirati") psql varijable u SQL naredbe stavljanjem dvotočka ispred imena varijable.
- **Uređivanje komandnih linija** (engl. command-line editing) – *psql* koristi *GNU readline* biblioteku koja pruža set funkcija za uređivanje naredbenih linija, sa podrškom za automatsko dopunjavanje djelomično napisanih naredbi[6] (Alagiannis i Borovica-Gajic, 2019).

4.1.2. Grafička sučelja

Standardna distribucija PostgreSQL-a ne sadrži nikakve grafičke alate. Međutim, postoji nekoliko alata otvorenog koda kao i komercijalnih alata za grafičko korisničko sučelje za zadatke kao što su SQL razvoj, administriranje baze podataka, modeliranje odnosno dizajn baze podataka i generiranje izvještaja. Grafički alati za razvoj SQL-a i administraciju baze podataka uključuju pgAdmin, PgManager i RazorSQL.

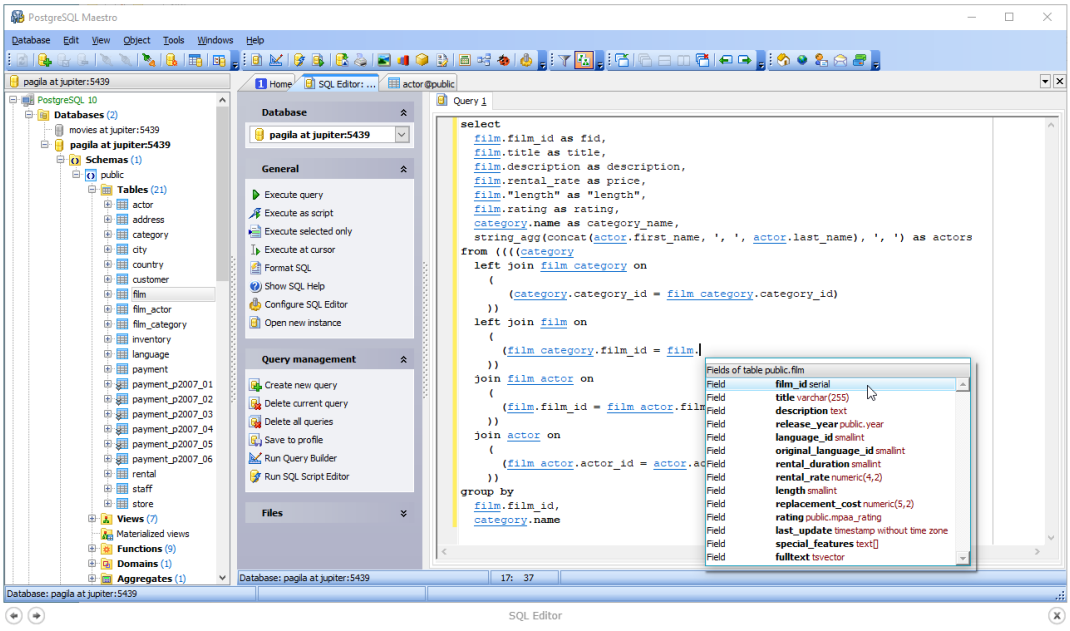
[6] Automatsko dopunjavanje djelomično napisanih naredbi – eng. tab-completion



Slika 2. Sučelje pgAdmin-a

Izvor: https://www.pgadmin.org/

Alati za dizajn baze podataka uključuju TOra, Power * Architect i PostgreSQL Maestro. Nadalje, PostgreSQL surađuje s nekoliko komercijalnih oblika alata za oblikovanje i generiranje izvještaja poput Reportizer, dbForge i Tour Tour Database (Alagiannis i Borovica-Gajic, 2019).



Slika 3. Sučelje PostgreSQL maestro-a

Izvor: https://www.sqlmaestro.com/products/postgresql/maestro/

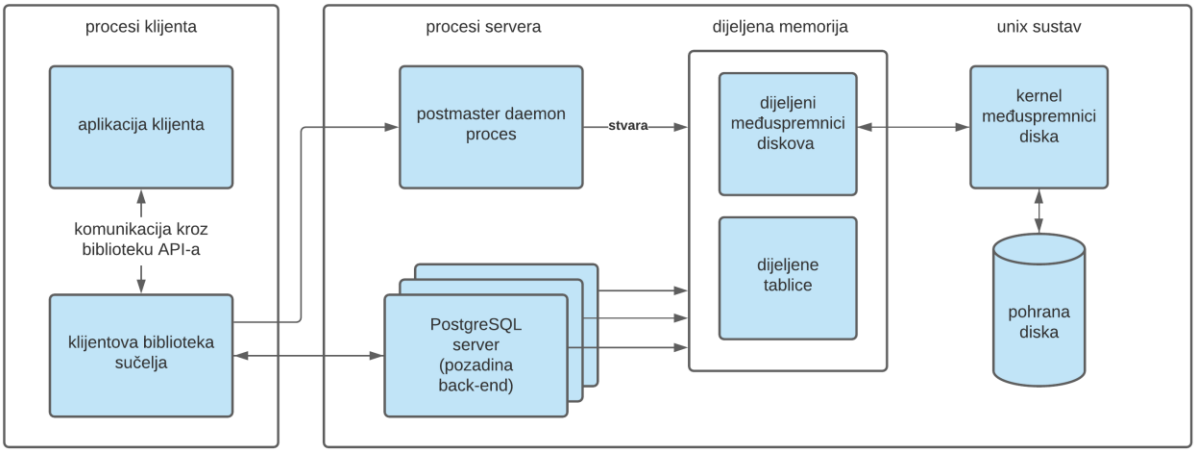
4.1.3 Programiranje jezičnih sučelja

PostgreSQL standardna distribucija uključuje dva korisnička sučelja a to su libpq i *ECPG*. Libpq je skup knjižničnih funkcija koje omogućuju klijentskim programima da prosljeđuju upite PostgreSQL pozadinskom serveru i da dobiju rezultate tih upita. (PostgreSQL)

Biblioteka *libpq* podržava sinkrono i asinkrono izvršavanje SQL naredbi i pripremljenih izjava, putem sučelja ponovnog ulaska (engl. reentrant) i sučelja sigurne dretve (engl. thread-safe). Parametri veze libpq-a mogu se konfigurirati na nekoliko fleksibilnih načina. Jedan od načina je postavljanje varijabli okruženja, postavljanje postavki u lokalnu datoteku ili stvaranje unosa na LDAP (Lightweight Directory Access Protocol) poslužitelju. ECPG je ugrađeni SQL predprocesor za C jezik i PostgreSQL. Omogućuje pristup PostgreSQL-u pomoću C programa s ugrađenim SQL kodom. ECPG pruža fleksibilno sučelje za povezivanje s poslužiteljem baze podataka, izvršavanje SQL izraza i dobivanje rezultata upita. Osim korisničkih sučelja koja su uključena u standardnu distribuciju PostgreSQL-a, postoje i klijentska sučelja koja su ustupljena od vanjskih projekata. Oni uključuju izvorna sučelja za ODBC (Open Database Connectivity) i JDBC (Java Database Connectivity), kao i povezivanje za većinu programskih jezika, uključujući C, C ++, PHP, Perl, Tcl / Tk, JavaScript, Python i Ruby (Alagiannis i Borovica-Gajic, 2019).

4.2 Arhitektura PostgreSQL sustava

PostgreSQL sustav temelji se na više procesnoj arhitekturi, kao što je prikazano na sljedećoj slici (Slika 4).



Slika 4. Arhitektura PostgreSQL sustava

Izvor: https://www.db-book.com/db7/online-chapters-dir/32.pdf

Postmaster je središnji koordinacijski proces i odgovoran je za inicijalizaciju sustava (uključujući dodjelu zajedničke memorije i pokretanje pozadinskih procesa), te za gašenje poslužitelja. Također, postmaster upravlja vezama s klijentskim aplikacijama i dodjeljuje svakog novog povezujućeg klijenta sa pozadinskim (engl. backend) serverskim procesom za izvršavanje upita u ime klijenta i vraćanje rezultata klijentu.

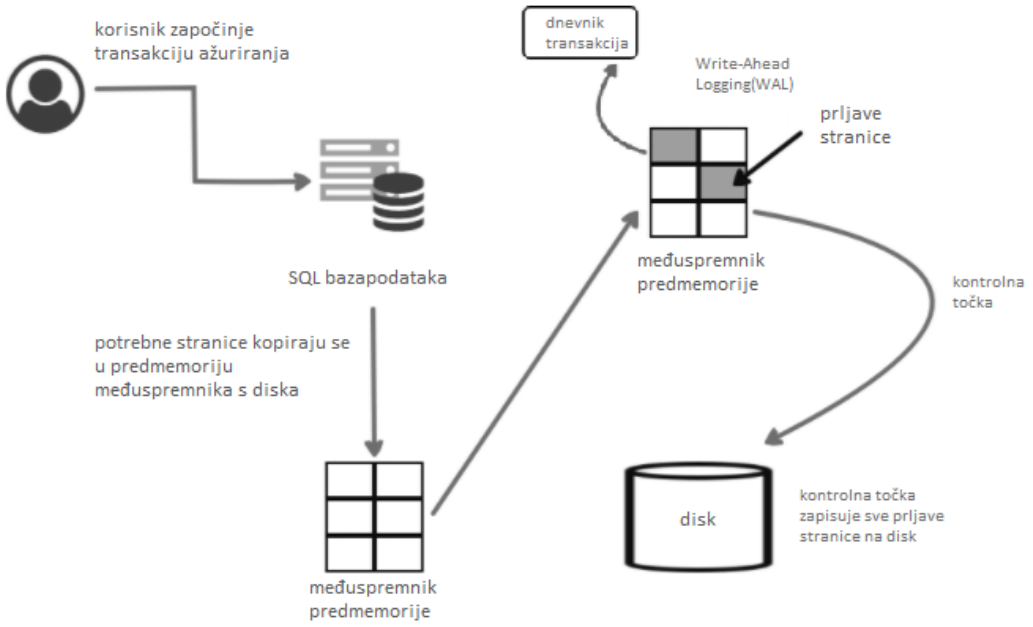
Klijentske aplikacije mogu se povezati na PostgreSQL poslužitelj i slati upite putem jednog od mnogih sučelja aplikacijskog programa baze podataka koje podržava PostgreSQL (libpq, JDBC, ODBC) koje su ponuđene kao biblioteke klijentske strane. Primjer klijentske aplikacije je program *psql* naredbenog retka (engl. command-line), uključen u standardnu PostgreSQL distribuciju. Postmaster je odgovoran za rukovanje početnim vezama s klijentima. Zbog toga stalno osluškuje nove veze na poznatom portu. Kad primi zahtjev za povezivanje, postmaster prvo izvršava korake inicijalizacije, poput provjere autentičnosti korisnika, a zatim dodjeli neaktivan pozadinski serverski proces (ili stvara novi ako je potreban) radi rukovanja s novim klijentom. Nakon ove početne veze, klijent stupa u interakciju samo s procesom pozadinskog servera, podnoseći upite i primajući rezultate upita. Dok je aktivna veza s klijentom, dodijeljeni pozadinski serverski proces posvećen je samo toj vezi klijenta. Stoga PostgreSQL koristi proces-po-konekciji model za pozadinski server.

Pozadinski serverski proces je odgovaran za izvršavanje upita koje je klijent poslao izvodeći potrebne korake izvršenja upita, uključujući raščlanjivanje (engl. parsing), optimizaciju i izvršavanje. Svaki pozadinski serverski proces može istovremeno obraditi samo jedan upit. Aplikacija koja želi istodobno izvršiti više upita mora održavati više veza s poslužiteljem. U bilo koje trenutku na sustav može biti spojeno više klijenata i zbog toga se višestruki pozadinski serverski procesi mogu izvršavati istovremeno.

Pored postmaster-a i pozadinskog serverskog procesa, PostgreSQL koristi nekoliko radnih pozadinskih procesa za obavljanje zadataka upravljanja podacima, uključujući pozadinski pisač (engl. backend writer), sakupljač statistike (engl. statistics collector), WAL (the write-ahead log) pisač i procese kontrolnih točaka (engl. checkpointer process) (Alagiannis i Borovica-Gajic, 2019).

Proces pozadinskog pisača odgovaran je za povremeno pisanje „prljavih stranica“ (engl. dirty pages) iz zajedničkih međuspremnika u trajno pohranjivanje:

- Prvo, SQL Server pokušava pronaći stranicu u međuspremniku predmemorije. Ako ne pronađe stranicu, SQL Server dobiva tu stranicu s diska u međuspremniku. Ova je stranica poznata kao čista stranica jer ne sadrži nikakve promjene.
- SQL Server nabavlja zaključavanja na stranicama, na razini retka i vrši ažuriranje zapisa. Modificirana stranica poznata je kao „prljava stranica“. Na dolje navedenoj slici mogu se vidjeti prljave stranice (Slika 5).
- To stvara zapise dnevnika koji opisuje izvršene promjene. SQL Server zapisuje zapise dnevnika u dnevnik transakcija te korisniku šalje potvrdu. Promijenjena stranica je i dalje u međuspremniku predmemorije.
- Kontrolna točka zapisuje sve prljave stranice (dostupne u međuspremniku predmemorije) i dnevnik transakcijski zapisa na disk. Također bilježi podatke o kontrolnoj točki u dnevniku transakcija (Gupta 2019).



Slika 5. Zapisivanje prljavih stranica (engl. Dirty pages)

Izvor: https://www.sqlshack.com/sql-server-checkpoint-lazy-writer-eager-writer-and-dirty-pages-in-sql-server/

Proces sakupljača statistike kontinuirano prikuplja statističke podatke o pristupima tablici i broju redaka u tablicama. Proces WAL pisača periodično pošalje podatke WAL-a u trajno pohranjivanje, dok proces kontrolnog pokazivača provodi kontrolne točke baze podataka kako bi se ubrzao oporavak. Ovi pozadinski procesi pokreću se procesom *postmastera*.

Kada je u pitanju upravljanje memorijom u PostgreSQL-u, postoje dvije različite kategorije a to su lokalna memorija i zajednička memorija. Svaki pozadinski proces dodjeljuje lokalnu memoriju za vlastite zadatke kao što su obrada upita (hash tablice internih operacija sortiranja i privremene tablice) i operacije održavanja (kreiranje indeksa).

S druge strane područje glavne memorije (eng. buffer pool) koje je upravitelj baze podataka dodijelio za predmemoriranje tablica i podataka indeksa dok se čitaju s diska, ono se nalazi u zajedničkoj memoriji. Na taj način mu mogu pristupiti svi procesi, uključujući i pozadinske serverske procese i ostale pozadinske procese. Zajednička memorija također se koristi za pohranjivanje tablica zaključavanja i drugih podataka koje server i pozadinski procesi moraju dijeliti (IBM Knowledge Center).

Zbog korištenja zajedničke memorije kao međuprocesnog komunikacijskog medija, PostgreSQL poslužitelj trebao bi se pokrenuti na jednom uređaju s zajedničkom memorijom. Web mjesto s jednim poslužiteljem se ne može izvršiti na više uređaja koji ne dijele memoriju bez pomoći paketa trećih strana. Međutim, moguće je izgraditi sustav paralelnih baza podataka sa shared-nothing arhitekturom s instancom PostgreSQL-a koja se izvodi na svakom čvoru. Shared-nothing arhitektura se bazira na podjeli podataka po serveru, gdje je bi svaki server bio zasebno odvojeno skladište podataka (Alagiannis i Borovica-Gajic, 2019).

4.3. PostgreSQL u Google Cloudu

Cijena SQL-a u oblaku za PostgreSQL sastoji se od sljedećih troškova:

- Cijena instance
- Cijena CPU[7]-a
- Cijena skladišta
- Cijena mreže

1. **Cijena instance**

   Cijene instanci za Cloud SQL za PostgreSQL primjenjuju se samo na instance s dijeljenom jezgrom. Namjenske jezgre, koje mogu sadržavati do 96 vCPU-a (virtual CPU) i 416 GB memorije, naplaćuju se po broju jezgara i količini memorije koju imaju. Kopije za čitanje naplaćuju se po istoj stopi kao i zasebne instance. Cijene instanci također ovise o regiji u kojoj se nalazi instanca. Na dolje priloženoj slici (Slika 6) su iskazane mjesečne cijene ukoliko se za regiju instance odabere London (Google Cloud).

[7] CPU (Central processing unit) – Središnja jedinica za obradu – procesor.

Tip uređaja sa zajedničkom jezgrom	Viirtuali CPU-ovi	RAM (GB)	Maksimalni skladišni kapacitet (GB)	Cijena $
db-f1-micro*	dijeljeni	0.6	3,062	9.20
db-g1-small*	dijeljeni	1.7	3,062	30.66
db-f1-micro*	dijeljeni	0.6	3,062	18.40
db-g1-small*	dijeljeni	1.7	3,062	61.32

HA (high availability) cijene primjenjuju se za instance konfigurirane za visoku dostupnost te se one još nazivaju u regionalne instance.

Slika 6. Cijena instance CloudSQL-a za regiju London

Izvor: Google Cloud

2. Cijena CPU-a i memorije

Za slučajeve instanci namjenskih jezgri (engl. dedicated-core instances), odabire se željeni broj CPU-a i količina memorije. Maksimalan broj CPU-a je 96 te memorije do 416 GB. Cijena CPU-a i memorije ovisi o regiji u kojoj se nalazi instanca (Google Cloud).

	Cijena ($)	jednogodišnja obveza ($)	trogodišnja obveza ($)
količina vCPU-ova	dijeljeni	27.13	17.37
memorija	dijeljeni	4.60	2.94
HA količina vCPU-ova	dijeljeni	54.26	34.72
HA meorija	dijeljeni	9.20	5.89

HA (high availability) cijene primjenjuju se za instance konfigurirane za visoku dostupnost te se one još nazivaju u regionalne instance.

Slika 7. Cijena CPU-a i memorije za regiju London

Izvor: Google Cloud

3. Cijena pohrane i mreže

PostgreSQL pohrana i umrežavanje cijene ovise o regiji u kojoj se nalazi instanca (Slika 8). (Google Cloud)

	Cijena
Pohrana	* 0,204 $ po GB / mjesečno za kapacitet SSD-a * 0,108 $ po GB / mjesečno za kapacitet HDD-a * 0,096 $ po GB / mjesečno za izradu sigurnosnih kopija
HA pohrana	* 0,408 $ po GB / mjesečno za kapacitet SSD-a * 0,216 $ po GB / mjesečno za kapacitet HDD-a * 0,096 $ po GB / mjesečno za izradu sigurnosnih kopija
Mreža	Ulazni promet u Cloud SQL: Besplatno Izlazazni promet iz Cloud SQL-a: Navedeno dolje niže IPv4 adrese: 0,012 $ na sat u mirovanju

Slika 8. Cijena pohrane i umrežavanja za regiju London

Izvor: Google Cloud

4. Cijena izlaznog mrežnog prometa

Kada mrežni promet napusti Cloud SQL instancu, primijenjena naknada ovisi o odredištu prometa, a u nekim slučajevima i je li partner uključen. Izlazni internet promet (engl. egress) je mrežni promet koji ostavlja CloudSQL instancu klijentu koji nije Googleov proizvod, kao što je primjerice korištenje lokalnog poslužitelja za čitanje podataka iz Cloud SQL-a (Google Cloud).

	Cijena
Instance Compute Engine-a i kopije križnih regija CloudSQL-a	U istoj regiji: besplatno Između regija unutar Sjeverne Amerike: 0,12 USD / GB Između regija izvan Sjeverne Amerike: 0,12 USD / GB
Googleovi proizvodi (izuzev Instance Compute Engine-a i kopija križnih regija CloudSQL-a)	Unutar kontinentalni: besplatno Međukontinentalni: 0,12 $ / GB
Ulazni internet promet koristeći Cloud Interconnect	0.05$ / GB
Ulazni internet promet bez korištenja Cloud Interconnect-a	0.19$ / GB

Slika 9. Cijena izlaznog mrežnog prometa

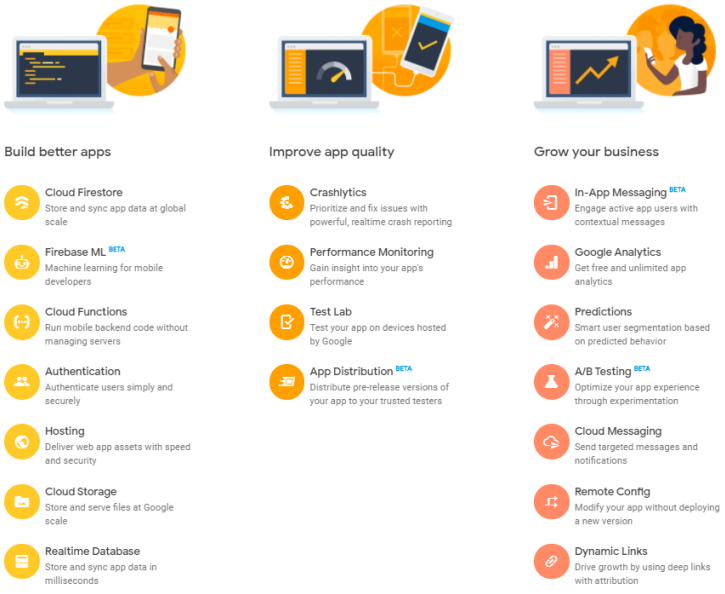
Izvor: Google Cloud

5. Firebase

Firebase pruža Realtime Database Firebase (Baza podataka u stvarnom vremenu) i Cloud Firestore baze podataka. Obje baze podataka su temeljene na oblaku, te predstavljaju rješenja pristupačna za klijente koja podržavaju sinkronizaciju podataka u stvarnom vremenu.

Firebase je nastao u travnju 2012., a Google ga je kupio 2014. godine kako bi dao podršku programerima. Kombinacija značajki u Firebaseu ubrzava integraciju baze podataka kako za web tako i mobilne aplikacije. Firebase je Baas (Backend as a Service – Backend kao usluga), što smanjuje konfiguraciju i logičko podešavanje bez trošenja vremena. (Yahiaoui 2017.)

Firebase je pružio niz razvojnih rješenja dizajniranih kako bi se ubrzala integracija značajki utemeljenih na oblaku za mobilne i web aplikacije. Nakon kupnje tvrtke, Google je kombinirao usluge Firebase-a s nizom komplementarnih značajki koje su prethodno bile uključene u dio Google Cloud Platform-a. Kombinirane značajke dviju platformi danas su jednostavno poznate kao Firebase.



Slika 10. Firebase proizvodi

Izvor: Firebase

5.1 Realtime baza podataka

Realtime odnosno baza podataka u realnom vremenu je baza podataka koja se nalazi u oblaku. Omogućuje sigurno pohranjivanje podataka na Google Cloud poslužiteljima i sinkronizaciju u stvarnom vremenu sa svim klijentima koji dijele istu bazu podataka. Značajka Realtime baze podataka Firebase-a je da omogućuje dijeljenje podataka između više korisnika, pri čemu „korisnik“ može imati oblik aplikacije koji radi na Android i iOS mobilnim uređajima, ili JavaScript koji se izvode na web poslužitelju.

Glavni cilj sustava je osigurati siguran, pouzdan i brz način za sinkronizaciju podataka s minimalnom količinom napora kodiranja od strane programera. Sustav baze podataka također je dizajniran kako bi podržao milijune korisnika. Baza podataka se naziva „realtime“ jer je brzina kojom se podaci sinkroniziraju na klijentima vrlo vjerojatno jednako bliska realnom vremenu koliko je to za sad moguće (uzimajući u obzir fizičko ograničenje prijenosa podataka putem interneta i bežične veze). Realtime baza podataka također omogućuje postojanje podataka tako što ih sprema lokalno, te to uključuje mogućnost dohvata podataka čak i kad je uređaj izvan mreže.

Firebase koristi NoSQL bazu podataka za pohranu podataka u Realtime bazu podataka. To znači da se podaci ne pohranjuju u tablice i retke koji se nalaze u sustavima za upravljanje relacijskim bazama podataka. Također podacima se ne pristupa preko SQL izraza. Umjesto toga podaci se pohranjuju u obliku JSON objekta i on definira sintaksu koja se koristi za prijenos podataka koji je lagan i jednostavan za čitanje, pisanje i razumijevanje.

Ključne karakteristike:

- Realno vrijeme - Umjesto uobičajenih HTTP zahtjeva, Firebase Realtime baza podataka koristi sinkronizaciju podataka. Svaki put kada se podaci promijene, svaki povezani uređaj primi to ažuriranje u nekoliko milisekundi.

- Izvan mreže – Firebase aplikacije ostaju responzivne čak i kad su izvan mreže jer SDK Realtime baze podataka zadržava sve podatke na disku. Jednom kada se uspostavi povezanost, uređaj klijenta prima sve promjene koje je propustio sinkronizirajući ih s trenutnim stanjem poslužitelja.
- Dostupna s klijentskih uređaja – Firebase bazi podataka se može pristupiti izravno s mobilnog uređaja ili web preglednika, to jest nema potrebe za aplikacijom. Sigurnost i provjera podataka dostupni su putem Firebase Realtime sigurnosnih pravila baze podataka te su to pravila koja se temelje na izrazima, a koja se izvršavaju kada se podaci čitaju ili pišu.
- Skaliranje preko više baza podataka – Pomoću Firebase Realtime baze podataka, postoji mogućnost podrške za zahtjeve podataka aplikacije sa dijeljenjem podataka na više instanci baza podataka u istom Firebase projektu. Pruža mogućnost provjere autentičnosti sa *Firebase Authenticaion*-om na projektu u svim instancama baze podataka. Također je moguće kontrolirati pristup podacima u svakoj bazi podataka sa prilagođenim *Firebase Realtime Database Rules* pravilima (Firebase 2020).

Realtime baza podatka je NoSQL baza podataka i kao takva ima različite optimizacije i funkcionalnosti u odnosu na relacijsku bazu podataka. API Realtime baze podataka je dizajniran tako da dopušta samo one operacije koje se mogu brzo izvršiti. To omogućuje vrlo dobro iskustvo u realnom vremenu koje može poslužiti milijune korisnika bez narušavanja reaktivnosti (Firebase 2020).

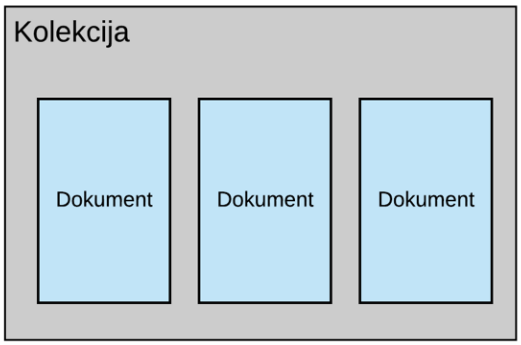
5.2. Firestore

U proteklih nekoliko godina Firebase proizvodi su se razvijali i s Googleovim integracijama, Firebase, pogonjen Googleom, jednostavno je postao snažniji nego prije. Takva snaga značila je ograničenja za trenutne proizvode. Programeri su puno puta smatrali zbunjujućim s obzirom kako ih treba koristiti za jednostavno spremanje podataka (Yahiaoui, 2017:256).

Cloud Firestore je najnovija baza podataka Firebase-a za razvoj mobilnih aplikacija. Gradi se na uspjesima baze podataka u stvarnom vremenu no s novim, intuitivnijim modelom podataka. Cloud Firestore također sadrži bogatije i brže upite te skalira dalje od baze podataka u stvarnom vremenu (Firebase, 2020).

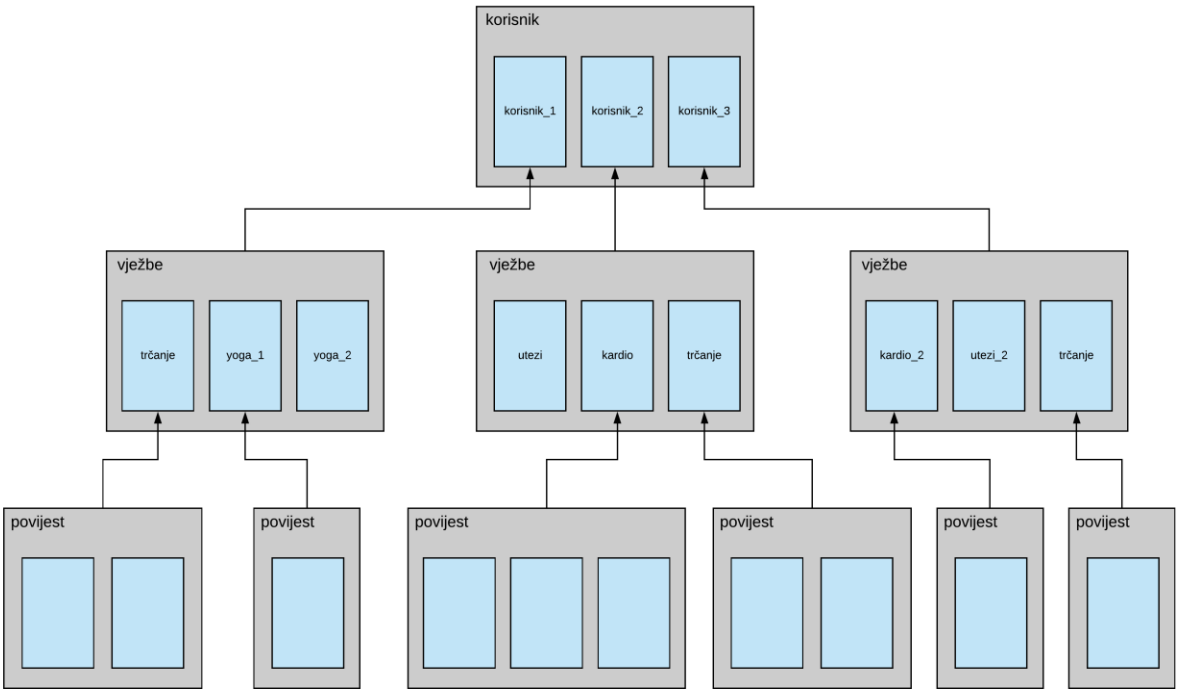
5.2.1. Razlika Firestore-a i Realtime baze podataka

Dok je Firebase Realtime baza podataka u osnovi ogromno JSON stablo gdje sve ide i vlada bezakonje, Cloud Firestore je strukturiraniji. Cloud Firestore je dokument-model baza podataka, što znači da su svi podaci pohranjeni u objektima koji se nazivaju dokumenti koji se sastoje od parova ključ-vrijednost, a te vrijednosti mogu sadržavati bilo koji broj podataka, kao što su stringovi, float podaci, binarni podaci te JSON slični objekti. Ti dokumenti su grupirani u kolekcije.



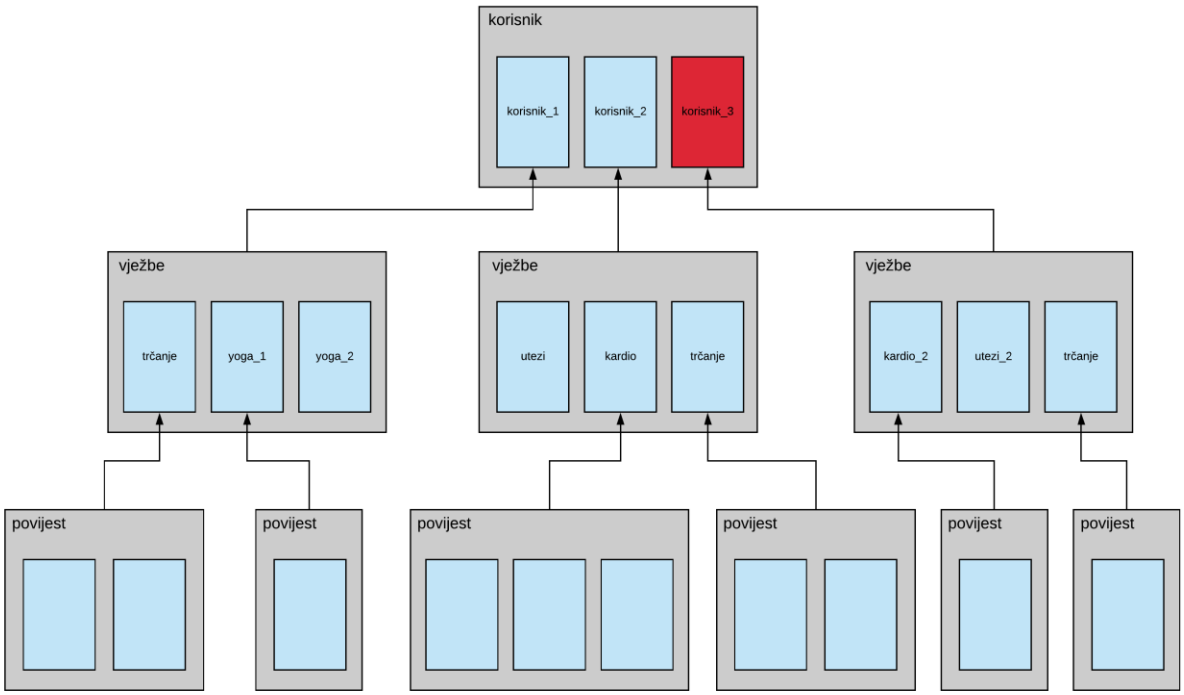
Slika 11. Struktura kolekcije

Cloud Firestore baza podataka vjerojatno će sastojati od nekoliko kolekcija koje sadrže dokumente koji upućuju na podkolekcije. Ove podkolekcije sadržavat će dokumente koji upućuju na druge podkolekcije i tako dalje (Kerpelman, 2017).



Slika 12. Struktura kolekcije sa podkolekcijama

Ova nova struktura daje nekoliko važnih prednosti kod pretraživanja podataka. Za početak svi upiti su površni, što znači da se dokument može jednostavno preuzeti bez potrebe za dohvaćanjem svih podataka sadržanih u bilo kojoj od povezanih podkolekcija. To znači da je hijerarhijski moguće podatke pohraniti na način koji ima logično smisla bez brige o preuzimanju tona nepotrebnih podataka (Kerpelman, 2017).



Slika 13. Primjer dohvaćanja dokumenta iz kolekcije

Cloud Firestore ima jače mogućnosti upita od Realtime baze podataka. U Realtime bazi podataka pokušaj kreiranja upita kroz više polja obično znači puno posla i obično uključuje denormalizaciju[8] podataka (Kerpelman, 2017).

[8] Denormalizacija - znači spremanje istih podataka na više mjesta čime se povećava zalihost (redundancija). Time se povećava brzina odgovora na račun povećanja baze.

Primjerice ako u bazi postoji popis gradova, a želi se pronaći popis svih gradova u Kaliforniji sa populacijom većom od 500 000.

Slika 14. Primjer Firebase Realtime baze podataka

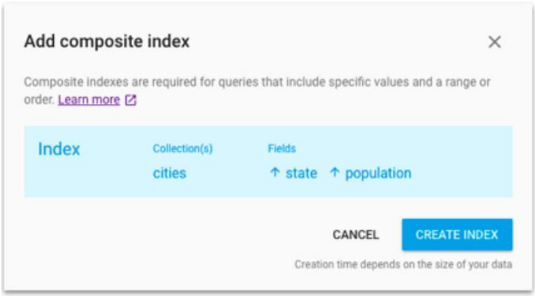
Izvor: https://firebase.googleblog.com/2017/10/cloud-firestore-for-rtdb-developers.html

U Realtime bazi podaka je potrebno provesti ovo pretraživanje izradom eksplicitnog polja "države plus populacija", a zatim pokrenuti upit sortiran na tom polju.

Slika 15. Primjer izrade eksplicitnog polja u Realtime bazi podataka

Izvor: https://firebase.googleblog.com/2017/10/cloud-firestore-for-rtdb-developers.html

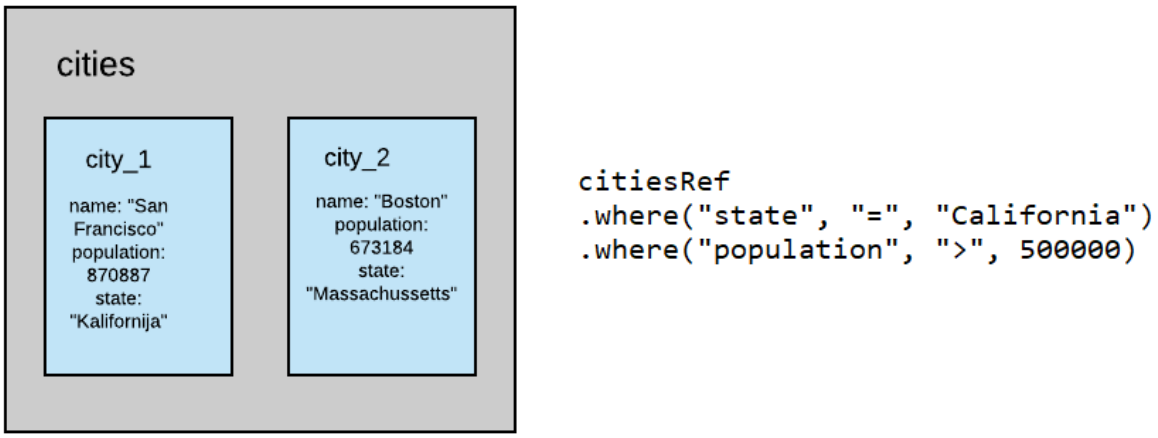
S Cloud Firestore-om to više nije potrebno. U nekim slučajevima Cloud Firestore može automatski pretraživati u više polja. U drugim slučajevima, poput ovog primjera sa gradovima, Cloud Firestore će korisnika voditi prema automatskom sastavljanju indeksa potrebnog za omogućavanje takvih upita (Kerpelman, 2017).



Slika 16. Cloud Firestore automatsko sastavljanje indeksa

Izvor: https://firebase.googleblog.com/2017/10/cloud-firestore-for-rtdb-developers.html

Nakon toga je lako pretraživati kroz više polja kao što je prikazano na sljedećoj slici (Slika 17).



Slika 17. Primjer pretraživanja u Cloud Firestore-u kroz više polja

Cloud Firestore će automatski zadržati ovaj indeks tokom životnog vijeka aplikacije. Nije potrebno kreiranje kombiniranog polja.

5.2.2. Obilježja Firestore-a

1. **Dizajniran za skaliranje**

Dok Realtime baza podataka radi skaliranje kako bi se udovolljila potreba za mnoge aplikacije, stvari mogu postati otežavajuće ukoliko aplikacija postane popularna illi ako skup podataka postane jako masivan.

S druge strane, Cloud Firestore je izgrađen na istoj Google Cloud infrastrukturi koja pokreće neke prilično popularne aplikacije. Na taj način moguće je puno lakše skalirati i na što većem skupu podataka nego što može Realtime baza podataka. Dok Realtime baza podataka podržava otprillike oko 100 000 istodobnih veza, Coud Firestore prihvaća oko 1 000 000 istodobnih veza klijenata po bazi podataka.

Cloud Firestore također ima robusniji Service Level Agreement (Ugovor o razini usluge) u odnosu na Realtime bazu podataka. Cloud Firestore jamči 99,999% neprekidnog rada u više i 99,99% neprekidnog rada u regionalnim slučajevima.

Uz novu strukturu upita, svi Cloud Firestore upiti skaliraju se prema veličini skupa rezultata, a ne prema veličini podataka. To znači da će primjerice potraga za najboljih 10 restorana u nekom gradu sa aplikacijom za recenziju restorana potrajati isto vrijeme, bilo da u bazi podataka ima 300 restorana, 300 tisuća ili 30 milijuna (Kerpelman, 2017).

2. **Olakšano ručno dohvaćanje podataka**

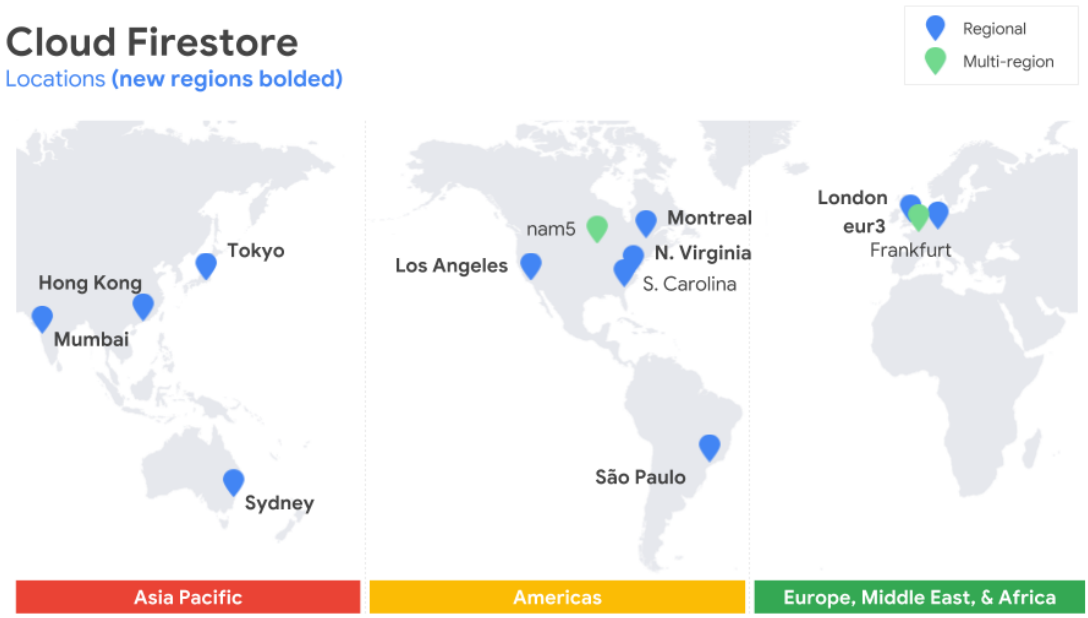
Iako baza podataka u stvarnom vremenu podržava ".once" upite, oni ponekad mogu biti neprirodni za korištenje. S Cloud Firestore-om, postavljanje jednostavnih jednokratnih upita je puno prirodnije i izrađeno je kao primarna funkcija unutar Firestore API-a. Moguće je postaviti podršku za "slušatelje" (engl. listener[9]), tako da klijenti imaju mogućnost primanja ažuriranja kad god se podaci promijene u bazi podataka.

[9] Slušatelj - zaseban proces koji se izvodi na računalu poslužitelja baze podataka. Ona prima dolazne zahtjeve za povezivanje klijenta i upravlja prometom tih zahtjeva na poslužitelju baze podataka.

3. Više-regionalna podrška

Cloud Firestore ima podršku za multi-regionalne lokacije. To znači da se podaci automatski kopiraju u više geografski zasebnih regija odjednom. Dakle, ako bi neka nepredviđena katastrofa učinila da se podatkovni centar, ili čak cijela regija, postavi izvan mreže, postoji sigurnost da će se podaci i dalje posluživati. Cloud Firestore nudi snažnu konzistenciju, što znači da pruža prednosti multi-regionalne podrške, a istovremeno šalje najnoviju verziju podataka kad god klijent zahtjeva.

Trenutno postoje dvije različite multi-regionalne lokacije koje se mogu koristiti za hostiranje podataka (jedan u Sjevernoj Americi i jedan u Europi). Cloud Firestore također nudi nekoliko regionalnih primjeraka na lokacijama širom svijeta. Suprotno tome, Realtime baza podataka nalazi se samo u Sjevernoj Americi (Kerpelman, 2017).



Slika 18. Dostupne lokacije Cloud Firestore-a

Izvor: https://firebase.googleblog.com/2017/10/cloud-firestore-for-rtdb-developers.html

4. **Različiti modeli cijena**

Dvije baze podataka imaju prilično različite modele cijena. Realtime baza podataka primarno određuje trošak na temelju količine preuzetih podataka kao i količine podataka pohranjenih u bazi podataka. Iako Cloud Firestore također naplaćuje te stvari, one su znatno niže od onoga u Realtime bazi podataka. Umjesto toga, cijene Cloud Firestore-a prvenstveno se temelje na broju zapisa, čitanja i brisanja koje naprave korisnici. Za neku tradicionalnu mobilnu aplikaciju u kojoj klijent povremeno zahtjeva veće količine podataka iz baze podataka, primjerice aplikacija za prikaz vijesti, model cijene Cloud Firestorea je mnogo povoljniji nego da se ista aplikacija vodi preko Realtime baze podataka.

S druge strane ako aplikacija stvara vrlo velik broj čitanja i pisanja u sekundi po klijentu (na primjer, aplikacija za grupni whiteboarding[10] , gdje je svačije ažuriranje crteža potrebno emitirati svima ostalima nekoliko puta u sekundi, Cloud Firestore bi vjerojatno bio skuplje rješenje (Kerpelman, 2017).

5.2.3. Zašto i dalje koristiti Realtime bazu podatka

S gore navedenim popisom promjena možda se čini kako je Cloud Firestore jednostavno bolji od baze podataka u stvarnom vremenu. I dok Cloud Firestore ima popriličan broj poboljšanja u odnosu na Realtime bazu podatka, još uvijek postoji nekoliko situacija u kojima bi trebalo razmotriti korištenje Realtime baze podataka. Neki od primjera:

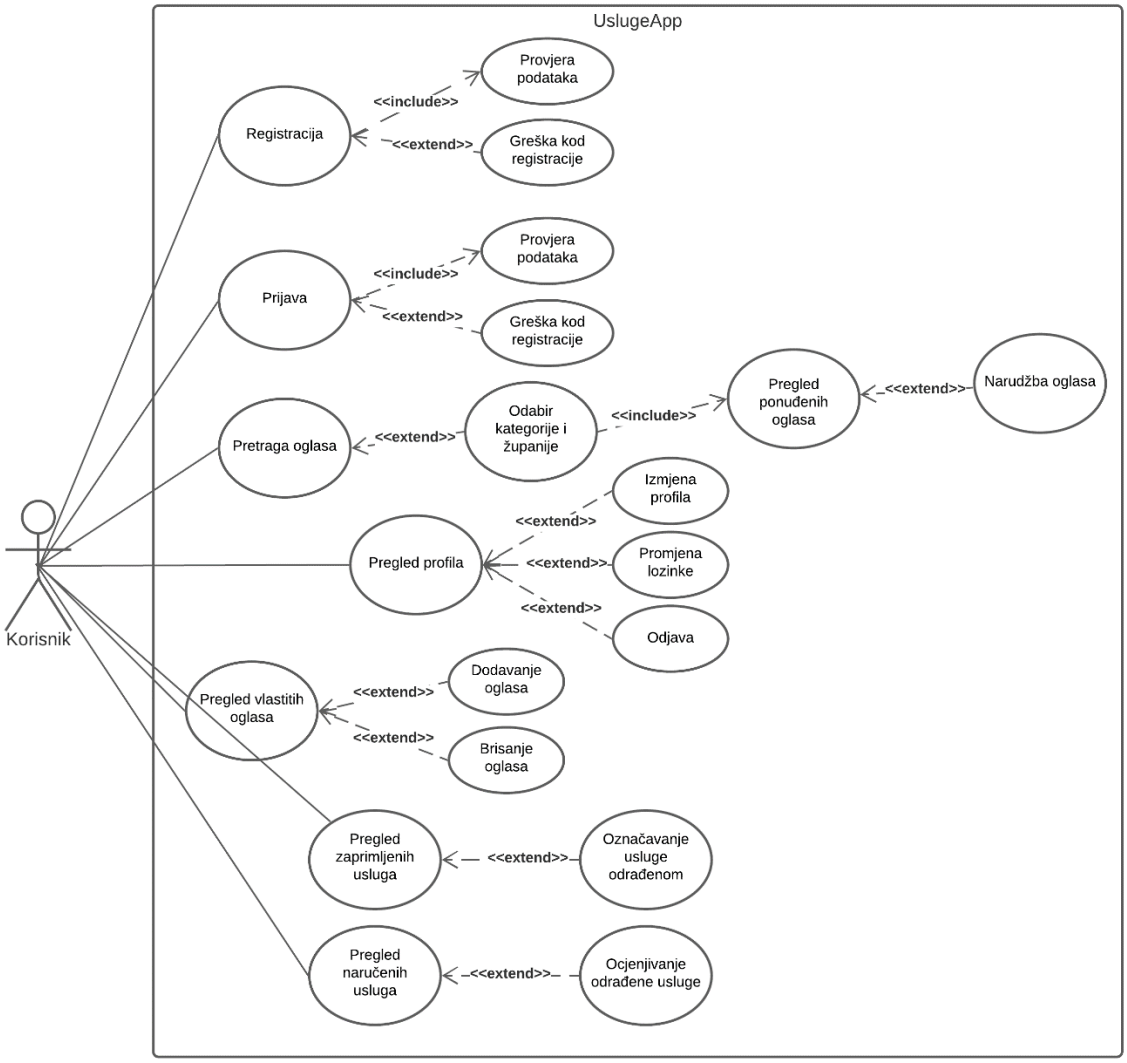
- Realtime baza podataka ima izvornu podršku za prisutnost, odnosno mogućnost prepoznavanja kada je korisnik online ili kada je izvan mreže.
- Realtime baza podataka ima malo bolje inicijalno kašnjenje (engl. latency[11]). Obično to ne čini toliku razliku ali ako je potrebna aplikacija sa što manjim inicijalnim kašnjenjem, Realtime baza podataka je svakako bolji izbor.

[10] Whiteboarding – postavljanje zajedničkih datoteka na zaslonsko prijenosnu bilježnicu
[11] Inicijalno kašnjenje (engl. latency) – vrijeme potrebno da podaci odu sa jednog mjesta na drugo

- Model cijene Cloud Firestore znači da bi aplikacije koje izvode vrlo velik broj malih čitanja i pisanja u sekundi po klijentu mogle biti znatno skuplje od aplikacije koja se izvodi u Realtime bazi podataka.

6. Aplikacija UslugeApp

UslugeApp je aplikacija koja nudi korisnicima da pretražuju te naruče usluge drugih korisnika, odnosno da i sami postave svoje oglase za usluge. Aplikacija je razvijena u *Android studio* razvojnom okruženju korištenjem programskog jezika *Java* te *XML*. Kao baza podatka korišten je *Firebase Firestore*.



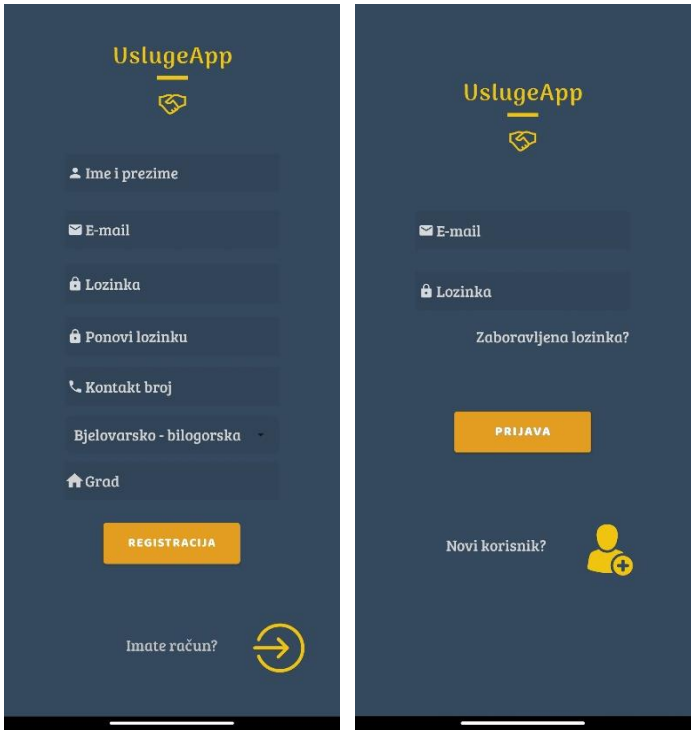
Slika 19. Use case dijagram

Korisnik ima nekoliko slučajeva korištenja a to su:

- Registracija
- Prijava
- Pregled profila (moguća izmjena profila, promjena lozinke i odjava)
- Pretraga oglasa (moguća narudžba)
- Pregled naručenih usluga (moguće ocjenjivanje usluge)
- Pregled zaprimljenih usluga (moguće označavanje usluge odrađenom)
- Pregled vlastitih usluga (moguće dodavanje novih i brisanje postojećih)

6.1. Registracija i prijava

Kod prvog pokretanja aplikacije, otvara se aktivnost za prijavu. Ukoliko je korisnik registriran od prije, dovoljno je da se prijavi sa e-mailom i lozinkom. U slučaju da nije, korisnik obavlja registraciju.



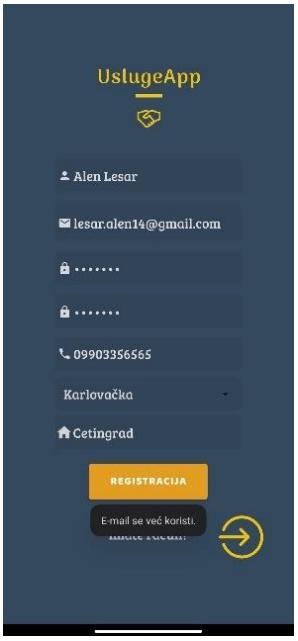
Slika 20. Mobilna aplikacija – registracija i prijava

Podatke koji su potrebni za registraciju su: *ime i prezime, e-mail, lozinka, kontakt broj, odabir županije* te *grad*. Ukoliko korisnik ne unese neki podatak ili ih pogrešno upiše, aplikacija će ga upozoriti na to (Slika 21).

```
if (TextUtils.isEmpty(fullName)){
    mFUllName.setError("Ime i prezime je obavezno.");
    return;
}
if (TextUtils.isEmpty(email)){
    mEmail.setError("E-mail je obavezan.");
    return;
}
if (TextUtils.isEmpty(password)){
    mPassword.setError("Lozinka je obavezna.");
    return;
}
if (password.length() < 6) {
    mPassword.setError("Lozinka mora imati minimalno 6 znakova.");
    return;
}
if (!password.equals(passwordRepeat)){
    mPasswordRepeat.setError("Lozinke se ne podudaraju.");
    return;
}
if (TextUtils.isEmpty(phone)){
    mPhone.setError("Kontakt broj je obavezan.");
    return;
}
if (TextUtils.isEmpty(city)){
    mCity.setError("Grad je obavezan.");
    return;
}
```

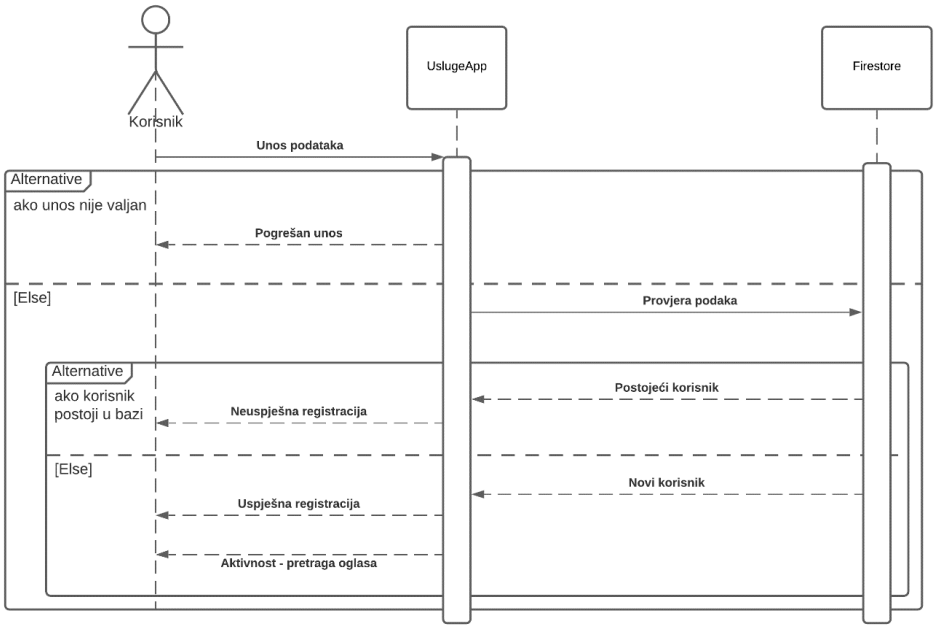
Slika 21. Programski kod i mobilna aplikacija - provjera podataka

Ukoliko su svi podaci ispravno uneseni, aplikacija šalje podatke do baze podataka tj. Cloud Firestorea koji provjerava da li već postoji korisnik sa istim e-mailom. Ako ne postoji, korisnik se uspješno registrira i automatski prijavljuje u aplikaciju te mu se otvara aktivnost „SearchAds“ u kojoj može početi pretraživati oglase o uslugama. Ukoliko korisnik sa istim e-mailom već postoji u bazi, aplikacija će o tome obavijestiti korisnika (Slika 22).



Slika 22. Mobilna aplikacija – obavijest o korištenom e-mailu

Na sljedećoj slici je prikazan UML sequence dijagram registracije unutar aplikacije (Slika 23).



Slika 23. UML sequence dijagram – registracija

Na dolje navedenoj slici je prikazan programski kod registracije koji se izvršava ukoliko je korisnik ispravno unio sve podatke (Slika 24). Podaci korisnika se unose preko Firebase funkcije *createUserWithEmailAndPassword*. Unutar funkcije se kreira DocumentReference[12] i HashMap[13] pomoću kojih će se spremiti svi korisnikovi podaci u bazu podataka.

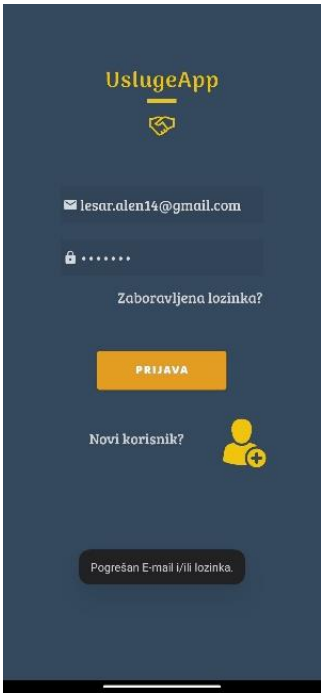
```
fAuth.createUserWithEmailAndPassword(email,password).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
    @Override
    public void onComplete(@NonNull Task<AuthResult> task) {
        if (task.isSuccessful()){
            userID = fAuth.getCurrentUser().getUid();
            DocumentReference documentReference = fStore.collection( collectionPath: "users").document(userID);
            Map <String, Object> user = new HashMap<>();
            user.put("fname", fullName);
            user.put("email", email);
            user.put("phone", phone);
            user.put("county", county);
            user.put("city", city);
            documentReference.set(user).addOnSuccessListener((OnSuccessListener) (aVoid) → {
                    Log.d(TAG,  msg: "onSuccess: user Profile is created for" + userID);
            }).addOnFailureListener((e) → {
                    Log.d(TAG,  msg: "onFailure: " + e.toString());
            });
            Toast.makeText( context: Register.this,  text: "Uspješna registracija.", Toast.LENGTH_SHORT).show();
            startActivity(new Intent(getApplicationContext(), SearchAds.class).setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK));
        }else {
            Toast.makeText( context: Register.this,  text: "E-mail se već koristi.", Toast.LENGTH_LONG).show();
            progressBar.setVisibility(View.GONE);
        }
    }
});
```

Slika 24. Programski kod – registracija

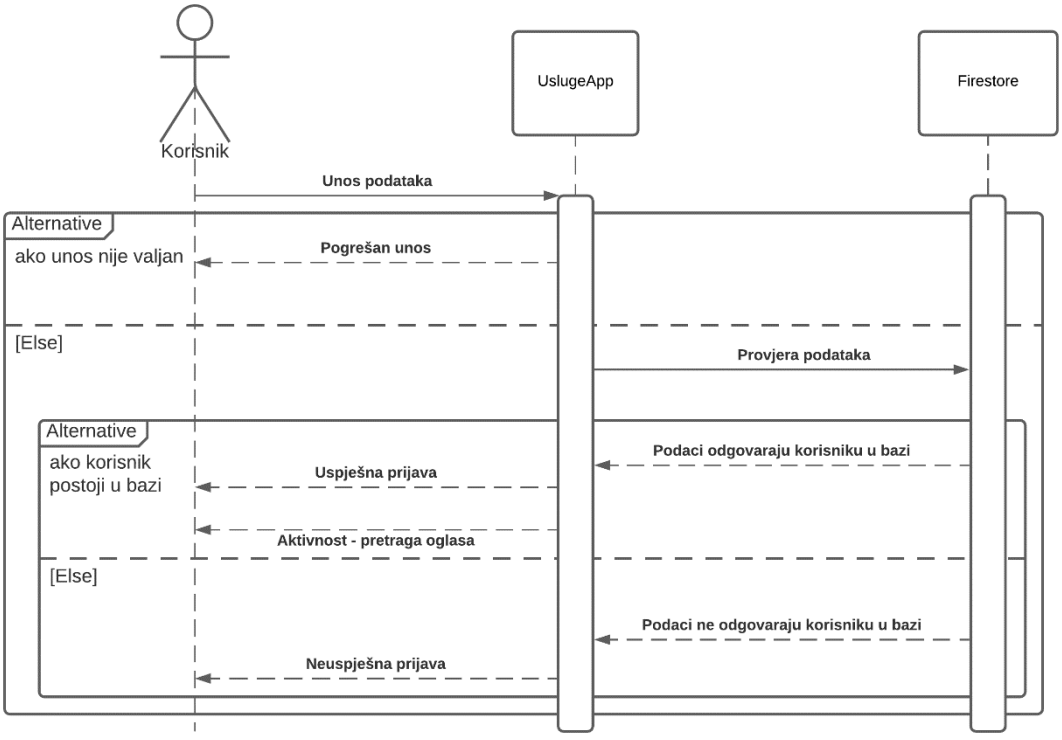
Za prijavu u aplikaciju korisnik navodi *e-mail* te *lozinku*. Prilikom prijave aplikacija provjerava unesene podatke te obavještava korisnika ukoliko je neki podatak krivo unesen. Ako su svi podaci ispravno uneseni, aplikacija šalje podatke do baze podataka koja provjerava postoji li korisnik sa navedenim e-mailom i odgovara li lozinka za navedeni e-mail, te ako ne postoji, aplikacija obavještava korisnika kako su navedeni e-mail i/ili lozinka pogrešno uneseni (Slika 25). Ukoliko su uneseni ispravni podaci, korisnik se uspješno prijavljuje te mu se otvara početna aktivnost „SearchAds“ gdje može početi pretraživati usluge. Na dolje navedenoj slici (Slika 26) prikazan je UML sequence dijagram prijave.

[12] DocumentReference – odnosi se na lokaciju dokumenta u Firestore bazi podataka i može se koristiti za čitanje, pisanje ili slušanje lokacije

[13] HashMap – kolekcijska klasa koja se koristi za pohranu parova ključ i vrijednost.



Slika 25. Mobilna aplikacija - pogrešan E-mail/lozinka

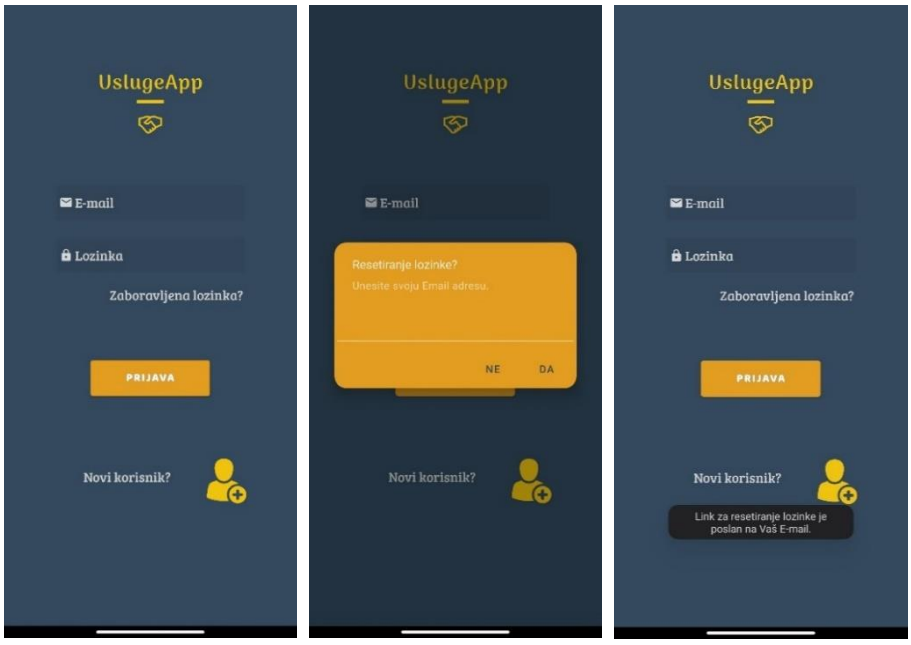


Slika 26. UML sequence dijagram – prijava

```
String email = mEmail.getText().toString().trim();
String password = mPassword.getText().toString().trim();

if (TextUtils.isEmpty(email)) {
    mEmail.setError("E-mail je obavezan.");
    return;
}

if (TextUtils.isEmpty(password)) {
    mPassword.setError("Lozinka je obavezna.");
    return;
}

if (password.length() < 6) {
    mPassword.setError("Lozinka ima minimalno 6 znakova.");
    return;
}

progressBar.setVisibility(View.VISIBLE);

fAuth.signInWithEmailAndPassword(email, password).addOnCompleteListener((task) → {
        if (task.isSuccessful()) {
            Toast.makeText( context: Login.this,  text: "Uspješna prijava.", Toast.LENGTH_SHORT).show();
            startActivity(new Intent(getApplicationContext(), SearchAds.class).setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK));

        } else {
            Toast.makeText( context: Login.this,  text: "Pogrešan E-mail i/ili lozinka.", Toast.LENGTH_SHORT).show();
            progressBar.setVisibility(View.GONE);
        }
});
```

Slika 27. Programski kod - prijava

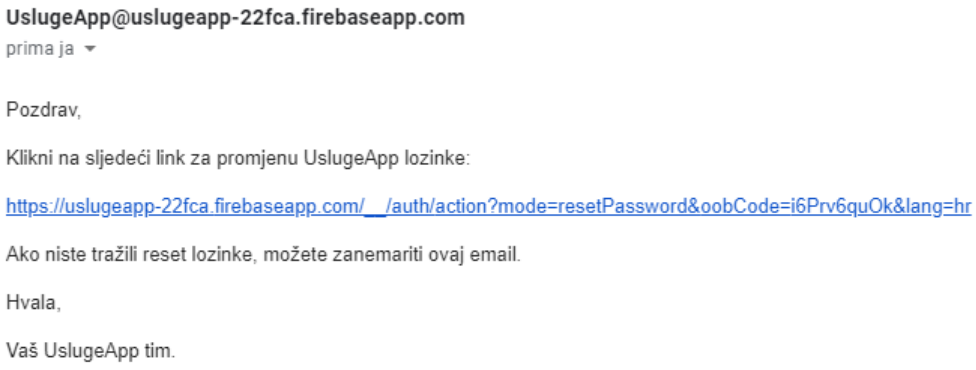
Ukoliko je korisnik zaboravio svoju lozinku, postoji mogućnost da ju promjeni klikom na *TextView*[14] „Zaboravljena lozinka?" kojim se otvara *dialog box*[15] (Slika 28).



Slika 28. Mobilna aplikacija - resetiranje lozinke

[14] TextView - element korisničkog sučelja koji prikazuje tekst korisniku.
[15] Dialog box - mali prozor koji od korisnika traži donošenje odluke ili unošenje dodatnih podataka.

Korisnik unosi e-mail adresu te ju aplikacija prosljeđuje do Cloud Firestorea. Ukoliko postoji korisnik sa navedenim mailom u bazi podataka, aplikacija obavještava korisnika da mu je na e-mail poslan link putem kojeg je moguće promijeniti lozinku (Slika 28).



Slika 29. E-mail za resetiranje lozinke

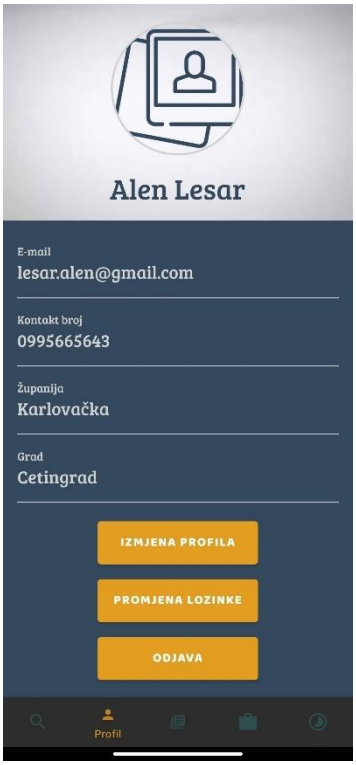
U slučaju da korisnik nije unio dobar e-mail, aplikacija ga obavještava kako zapis o navedenom e-mailu ne postoji.

```
resetPassword.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        final EditText resetMail = new EditText(v.getContext());
        MaterialAlertDialogBuilder resetPassword = new MaterialAlertDialogBuilder( context: Login.this);
        resetPassword.setTitle("Resetiranje lozinke?");
        resetPassword.setMessage("Unesite svoju Email adresu.");
        resetPassword.setBackground(getResources().getDrawable(R.drawable.alert_dialog_bg));
        resetPassword.setView(resetMail);

        resetPassword.setPositiveButton( text: "Da", (dialog, which) -> {
                String mail = resetMail.getText().toString();
                if (mail.length() < 1) {
                    Toast.makeText( context: Login.this,  text: "Unesi svoj E-mail.", Toast.LENGTH_SHORT).show();
                } else {
                    fAuth.sendPasswordResetEmail(mail).addOnSuccessListener((OnSuccessListener) (aVoid) -> {
                            Toast.makeText( context: Login.this,  text: "Link za resetiranje lozinke je poslan na Vaš E-mail.", Toast.LENGTH_SHORT).show();
                    }).addOnFailureListener((e) -> {
                            Toast.makeText( context: Login.this,  text: "Greška! " + e.getMessage(), Toast.LENGTH_SHORT).show();
                    });
                }
        });
        resetPassword.setNegativeButton( text: "Ne", (dialog, which) -> {
                // close
        });
        resetPassword.show();
    }
});
```

Slika 30. Programski kod - resetiranje lozinke

6.2. Pregled i uređivanje profila

Otvaranjem aktivnosti „Profil“ korisnik dobiva pregled svog profila, odnosno informacije koje je unio prilikom registracije (Slika 31).

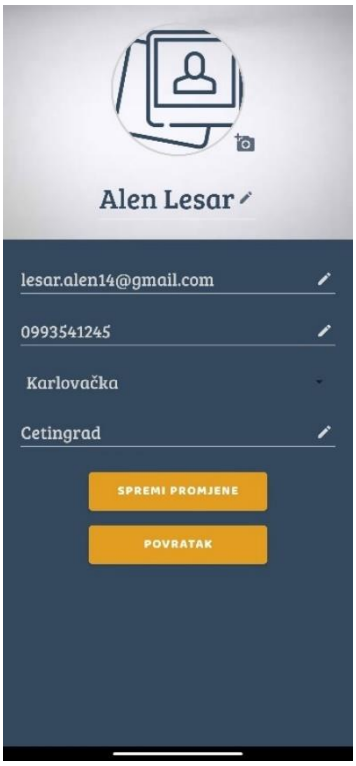


Slika 31. Mobilna aplikacija – profil

Korisnik ima mogućnosti izmijeniti profil, promijeniti lozinku ili se odjaviti. Pritiskom na gumb „Izmjena profila” otvara se aktivnost “ProfileEdit” (Slika 33). Podaci se preko *intenta* šalju iz aktivnosti „Profile“ do aktivnosti „ProfileEdit“ kako ne bi morali ponovno slati upit na bazu podataka i povlačiti podatke iz nje (Slika 32).

```
Intent i = new Intent(v.getContext(), ProfileEdit.class);
i.putExtra( name: "fullName", fullName.getText().toString());
i.putExtra( name: "email", email.getText().toString());
i.putExtra( name: "phone", phoneNum.getText().toString());
i.putExtra( name: "city", city.getText().toString());
i.putExtra( name: "county", countyId);

startActivity(i);
```

Slika 32. Programski kod – slanje podataka do druge aktivnosti



Slika 33. Mobilna aplikacija – izmjena profila

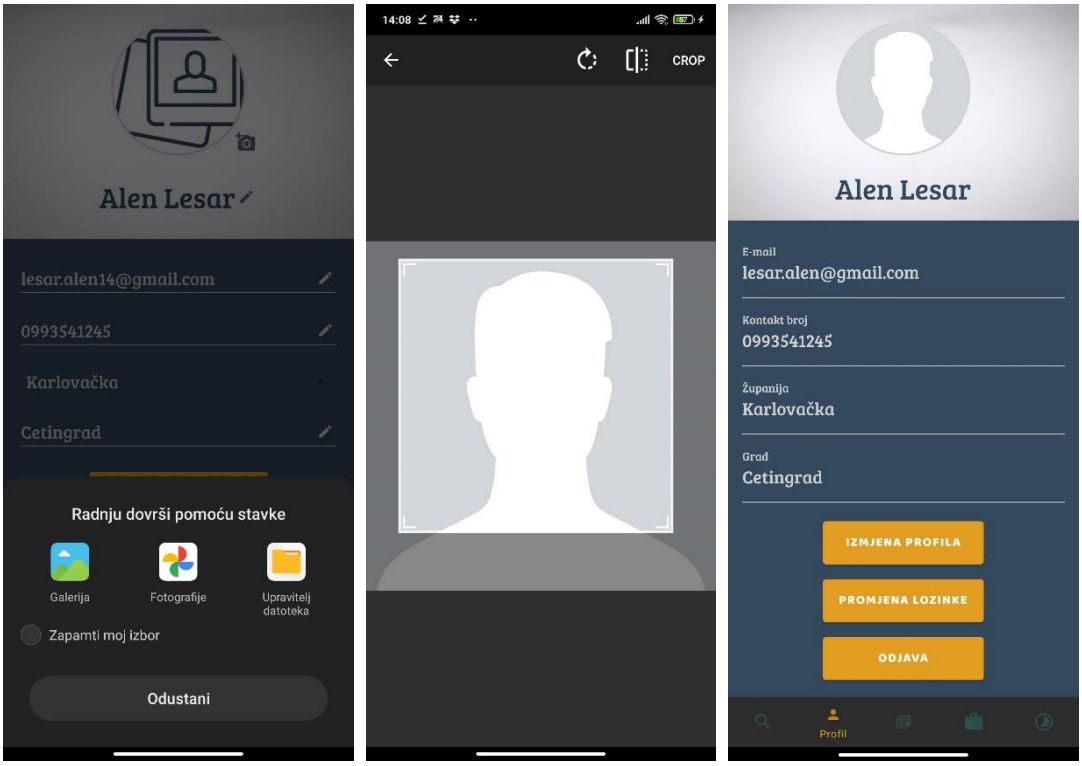
Dohvat podataka unutar aktivnosti „ProfileEdit“ također se vrši preko *intenta (Slika 34).*

```
Intent data = getIntent();
profileFullName.setText(data.getStringExtra( name: "fullName"));
profileEmail.setText(data.getStringExtra( name: "email"));
profilePhoneNum.setText(data.getStringExtra( name: "phone"));
profileCity.setText(data.getStringExtra( name: "city"));
profileCounty.setSelection(Integer.parseInt(data.getStringExtra( name: "county")));
```

Slika 34. Programski kod - dohvat podataka iz prethodne aktivnosti

Korisnik klikom na određeni *EditText*[16] ima mogućnost izmijeniti bilo koji podatak. Pritiskom na *ImageView* otvara se galerija na uređaju te korisnik ima mogućnost ažurirati profilnu sliku. Nakon što odabere sliku iz galerije, postoji mogućnost rezanja i rotiranja slike (Slika 35).

[16] EditText - Element korisničkog sučelja za unošenje i izmjenu teksta.



Slika 35: Mobilna aplikacija - izmjena profilne slike

Na slijedećoj slici je programski kod kojim se preko *intenta* otvara galerija na uređaju (Slika 36).

```
profileImage.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent openGalleryIntent = new Intent(Intent.ACTION_PICK, MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
        startActivityForResult(openGalleryIntent,  requestCode: 1000);
    }
});
```

Slika 36. Programski kod - otvaranje galerije

Nakon otvaranja galerije, korisnik odabire sliku i ta slika se učitava u aktivnost za rezanje kako bi ju korisnik mogao urediti kako želi.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == 1000 && resultCode == Activity.RESULT_OK ) {
        Uri imageUri = null;
        if (data != null) {
            imageUri = data.getData();
            CropImage.activity(imageUri)
                    .setAspectRatio( aspectRatioX: 1, aspectRatioY: 1)
                    .start( activity: this);
        }
    }
    if (requestCode == CropImage.CROP_IMAGE_ACTIVITY_REQUEST_CODE){
        CropImage.ActivityResult result = CropImage.getActivityResult(data);
        Uri imageUri = null;
        if (result != null) {
            imageUri = result.getUri();
            Picasso.get().load(imageUri).into(profileImage);
            imgUri = imageUri;
        }
    }
}
```

Slika 37. Programski kod - dohvaćanje i rezanje slike

```
private void uploadImageToFirebase(Uri imageUri) {
    final StorageReference fileRef = storageReference.child("users/" + Objects.requireNonNull(fAuth.getCurrentUser()).getUid() + "profile.jpeg");
    fileRef.putFile(imageUri).addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {
        @Override
        public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
            Log.d(TAG, msg: "Success");
        }
    }).addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            Toast.makeText( context: ProfileEdit.this, text: "Neuspješan prijenos slike." + e.getMessage(), Toast.LENGTH_SHORT).show();
        }
    });
}
```

Slika 38. Programski kod - spremanje slike u bazu podataka

Pritiskom na gumb „Spremi promjene“ u bazu podataka se unose sve promjene koje je korisnik unio (Slika 39). Promjene se unose preko Firebase funkcije *updateMail*. Unutar te funkcije se kreiraju se DocumentReference i HashMap pomoću kojih će se spremiti svi korisnikovi podaci u bazu podataka. Za pohranu slike profila u bazu podataka koristi se

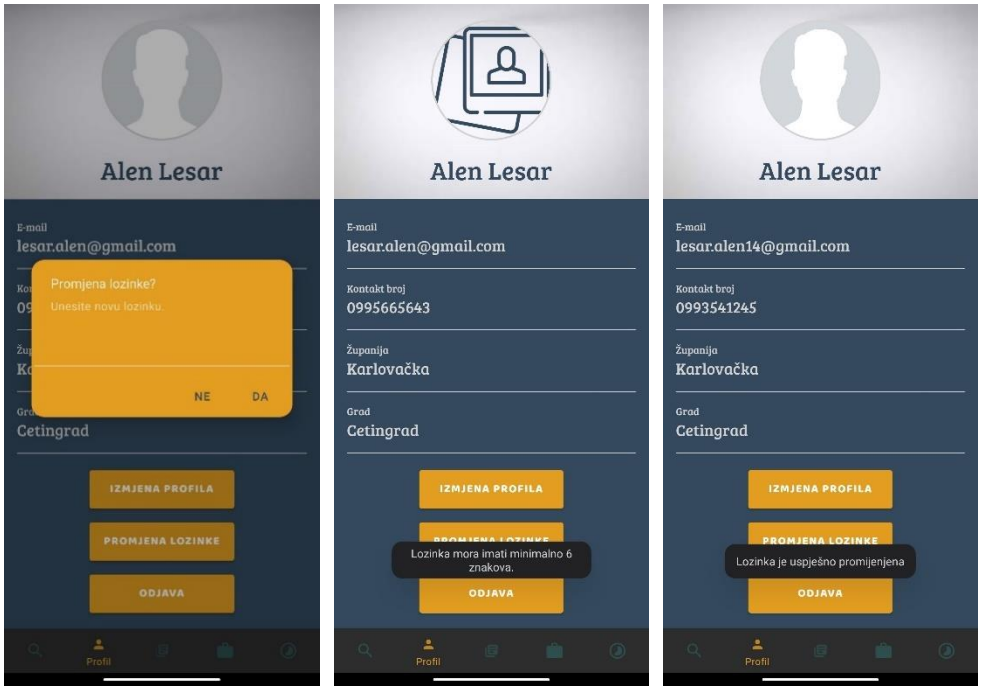
funkcija „uploadImageToFirebase“ (Slika 38) koja u sebi ima deklariran *StorageReference*[17] u kojem je definiran put gdje će se slika pohraniti.

```
saveProfileBtn.setOnClickListener((v) → {
        if (profileFullName.getText().toString().isEmpty() || profileEmail.getText().toString().isEmpty() ||
                profilePhoneNum.getText().toString().isEmpty()){
            Toast.makeText( context: ProfileEdit.this,  text: "Jedno ili više polja su prazna.", Toast.LENGTH_SHORT).show();
            return;
        }
        final String email = profileEmail.getText().toString();
        user.updateEmail(email).addOnSuccessListener((OnSuccessListener) (aVoid) → {
                DocumentReference documentReference = fStore.collection( collectionPath: "users").document(user.getUid());
                Map<String, Object> edited = new HashMap<>();
                edited.put("email", email);
                edited.put("fname", profileFullName.getText().toString());
                edited.put("phone", profilePhoneNum.getText().toString());
                edited.put("city", profileCity.getText().toString());
                edited.put("county", String.valueOf(profileCounty.getSelectedItemId()));
                documentReference.update(edited).addOnSuccessListener((OnSuccessListener) (aVoid) → {
                        if (imgUri!=null){
                        uploadImageToFirebase (imgUri);
                        }
                        Toast.makeText( context: ProfileEdit.this,  text: "Profil ažuriran.", Toast.LENGTH_SHORT).show();
                        startActivity(new Intent(getApplicationContext(), SearchAds.class));
                        finish();
                }).addOnFailureListener((e) → {
                        Toast.makeText( context: ProfileEdit.this,  text: "Profil nije ažuriran. " + e.getMessage(), Toast.LENGTH_SHORT).show();
                });
        }).addOnFailureListener((e) → {
                Toast.makeText( context: ProfileEdit.this, e.getMessage(), Toast.LENGTH_SHORT).show();
        });
});
```

Slika 39. Programski kod - spremanje promjena na profilu

Ukoliko korisnik želi promijeniti lozinku, ima mogućnost kliknuti na gumb „Promjena lozinke“. Pritiskom na gumb se otvara *dialog box* gdje ga se traži da unese novu lozinku. Ukoliko korisnik unese lozinku koja je kraća od 6 znakova, aplikacije će ga upozoriti na to, te isto tako ukoliko pravilno unese lozinku, aplikacija će ga obavijestiti kako je lozinka uspješno promijenjena (Slika 40).

[17] StorageReference - Predstavlja referencu na Google Cloud Storage objekt. Omogućava učitavanje i preuzimanje objekata, uzimanje / postavljanje meta podataka objekta i brisanje objekta na navedenom putu.



Slika 40. Mobilna aplikacija - promjena lozinke

```
resetPasswordLocal.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        final EditText newPass = new EditText(v.getContext());
        newPass.setInputType(InputType.TYPE_CLASS_TEXT | InputType.TYPE_TEXT_VARIATION_PASSWORD);
        MaterialAlertDialogBuilder resetPassword = new MaterialAlertDialogBuilder( context: Profile.this);
        resetPassword.setTitle("Resetiranje lozinke?");
        resetPassword.setMessage("Unesite novu lozinku.");
        resetPassword.setBackground(getResources().getDrawable(R.drawable.alert_dialog_bg));
        resetPassword.setView(newPass);

        resetPassword.setPositiveButton( text: "Da", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                String newPassword = newPass.getText().toString();
                if (newPassword.length() < 6) {
                    Toast.makeText( context: Profile.this, text: "Lozinka mora imati minimalno 6 znakova.", Toast.LENGTH_SHORT).show();
                }
                else {
                    user.updatePassword(newPassword).addOnSuccessListener(new OnSuccessListener<Void>() {
                        @Override
                        public void onSuccess(Void aVoid) {
                            Toast.makeText( context: Profile.this, text: "Lozinka je uspješno promijenjena", Toast.LENGTH_SHORT).show();
                        }
                    }).addOnFailureListener(new OnFailureListener() {
                        @Override
                        public void onFailure(@NonNull Exception e) {
                            Toast.makeText( context: Profile.this, text: "Neuspješna promjena lozinke" + e.getMessage(), Toast.LENGTH_SHORT).show();
                        }
                    });
                }
            }
        });
```

Slika 41. Programski kod - promjena lozinke

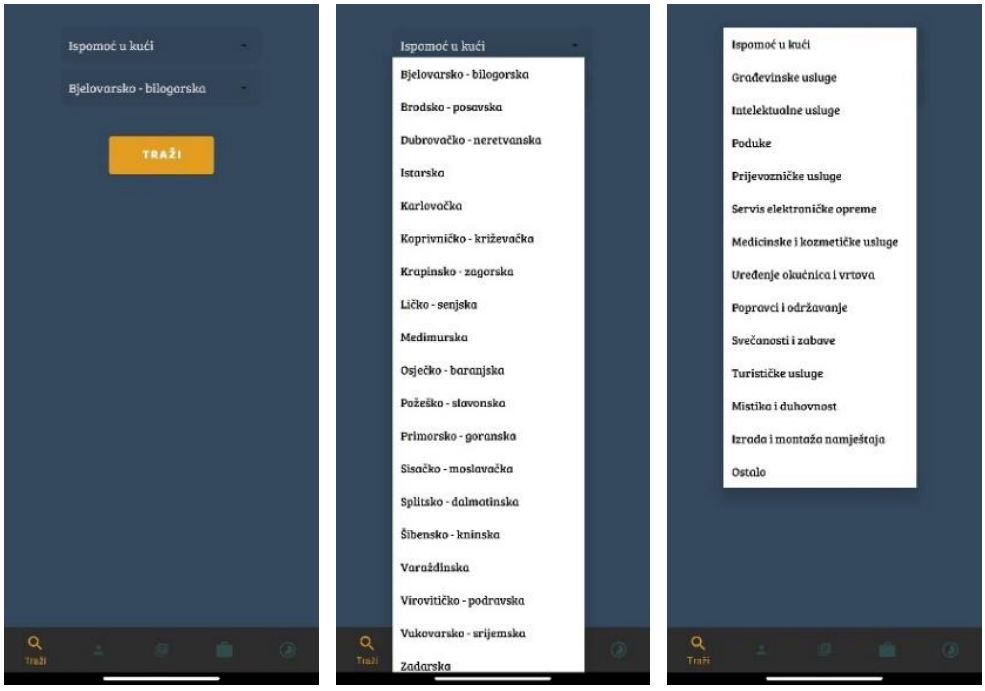
Posljednja mogućnost koju korisnik ima unutar aktivnosti „Profile“ je odjava. Klikom na gumb „Odjava“ korisnik se odjavljuje iz aplikacije te mu se otvara aktivnost za prijavu „Login“.

```
public void logout(View view) {
    FirebaseAuth.getInstance().signOut();
    Intent intent = new Intent( packageContext: this, Login.class);
    intent.putExtra( name: "finish",  value: true); // if you are checking for this in your other Activities
    intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |
            Intent.FLAG_ACTIVITY_CLEAR_TASK |
            Intent.FLAG_ACTIVITY_NEW_TASK);
    startActivity(intent);

}
```

Slika 42. Programski kod – odjava

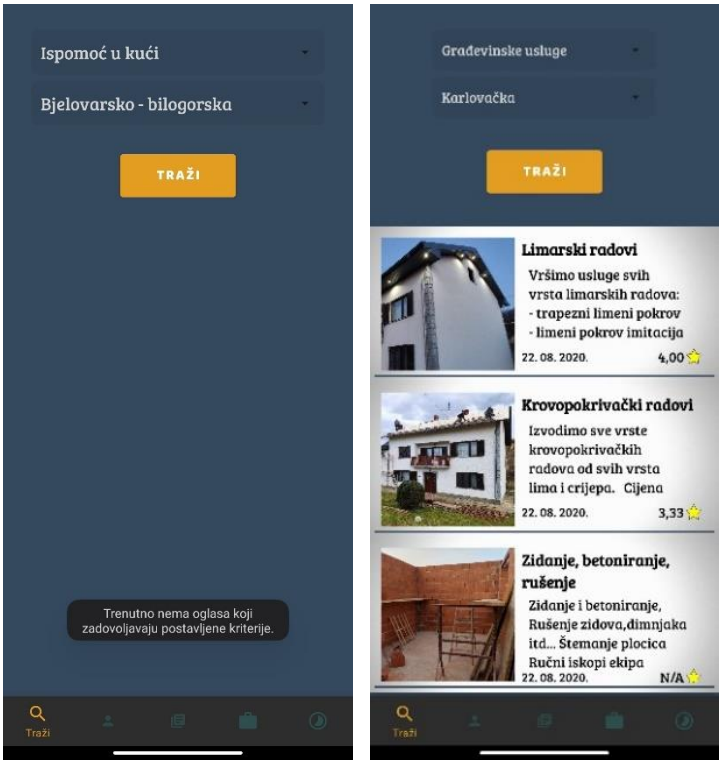
6.3. Pretraživanje i narudžba usluge

Kod pretrage oglasa korisnik odabire *kategoriju* usluge te ***županiju*** kao kriterije pretrage (Slika 43).



Slika 43. Mobilna aplikacija - pretraga usluga

Ukoliko nema oglasa o uslugama prema zadanim parametrima, aplikacija će o tome obavijestiti korisnika. Ukoliko ima, ispisati će se svi dostupni oglasi sa informacijama kao što su *naziv oglasa, slika oglasa, kratki opis, datum dodavanja* i *prosječna ocjena*. (Slika 44).



Slika 44. Mobilna aplikacija – pregled dostupnih usluga

Na sljedećoj slici (Slika 45) je prikazan programski kod koji se koristi za pretragu usluga prema zadanim kriterijima. Dohvaćaju se vrijednosti *ID kategorije* i *županije* te se pomoću njih kreira upit *Query*.

```
String categoryID = String.valueOf(adCategorySearch.getSelectedItemId());
String county = String.valueOf(countySearch.getSelectedItemId());

Query query = firebaseFirestore.collection( collectionPath: "adCategory").document(categoryID).
        collection( collectionPath: "ads").whereEqualTo( field: "adCounty", county);
```

Slika 45. Programski kod – pretraga usluga

Za pregled dostupnih usluga koristi se *RecyclerView*[18] sa paginacijom (engl. pagination) (Slika 46). Prvo se kreira konfiguracija za paginaciju sa *PageList.Config* gdje se konstruira i definira prilagođeno ponašanje učitavanja. Prva opcija je postavljanje *setInitialLoadSizeHint* u kojoj se definira koliko stavki treba učitati kod prvog učitavanja. Druga opcija je *setPageSize* koja definira broj učitanih stavki nakon što su učitane prve stavke sa opcijom *setInitialLoadSizeHint.* Primjerice ako je slučaj kao u aplikaciji UslugeApp da je *setInitialLoadSizeHint(10)* tada će se prilikom prvog učitavanja učitati prvih 10 stavki, te ako je *setPageSize(3),* kad korisnik dođe do stavke 10, učitat će se stavke 11, 12 i 13.

Nakon toga kreira se *FirestorePagingOptions*. Prva opcija je *setLifecycleOwner(SearchAds.this)* koja omogućava da se ne moraju ručno koristiti *start* i *stop* funkcije za slušanje promjena već će to automatski obaviti *FirebaseUI*[19]. Drugoj opciji *setQuery* se predaju kreirani upit *query,* konfiguracija *config* te klasa *AdModel.class* koja sadrži *get* metode pomoću kojih se dohvaćaju podaci iz baze podataka. Nadalje pomoću klase *AdsViewHolder* kreira se lista usluga tako što se svaka usluga postavi u novi *layout*[20] koji je definiran *list_item_single.xml* datotekom (Slika 47).

[18] RecyclerView - spremnik za pregled većeg skupa podataka, zamjena za ListView, GridView.
[19] FirebaseUI – biblioteka izgrađena nad Firebase Authentication SDK-u.
[20] Layout – definira struktura korisničkog sučelja unutar aplikacije

```
PagedList.Config config = new PagedList.Config.Builder()
        .setInitialLoadSizeHint(10)
        .setPageSize(3)
        .build();

FirestorePagingOptions<AdModel> options = new FirestorePagingOptions.Builder<AdModel>()
        .setLifecycleOwner(SearchAds.this)
        .setQuery(query, config, AdModel.class)
        .build();

adapter = new FirestorePagingAdapter<AdModel, AdsViewHolder>(options) {
    @NonNull
    @Override
    public AdsViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.list_item_single, parent, attachToRoot: false);
        return new AdsViewHolder(view);
    }

    @Override
    protected void onBindViewHolder(@NonNull AdsViewHolder holder, int position, @NonNull AdModel model) {
        if (user.equals(model.getUserID())) {
            holder.itemView.setVisibility(View.GONE);
            holder.itemView.setLayoutParams(new RecyclerView.LayoutParams( width: 0,  height: 0));
        } else {
            holder.ad_name.setText(model.getAdName());
            holder.ad_desc.setText(model.getAdDesc());
            Double rating = model.getAdRating();
            holder.ad_rating_txt = String.format("%.2f", rating);
            if (rating.equals(0.0)) {...} else {...}
            Picasso.get().load(model.getAdImageUrl()).into(holder.ad_image);
            holder.ad_ID = model.getAdID();
            holder.ad_county = model.getAdCounty();
            holder.ad_category = model.getAdCategory();
            holder.ad_date.setText(model.getAdDate());
        }
    }
    @Override
    protected void onLoadingStateChanged(@NonNull LoadingState state) {...}
};
mFirestoreList.setHasFixedSize(true);
mFirestoreList.setLayoutManager(new LinearLayoutManager( context: SearchAds.this));
mFirestoreList.setAdapter(adapter);
```

Slika 46. Programski kod – pretraga i ispis usluga

Slika 47. Mobilna aplikacija - list_item_single.xml

Nakon što su se ispisali svi dostupne usluge, korisnik ima mogućnost otvoriti oglas te se usluge te vidjeti detaljnije informacije. Otvaranjem oglasa otvara se aktivnost „AdDetails“ u kojoj korisnik dobiva informacije o tome tko je objavio oglas, koja je lokacija oglasa, kontakt broj oglašivača te ocjenu usluge (Slika 48). Također korisnik ovdje ima mogućnost naručiti uslugu.

Slika 48. Mobilna aplikacija - detalji oglasa

Za pregled detalja usluge otvara se aktivnost “AdDetails”. Njoj se preko *intenta* šalju podaci iz “SearchAds“ aktivnosti kako ne bi morali ponovno slati upit na bazu podataka i povlačiti podatke iz nje. Slanje podataka preko *intenta* je prikazano na sljedećoj slici (Slika 49).

```
Intent i = new Intent(v.getContext(), AdDetails.class);
i.putExtra( name: "adID", ad_ID);
i.putExtra( name: "adName", ad_name_txt);
i.putExtra( name: "adDesc", ad_desc_text);
i.putExtra( name: "adImage", imageUrl);
i.putExtra( name: "adCounty", ad_county);
i.putExtra( name: "adCategory", ad_category);
i.putExtra( name: "adRating", ad_rating_txt);
i.putExtra( name: "userID", user_id);
startActivity(i);
```

Slika 49. Programski kod - slanje podataka preko intenta

Dohvat podataka poslanih iz „SearchAds“ aktivnosti unutar „AdDetails“ aktivnosti i postavljanje vrijednosti TextView-ova i ImageView-a prikazano je na sljedećoj slici (Slika 50).

```
Intent data1 = getIntent();
adName.setText(data1.getStringExtra( name: "adName"));
adDesc.setText(data.getStringExtra( name: "adDesc"));
Picasso.get().load(data1.getStringExtra( name: "adImage")).into(adImage);
adAdvertiserID = data1.getStringExtra( name: "userID");

String rating = data1.getStringExtra( name: "adRating");
if (rating.equals("0,00")) {
    adRatingTV.setText("N/A");
} else {
    adRatingTV.setText(rating);
}
```

Slika 50. Programski kod - dohvat podataka preko intenta

Sa *DocumentReferenceom* i *DocumentSnapshotom*[21], aplikacija dolazi do podataka o oglašivaču usluge koji se nalaze u kolekciji „users“ (Slika 51). Podaci koji se dohvaćaju su *ime i prezime oglašivača*, *lokacija* te *kontakt broj*.

[21] DocumentSnapshot – sadrži podatke pročitane iz dokumenta Firestore baze podataka

```
DocumentReference documentReference1 = fStore.collection( collectionPath: "users").document(String.valueOf(adAdvertiserID));
documentReference1.get().addOnCompleteListener(new OnCompleteListener<DocumentSnapshot>() {
    @Override
    public void onComplete(@NonNull Task<DocumentSnapshot> task) {
        if (task.isSuccessful()) {
            DocumentSnapshot snapshot = task.getResult();
            adAdvertiserName = snapshot.getString( field: "fname");
            adAdvertiser.setText(adAdvertiserName);

            adCityTxt = snapshot.getString( field: "city");
            adCity.setText(adCityTxt);

            advertiserPhone = snapshot.getString( field: "phone");
            PhoneTV.setText(advertiserPhone);
        }
    }
```

Slika 51. Programski kod – documentSnapshot dohvat podataka o oglašivaču usluge

Korisnik naručuje uslugu pritiskom na dugme „Naruči” (Slika 48). Kod narudžbe usluge, prvo se dohvaćaju podaci o trenutnom korisniku. Nakon toga, ti podaci se zajedno sa podacima o usluzi koji su ranije dohvaćeni, upisuju se u podkolekciju „AdsToDo” unutar *dokumenta* oglašivača usluge (Slika 52).

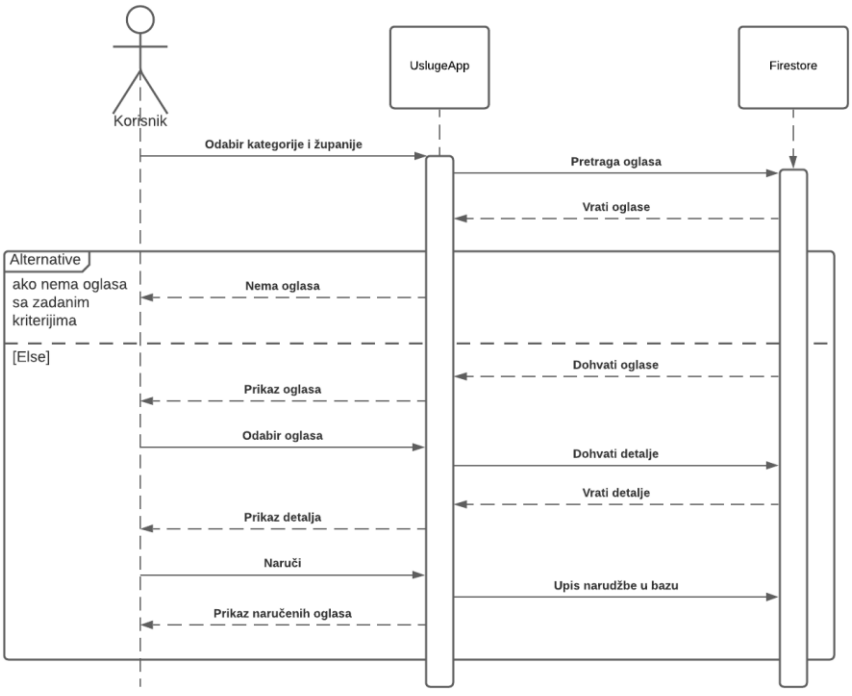
```
DocumentReference documentReference1 = fStore.collection( collectionPath: "users").document(currentUser);
documentReference1.get().addOnCompleteListener(new OnCompleteListener<DocumentSnapshot>() {
    @Override
    public void onComplete(@NonNull Task<DocumentSnapshot> task) {
        if(task.isSuccessful()){
            DocumentSnapshot snapshot = task.getResult();
            adClientName = snapshot.getString( field: "fname");
            adClientPhone = snapshot.getString( field: "phone");
            adClientCity = snapshot.getString( field: "city");

            DocumentReference documentReference = fStore.collection( collectionPath: "users").document(adAdvertiserID).
                    collection( collectionPath: "adsToDo").document( documentPath: adID+currentUser);

            Map<String, Object> ad = new HashMap<>();
            ad.put("adID", adID);
            ad.put("adName", adNameAdvertised);
            ad.put("adDesc", adDescAdvertised);
            ad.put("adImageUrl", adImageAdvertised);
            ad.put("adClient", currentUser);
            ad.put("adCategory", adCategory);
            ad.put("adToDoID", adID+currentUser);
            ad.put("clientName", adClientName);
            ad.put("clientPhone", adClientPhone);
            ad.put("clientCity", adClientCity);
            ad.put("adDate", adDate);

            documentReference.set(ad).addOnSuccessListener((OnSuccessListener) (aVoid) -> {
                    Log.d(TAG, msg: "Success");
            }).addOnFailureListener((e) -> {
                    Log.d(TAG, msg: "onFailure: " + e.toString());
            });
```

Slika 52. Programski kod - narudžba oglasa (1. dio)

Na isti način se podaci o naručitelju usluge zajedno sa ranije dohvaćenim podacima o usluzi upisuju u podkolekciju „OrderedAds" unutar *dokumenta* naručitelja usluge (Slika 53). Nakon što je korisnik naručio uslugu, otvara se aktivnost „OrderedAds" gdje korisnik može vidjeti sve usluge koje je naručio.

```
DocumentReference documentReference2 = fStore.collection( collectionPath: "users").document(currentUser)
        .collection( collectionPath: "orderedAds").document(adID);
Map<String, Object> ad1 = new HashMap<>();

ad1.put("adID", adID);
ad1.put("adName", adNameAdvertised);
ad1.put("adDesc", adDescAdvertised);
ad1.put("adImageUrl", adImageAdvertised);
ad1.put("adCity", adCityTxt);
ad1.put("adAdvertiserName", adAdvertiserName);
ad1.put("adAdvertiserID", adAdvertiserID);
ad1.put("adCategory", adCategory);
ad1.put("advertiserPhone", advertiserPhone);
ad1.put("adDate", adDate);
ad1.put("adRatingBoolean", false);

documentReference2.set(ad1).addOnSuccessListener((OnSuccessListener) (aVoid) → {
        Log.d(TAG,  msg: "Success");
        Toast.makeText( context: AdDetails.this,  text: "Uspješno naručeno.", Toast.LENGTH_SHORT).show();
}).addOnFailureListener((e) → {
        Log.d(TAG,  msg: "onFailure: " + e.toString());
});
```

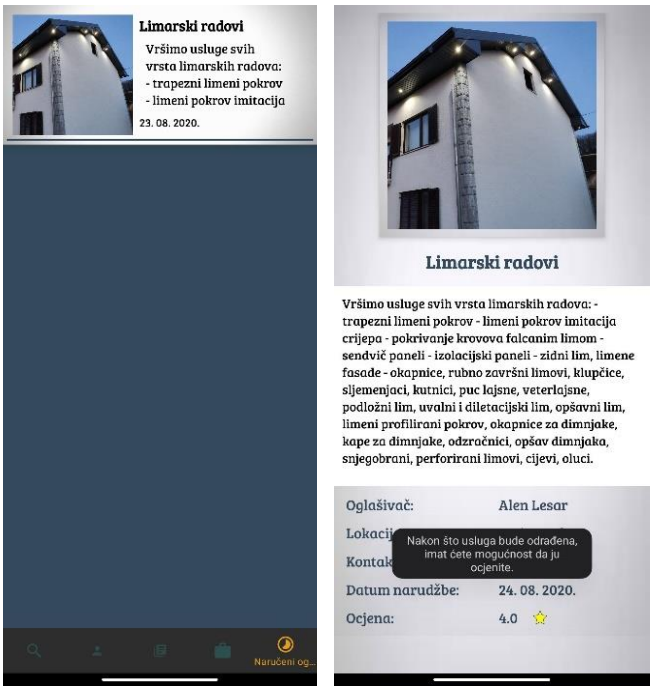
Slika 53. Programski kod - narudžba oglasa (2. dio)



Slika 54. UML sequence dijagram - narudžba usluge

6.4. Pregled naručenih usluga

Pokretanjem aktivnosti „AdsOrdered” korisnik dobiva uvid u sve usluge koje je naručio, a koje su ponudili ostali korisnici. Za prikaz naručenih oglasa se koristi *RecyclerView* (Slika 55). Programski kod za dohvaćanje svih naručenih usluga je veoma sličan onome za prikaz svih usluga koje su drugi korisnici naručili od korisnika.



Slika 55. Mobilna aplikacija – usluge koje je korisnik naručio

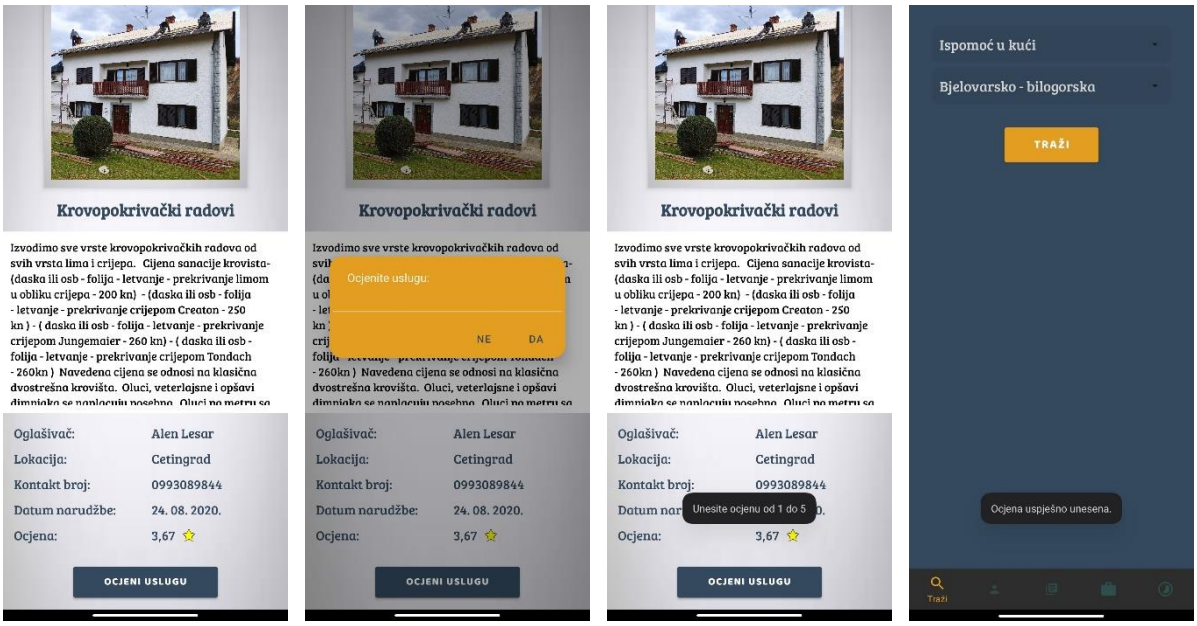
Razlika je što se podaci povlače iz podkolekcije korisnika „orderedAds“ (Slika 56).

```
query = firebaseFirestore.collection( collectionPath: "users").document(user).collection( collectionPath: "orderedAds");
```

Slika 56. Programski kod – upit (naručene usluge)

Otvaranjem naručene usluge, pokreće se aktivnost „AdDetails“ gdje korisnik ima pristup o informacijama kao što su *ime i prezime oglašivača*, *lokacija* ,*kontakt broj*, *datum narudžbe* te *trenutnu ocjenu*. Ti podaci šalju preko *intenta* iz prethodne aktivnosti. Također korisnik dobiva obavijest da će imati mogućnost ocijeniti uslugu nakon što usluga bude označena kao odrađena od strane oglašivača (Slika 55).

Nakon što određena usluga bude označena kao odrađena od strane oglašivača, korisnik ima mogućnost ocijeniti oglas. Klikom na gumb „Ocjeni uslugu“ otvara se *dialog box*. Usluge se ocjenjuju sa ocjenama od 1 do 5 te ukoliko korisnik upiše ocjenu van tog raspona, aplikacija će ga upozoriti na to. Nakon ocjenjivanja usluge, korisnik dobiva obavijest kako je ocjena uspješno unesena te mu se ponovno otvara aktivnost za pretraživanje usluga (Slika 57).



Slika 57. Mobilna aplikacija - ocjenjivanje oglasa

Kod ocjenjivanja prvo se kreira *DocumentReference* do oglasa koji se ocjenjuje. Nakon toga se kreira *DocumentSnapshot* pomoću kojega provjeravamo da li oglas još uvijek postoji ili ga je oglašivač možda obrisao. Ukoliko oglas još uvijek postoji, izvršava se sljedeći kod (Slika 58). Prvo se kreira *dialog box* u koji korisnik unosi ocjenu. Unesena ocjena mora biti između 1 i 5, ako nije aplikacija će na to upozoriti korisnika. Ukoliko je unesena ocjena između 1 i 5, kreira se novi *DocumentReference* te *HashMap* pomoću kojih će se upisati korisnikova ocjena. Nova ocjena se izračunava se na sljedeći način:

$$\frac{\text{posljednja ocjena} \ * \ \text{ukupan broj ocjena} \ + \ \text{korisnikova ocjena}}{ukupan\ broj\ ocjena \ + \ 1}$$

Nova izračunata ocjena se unosi preko *DocumentReference update* funkcije. Nakon toga kreira se još jedan DocumentReference s kojim se pomoću *delete* funkcije briše oglas iz korisnikove podkolekcije „OrderedAds“.

```
adRatingBtn.setOnClickListener((v) → {
        DocumentReference documentReference = fStore.collection( collectionPath: "adCategory").
                document(adCategoryOrdered).collection( collectionPath: "ads").document(adID);
        documentReference.addSnapshotListener((documentSnapshot, e) → {
                if (documentSnapshot != null && documentSnapshot.exists()) {
                    final EditText adRatingUser = new EditText(v.getContext());
                    adRatingUser.setInputType(InputType.TYPE_CLASS_NUMBER);
                    MaterialAlertDialogBuilder rating = new MaterialAlertDialogBuilder( context: AdDetails.this);
                    rating.setTitle("Ocjenite uslugu: ");
                    rating.setView(adRatingUser);
                    rating.setBackground(getResources().getDrawable(R.drawable.alert_dialog_bg));
                    rating.setPositiveButton( text: "Da", (dialog, which) → {
                            String ratingString = adRatingUser.getText().toString();
                            if (ratingString.length() < 1) {...} else {
                                final Double rating = Double.parseDouble(ratingString);
                                if (rating < 1 || rating > 5) {...} else {
                                    DocumentReference documentReference1 = fStore.collection( collectionPath: "adCategory").
                                            document(adCategoryOrdered).collection( collectionPath: "ads").document(adID);
                                    Map<String, Object> ad = new HashMap<>();
                                    Double sum = adRating * numOfRatings + rating;
                                    Double newRating = sum / (numOfRatings + 1);
                                    ad.put("adRating", newRating);
                                    ad.put("numOfRatings", numOfRatings + 1);
                                    documentReference1.update(ad).addOnSuccessListener((OnSuccessListener) (aVoid) → {
                                            Log.d(TAG, msg: "Success");
                                            Toast.makeText( context: AdDetails.this, text: "Ocjena uspješno unesena.", Toast.LENGTH_SHORT).show();
                                    }).addOnFailureListener((e) → {
                                            Log.d(TAG, msg: "onFailure: " + e.toString());
                                            Toast.makeText( context: AdDetails.this, text: "Neuspješno ocijenjivanje", Toast.LENGTH_SHORT).show();
                                    });
                                    DocumentReference documentReference2 = fStore.collection( collectionPath: "users").document(currentUser).
                                            collection( collectionPath: "orderedAds").document(adID);
                                    documentReference2.delete().addOnSuccessListener((OnSuccessListener) (aVoid) → {
                                            Intent myIntent = new Intent( packageContext: AdDetails.this, SearchAds.class);
                                            AdDetails.this.startActivity(myIntent);
                                            finish();
                                    }).addOnFailureListener((e) → {
                                            Toast.makeText( context: AdDetails.this, text: "Error: " + e.getMessage(), Toast.LENGTH_SHORT).show();
                                    });
                                }
                            }
                    });
                    rating.setNegativeButton( text: "Ne", new DialogInterface.OnClickListener() {...});
                    rating.show();
                }
                else {...}
```

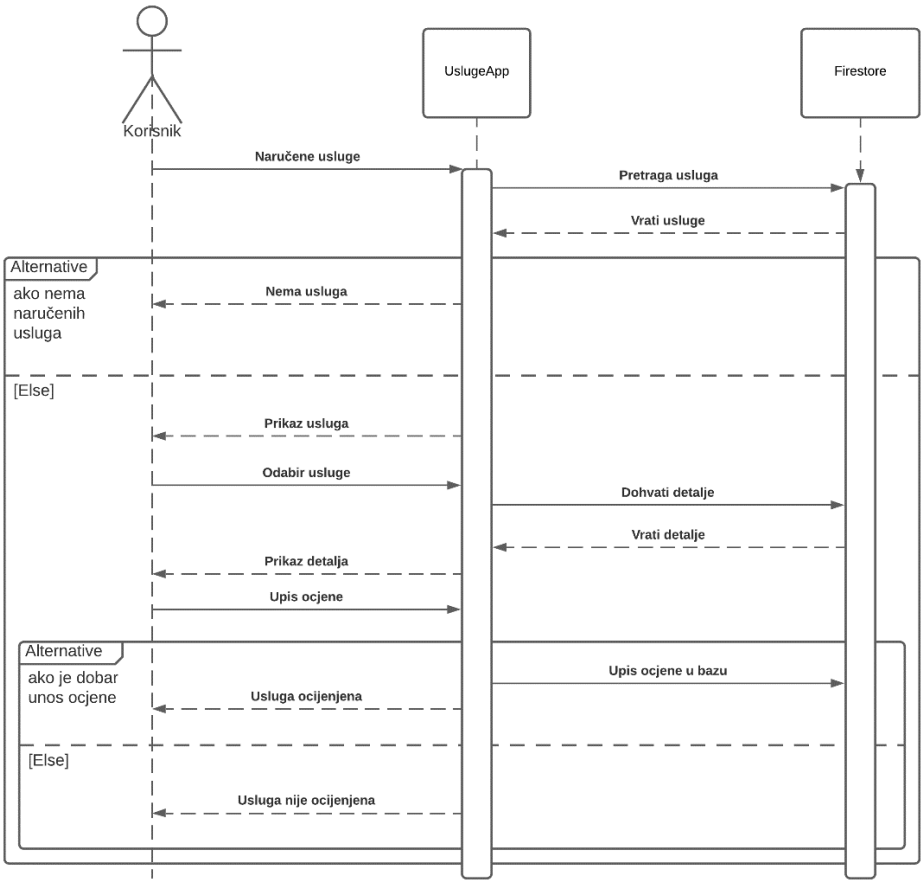
Slika 58. Programski kod – ocjenjivanje usluge (1. dio)

Ukoliko oglas više ne postoji u bazi podataka, izvršava se drugi dio koda. Kreira se novi DocumentReference s kojim se pomoću *delete* funkcije briše oglas iz korisnikove podkolekcije „OrderedAds“ te korisnik dobiva obavijest kako je oglašivač u međuvremenu izbrisao oglas (Slika 59).

```
DocumentReference documentReference2 = fStore.collection( collectionPath: "users").
        document(currentUser).collection( collectionPath: "orderedAds").document(adID);
documentReference2.delete().addOnSuccessListener(new OnSuccessListener<Void>() {
    @Override
    public void onSuccess(Void aVoid) {
        Intent myIntent = new Intent( packageContext: AdDetails.this, SearchAds.class);
        AdDetails.this.startActivity(myIntent);
        finish();
    }
}).addOnFailureListener(new OnFailureListener() {
    @Override
    public void onFailure(@NonNull Exception e) {
        Toast.makeText( context: AdDetails.this,  text: "Error: " + e.getMessage(), Toast.LENGTH_SHORT).show();
    }
});
Toast.makeText( context: AdDetails.this,  text: "Oglas je u međuvremenu obrisan.", Toast.LENGTH_SHORT).show();
```

Slika 59. Programski kod – ocjenjivanje usluge (2. dio)

Proces ocjenjivanja oglasa prikazan je UML sequence dijagramom (Slika 60).



Slika 60. UML sequence dijagram - ocjenjivanje oglasa

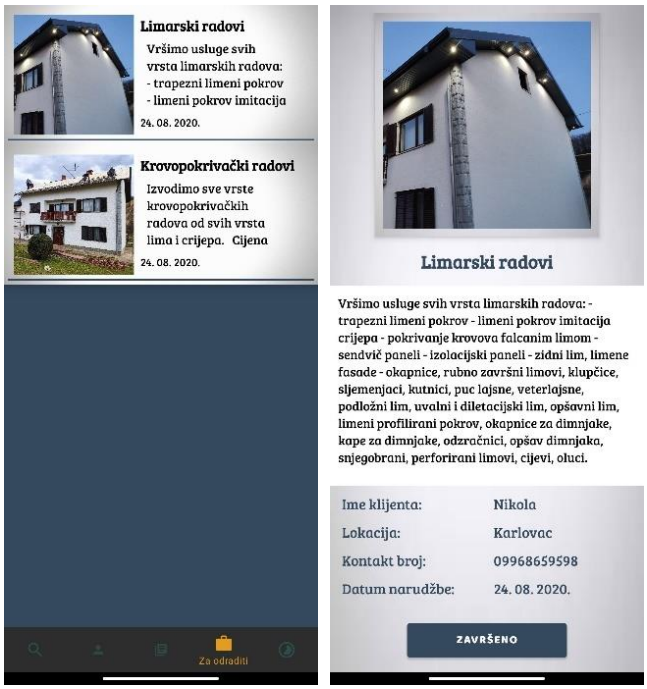
6.5. Pregled zaprimljenih usluga

Pokretanjem aktivnosti „AdsToDo“ korisnik dobiva u uvid popis svih usluga koje su od njega naručili drugi korisnici. Za prikaz naručenih oglasa se koristi *RecyclerView* kao i kod pretrage usluga (Slika 62). Programski kod za dohvaćanje svih usluga koje su drugi korisnici naručili je veoma sličan onome kod pretraživanja usluga, samo što se ovaj put podaci povlače iz podkolekcije korisnika „adsToDo“ (Slika 61).

```
query = firebaseFirestore.collection( collectionPath: "users").document(user).collection( collectionPath: "adsToDo");
```

Slika 61. Programski kod – upit (zaprimljene usluge)

Otvaranjem naručene usluge, otvara se aktivnost „AdDetails“ gdje korisnik ima pristup o informacijama kao što su *ime i prezime naručitelja, lokacija, kontakt broj* te *datum narudžbe* (Slika 62). Ti podaci se šalju preko *intenta* od aktivnosti „AdsToDo“ od aktivnosti „AdDetails“. Nakon što je korisnik odradio naručenu uslugu, klikom na gumb “Završeno” označava kako je usluga odrađena te da korisnik koji je naručio uslugu, ima mogućnost da ju ocjeni.



Slika 62. Mobilna aplikacija - usluge koje su drugi korisnici naručili

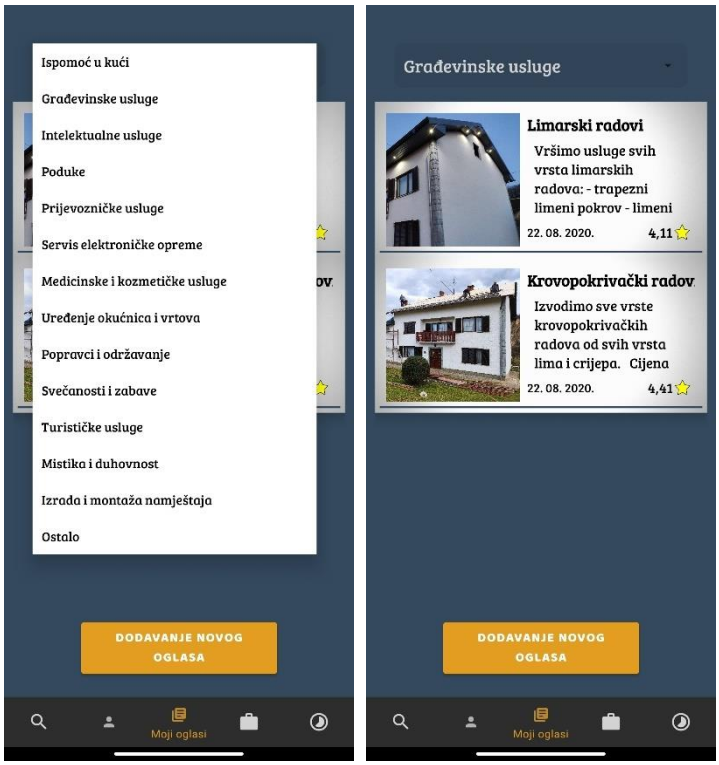
Kada korisnik klikne na gumb „Završeno“, aplikacija pomoću *DocumentReferenca* i *HashMapa* upisuje u bazu podataka u oglas koji je naručio drugi korisnik kako je usluga odrađena te će na taj način, taj drugi korisnik imati mogućnost da ju ocjeni. Nakon toga pomoću novog *DocumentReferencea* aplikacija briše uslugu iz korisnikove podkolekcije „AdsToDo“.

```
adDoneBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        DocumentReference documentReference = fStore.collection( collectionPath: "users").
                document(adClient).collection( collectionPath: "orderedAds").document(adID);
        Map<String, Object> edited = new HashMap<>();
        edited.put("adRatingBoolean", true);
        documentReference.update(edited).addOnSuccessListener(new OnSuccessListener<Void>() {
            @Override
            public void onSuccess(Void aVoid) {
                DocumentReference documentReference1 = fStore.collection( collectionPath: "users")
                        .document(currentUser).collection( collectionPath: "adsToDo").document(adToDoID);
                documentReference1.delete().addOnSuccessListener(new OnSuccessListener<Void>() {
                    @Override
                    public void onSuccess(Void aVoid) {
                        Intent myIntent = new Intent( packageContext: AdDetails.this, SearchAds.class);
                        AdDetails.this.startActivity(myIntent);
                    }
                }).addOnFailureListener(new OnFailureListener() {
                    @Override
                    public void onFailure(@NonNull Exception e) {
                        Toast.makeText( context: AdDetails.this,  text: "Error: " + e.getMessage(), Toast.LENGTH_SHORT).show();
                    }
                });
                Toast.makeText( context: AdDetails.this,  text: "Uspješno", Toast.LENGTH_SHORT).show();
                startActivity(new Intent(getApplicationContext(), SearchAds.class));
                finish();
            }
        }).addOnFailureListener(new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                Toast.makeText( context: AdDetails.this,  text: "Neuspješno.  " + e.getMessage(), Toast.LENGTH_SHORT).show();
            }
        });
    }
});
```

Slika 63. Programski kod – usluga završena

6.6. Pregled vlastitih oglasa

Otvaranjem aktivnosti „MyAds“ korisnik dobiva pregled usluga koje je on oglasio. Korisnik u padajućem izborniku odabiru kategoriju usluge te mu se tad prikazuju sve usluge koje je objavio a da su iz odabrane kategorije. Oglasi se i ovdje prikazuju preko *RecyclerView*-a (Slika 64).



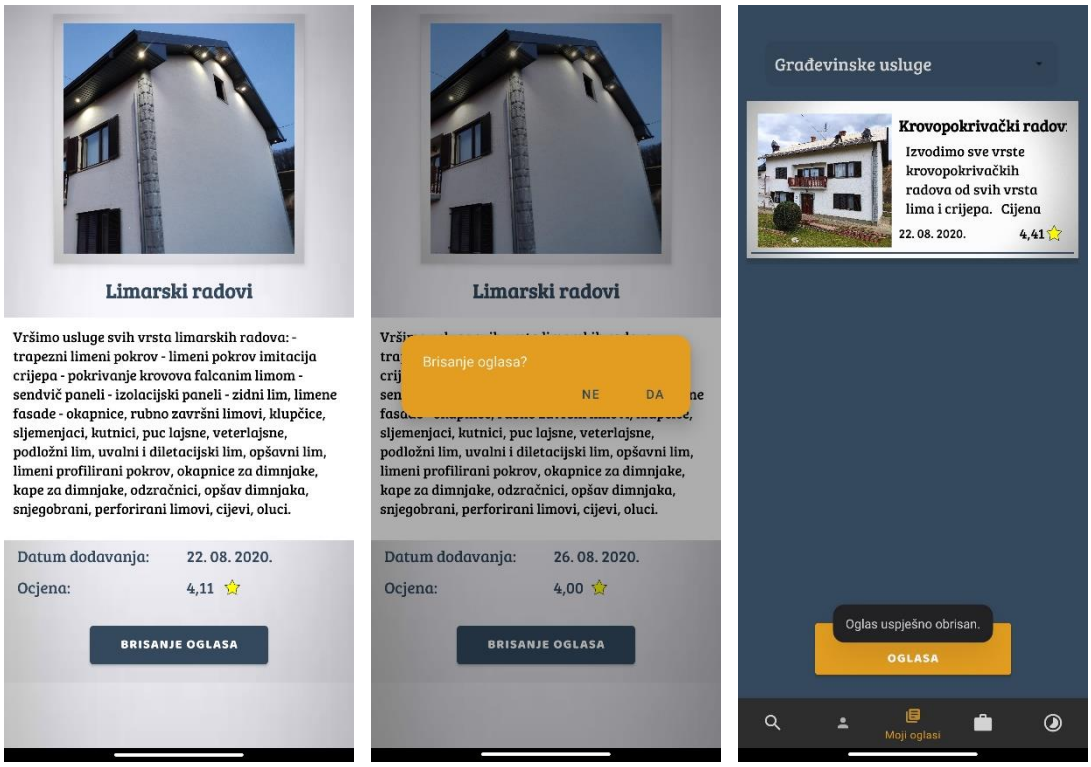
Slika 64. Mobilna aplikacija - Moji oglasi

Programski kod odnosno upit *Query* kojim se dohvaćaju korisnikovi oglasi je prikazan na slijedećoj slici (Slika 65). Pretražuju se svi oglasi u određenoj kategoriji koji imaju u sebi istu vrijednost „userID“-a kao i trenutni korisnik.

```
query = firebaseFirestore.collection( collectionPath: "adCategory").document(String.valueOf(position)).
        collection( collectionPath: "ads").whereEqualTo( field: "userID", userID);
```

Slika 65. Programski kod – upit (vlastiti oglasi)

Otvaranjem određene usluge otvara se aktivnost „AdDetails“ u kojoj korisnik ima mogućnost izbrisati uslugu. Klikom na gumb „Brisanje oglasa“ otvara se *dialog box* u kojem se korisnika pita da li želi obrisati oglas.



Slika 66. Mobilna aplikacija - Brisanje oglasa

Pritiskom na gumb „Brisanje oglasa“ otvara se *dialog box* sa naslovom „Brisanje oglasa“ te dvije ponuđene mogućnosti odgovora a to su „Da“ i „Ne“. U slučaju da korisnik klikne „Da“, kreira se *DocumentReference* sa putom do oglasa te se pomoću njega briše iz baze podataka. Nakon toga se pomoću *StorageReferenca* slika oglasa briše iz baze podataka.

```
deleteAd.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        MaterialAlertDialogBuilder delete = new MaterialAlertDialogBuilder( context: AdDetails.this);
        delete.setTitle("Brisanje oglasa?");
        delete.setBackground(getResources().getDrawable(R.drawable.alert_dialog_bg));
        delete.setPositiveButton( text: "Da", (dialog, which) -> {
                DocumentReference documentRef = fStore.collection( collectionPath: "adCategory").
                        document(adCategory).collection( collectionPath: "ads").document(adID);
                documentRef.delete().addOnSuccessListener((OnSuccessListener) (aVoid) -> {
                        Toast.makeText( context: AdDetails.this,  text: "Oglas uspješno obrisan.", Toast.LENGTH_SHORT).show();

                        Intent myIntent = new Intent( packageContext: AdDetails.this, MyAds.class);
                        AdDetails.this.startActivity(myIntent);
                }).addOnFailureListener((e) -> {
                        Toast.makeText( context: AdDetails.this,  text: "Error: " + e.getMessage(), Toast.LENGTH_SHORT).show();
                });
                StorageReference storageRef = null;
                if (adImageUrl != null) {
                    storageRef = FirebaseStorage.getInstance().getReferenceFromUrl(adImageUrl);
                }
                if (storageRef != null){
                storageRef.delete().addOnSuccessListener((OnSuccessListener) (aVoid) -> {
                        Log.e( tag: "firebasestorage",  msg: "onSuccess: deleted file");
                }).addOnFailureListener((exception) -> {
                        Log.e( tag: "firebasestorage",  msg: "onFailure: did not delete file");
                });
                }
        });
        delete.setNegativeButton( text: "Ne", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
            }
        });
        delete.show();
    }
});
```

Slika 67. Programski kod - brisanje oglasa

Unutar aktivnosti „MyAds“ korisnik također ima mogućnost dodati novi oglas za uslugu. Pritiskom na dugme „Dodavanje novog oglasa“ otvara se aktivnost „newAd“ (Slika 68). U toj aktivnosti korisnik prvo navodi „naziv“ oglasa, zatim u padajućem izborniku odabire „kategoriju“ usluge, potom „opis“ i na kraju odabire sliku koju učitava sa svog uređaja. Kao i kod učitavanja nove profine slike, tako i ovdje korisnik ima mogućnost rezanja i okretanja slike. Ukoliko korisnik nije naveo neki od potrebnih podataka, aplikacija će ga upozoriti na to. Nakon što su svi podaci ispravno uneseni, pritiskom na gumb „Dodaj oglas“, aplikacija obavještava korisnika da je oglas uspješno dodan.

Slika 68. Mobilna aplikacija - dodavanje novog oglasa

Kod pohrane novog oglasa u bazu podataka, prvo se dohvaćaju potrebni podaci o korisniku kao što su *ime i prezime, kontakt broj* te ***županija***. Ti podaci se dohvaćaju pomoću *DocumentReference-a* i *DocumentSnapshot-a* (Slika 69).

```
DocumentReference documentReference = fStore.collection( collectionPath: "users").document(userID);
documentReference.addSnapshotListener( activity: this, (documentSnapshot, e) → {
        assert documentSnapshot != null;
        adCounty = documentSnapshot.getString( field: "county");
        phoneNum = documentSnapshot.getString( field: "phone");
        advertiserName =documentSnapshot.getString( field: "fname");
});
```

Slika 69. Programski kod - dohvat podataka o korisniku

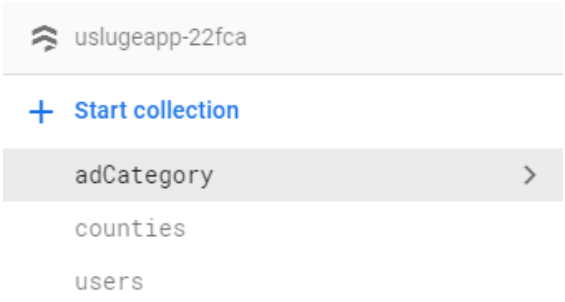
Nakon toga se pomoću novog *DocumentReferenca* te *Hashmapa* oglas sprema u bazu podataka. Unose se dohvaćeni podaci o korisniku, podaci koje je korisnik unio za uslugu te se kao *ocjena usluge* i *ukupni broj ocjena* postavlja „0“ kako bi se poslije mogla izračunati nova ocjena nakon što drugi korisnici ocjene uslugu (Slika 70).

```
DocumentReference documentReference = fStore.collection( collectionPath: "adCategory").
        document(category).collection( collectionPath: "ads").document();
Map <String, Object> ad = new HashMap<>();

ad.put("userID", userID);
ad.put("adName", name);
ad.put("adDesc", desc);
ad.put("adCounty", adCounty);
ad.put("adImageUrl", adImageUrl);
ad.put("adCategory", category);
ad.put("adID", documentReference.getId());
ad.put("adRating", 0);
ad.put("numOfRatings", 0);
ad.put("phoneNum", phoneNum);
ad.put("advertiserName", advertiserName);
ad.put("adDate", adDate);
documentReference.set(ad).addOnSuccessListener((OnSuccessListener) (aVoid) → {
        Log.d(TAG,  msg: "onSuccess");
        Toast.makeText( context: NewAd.this,  text: "Oglas uspješno dodan.", Toast.LENGTH_SHORT).show();
}).addOnFailureListener((e) → {
        Log.d(TAG,  msg: "onFailure: " + e.toString());
        Toast.makeText( context: NewAd.this,  text: "Neuspješno" + e.getMessage(), Toast.LENGTH_SHORT).show();
});
startActivity(new Intent(getApplicationContext(),MyAds.class));
```

Slika 70. Programski kod - spremanje novog oglasa

6.7. Baza podataka

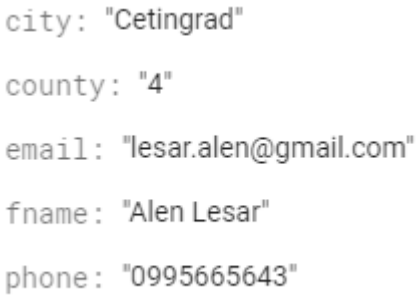
Kao baza podataka za aplikaciju je korišten Cloud Firestore. Podaci su podijeljeni u tri glavne kolekcije a to su *adCategory, counties* te *users* (Slika 71).



Slika 71. Baza podataka – glavna podjela kolekcija

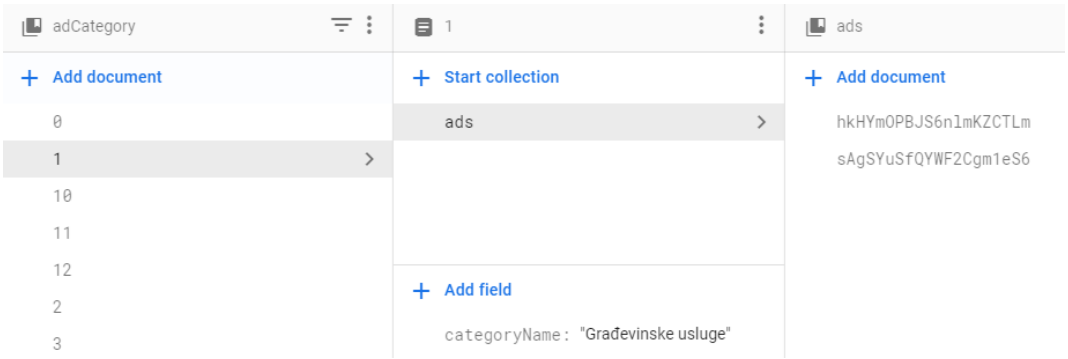
Unutar kolekcije *users* se nalazi dokument za svakog korisnika unutar kojeg su pohranjeni svi podaci o korisniku. Nakon što se korisnik registrira, kreira se dokument

sa automatski ID-em te se unutar njega pohranjuju svi podaci koje je korisnik unio prilikom registracije (Slika 72).



Slika 72. Baza podataka – dokument korisnika

Unutar kolekcije *adCategory* nalaze se kategorije usluga kojih je ukupno 13 te svaka ima polje *categoryName*. Svaka od tih 13 kategorija ima i podkolekciju *ads* u kojoj se nalaze svi oglasi o uslugama te kategorije. Na sljedećoj slici prikazana je struktura jedne od kategorija (Slika 73).

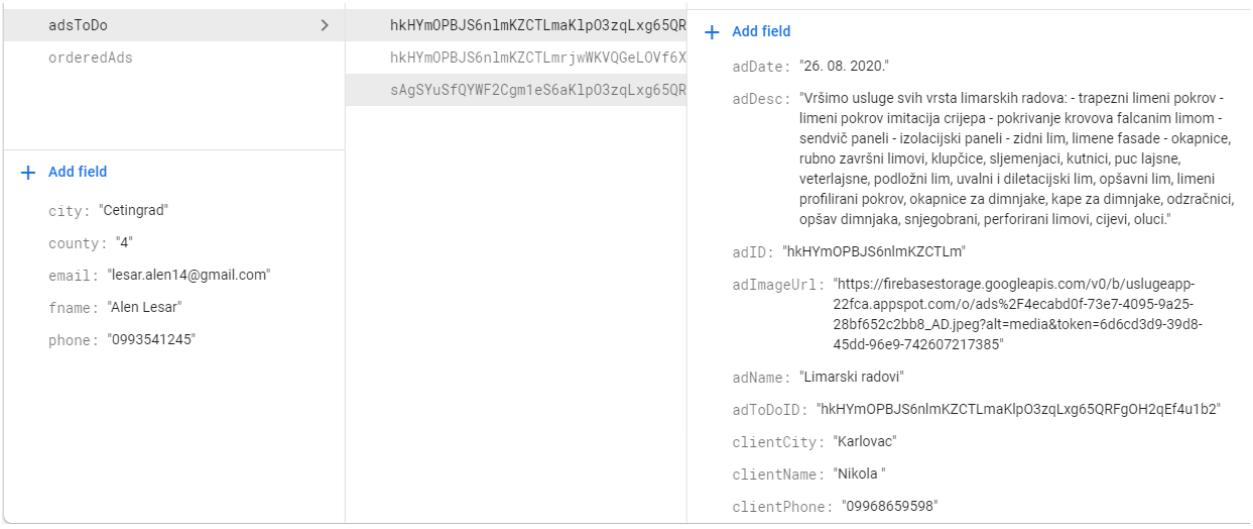


Slika 73. Baza podataka – kategorije usluga

Svaki dokument oglasa za uslugu sadrži podatke koje je korisnik unio prilikom kreiranja oglasa te se uz njih još automatski upisuje datum kreiranja oglasa te se još dodatno povlače određeni podaci o kreatoru oglasa.

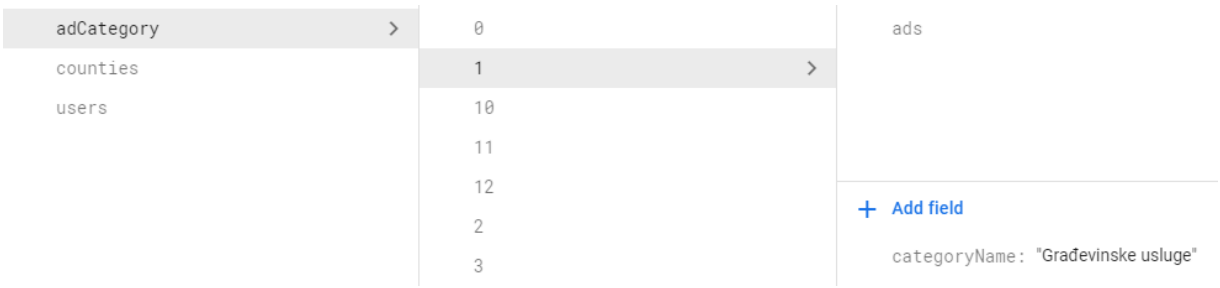
Slika 74. Baza podataka – dokument oglasa usluge

Kada korisnik naruči određenu uslugu, unutar kolekcije *users,* u dokumentu korisnika se stvara podkolekcija *orderedAds*_u kojoj se kreira novi dokumet sa informacijama o naručenoj usluzi. Isto tako kod korisnika koji nudi tu uslugu, stvara se nova podkolekcija *adsToDo*_u kojoj se nalaze dokumenti sa informacijama o uslugama koje su korisnici naručili.



Slika 75. Baza podataka – naručeni oglasi

Unutar kolekcije *counties* nalaze se dokumenti sa županijama pomoću kojih je omogućeno pretraživanje oglasa po županijama.



Slika 76. Baza podataka – kolekcija kategorije usluge

7. Zaključak

Diplomski rad se sastoji od teorijskog i praktičnog dijela. Trenutno za područje Hrvatske ne postoji aplikacija koja je fokusirana samo na usluge, odnosno njihovu ponudu i potražnju. Postoji nekoliko aplikacija koje nude tu mogućnost no više su usmjere na ponudu i potražnju dobara te stoga ne pružaju puno mogućnosti što se tiče samih usluga.

Odabir ispravne baze podataka je vrlo važna odluka. Izbor između SQL-a i NoSQL-a u potpunosti ovisi o pojedinačnim okolnostima jer obje imaju prednosti kao i nedostatke. Iako NoSQL baze podataka dobivaju na popularnosti zbog njihove brzine i skalabilnosti, još uvijek postoje situacije u kojima može biti poželjna visoko strukturirana SQL baza podataka.

Jedan od razloga za korištenjem SQL baze podataka je ukoliko je potrebno osigurati ACID svojstva. Osiguravanjem ACID svojstava se smanjuju anomalije i štiti integritet baze podataka propisujući točnu interakciju transakcija sa bazom podataka. NoSQL baze podataka žrtvuju ACID svojstva zbog fleksibilnosti i brzine obrade. Drugi razlog za odabir SQL baze podataka je taj da su podaci strukturirani i nepromjenjivi.

Kad su sve ostale komponente aplikacije na strani poslužitelja dizajnirane tako da budu brze, NoSQL baze podataka sprječavaju da podaci budu usko grlo. NoSQL baza podataka je bolji izbor ukoliko je potrebno pohranjivanje velikih količina podataka koji često imaju malo ili nimalo strukture. Baza podataka ne postavlja ograničenja za vrste podataka koji se mogu zajedno pohraniti i omogućuje dodavanje novih vrsta ukoliko se javi potreba za njima. Također omogućuje pohranjivanje podataka bez potrebe da se unaprijed definiraju vrste podataka. Ukoliko je potreban brz razvoj, te se radi sa puno iteracija ili je potrebno učestalo ažurirati strukturu podataka bez puno zastoja između verzija, relacijska baza podataka bi usporavala taj razvoj za razliku od nerelacijske baze podataka.

Razvijena aplikacija nudi korisnicima pretraživanje drugih oglasa o uslugama, odnosno postavljanje vlastitih. Prilikom pretraživanja oglasa korisnik ima mogućnost naručiti

određenu uslugu. Također ima pregled svih naručenih usluga kao i pregled svih usluga koje su drugi korisnici naručili od njega. Nakon što određena usluga bude označena kao odrađena, korisnik dobiva mogućnost ocijeniti odrađenu uslugu te na taj način ostali korisnici prilikom pretraživanja usluga imaju uvid u kvalitetu usluge ovisno o njezinoj ocjeni.

8. Literatura

Knjige:

1) Vučemilović V. i Blažević Z. (2016), *Marketing usluga – Autorizirana predavanja s primjerima iz prakse – elektroničko izdanje*, Dostupno na: https://vsmti.hr/wp-content/uploads/2018/04/Marketing-usluga-fin.pdf, [Pristupljeno: 2. 8. 2020.]
2) Yahiaoui H. (2017), *Firebase Cookbook*, Packt Publishing Ltd., Birmingham

Internet izvori:

1) Lumen Learning – *Boundless Marketing*, Dostupno na: https://courses.lumenlearning.com/boundless-marketing/chapter/services-versus-products/ , [pristupljeno 2.8.2020]
2) Anderson B. i Nichlson B. (2020), SQL vs. NoSQL Databases: What's the Difference?, Dostupno na: https://www.ibm.com/cloud/blog/sql-vs-nosql, [pristupljeno 6.8.2020]
3) Shiff L. i Rowe W. (2018), SQL vs. NoSQL Databases: What's the Difference?, Dostupno na: https://www.bmc.com/blogs/sql-vs-nosql/, [pristupljeno 6.8.2020]
4) Firebase, CloudFirestore, Dostupno na: https://firebase.google.com/docs/database/rtdb-vs-firestore , [pristupljeno 7.8.2020]
5) Firebase, Firebase Realtime Database, Dostupno na: https://firebase.google.com/docs/database , [pristupljeno 7.8.2020]
6) Kerpelman T. (2017), *Cloud Firestore vs the Realtime Database: Which one do I use?,* Dostupno na: https://firebase.googleblog.com/2017/10/cloud-firestore-for-rtdb-developers.html , [pristupljeno 10.8.2020]
7) Google Cloud , *Cloud SQL for PostgreSQL pricing* , Dostupno na: https://cloud.google.com/sql/docs/postgres/pricing#pg-cpu-mem-pricing, [pristupljeno 12.8.2020],

8) PostgreSQL, *Chapter 31.libpq – C Library* Dostupno na: https://www.postgresql.org/docs/9.5/libpq.html , [pristupljeno 5.8.2020]
9) Alagiannis I. Borovica-Gajic R. (2019), *PostgreSQL,* Dostupno na: http://renata.borovica-gajic.com/data/bk_chapter32.pdf> , [pristupljeno 10.8.2020]
10) IBM Knowledge Center, *Buffer pools***,** Dostupno na: https://www.ibm.com/support/knowledgecenter/SSEPGG_11.5.0/com.ibm.db2.luw.admin.dbobj.doc/doc/c0052482.html , [pristupljeno 14.8.2020]
11) Gupta R (2019), *SQL server CHECKPOINT, Lazy Writer and Dirty pages in SQL Server*, Dostupno na: https://www.sqlshack.com/sql-server-checkpoint-lazy-writer-eager-writer-and-dirty-pages-in-sql-server/ , [pristupljeno 12.8.2020]

Popis slika

Sažetak

Ovaj diplomski rad se sastoji od teorijskog dijela i praktičnog dijela. U teorijskom dijelu ukratko je objašnjen pojam usluge. Nakon toga su opisane *SQL* i *NoSQL* baze podataka te su navedene prednosti i nedostaci jednih i drugih. Zatim su kao primjer za *SQL* odnosno *NoSQL* baze podataka objašnjeni *PostgreSQL* i *Firebase*. Kod *Firebasea* je u prvom dijelu opisana *Realtime* baza podataka a u drugom dijelu *Cloud Firestore*. Aplikacija *UslugeApp* nudi praktično rješenje za pretraživanje usluga odnosno objavu vlastitih. Svaka usluga ima prosječnu ocjenu koja se računa na temelju danih ocjena korisnika kojima je bila isporučena.

Ključne riječi: mobilna aplikacija, android ,usluga, baza podataka, PostgreSQL, Firebase

Summary

This thesis consists of a theoretical part and a practical part. The theoretical part briefly explains the concept of service. After that, SQL and NoSQL databases are described and the advantages and disadvantages of both are listed. Then PostgreSQL and Firebase are explained as examples for SQL and NoSQL databases. For Firebase, the Realtime database is described in the first part and the Cloud Firestore in the second part. The *UslugeApp* application offers a practical solution for searching services or publishing your own. Each service has an average rating that is calculated based on the given ratings of the users to whom it was delivered.

Keywords: mobile application, android, service, database, PostgreSQL, Firebase