Android aplikacija za rezervaciju apartmana

Štajcer, Ivan

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

Sveučilišni studij

Android aplikacija za rezervaciju apartmana

Završni rad

Ivan Štajcer

Osijek, 2020.

Sadržaj

1. UVOD

U današnje vrijeme s tolikim porastom tehnologije i standarda života, olako se uzimaju pogodnosti koje su pružene. U svakodnevnici mogu se vidjeti neki pametni uređaji kao što su pametni mobiteli, televizori, tableti, satovi, i tako dalje. Svaki od tih uređaja koristi aplikacije pomoću kojih korisnik upravlja tim uređajima. To su računalni programi koji su napravljeni sa svrhom pomaganja korisniku pri izvršavanju jednog ili više zadataka [1]. Kada se priča o podijeli aplikacija bitno je navesti o kojoj se platformi radi. Postoje iOS i Android operacijski sustav i svaki ima svoje prednosti i nedostatke. iOS je operacijski sustav kojeg je razvila tvrtka Apple radi upotrebe na iPad i iPhone uređajima te nije slobodan za korištenje drugim tvrtkama. Android je razvila tvrtka Google, slobodan je za korištenje drugim tvrtkama te je zato najbrže rastući i trenutno dominantan operacijski sustav što se tiče mobilnih uređaja [2]. Za izrađenu aplikaciju izabrana je Android platforma, koja je napravljena pomoću novijeg programskog jezika zvanog Dart koji ide odlično s priborom za razvijanje softvera zvanim Flutter. Oboje je proizvela tvrtka Google i slobodni su za korištenje.

U daljnim poglavljama će biti navedene karakteristike Android operacijskog sustava, prednosti i nedostaci s obzirom na iOS. U trećem poglavlju će biti opisan Flutter i Dart, pogodnosti koje pružaju i kako se razlikuju od drugih tehnologija te kako razviti aplikaciju u Flutteru. U četvrtom poglavlju je objašnjen tijek korištenja i svrha izrađene aplikacije. Peto poglavlje nudi prikaz rješenja pisanja koda i strukturiranja same aplikacije.

1.1. Zadatak završnog rada

Kratko opisati postupak rezervacije apartmana sa stajališta vlasnika i klijenta. Dati opis tehnologija koje se koriste u izradi aplikacije kao i funkcionalnosti koje bi Android aplikacija za rezervaciju apartmana trebala sadržavati. Izraditi Android aplikaciju koja će pristupati već pripremljenoj bazi podataka s podacima o dostupnim terminima za iznajmljivanje apartmana te funkcionalnosti za rezervaciju termina u nekom od apartmana. Kratko opisati proces izrade aplikacije i podržanih funkcionalnosti.

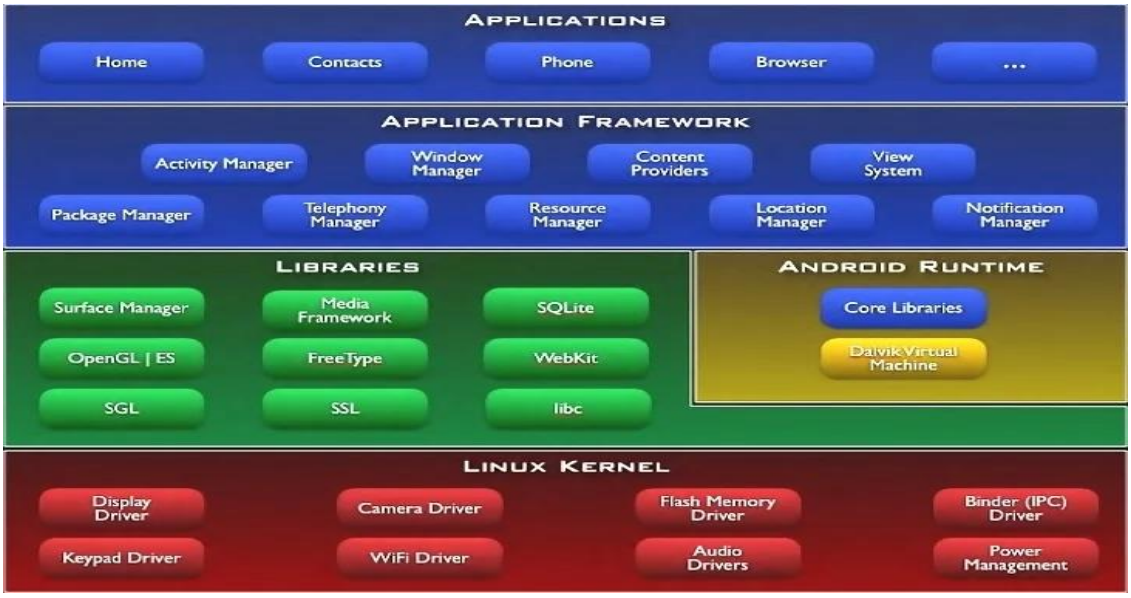
2. PREGLED KORIŠTENIH TEHNOLOGIJA

U ovom poglavlju prikazan je kratak pregled Android operacijskog sustava. Objašnjen je njegov nastanak, struktura operacijskog sustava te usporedba s iOS. Prikazan je opis o Flutter frameworku, kako je struktuiran te načinima rada u njemu. Opisane su mogućnosti Dart programskog jezika koji se koristio za izradu aplikacije. Objašnjeno je što je Firestore i moderna riješenja koje nudi.

2.1. Android operacijski sustav

2.1.1. Kratki opis Android operacijskog sustava

Android je široko prihvaćen operacijski sustav baziran na jezgri Linux 2.6 i napisan u C/C++ programskom jeziku, kojeg je prvobitno razvila tvrtka Android Inc. te je kasnije prodana firmi Google. Koriste ga razni uređaji koji su dio ljudske svakodnevnice kao što su mobilni uređaji, pametni satovi, televizori…U današnje vrijeme ne može se izbjeći susret s jednim od Android uređaja [3]. Google aktivno razvija Android platformu i daje dio nje besplatno proizvođačima hardvera i korisnicima mobitela koji žele tu platformu na svojim uređajima. Android je domnantni mobilni operacijski sustav baš iz razloga što veliki broj aplikacija podržava tu platformu. Problem kod Android aplikacija je to što su u usporedbi s aplikacijama za iPad i iPhone uređaji često niže kvalitete i u sebi mogu sadržavati zlonamjerni kod [4]. Arhitektura trenutnog Android operacijskog sustava prikazana je na slici slici .2.1.

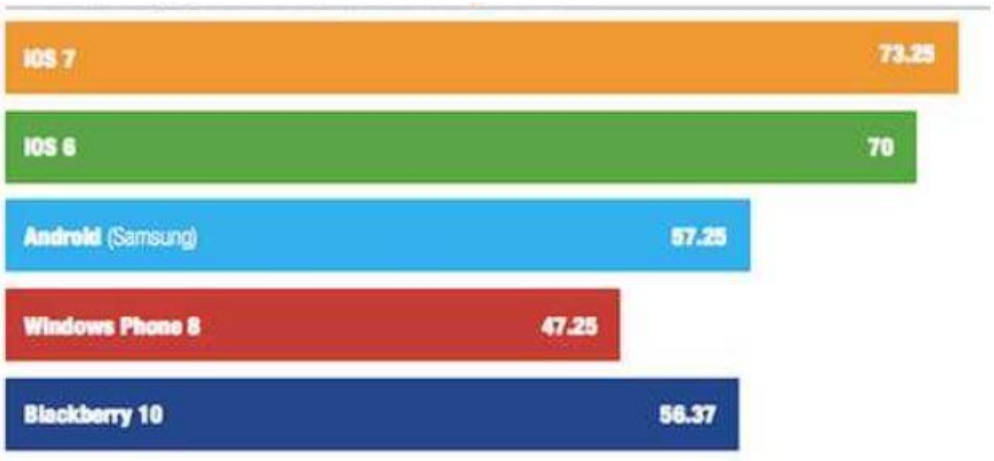


Slika 2.1. Arhitektura Android aplikacije, izvor:
https://informatika.buzdo.com/pojmovi/mobile-3.htm

Arhitektura je struktuirana tako da drajvere uređaja osigurava Linux kernel, koji upravlja procesima i memorijom sustava. Iznad njega se nalazi sloj biblioteka (Libraries) koji je pisan u Java programskom jeziku te se u njemu nalazi Android specifična biblioteka [5]. U sloju biblioteka je ugrađen Android sloj za pokretanje aplikacija u kojem se nalazi Dalvik virutalni uređaj i biblioteke jezgre koje su zaslužne za većinu funkcionalnosti operacijskog sustava. Najviši sloj sustava je Aplikacijski sloj u kojemu se nalaze sve aplikacije, kao na primjer kalendar ili web preglednik. Također, u ovom sloju se nalazi upravitelj *Activity Manager* koji kontorlira životni tijek aplikacije, *Resource Manager* koji kontrolira pristup resursima, Package manager koji sadrži i kontrolira podatke o pojedinim instaliranim paketima na uređaju, *Window Manager* koji je zaslužan za kreiranje pogleda i *layout*, *Location Manager* koji dohvaća lokaciju uređaja te *Telephony Manager* koji se koristi za upravljanje usluga na uređaju i mrežnim postavkama [6].

2.1.2. Android naspram iOS

Android i iOS operacijski sustavi se prije svega koriste u mobilnoj tehnologiji i uređajima kao što su su tableti i pametni telefoni. U usporedbi s iOS-a Android je više prilagodiv, jer svojim korisnicima dopušta prilagođavanje njegovog sučelja i osnovne značajke od vrha do dna. No, iOS se pokazao s dizajnerske strane boljim za korisnike, vidljivo na slici 2.2., jer se fokusira na glatko izvođenje animacija dok se Android fokusira više na brzinu izvođenja.



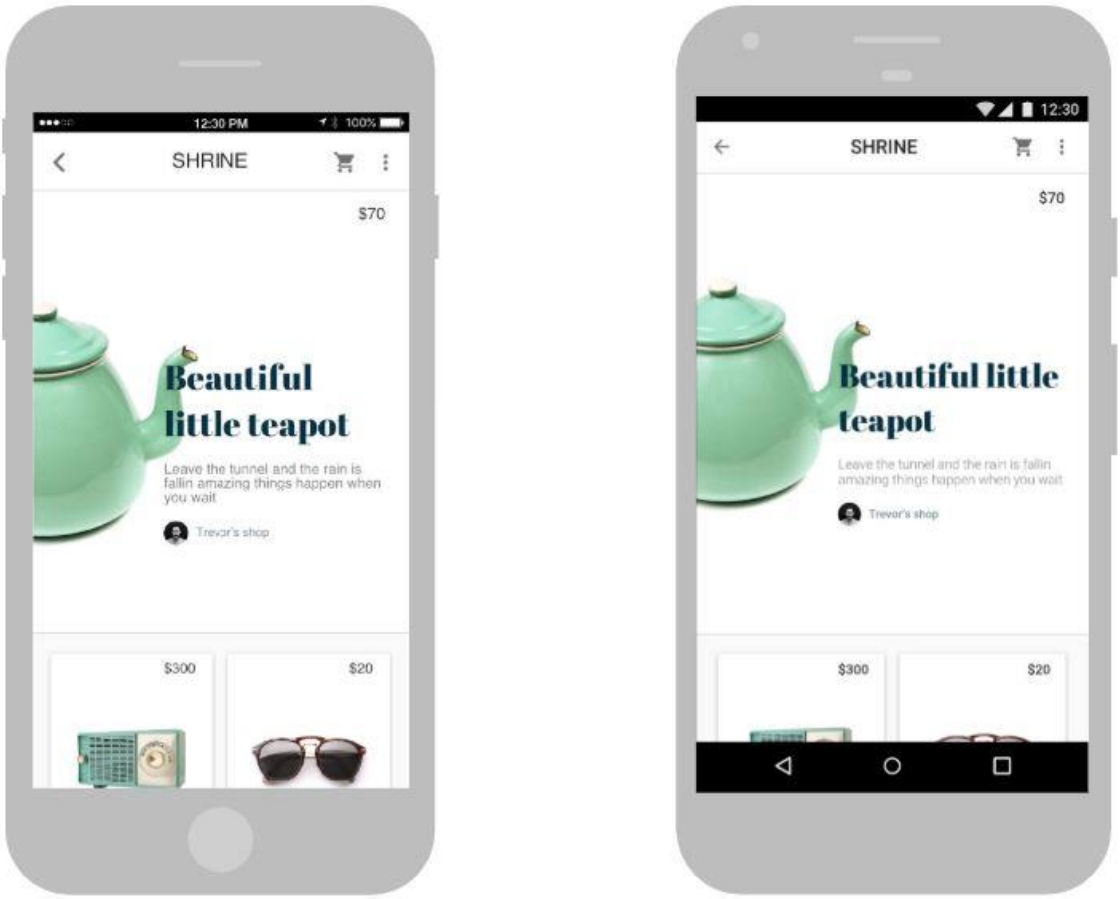
Slika 2.2. Ocjena korisničkog iskustva, izvor: https://hr.betweenmates.com/android-vs-ios

Pri odabiru operacijskog sustava je potrebno pažljivo izabrati jer za prelazak s jednog operacijskog sustava na drugi bit će potrebna ponovna kupnja aplikacija na Apple App Store-u ili Google Play trgovini. Pregled ocjena Androida naspram iOSa u rasponu ocjene 1-5 iz ukupno 3177 prikupljenih ocjena za Android i 2919 prikupljenih ocjena za iOS ukupni rezultati su 4,17/5 za Android i 3,19/5 za iOS prema [7]. No nije dovoljno reći samo ocjene, potrebno je uzeti u obzir pojedinačne karakteristike svake platforme. Kao što je već rečeno, iOS ima bolje korisničko iskustvo naspram Androida što je mnogima bitan čimbenik pri odabiru mobitela. Ako se gleda mogućnost prilagodbe tu je Android moćniji jer se može promjeniti gotovo sve dok je iOS ograničen. Uz Android dostupno je preko sto jezika dok je u iOS dostupno 34 jezika. Uređaji koji koriste iOS su iPod Touch, iPad, iPhone i Apple TV, dok Android pokriva mnogo telefona i tableta. Neki od glavnih proizvođača su Oppo, Samsung, Vivo, Xiaomi, OnePlus i Honor. Android za pozive i razmjene poruke koristi Google poruke, dok iOS iMessage i FaceTime samo za druge Apple uređaje, dok oboje koriste aplikacije treće strane popout Facebook-a, Messenger-a, WhatsApp, Discord itd. Najnovije stabilno izdanje Androida je Android 8.1.0 koji je objavljen u svibnju 2019. godine, a iOS 12.3.1 objavljen nešto kasnije, također u svibnju 2019. godine. Mnogo je različitih kategorija koje služe za uspoređivanje ove dvije platforme, no kao razvojni programer aplikacija treba se odlučiti koju platform koristiti. Baš to je razlog korištenja tehnologije pri izradi ove aplikacije, jer u Flutteru je moguće proizvoditi oboje.

2.2. Flutter i Dart

2.2.1. Flutter

Flutter je alat za razvoj softvera koji omogućava dobre preformanse i razvoj aplikacija za iOS, Android, aplikacije na webu (još je u *beta* verziji) i radnoj površini iz istog koda [8]. Flutter uzima pojedinačne značajke različitih platformi te omogućava prilagodbu s obzirom na to koju platform korisnik koristi kao što su ikone, ponašanja, geste prstiju i slično, kao što je vidljivo na slici 2.3. Kao jezik u kojemu se piše programski kod koristi se Dart.

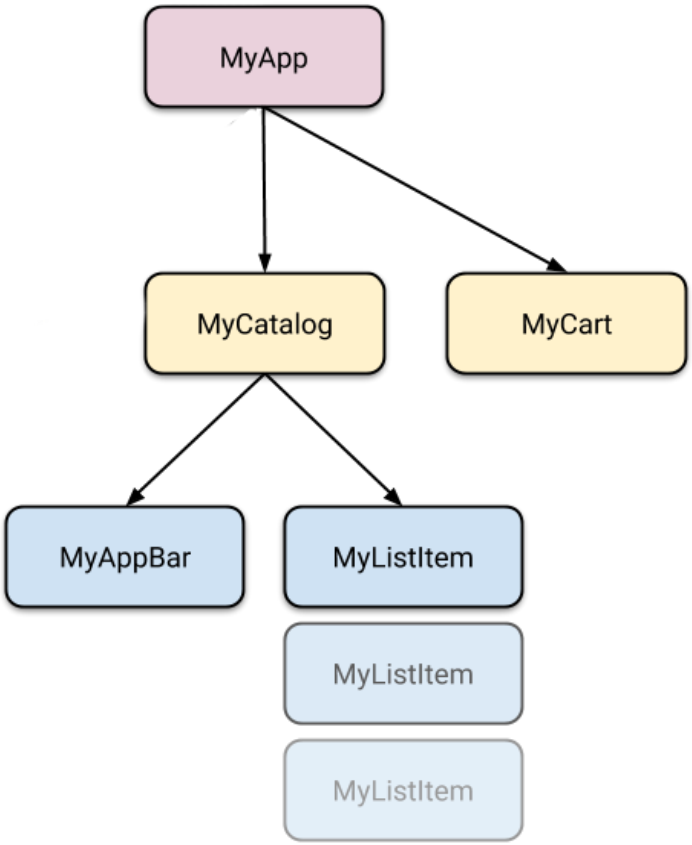


Slika 2.3. Flutter aplikacija na iOS i Androidu, izvor: https://bsgroup.eu/flutter-hail-new-cross-platform-king/

2.2.2. Widget

U programskim jezicima sve je obično objekt, to jest klasa koja predstavlja osnovni element gradnje programa u objektno orijentiranim programskim jezicima. Klase imamo i u Flutteru samo što kao osnovna struktura za gradnju aplikacije je nešto što se zove widget. Widget-i su osnovna struktura pomoću koje se gradi aplikacija u Flutter-u. Za razliku od drugih Flutter ne razdvaja poglede, kontrolere pogleda, raspored ekrana, ono za sve to ima jedan konzistentan model objekta, widget. Widget može predstavljati bilo što, kao na primjer neki element strukture kao što je gumb, neki element stila kao što je font ili boja, pozicioniranje na ekranu kao što je razmak i slično. Postoji razlika između widgeta koji se odnose na Android način prikazivanja i iOS. Za Android stil postoji *Material* widgeti, dok za iOS *Cupertino* widgeti. Oni u pozadi rade isto, u oba operacijska sustava su na kraju osnovni objekt (widget), samo je gumb

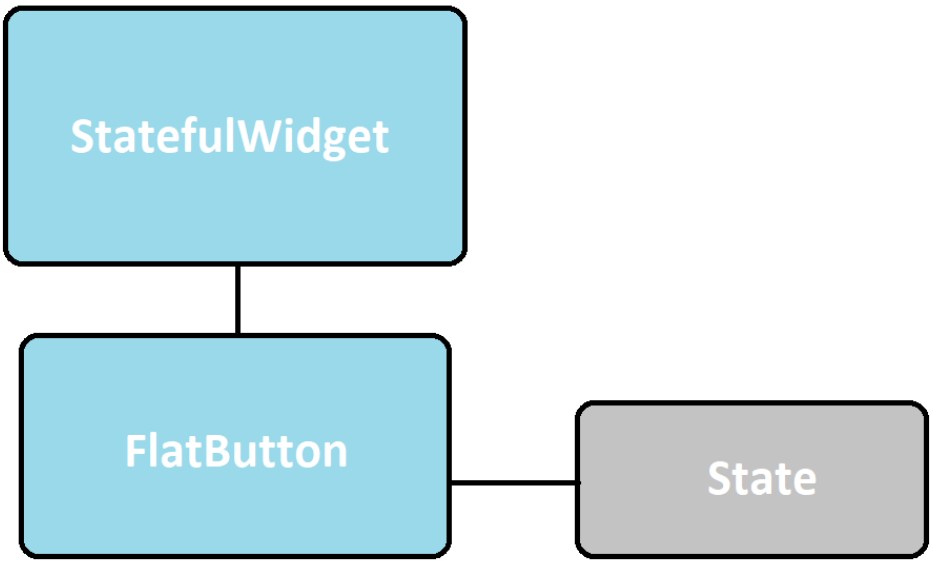
jednog stila u Androidu, a drugoga na iOS. Widgeti rade zajedno tako što formiraju hijerarhiju baziranu na kompoziciji, kao što je prikazano na slici 2.4. Dakle, svaki widget je sadržan unutar nekog drugog widgeta, osim korijenskog widgeta, i nasljeđuje njegova svojstva. Widget koji je iznad u hijerarhiji naziva se roditelj, dok ispod se naziva dijete.



Slika 2.4. Primjer hierarhije widgeta, izvor: https://flutter.dev/docs/development/data-and-backend/state-mgmt/simple

Svaki widget ima metodu „*build*" koja se poziva pri stvaranju tog widgeta, u njoj su sadržani podaci za njegovu izgradnju. Kao način reakcije na neke događaje u aplikaciji, kao što je na primjer komuniciranje s korisnikom, Flutter može zamijeniti određeni widget unutar hijerarhije s nekim drugim widgetom. Zatim uspoređuje stare i nove widgete te obnovi ekran. Flutter pruža već predefinirane widgete za korištenje [9], a često su ti widgeti sastavljeni od mnogo manjih widgeta. Kao što je bolje umjesto jedne velike klase koja radi više stvari, razbiti klasu na manje klase koje rade samo jednu stvar, isto tako je bolje imati više manjih widgeta nego jedan ogroman. Pojedinačni widgeti nisu sposobni raditi kompleksne stvari, ali zajedno mogu postići kompleksne radnje. Postoje dvije vrste widget-a, to su „*StatefulWidget*" i „*StatelessWidget*".

Razlikuju se po tome koju svrhu služe, mjenja li se unutrašnje stanje widgeta ili ne. Na primjer, ako neke karakteristike widgeta trebaju biti promjenjenje bazirano na komunikaciji s korisnikom, kao što je gumb koji mjenja boju na dodir, onda on mjenja svoje stanje te nije *StatelessWidget* već *StatefulWidget*. U ovom slučaju boja je stanje koje je očuvano u widgetu, a kada se ono promjeni, mora se promjeniti i izgled widgeta. Takvi widgeti pohranjuju svoje stanje u poseban objekt koji se zove *State,* kao što je prikazano na slici 2.5. Kako bi Flutter znao da se stanje promjenilo mora se pozvati funkcija koja se zove „*setState*", nakon poziva funkcije Flutter zna da treba promjeniti nešto na ekranu i pozvati „*build*" metodu objekta *State*.



Slika 2.5. Podjela *StatefulWidget*-a

2.2.3. Brzi rad u Flutteru

Flutter ima značajku brzog rada pri izradi aplikacija („*Hot reload*"). Naime, ona omogućava jednostavno eksperimentiranje s različitim widgetima, stvaranje korisničkih sučelja, dodavanje novih funkcionalnosti i popravljanje logičkih greški u kodu. Radi tako što su sve promjene spremljene u novostvorene datoteke koje su sadržane unutar virtualne mašine samog Dart jezika. Nakon što se virtualna mašina osvježi klase s novim verzijama njihovih polja i funkcija,

Flutter automatski zna osvježiti stablo widgeta, omogučavajući vidne promjene u kratkom vremenu. Postoje još dvije funkcionalnosti zvane „*Hot restart*" i „*Full restart*" [10]. Razlike između njih su:

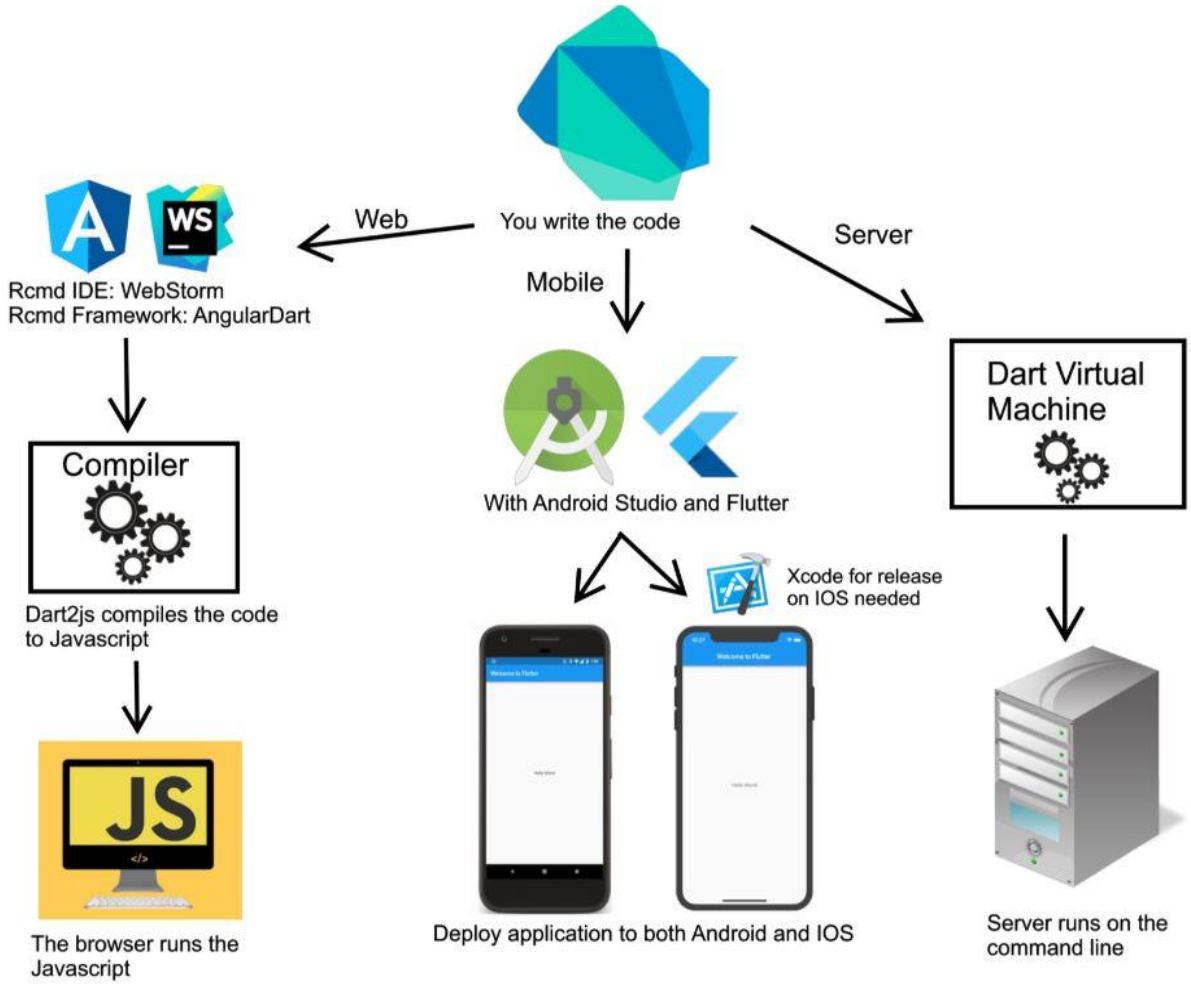
- „*Hot reload*" sprema promjene koda unutar virtualne mašine i osvježava stablo widgeta, uz očuvanje trenutnog stanja aplikacije.

- „*Hot restart*" također sprema promjene unutar virtualne mašine, ali ponovo pokreće aplikaciju te time gubi trenutno stanje aplikacije.

- „*Full restart*" ponovo pokreće čitavu aplikaciju i ova akcija traje dulje budući da ponovo procesira čitav kod u pozadini. Za ovo ne postoji specifični gumb, za razliku od prve dvije spomenute funkcionalnosti, nego jednostavno je potrebno zaustaviti i ponovo porenuti aplikaciju.

2.2.4. Dart i nativne preformanse

Dart je programski jezik otvoren za korištenje namijenjen za generalno korištenje i ne cilja jednu platform, a razvila ga je tvrtka Google. Novi je programski jezik namijenjen serverima, a također i web preglednicima. Dart dolazi sam sa svojim kompajlerom i Dart virtualnom mašinom, vidljivo na slici 2.6. Widgeti u Flutteru uzimaju u obzir sve kritične razlike u platformama na kojima se aplikacije napravljene u Flutteru pokreću. Takve razlike su navigacija, ikone, listanje i fontovi. Kod koji je napisan Flutter prevara u nativni kod za virtualnu mašinu pomoću Dart-ovih nativnih kompajlera. To sve omogućuje Dart programski jezik jer on sadrži virtualnu mašinu s JIT („*just-in-time*") kompaliranjem i AOT („*ahead-of-time*") kompajlerom za stvaranje strojnog jezika.



Slika 2.6. Funkcionalnost Dart-a, izvor:
https://www.reddit.com/r/dartlang/comments/9p8kyo/i_made_a_little_illustration_of_the_dart

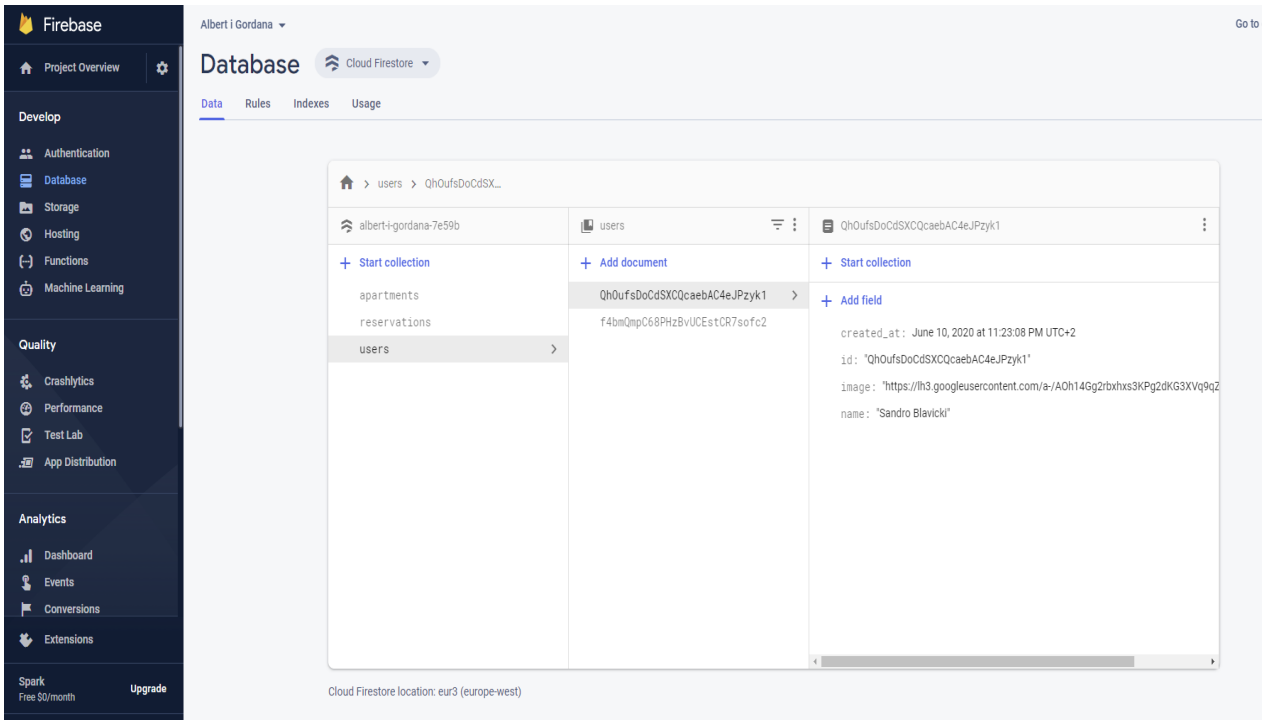
2.3. Firestore

Firestore je NoSQL, fleksibilna, lako proširiva baza podataka koja omogućuje razne funkcionalnosti za izrađivanje web i mobilnih aplikacija [11]. Ova baza podataka omogućuje razmjenu podataka u stvarnom vremenu i sinkronizaciju podataka kroz više uređaja. Osim mnoštva funkcionalnosti koje nudi tijekom uspostavljenje veze između uređaja i poslužitelja, brine se o tome da su podaci dostupni i bez spajanja na internet, tako da napravljen proizvod radi i bez internetske veze te se podaci obnove pri uspostavi veze s poslužiteljom. Ovaj proizvod napravila je tvrtka Google kao moderno rješenje proizvođačima sofwera, tako da se ne moraju brinuti o samoj bazi podataka, održavanju, algoritmima koje će koristiti pri velikim ili manjim pristupima odacima, kako će struktuirati bazu, kako će osigurati sigurnost podataka koje se

sprema unutar baze, dostupnost podacima te brzina pristupa samoj bazi. Ukratko navedeno, ovo su funkcionalnosti koje ističu ovaj proizvod:

- Fleksibilnost
- Moćni načini dohvaćanja i filtriranja podataka
- Sinkronizacija svih spojenih uređaja
- Mogućnost pristupa podataka bez internetske veze
- Efikasnost pri malim i velikim aplikacijama

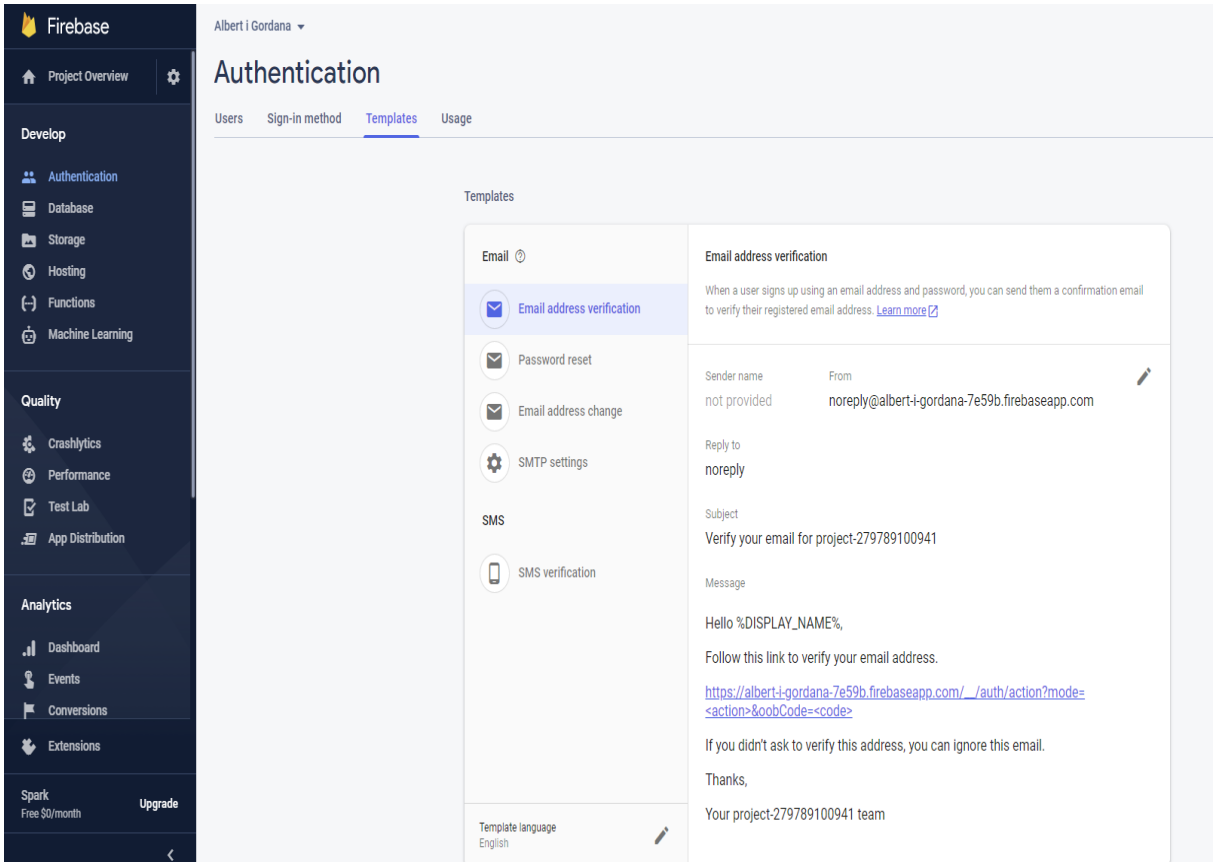
Ova aplikacija strukturira svoje podatke kao kolekcije, dokumente i podatke. Pomoću njih ona zapravo predstavlja apstrakciju složenosti koja se krije iza njih. Kolekcija predstavlja najveći skup podataka u Firestoru te svaka baza počinje od korijenske kolekcije, kao što stablo počinje od korijena. Svaka kolekcija sadržava dokumente, koji predstavljaju mjesto u bazi u kojem mogu biti spremljeni bilo kakvi podaci. Dokumenti u sebi sadrže imena podataka koje preslikava svojim pripadajućim vrijednostima, vrlo poznat oblik podatka (JSON). Neki od podataka koje podržava ova baza su cijeli brojevi, realni brojevi, boolean („*true*" ili „*false*"), TimeStamp, tip podatka koji predstavlja neki vremenski trenutak, složenije tipve podatka strukturiane u JSON formatu, polja, reference na druge lokacije unutar stabla. Bitna stavka je da dokument može sadržavati druge kolekcije te se time mogu konstruirati kompleksne hijerarhije podataka uz dobru preglednost nad podacima koja je po načinu slaganja bliža čovjeku nego računalu. Kao što je prikazano na slici 2.7., može se vidjeti UI Firestore baze podataka na kojemu je prikazan način strukturiranja podataka, bez čitanja dokumentacije može se razumjeti princip slaganja podataka. Vidljivo je da je Firebase samo dio Google platforme koja se zove Firebase u kojoj ima mnoštvo drugih funkcionalnosti vezanih za firestore i druge usluge.



Slika 2.7. UI Firestore

Osim spremanja podataka Firestore radi s jednostavnim načinom prijave korisnika u sustav s „*Authentication*" postavkama u kojim je rad s različitim korisnicima učinjen jednostavnim te proizvođači softvera ne moraju brinuti o sigurnosti podataka svojih korisnika. Osim sigurnosti pruženi su i različiti načini prijave u sustav kao što su elektronske pošte, Google račun, mobilni uređaj, Yahoo, Microsoft račun, Facebook i drugi poslužitelji. Prilikom prijave ponuđeni su već pripremljeni obrasci za slanje potvrde napravljenog korisničkog računa. Te obrasce je moguće izmjeniti po želji i kroz njih postaje lako implementirati funkcionalnost kao slanje SMS potvrde, zahtjeva za promjenu lozinke, promjena adrese elektronske pošte, što se može vidjeti na Slici 2.8. Također, za sam pristup bazi moguće je postaviti određena pravila o tome tko može raditi određene operacije nad bazom. Uz svaku funkcionalnost gleda se promet korištenja pojedine funkcionalnosti kao što su pisanje i čitanje iz baze te ovisno o tome naplaćuje korisniku usluge. Stoga se nije potrebno brinuti hoće li cijena biti prevelika za premali promet, jer svatko plaća onoliko koliko je potrošio. Google nudi određenu količinu sredstava za korištenje bez uplate sredstava korisnika, što je omogučilo da ovakva moćna platform bude korištena za hobije i razne radove. Još bitnije, omogućava pristup učenja korištenja ove usluge prije potrebe za plaćanjem. Firestore pruža jednostavno programsko sučelje za korištenje te uz različite ugradnje

paketa za pojedine platforme i programske jezike, postaje jednostavno koristiti moćan alat kao što je Firestore.



Slika 2.8. Korisnici u Firestore platform

3. FUNKCIONALNOST APLIKACIJE

U ovom poglavlju naveden je tijek izvođenja aplikacije, izgled te funkcionalnosti koje ona obavlja i kriterije koje treba zadovoljiti. Na kraju su navedeni primjeri sličnih aplikacija, koje funkcionalnosti izrađene aplikacije su slične već postojećim aplikacijama te mogućnosti proširenja izrađene aplikacije.

3.1. Tijek rada aplikacije

Svrha aplikacije je olakšati vlasniku raspored posjetitelja u apartmane u njegovom posjedu, pregled financijskog stanja pojedinih apartmana te mogućnost kontrole financija simuliranjem priljeva ili odljeva novca unošenjem željenog iznosa unutar aplikacije. Aplikacija je namjenjena vlasniku apartmana, a ne korisnicima koji žele rezervirati termin u ponuđenim apartmanima. Uz aplikaciju postoji web stranica koja koristi istu bazu podataka kao i aplikacija, što omogućava osvježavanje podataka s aplikacije i web stranice. Aplikacija je izrađena u prethodno navedenim tehnologijama, Flutter i Dart, a za bazu podataka koristi se SQLite, na uređaju, i Firestore kojeg je također kao Flutter i Dart proizveo Google. U komunikaciji sudjeluju tri strane:

- Web stranica
- Server
- Aplikacija

Korisnici pregledavaju ponuđene apartmane koji su izloženi na web stranici te odabiru apartman koji im odgovara. Zatim pregledavaju datume slobodne za korištenje koji su naznačeni posebnom bojom unutar prostora za odabir datuma. Nakon toga pošalju zahtjev za odobrenjem vremena boravka koji vlasnik odobrava ili ne odobrava, koji su u aplikaciji vidljivi pod posebnom katergorijom „ZAHTJEVI”. Vlasnik dobiva obavijest o poslanom zahtjevu za rezervaciju apartmana te ju može odobriti ili odbiti. Ako vlasnik ništa ne poduzme neće biti vidljive nikakve promjene. Podaci u bazi podataka se mjenjaju odobrenjem te datumi odobrene rezervacije postaju zauzeti na web staranici i na aplikaciji. Drugi način zauzeća datuma je korištenjem aplikacije, tako da vlasnik osobno upiše određene termine te ih označi zauzetim, što uzrokuje promjenu u bazi podataka te promjenu na web stranici. Iznosi cijena rezervacija nisu ponuđeni na internetskoj stranici, stoga je vlasniku aplikacijom omogućeno pri pravljenju rezervacija navesti cijenu te ime korisnika koji nije registriran na web stranici. Također, s aplikacijom je ponuđena kontrola prihoda pojedinog apartmana unošenjem pozitivnog ili negativnog iznosa novca te je prikazana statistika ukupnog iznosa zarade trenutne godine, ili

gubitka ako je u minusu, i prosječni mjesečni prihod u dosadašnjoj godini. Nakon što se pokrene aplikacija prikazuje se ekran na kojemu je prikazan popis svih apartmana registriranih na web stranici zajedno s njihovim pripadajućim imenima u bazi podataka, kao što je prikazano na slici 3.1.

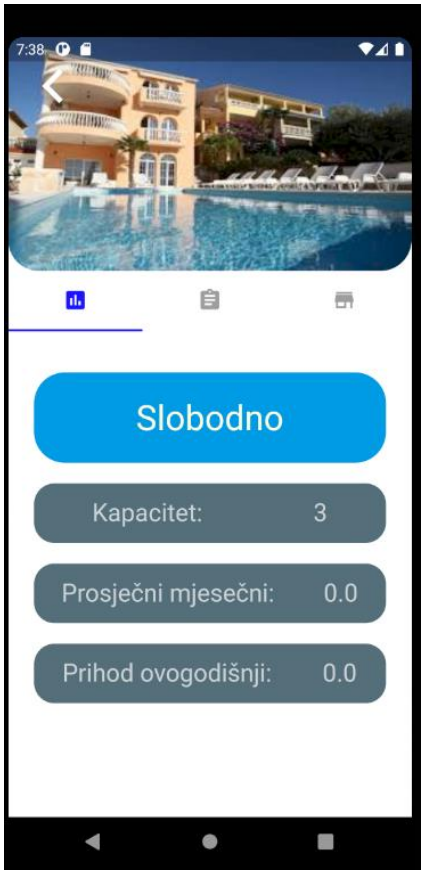
Slika 3.1. Početni ekran aplikacije

Vidljivo je da je dizajn jednostavan, budući da je svrha ovog ekrana odabir apartmana te je sukladno tome je izrađen prikaz na ekranu. Vlasnik na početnom prikazu ekrana aplikacije može listati kroz listu ponuđenih apartmana, koji su prikazani sa slikom i imenom pojedinog apartmana, dok ne dođe do željenog apartmana. Zatim, klikom na ime apartmana prikazuje se prikaz ekrana izabranog apartmana koji se sastoji od slike izabranog apartmana te ponuđenog kliznog izbornika koji ima tri pregleda:

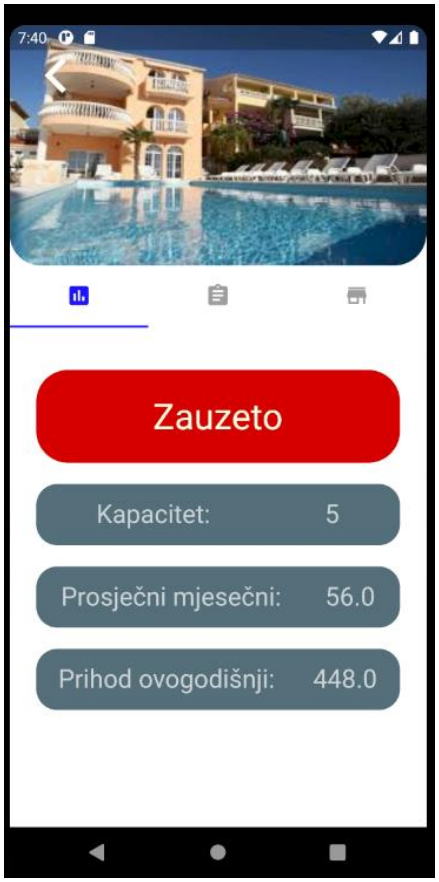
- Prikaz statistike izabranog apartmana

- Prikaz izbornika za kontrolu rezervacija
- Prikaz kontrole apartmana

Izbornici su prikazani kao ikone koje približno vidno naslućuju na prikazani sadržaj pojedinog prikaza. Prilikom mjenjanja pojedinih prikaza kliznog izbornika, mjenja se boja ikone trenutno odabranog prikaza, što naznačuje koji je prikaz trenutno odabran. Vlasnik mjenja prikaze pritiskom na ikone ili kliznim pokretom prsta u lijevu ili desnu stranu. Prvobitno je ponuđen prikaz statistike izabranog apartmana, na kojem se vidi par karakteristika apartmana. Prije svega najbitnija od njih, zato najviše istaknutija, je informacija o trenutnom stanju zauzeća apartmana. Ako je apartman trenutno slobodan, to je naznačeno svijetlo plavom bojom i tekstom „Slobodno", kao što vidimo na slici 3.2. U suprotnom ako apartman nije slobodan, to je naznačeno crvenom bojom i tekstom „Zauzeto", što je vidljivo na slici 3.3. Kao tema kroz čitavu aplikaciju odabrana je plava boja, stoga ako se nešto prikaže u drugoj boji kao što je crvena zahtjeva vlasnikovu pozornost i prvo mu upada u oči. Osim ovoga prikazano je i broj osoba koji apartman može primiti te financijsko stanje apartmana za trenutnu godinu i prosječni mjesečni prihod do trenutnog mjeseca u godini.



Slika 3.2. Slobodan apartman

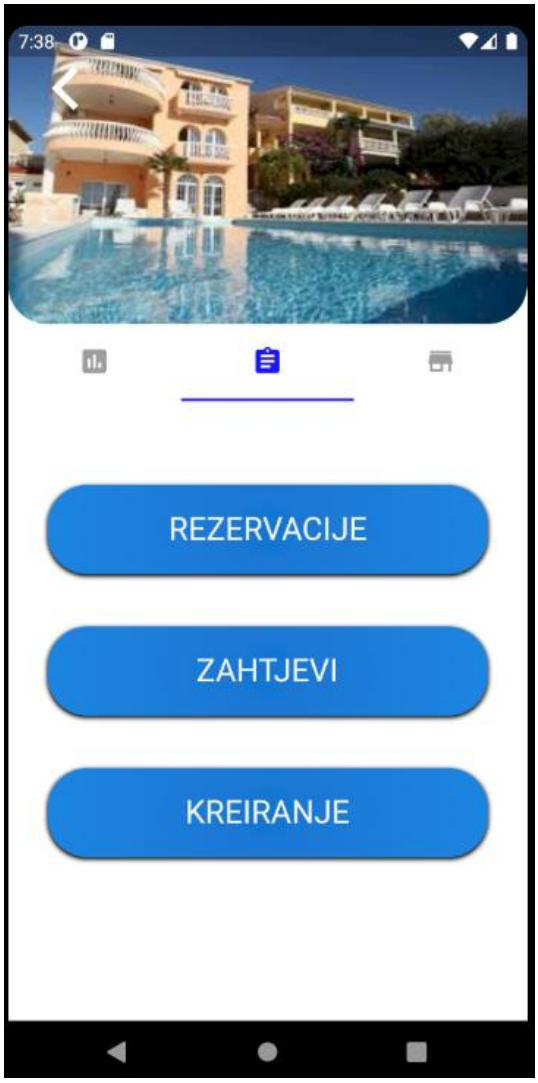


Slika 3.3. Zauzet apartman

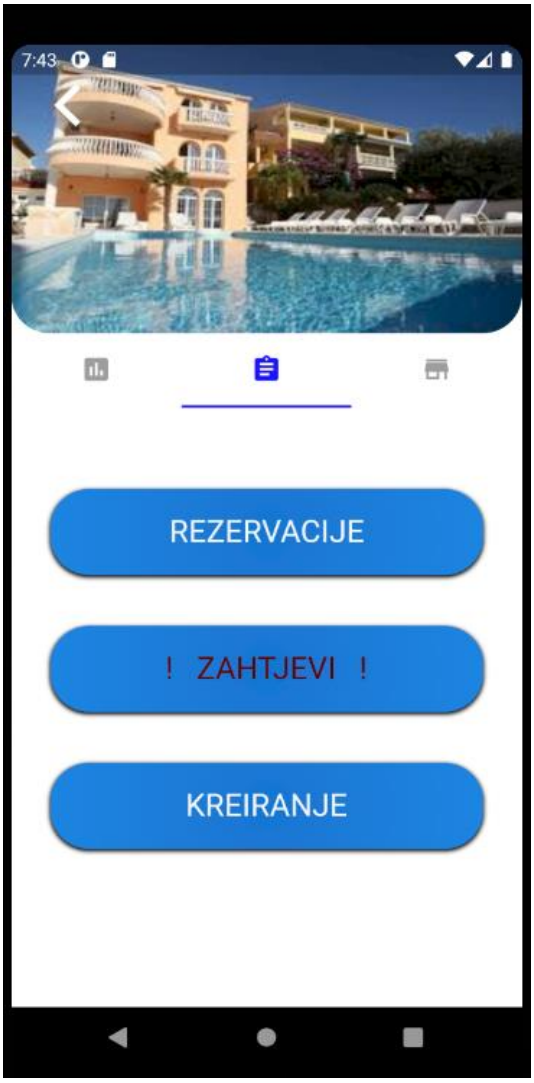
Klikom na prikaz pregleda za kontrolu rezervacija prikazuje se sučelje na su prikazana tri gumba, kao što je vidljivo na slici 3.4., koja su u sukladno s temom aplikacije plave boje:

- REZERVACIJE
- ZAHTJEVI
- KREIRANJE

Ovdje vlasnik može navigirati kroz kontrole rezervacija. Prikazano na slici 3.5., ako korisnik pošalje zahtjev za zauzeće apartmana na web stranici, gumb „ZAHTJEVI" promijenit će izgled tako što se tekst zacrveni i pojave se dva uskličnika na gumbu. Time je uočjivo da je potreba vlasnikova akcija, kao i na predhodnom prikazu boja teksta se mjenja u crvenu i ističe se u odnosu na ostale. Time vlasnik ne može ne primjetiti nadolazeći zahtjev za rezervaciju.



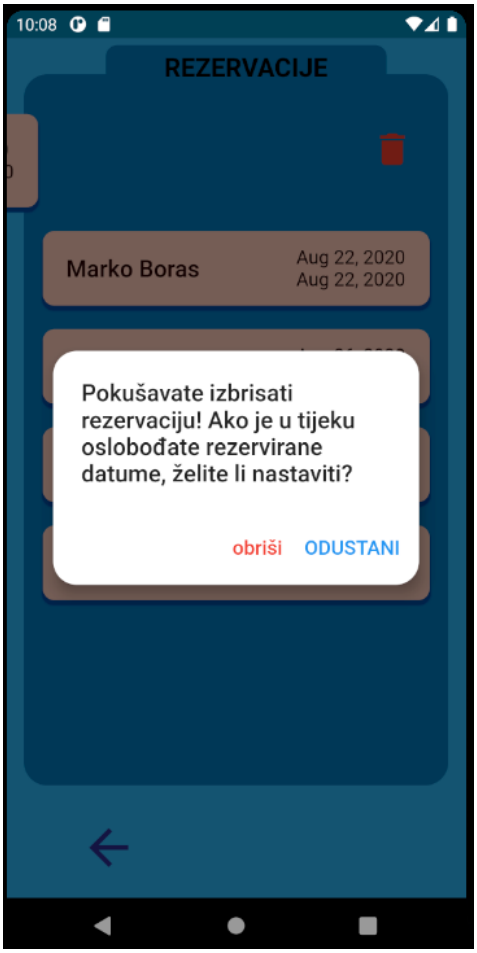
Slika 3.4. Izbornik kontrole rezervacija



Slika 3.5. Zahjev na čekanju

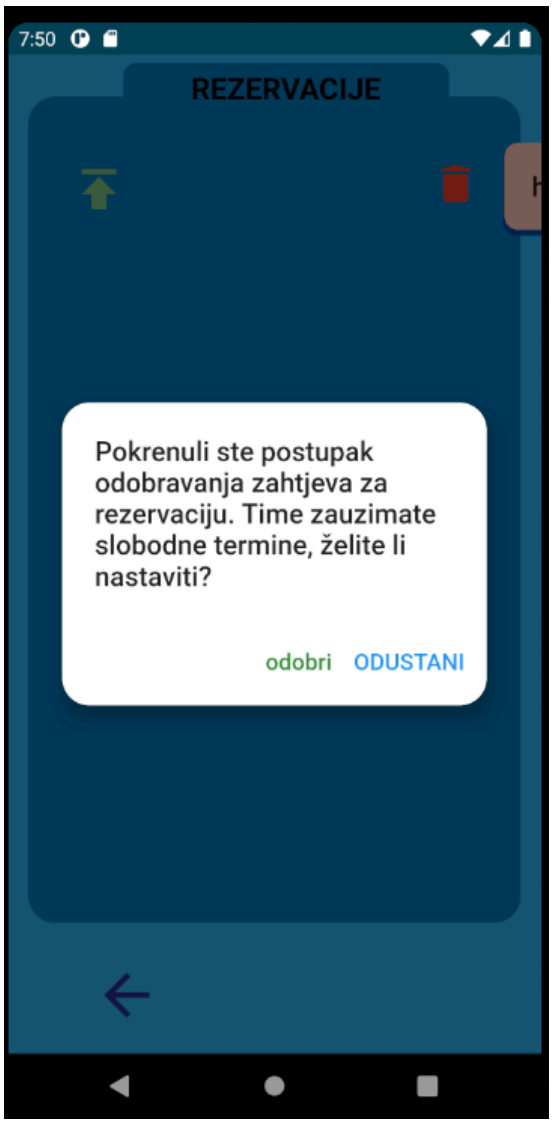
Odluči li se vlasnik pregledati postojeće rezervacije može odabrati „REZERVACIJE”. Takvim odabirom se pikazuje novi ekran s listom svih rezervacija koje se drže u bazi podataka koje je vlasnik odobrio, to jest kojima je *approved* zastavica postavljena na *true*. Svaka rezervacija prikazna je kao kartica koja sadrži ime korisnika koji je zatražio rezervaciju te datum početka ikraja rezervacije apartmana, vidljivo slici 3.6. Kako se rezervacije čuvaju dok ih vlasnik ne izbriše, one rezervacije kojima je datum završetka prošao naznačene su crvenom bojom. Rezervacija koja nije završila, naznačena je plavom bojom. Rezervacije su poredane od vrha prikaza ekrana do dna prema datumima, starije rezervacije si pri vrhu, a novije prema dnu. Tako da vlasnik ne mora tražiti trenutnu rezervaciju, a pomoću boja se točno vidi granica između isteklih, trenutno aktivnih i nadolazećih rezervacija. Vlasnik može obrisati rezervaciju kliznim pokretom prsta u lijevu stranu, time se pojavljuje crveni koš za smeće kao naznaka operacije brisanja. Kao što je vidljivo na slici 3.7., prije konačnog brisanja rezervacije, prikazuje se poruka upozorenja te je potrebno potvrditi nastavak radnje ukoliko je vlasnik siguran da je želi obrisati.

Slika 3.6. Lista odobrenih rezervacija



Slika 3.7. Tekst brisanja rezervacije

Klikom na gumb za povratak u donjem lijevom kutu, prikazan kao strelica u lijevo, vlasnik se vraća na izbornik za kontrolu rezervacija. Ako ima nadolazećih zahtjeva može odabrati gumb „ZAHTJEVI” te će mu se prikazati identični ekran s prikazom rezervacija koje nisu još odobrene i stoga ne utječu na datume. Kao i već odobrene rezervacije, zahtjevi su poredani po redu od vrha do dna po datumima, a istekli zahjtevi prikazani su crvenom bojom. Zahtjevi ostaju izloženi dok ih vlasnik ne potvrdi ili ne obriše. Brisanje zahtjeva također se pokreće kliznim pokretom prsta u lijevu stranu s prikazom crvenog koša za smeće, vidljivo na slici 3.9. Vlasnik potvrđuje zahtjev za rezervaciju kliznim pokretom prsta u desnu stranu, pojavljuje se zelena ikonica za pohranu podataka, kao što je prikazano na slici 3.8. Prije nego li se izvrše operacije brisanja ili dodavanja, vlasnik je upozoren i mora potvrditi akcij za daljni nastavak.

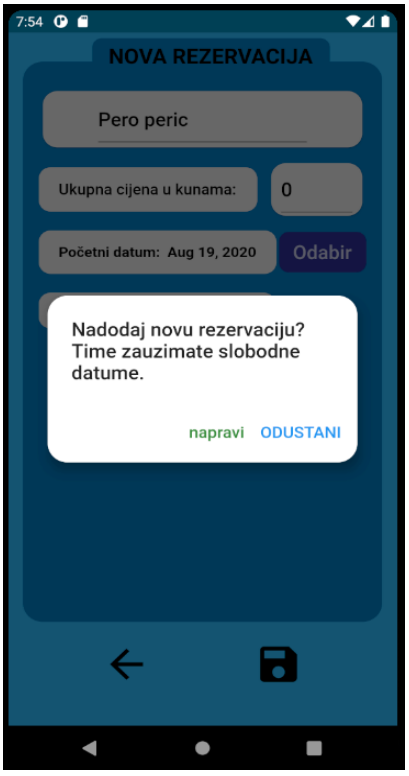


Slika 3.8. Potvrda zahtijeva

Slika 3.9. Odbijanje zahtijeva

Ponovnim pritiskom gumba za povratak, vlasnik je vraćen na prikaz za kontrolu rezervacija. Osim pregleda rezervacija i zahtjeva, moguće je kreirati vlastitu rezervaciju klikom na gumb „KREIRANJE”. Prikazuje se nova stranica za izradu rezervacije, koja nudi unos potrebnih parametara za izradu, vidljivo na slici 3.10. Potrebni podaci su ime i prezime na koje je rezervacija napravljena, nije moguće napraviti rezervaciju bez imena, iznosa novca za koji je rezervacija dana, moguće je upisati samo pozitivne iznose, uključujući nulu te je potreban unos. Osim toga, prikazana su dva datuma koja predstavljaju izabrani početni i završni datum rezervacije koju vlasnik želi napraviti. Pritiskom na gumb „Odabir” prikazuje se kalenar na kojemu je moguće odabrati raspon datuma unutar kojeg će trajati rezervacija te se odabirom mjenja prikaz odabranih početnog i završnog datuma. Datume koji su prošli nije moguće izabrati, kao i datume već zauzete za druge rezervacije te su oni prikazani crvenom bojom. Ako vlasnik pokuša odabrati raspon datuma u kojeg ulaze jedan od već zauzetih datuma, prikazat će se obavijest o neispravnom odabiru datuma. Nakon što vlasnik ispuni sve podatke potrebne za izradu, pritiskom na ikonu za spremanje u donjem desnom kutu, prikazuje se obavijest o trenutnoj radnji i vlasnik mora potvrditi radnju kako bih ona bila izvršena, prikazano na slici 3.11. Ako vlasnik odluči odustati, ostaje na trenutnom ekranu s ispunjenim podacima. Odluči li nastaviti, rezervacija je dodana pod odobrene rezervacije i vlasnik je vraćen na prikaz za kontrolu rezervacija.

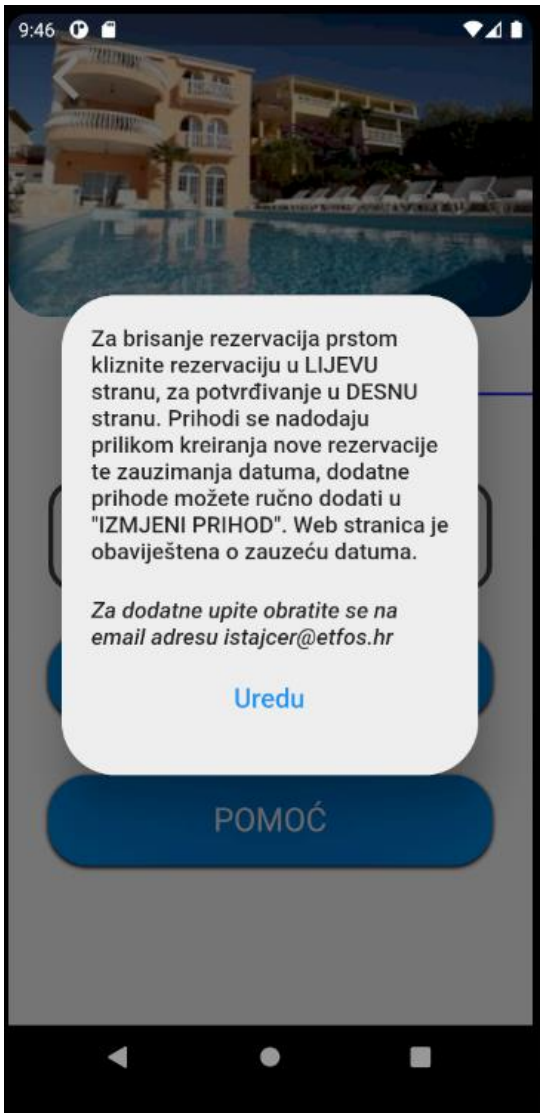
Slika 3.10. Kreiranje rezervacije



Slika 3.11. Potvrda kreiranja

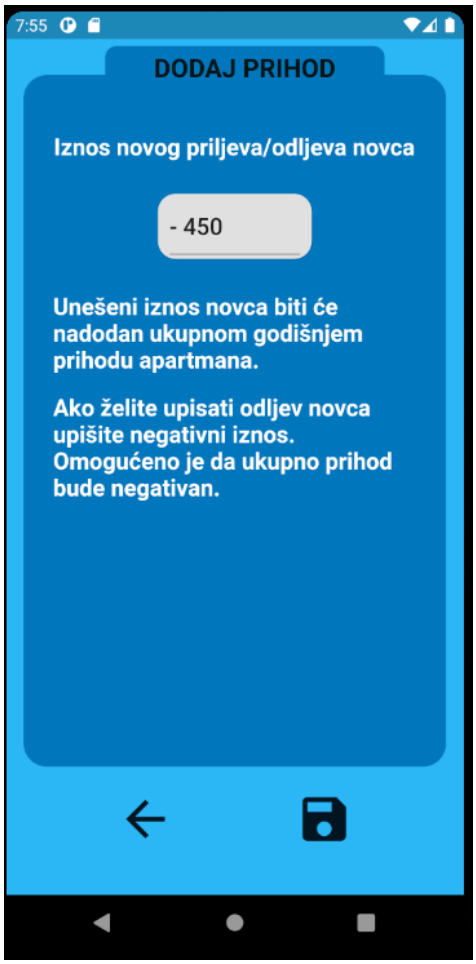
Nakon toga vlasnik može kliznim pokretom u lijevu stranu ili pritiskom na slijedeću ikonu, prikazanu kao apartman, doći do pregleda kontrole apartmana. Na ovom pregledu vlasnik može vidjeti informaciju o prvom nadolazećem slobodnom datumu za trenutni apartman, što je često potrebna informacija. Prikazan je gumb za izmjenu prihoda apartmana i gumb za pomoć, vidljivo na slici 3.12. Kao što je prikazano na slici 3.13., gumb za pomoć prikazuje kratke informacije o tome kako se izvode informacije brisanja rezervacija, dodavanja rezervacija. Pomoćni tekst opisuje što se događa pri izmjenama i kako se može kontrolirati prihod apartmana. Za ostale informacije vezane uz aplikaciju izložena je adresa elektronske pošte na koju se može kontaktirati vlasnika. Nakon čitanja vlasnik može pritisnuti gumb „Uredu" te je vraćen nazad na prikaz kontrole apartmana.

Slika 3.12. Prikaz kontrole apartmana



Slika 3.13. Pomoć

Ako vlasnik želi promijeniti ukupni prihod apartmana, to može učiniti pritiskom na gumb „IZMJENI PRIHOD”. Kao što je prikazano na slici 3.14., pritiskom na gumb „IZMJENI PRIHOD” prikazuje se ekran za izmjenu prihoda apartmana. Ponuđeno jedno polje za upis podataka, unutar kojeg vlasnik može upisati iznos novca koji se nadodaje ukupnom prihodu apartmana za trenutnu godinu i time mjenjati prihode apartmana radi drugih faktora, osim rezervacija. Za razliku od unošenja iznosa cijene pri kreiranju rezervacije, gdje je moguće unjeti samo pozitivne iznose, ovdje vlasnik ima mogućnost simulirati odljev novca unošenjem negativnog iznosa. Time vlasnik može upisati troškove apartmana koji nastaju tijekom godine te vidjeti donosi li apartman profit ili trošak. Nakon unošenja iznosa, vlasnik može pritisnuti ikonu za spremanje u donjem desnom kutu te se prikazuje poruka o potvrdi obavljanja radnje, vidljivo na slici 3.15. Ako vlasnik odluči nastaviti s obavljanjem radnje, ukupan prihod apartmana se mjenja te je to vidljivo na prikazanim statistikama apartmana. Također, uspješnim izvođenjem operacije vlasnik je vraćen na prethodni ekran.



Slika 3.14. Izmjena prihoda

Slika 3.15. Potvrda izmjene prihoda

3.2. Funkcije aplikacije

Glavne funkcije aplikacije su primanje obavijesti o novim zahtjevima za rezervaciju, izmjena prihoda pojedinih apartmana, pregled apartmana, brisanje rezervacija i dodavanje novih rezervacija. Aplikacija je izrađena da bude jednostavna za korištenje te otvorena za moguća buduća proširenja. Kroz cijelu aplikaciju tema aplikacije je plava boja i vlasnik to može prepoznati kao pozitivnu stvar u aplikaciji, kao na primjer kod trenutno aktivne rezervacije, dok su crvenom naznačena upozorenja. Vlasnik aplikacijom ne može dodavati nove apartmane, apartmani koje vlasnik izlaže na svojoj stranici su prikazani na aplikaciji. Aplikacija i web stranica koriste istu bazu podataka. Stoga, ako vlasnik doda još neki apartman i izloži ga na web stranici, on će se automatski prikazati na mobilnoj aplikaciji.

3.3. Slične aplikacije

3.3.1. Trivago

Trivago mobilna aplikacija je namjenjena za korisnike i prikazuje hotele koji su u blizini korisnika ili na nekoj drugoj željenoj lokaciji. Nudi mogučnosti označavanja najdražih hotela i lokacija posebno za sakog korisnik te uspoređivanje cijena više različitih hotela. Trivago aplikacija ne zahtijeva kreiranje korisničkog računa i napravljena je za široku primjenu.

3.3.2. Airbnb

Airbnb mobilna aplikacija je namjenjana za korisnike i nudi mogučnost prikazivanja lokacija koje se nalaze u blizini korisnika. Aplikacija nudi pretraživanje putovanja za različite lokacije i prosječni iznos noćenja na određenoj lokaciji. Airbnb zahjteva kreiranje korisničkog računa i time personalizira korisničko iskustvo pojedinog korisnika, omogućuje korisniku označavanje najdražih mjesta i putovanja te radi korisničkog računu moguće je koristiti na više različitih uređaja. Airbnb ima kao funkcionalnost pružanje pomoći putem online razgovoa unutar aplikacije, stoga korisnici mogu direktno komunicirati s timom za pružanje pomoći korisnicima preko aplikacije i time poboljšava korisničko iskustvo.

3.3.3. Sličnosti s izrađenom aplikacijom

Izrađena aplikacija je namijenjena za vlasnika, radi pregleda datuma i financija apartmana na raspolaganju vlasnika. Aplikaciju je moguće proširiti na korisnike dodavanjem novih funkcionalnosti, kao što je korisnički račun i mogućnost slanja zahtjeva putem aplikacije, kao što je to unutar Airbnb aplikacije. Kako bi korisnici mogli koristit izrađenu aplikaciju, potrebno je ograničavanje određenih funkcionalnosti kao što je pregled financija, što je podatak namjenjen samo za vlasnika apartmana i nije prikazan unutar Airbnb ni Trivago aplikacije. Također, moguće je dodati prikaz lokacija apartmana na karti i omogućiti korisnicima ocjenjivanje pojedinih apartmana s obzirom kako im se svidjelo noćenje kao što je to unutar Trivago aplikacije. S dodavanjem i oduzimanjem određenih funkcionalnosti, izrađenu aplikaciju je moguće prilagoditi za namjenu korisnika, kao što je i namjena njoj sličnim aplikacijama poput Airbnb i Trivago aplikacije.

4. RAZVOJ APLIKACIJE

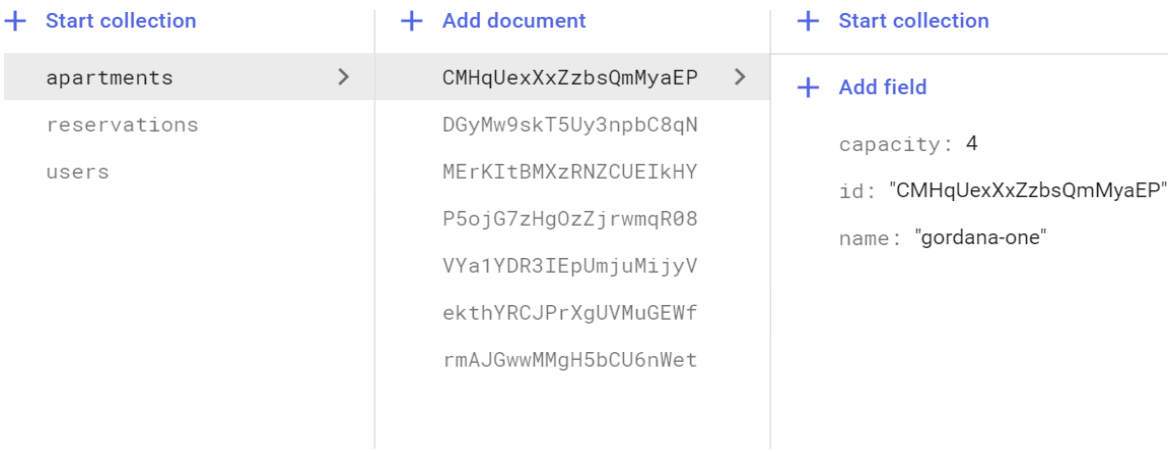
U ovom poglavlju prikazuju se rješenja pri razvoju aplikacije kao što su komuniciranje s bazom podataka na serveru pomoću Firestore usluge. Prikazana razmjena podataka između widgeta kroz aplikaciju pomoću *Provider* obrasca. Objašnjena je komunikacija s lokalnom bazom podataka i korištenje asinkronog koda.

4.1. Komuniciranje s Firestore bazom podataka

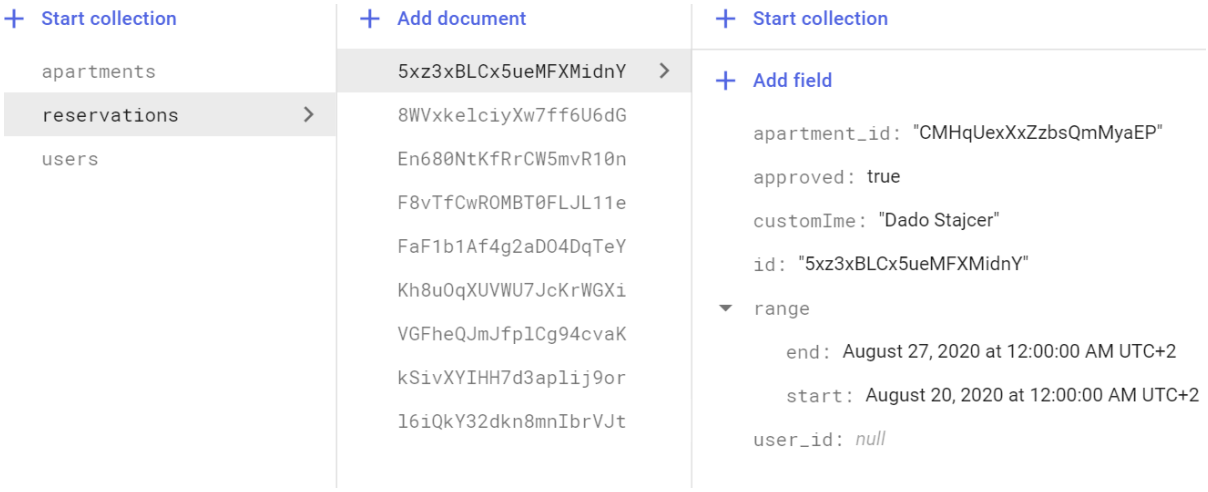
Podatke spremljene unutar Firestore baze podataka koriste web stranica i mobilna aplikacija. Njihova struktura je radi bolje organizacije podataka podjeljena u tri kolekcije:

- Apartments
- Users
- Reservations

Svaka od kolekcija ima listu dokumenata koji predstavljaju pojedini element unutar kolekcije. Kao što je prikazano na slici 4.1., pod kolekcijom „*Apartments*" pojedini elementi imaju spremljene podatke o kapacitetu apartmana, identifikacijskom broju apartmana i nazivu apartmana. Pojedini dokumenti pod kolekcijom „*Reservations*", prikazani na slici 4.2., imaju spremljene podatke o identifikacijskom broju apartmana kojem pripadaju, identifikacijom broju korisnika koji je poslao zahtjev, ime korisnika koje upisuje vlasnik pri pravljenju nove rezervacije, raspon datuma od kojih počinje i završava trajanje rezervacije i podatak o tome je li rezervacija odobrena ili ne. Podaci u dokumentima koji pripadaju kolekciji „*Users*" sadrže identifikacijski broj korisnika, ime korisnika, vrijeme kreiranja korisnika i slika korisnika, kao što je vidljivo na slici 2.7. Aplikacija koristi svaki podatak, osim vremena kreiranja i slike korisnika.



Slika 4.1. Podazi Firestora za apartmane



Slika 4.2. Podaci Firestora za rezervacije

Sukladno s podacima na Firestoru, u aplikaciji su napravljeni pojedinačni modeli za podatke. To znači da su prikazane klase na slici 4.2. *Apartman*, *Resrvation* i *User* pomoću kojih konstruiramo spremljene podatke. Osim navedenih podataka u Firestore, *Apartman* model podatka sadrži još atribut *totalIncome* koji se koristi pri lokalnoj bazi za praćenje totalnog iznosa prihoda apartmana za tekuću godinu. Ta informacija nije potrebna korisnicima na internetu te je stoga nema potrebe izlagati izvan uređaja. Za komuniciranje s Firestore unutar Fluttera korištena je *cloud_firestore 0.14.0+1* biblioteka koja je napravljena baš za tu platformu. Prije svega, na mjestu gdje se želi koristiti ova ugradnja, potrebno je uključiti paket. Komunikacija započinje dohvaćanjem instance objekta pomoću kojega se komunicira. Na njemu se može pozvati metoda *.collections()* pomoću koje se određuje kojoj kolekciji pristupiti. Potrebno je uspostaviti *Stream* podataka koji vraća trenutnu sliku stanja sa *.snapshots()*. To je važno jer se u flutteru može koristiti widget koji olakšava rad sa Stream podacima i koji se zove

StreamBuilder, koji je prikazan u kodu 4.1. *Streambuilder* ima parameter *stream* kojem je potrebno pružiti promatrani *Stream*, u ovom slučaju to je:

Firestore.instance.collection("/reservations/").where("apartment_id",isEqualTo:_apartman.id).snapshots()

Drugi parameter *StreamBuilder* widgeta je *builder*, koji prima funkciju koja za parameter daje trenutni *context*, koji je metapodatak o trenutnoj poziciji widgeta u stablu, i *snapshot* , koji nam daje trenutna sliku stanja slanja podataka, jesu li podaci učitani ili ne. Ova metoda vraća widget koji će se prikazati na ekranu. Dok podaci nisu učitani iz baze podataka vraća se *CircularProgressIndicator,* koji daje jasnu vizualnu informaciju učitavanja podataka, tako što prikazuje vrteći krug na ekranu, *loading spinner*. Nako toga, potrebno je podatke dohvatiti iz *Streama* kroz trenutne preslike podataka iz *Streama*.

```
child: StreamBuilder(
    stream: Firestore.instance
        .collection("/reservations/")
        .where("apartment_id", isEqualTo: _apartman.id)
        .snapshots(),
    builder: (context, snapshot) {
      if (snapshot.connectionState == ConnectionState.waiting) {
        return Center(
          child: CircularProgressIndicator(),
        ); // Center
      }

      final _resData =
          snapshot.data.documents as List<DocumentSnapshot>;

      List<Reservation> _list = _resData.map((element) {
        return Reservation.fromMapNet(element.data);
      }).toList() as List<Reservation>;

      final List<Reservation> _listApproved = _list
          .where((element) => element.approved == true)
          .toList();
      final List<Reservation> _listNotApproved = _list
          .where((element) => element.approved == false)
          .toList();

      return Consumer<Reservations>(builder: (ctx, res, child) {
        res.setReservations(_listApproved);

        final String _occupation =
            res.validatePickedDateAllReservations(
                    DateTime.now())
                ? "Slobodno"
                : "Zauzeto";
        return TabBarView(
          children: [
            SingleChildScrollView(
              child: StatisticWidget(_apartman, _occupation),
            ), // SingleChildScrollView
```

Kod 4.1. *StreamBuilder*

Podaci su, u JSON obliku (imena podataka kojima su dodjeljene vrijednosti određenog tipa). Potrebno je to pretvoriti u model podatka koji je definiran unutar aplikacije, potrebno je omogućiti pretvorbu nazad u JSON oblik ako je potrebno slati nešto prema bazi. Radi toga su unutar samih modela podataka napisane metode za pretvorbu iz JSON podatka u model (*fromMap*) i iz modela u JSON (*toMap*), kao što je vidljivo na kodu 4.2. Nakon pretvorbe podataka, potrebno ih je proslijediti dalje unutar aplikacije gdje su potrebni, a *StreamBuilder* će pri promjeni podatka obnoviti trenutnu presliku podataka te će podaci biti obnovljeni.

```dart
static Reservation fromMap(Map<String, dynamic> map) {
  Reservation _newReservation = new Reservation(
    apartmanId: map['apartmanId'],
    reservationId: map['reservationId'],
    end: DateTime.parse(map['end']),
    start: DateTime.parse(map['start']),
    approved: map['approved'] == 1,
    userID: map['userID'],
  );
  if (map['customIme'] != null) {
    _newReservation.customIme = map['customIme'];
  }
  return _newReservation;
}

Map<String, dynamic> toMap() {
  return {
    "apartmanId": this.apartmanId,
    "reservationId": this.reservationId,
    "userID": this.userID,
    "start": this.start.toIso8601String(),
    "end": this.end.toIso8601String(),
    "customIme": this.customIme,
    "approved": this.approved,
  };
}
```

Kod 4.2. Primjer funkcija pretvorbi model/JSON

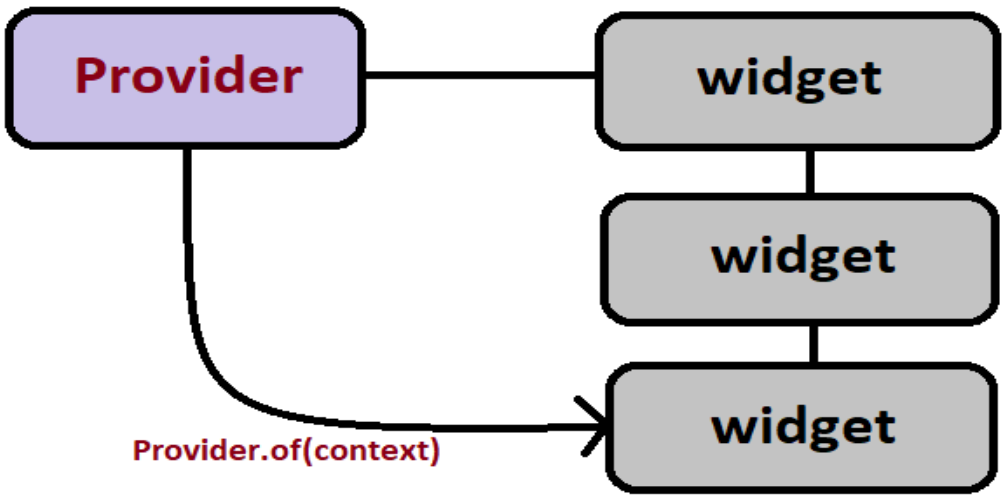
4.2. Kontrola stanja kroz aplikaciju

Aplikacije se u Flutteru konstruiraju pomoću widgeta, a oni se slažu kompozicijom te se na kraju dobije kao završni proizvod stablo widgeta. Broj widgeta može lako porasti u većim aplikacijama i ako se želi proslijediti određeni podatak kroz aplikaciju mora se provući kroz konstruktore pojedinih widgeta, što naravno nije cilj jer nije efikasno, osim kod slučajeva gdje to ima smisla. U slučaju predavanja podataka samo kroz konstruktore widgeta, određeni widget samo prosljeđuju podatak niz hierarhiju stabla. Time widgeti rade s podacima za koje ih nije briga samo zato što dijete widgeta ga koristi. Osim što je naporno za proizvođača aplikacije, ako se taj podatak promjeni, jer se proslijeđuje kroz više widgeta nepotrebno, svi widget kroz koje prolazi bi se morali ponovo učitati na ekranu, jer bi se ponovo pozvala *build* metoda, što je jako loše. Kako bi se izbjeglo nepotrebno pozivanje *build* metode potrebno je koristiti bolji način kontrole stanja widgeta. Stanje widgeta se odnosi na podatke koji utječu na korisničko sučelje i mogu se promijeniti tokom vremena. Stanje ima usku vezu s prikazom korisničkog

sučelja. Večinom ako se nešto promjeni, ta promjena je reflektirana na korisničkom sučelju. Na primjer, stanje u ovoj aplikaciji je podatak o tome je li rezervacija odobrena ili nije, s obzirom na to prikazana je u različitim prikazima ekrana i gumb „ZAHTIJEVI" možda promijeni svoj izgled. Stanje gledano izvana može se podijeliti na dva dijela:

- Aplikacijsko stanje
- Stanje widgeta (lokalno)

Aplikacijsko stanje se odnosi na podatke koji utječu na čitavu aplikaciju ili veliki dio nje. Primjer stanja koje utječe na čitavu aplikaciju je lista rezervacija, u kojoj se spremaju podaci o rezervacijama s Firestore baze podataka. Rezervacije se koriste u velikom dijelu aplikacije i potrebne su na više mjesta, negdje kompletne rezervacije, kao kod prikaza liste svih rezervacija, negdje samo dio njihove informacije, kao pri određivanju zauzetih datuma. Stanje widgeta ili lokalno stanje predstavlja stanje koje se koristi u pojedinom widgetu i utječe samo na njega, ne na ostale widgete. Primjer lokalnog stanja unutar aplikacije je podatak o tome treba li se prikazati *CirculProgressIndicator* ili lista apartmana, ovisno o tome jesu li svi potrebni podaci učitani, taj podatak je potreban samo jednom widgetu, ne cijeloj aplikaciji. Kako bi se riješio problem kontrole stanja kroz aplikaciju koristi se *Provider* paket i obrazac, prikazan na slici 4.3. Prije korištenja potrebno je unutar *pubspec.yaml* datoteke unutar aplikacije dodati pod ovisnostima *Provider* paket. Ideja iza ovog obrasca je da postoji globalno mjesto, spremnik podataka, iz kojeg widget mogu dohvatiti podatke i biti obaviješteni o promjeni podataka.



Slika 4.3. Provider obrazac

Kako bi naznačili određeni widget da treba moći slati obavijeti o promjeni podataka, potrebno je dodati *mixin* koji se zove *ChangeNotifier*, prikazan na kodu 4.3. *Mixin* je slično naslijeđivanju klasa, samo što je bitna razlika to što klasa koja koristi *mixin* ne postaje i sama tog tipa, za razliku od naslijeđivanja. U Dart programskom jeziku je dopušteno imati više *mixin* klasa, dok je dopušteno samo jedno direktno naslijeđivanje, implementiranje.

```
class Apartmani with ChangeNotifier {
  List<Apartman> _apartmani = [];

  Future<void> fetchApartmans() async {
    final _result = await DbProvider.db.getApartmans();
    _apartmani =
        _result.map((e) => Apartman.fromMap(e)).toList() as List<Apartman>;
  }

  Future<void> storeApartmans(List<Apartman> incomingApartmans) async {···

  Apartman getApartmanById(String id) {
    return _apartmani.firstWhere((element) => element.id == id,
        orElse: () => null);
  }

  Future<void> chargeApartman(String id, double price) async {
    int index = _apartmani.indexWhere((element) => element.id == id);

    if (index != -1) {
      var el = _apartmani.elementAt(index);

      el.totalIncome += price;
      await DbProvider.db.chargeApartman(el.id, el.totalIncome);

      notifyListeners();
      return;
    }
  }

  double getTotalIncome(String id) {
    return _apartmani.firstWhere((element) => element.id == id).totalIncome;
  }
}
```

Kod 4.3. *ChangeNotifier* mixin

ChangeNotifier mixin naznačuje da ta klasa koja sadrži taj *mixin* može obaviještavati druge klase promjeni podataka unutar nje. Kako bi im poslala obavijest o promjeni podataka, koristi se metoda koja dolazi s *ChangeNotifier* klasom koja se zove *notifyListeners*. Pozivanjem

metode *notifyListeners,* klase koje slušaju promjene su ponovo učitane, to jest ponovo im se pokreće *build* funkcija. Widget koji koristi *ChangeNotifier* je potrebno vezati za određeni widget unutar stabla te će svi widget ispod njega moći slušati promjene podataka unutar klase koja sadrži *ChangeNotifier mixin*. Widget za kojeg vežemo spremnik podataka ne mora biti korijenski widget, u ovoj aplikaciji je, ali ne mora biti. Ako widget želi slušati podatke, može to učiniti stvaranjem *Listenera* pomoću *of(context)* metode, zajedno s *InheritedWidgetom* kojeg *Provider* paket koristi kako bi uspostavio vezu između widgeta. Ako se podaci promjene, *build()* metoda widgeta koji sluša podatke se ponovo pokreće, time se eliminira ponovo pokretanje ostalih widgeta unutar lanca. Također, ako je potrebno samo dohvatiti određeni podatak i widgeta ne zanima ponovo pokretanje pri promjeni podataka, može se postaviti atribut *listen* na *false* pri uspostavljanju *Listenera*, vidljivo na kodu 4.4.

```
final _reservationsProvider =
    Provider.of<Reservations>(context, listen: false);
```

Kod 4.4. Uspostavljanje *Listenera*

Naravno, možemo imati više više spremnika podataka koje možemo vezati za različite widgete i uspostavljati *listener*e gdje su potrebni. Osim uspostavljanja *listener*a, može se koristiti widget koji dolazi s *Provider* paketom koji se zove *Consumer*. Kao što je prikazano na kodu 4.5., *Consumer* je widget koji konstruira stablo pomoću *builder* metode, koja kroz sebe daje:

- trenutni *context* u stablu,
- vezu na spremink podataka koji je vezan za određeni widget iznad u hijerarhiji
- *child* widet kojeg je moguće postaviti pomoću *child* parametra

Ovaj widget, za razliku od uspostavljanja *listener*a, uvijek sluša promjene podataka i time uvijek pri promjenama ponovo pokreće widgete unutar sebe. Prednosti ovog widgeta su to što ga je moguće staviti unutar *build* metode nekog widgeta te se neće ponovo pokretati wiget unutar kojeg se nalazi, već samo ono što *Consumer* obuhvaća. Osim toga, ako se postavi *child* parametar na određeni widget, on se neće ponovo pokretati. Svrha toga je da child widget, iako korišten unutar *Consumer* widgeta, neće se ponovo pokretati pri promjeni podataka, dok ostali dijelovi hoće. *Consumer* widget ima generički konstruktor, stoga je potrebno navesti s kojim podacima on treba raditi, točnije s kojim spremnikom podataka treba raditi.

```
return Consumer<Apartmani>(
  builder: (context, apartmani, child) {
    return Padding(
      padding: EdgeInsets.symmetric(horizontal: 15, vertical: 20),
      child: Container(
        alignment: Alignment.center,
        child: Column(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          children: <Widget>[
            child,
            Column(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: <Widget>[
                RowCell( // RowCell
                RowCell(
                  style: style,
                  leading: "Prosječni mjesečni:",
                  text: (apartmani.getTotalIncome(apartman.id) /
                      DateTime.now().month)
                  .toStringAsFixed(2)), // RowCell
                RowCell( // RowCell
              ], // <Widget>[]
            ), // Column
          ], // <Widget>[]
        ), // Column
      ), // Container
    ); // Padding
  },
  child: Row(
    children: <Widget>[
      Expanded(
        child: Container(
          padding: EdgeInsets.all(24),
          margin: EdgeInsets.symmetric(vertical: 10),
          alignment: Alignment.center,
          decoration: BoxDecoration( // BoxDecoration
          child: Text(
            _occupation,
            style: theme.textTheme.headline4
                .copyWith(color: Color.fromRGBO(255, 255, 225, 1)),
          ), // Text
        ), // Container
```

Kod 4.5. Consumer Widget

4.3. Komuniciranje s lokalnom bazom podataka

Osim Firestore baze podataka, koju koristi internetska stranica i aplikacija, aplikacija koristi dodatno lokalnu bazu podataka napravljenu u SQLite. Baza podataka je olakšana pomoću ugradnje *sqflite* biblioteke, koja olakšava rad fluttera s SQLite bazom podataka. Ona podržava transakcije, nudi pomoćne funkcije kod slanja zahtjeva bazi podataka te sve operacije s bazom izvršava na pozadinskoj niti na Androidu. Za korištenje paketa potrebno je u *pubspec.yaml* datoteci dodati sqflite paket pod ovisnostima, kao i bilo koji drugi paket. Unutar aplikacije rad s bazom podataka odvojen je u posebnu datoteku nazvanu „database.dart", koja sadrži sve operacije s bazom, tako da budu odvojene od aplikacijskog koda. Unutar „database.dart" datoteke nalazi se klasa *DbProvider* u kojoj se nalaze sve funkcije potrebne za komuniciranje s bazom podataka. Prije svega, ova klasa koristit obrazac stvaranja koji se zove *Singleton*, time će kroz cijelu aplikaciju postojati samo jedna instanca s kojom se komunicira. Klasa sadrži privatni kostruktor te kao parametar instancu klase *DbProvider* koja je inicijalizirana preko privatnog konstruktora i insancu *Database* klase koja dolazi od *sqflite* paketa. Instanca *DbProvider* klase služi da dohvatimo operacije, metode, koje se nalaze unutar klase te je ona statička pošto karakterizira samu klasu i potrebno joj je pristupiti preko imena klase. Pomoću instance *Database,* klase obavljaju se operacije s bazom podataka, stoga se želi osigurati da postoji samo jedna instance te klase, zato je potrebno napraviti *getter,* metodu za dohvaćanje, za tu instancu. Unutar metode za dohaćanje instance *Database,* navedeno je ako instanca postoji, vraća se postojeća, u suprotnom se poziva metoda za inizijaliziranje instance *initDb*. Ta metoda koristi metode dane iz paketa *sqflite* koje olakšavaju rad s lokalnom bazom podataka, pomoću njih na željenoj putanji izvršava SQL komande za stvaranje baze podataka s odgovarajućim tablicama. Treba uzeti u obzir da izvršavanje bilo kakih operacija se izršava asinkronim kodom, a ne sinkronim. Takav kod se izvršava paralelno s ostalim kodom, pošto bilo bi vrlo lose da čitav program treba čekati na izvršavanje tog koda. Dart programski jezik sadrži tip podatka koji se odnosi na asinkroni način pisanja koda, *Future* tip podatka koji je prikazan na kodu 4.6. *Future* tip podatka je baš ono što i znači, neki podatak koji se dobije u budućnosti. Taj podatak može biti bilo što, stoga je *Future* generičan i potrebno je navesti koji tip podatka koji će vratiti kada kod završi s izvršavanjem. Postoje dva načina pisanja asinkronog koda u Dart programskom jeziku, koristeći ugrađene funkcije i koristeći posebnu sintaksu pisanja. Prvi način uključuje dodavanje *then* funkcije na widget ili funkciju koja vraća *Future*. *Then* funkcija nam daje parametar koji je vraćen nakon završetka izvršavanja asinkronog koda i moguće je napisati dio koda koji će se izvršiti nakon završetka izvršavanja asinkronog koda.

U izrađenoj aplikaciji korišten je drugi način pisanja koda, jer je s programerske strane puno pregledniji. Korišteni način uključuje korištenje ključne riječi *await* prije radnje koja vraća *Future* te sve nakon linije s tom ključnom riječi će se izvršiti nakon završetka izvršavanja asinkronog koda. Također, pri korištenju ključne riječi *await*, potrebno je navesti da je funkcija u kojoj se ona koristi asinkrona s ključnom riječi *async*. U aplikaciji *initDb* funkcija vraća otvorenu bazu podataka, u povratni tip navodi se *Future* tip podatka koji će nakon izvršavanja vratiti *Database* tip podatka.

```
class DbProvider {
  DbProvider._();
  static final DbProvider db = DbProvider._();
  static Database _database;

  Future<Database> get database async {
    if (_database != null)
      return _database;
    else {
      _database = await _initDb();
      return _database;
    }
  }

  Future<Database> _initDb() async {
    return await openDatabase(
      join(await getDatabasesPath(), "apartmaniDatabase"),
      onCreate: (db, version) async {
        await db.execute('CREATE TABLE apartmani(id TEXT PRIMARY KEY,totalIncome DOUBLE)');
      },
      version: 1,
    );
  }
```

Kod 4.6. Asinkrono izvršavanje

U svrhu dohvaćanja i izmjene podataka iz baze podataka, napisane su funkcije koje služe kao sučelje pomoću kojega se komunicira s bazom podataka. Unutar aplikacije stvorena je tablica „*apartmani*" koja ima redove *id*, koji je ujedno i primarni ključ, tipa TEXT i *totalIncome* tipa DOUBLE. Tablica nije velika, zato što je korištena za spremanje podataka o prihodu pojedinog apartmana. Iako bi bilo lakše jednostavno ubaciti tu informaciju u bazu na serveru, informacija o prihodima nije potrebna nikome osim vlasniku i vlasnikovom mobilnom uređaju na kojem se aplikacija nalazi, ne treba je izlagati drugim uređajima. Pri komunikaciji s bazom podataka koristi se JSON oblik podatka. Radi toga korišten je *Map* tip podatka unutar kojega se spremaju odgovori i u tom se obliku šalju naredbe bazi. To je također generički tip podatka te je potrebno navesti koji se podaci nalaze unutra. *Map* radi tako da preslika određene vrijednosti na ključeve,

te vrijednosti mogu biti bilo što, a ključevi su najčešće *String* tip podatka. Unutar klase *DbProvider* napravljene su funkcije za dohvaćanje, umetanje, osvježavanje i brisanje podataka. Kao što se vidi na kodu 4.7., konkretne operacije s lokalnom bazom podataka napravljene su preko instance *Database* klase. Operacije se mogu napisati pomoću pomoćnih funkcija (*query*, *update*, *insert*, *delete*) ili pomoću čistog SQL koda (*rawQuery*, *rawUpdate*, *rawInser*, *rawDelete*).

```
Future<Map<String, dynamic>> getApartmanById(String id) async {
  final db = await database;
  final apartman =
      await db.query('apartmani', where: 'id=?', whereArgs: [id]);
  return apartman.isNotEmpty ? apartman.first : null;
}

Future<List<Map<String, dynamic>>> getApartmans() async {
  final db = await database;

  return await db.query('apartmani');
}

Future<void> deleteApartmans() async {
  final db = await database;
  return await db.delete('apartmani');
}

Future<void> chargeApartman(String id, double totalIncome) async {
  final db = await database;
  await db.update('apartmani', {'totalIncome': totalIncome},
      where: 'id=?', whereArgs: [id]);
}

Future<void> insertApartman(Map<String, dynamic> map) async {
  final res = await getApartmanById(map['id']);

  if (res != null) {
    return;
  }
  final db = await database;

  return await db.rawInsert('INSERT INTO apartmani(id,totalIncome)VALUES(?,?)',
  [map['id'],map['totalIncome'],]);
}
```

Kod 4.7. Operacije s bazom podataka

Unutar aplikacije korišene su pomoćne funkcije i pisan je čisti SQL kod, princip je isti i nema razlike u efikasnosti, osim ako se loše napiše čisti SQL kod. Unutar funkcije *insertApartman* vidi se primjer korištenja asinkronog koda, čeka se rezultat o tome postoji li već taki apartman unutar baze te s obzirom povratnu informaciju napravi se novi redak u tablici ili se prekida izvršavanje koda. Kako se ne bi konstantno zvale funkcije koje su napravljene unutar ove klase, koristi se *Provider* obrazac za raspodijelu podataka o apartmanima unutar aplikacije, kao što je prikazano na kodu 4.3. Klasa *Apartmani* je spremnik podataka koji sadrži listu apartmana dohvaćenu iz baze podataka, funkcije koje pozivaju metode klase *DbProvider* i funkciju za dohvaćanje totalnog prihoda apartmana *getTotalncome*. Time se postiže što manji rad sa samom bazom podataka i povećava efikasnost, jer nije potrebno konstantno dohvaćati podatke iz baze podataka već iz liste unutar klase *Apartmani*. Nakon uspostavljanja ovakve veze, dalje je u kodu moguća komunikacija pridržavajući se *Provider* obrasca. Time se odvaja sav rad s bazom podataka od glavnog koda za aplikaciju te se spremnik podataka korišten u *Provider* obrascu koristi kao i nekakav *Proxy* u komuniciranju s lokalnom bazom.

5. ZAKLJUČAK

Zadatak završnog rada je prikazati nove tehnologije korištene pri izradi aplikacija i način izvođenja cijelokupnog korisničkog iskustva. Potrebno je u objasniti platformu korištenu za izradu aplikacije te uvidjeti prednosti korištenja novih tehnologija u kombinaciji s tim platformama. Bitno je predočiti način korištenja novih tehnologija pri izradi ovakve aplikacije kroz dobre primjere upotrebe. Potrebno je navesti dobre prakse pisanja ovakvih aplikacija i načine pristupa pri pisanju.

Na temelju rezultata i dosad provedenih usporedbi preformansi pri korištenju ovakvih tehnologija te usporedbe platformi Android i iOS [5], može se zaključiti da Flutter u kombinaciji s Dart programskim jezikom pruža vrlo dobre preformanse pri izradi aplikacija te se brine o puno toga za samog programera. Također Flutter uzima u obzir najvažniju stavku čitave aplikacije, a to je korisničko sučelje. Flutter omogućava lakši pristup pri sastavljanju i samom dizajnu korisničkog sučelja, naspram drugih tehnologija za razvoj sličnih aplikacija. Stoga, Flutter zajedno s Dart-om se pokazao kao odličan kandidat i moćan konkurent u svijetu izrade mobilnih aplikacija.

LITERATURA

[1] Aplikacije definicija: Svi znamo što je, ali koja je točna definicija aplikacija? , https://www.rtl.hr/zivotistil/tehnologija/3546525/aplikacije-definicija-svi-znamo-sto-je-ali-koja-je-tocna-definicija-aplikacija/ , kolovoz 2019.

[2] Reverzni inženjering Android aplikacija, CERT.hr, https://www.cert.hr/wp-content/uploads/2020/01/Reverzni_inzenjering_Android_aplikacija.pdf , sječanj 2020.

[3] What Is Android?, https://www.lifewire.com/what-is-google-android-1616887 , Prosinac 2019.

[4] Costello, Sam. 5 Reasons iPhone Is More Secure Than Android. Lifewire. , https://www.lifewire.com/reasons-iphone-ismore-secure-than-android-2000308 , lipanj 2019.

[5]A.Harrison,J.Sxton "Android Arhitecture", http://meseec.ce.rit.edu/551projects/fall2015/1-3.pdf

[6] A.Shibly, "Android Operating System: Architecture, Security Challenges and Solutions", https://www.researchgate.net/profile/Ahamed_Shibly/publication/299394606_Android_Operating_System_Architecture_Security_Challenges_and_Solutions/links/56f3da5e08ae81582bef1a0b/AndroidOperatingSystemArchitectureSecurityChallengesandSolutions.pdf?_sg%5B0%5D=xO_ZVzWaxGt_CIKCkC2vXACPHrkVSFIV_DqGtmWHTKdPo6fgRvOsxFeNVYyhhWBOG1Ta1wZgjg3uCCea7TGfg.IiRnNg8kcguHsT3YwJIG1ZH8v7AJ2AHnF3jQe8erV6jVpACZ8HbTKGTf6j61V7HXIW3DZvTxMgatZRmI6TnBw&_sg%5B1%5D=C0pITpSoOstKf0GpyjWFk7mSNMLy8BD_bXj37QiKg3_IF719wPeUWJcJyStphjixh1pbqLshqBMsXOmc3LYthAy7bAU2hkYAd7XEVeIm1u.IiRnNg8kcguHsT3YwJIG1ZH8v7AJ2AHnF3jQe8erV6jVpACZ8HbTKGTf6j61 V7HXIW3DZv-TxMgatZRmI6TnBw&_iepl=, ožujak 2016.

[7] Android vs ios - razlika i usporedba – 2020 , https://hr.betweenmates.com/android-vs-ios , 2020.

[8] Fluter tehnical overview , https://flutter.dev/docs/resources/technical-overview , 2020.

[9] Widget catalog, https://flutter.dev/docs/development/ui/widgets , 2020.

[10] Flutter - Hot reload , https://flutter.dev/docs/development/tools/hot-reload , 2020.

[11] Firebase, Google, https://firebase.google.com/docs/firestore, kolovoz 2020.

SAŽETAK

Prvobitno, odabran je problem kojeg će aplikacija riješiti. Zatim, pogledom na dostupne platforme odlučeno je napraviti aplikaciju na Android platformi radi njene dostupnosti i raširenosti. Pregledom dostupnih tehnologije pri izradi mobilnih aplikacija odlučeno je uzeti najnoviju tehnologiju koja je dostupna, radi usporedbe s dosadašnjim i iskorištenja svih pogodnosti koje nudi pri izradi. Aplikacija je izrađena u suradnji sa web stranicom s kojom dijeli podatke preko servera. Pri izradi aplikacije korišteno je lijepo pisanje koda te je struktura same aplikacije i rukovanje stanjem kroz aplikaciju napravljeno u cilju poboljšanja preformansi. Provedena je usporeba preformansi, procesa pisanja aplikacije, izgleda te korisničkog iskustva pri korištenju. Donesen je zaključak da ova tehnologija pruža iznimno dobre preformanse uz održavanje dobrog korisničkog iskustva te da ima potencijal da bude ozbiljan konkurent.

Ključne riječi: korisničko iskustvo, preformanse, najnovija tehnologija

ABSTRACT

Android application for apartment booking

Firstly, the problem that this application will solve was choosen. Then, looking at the available platforms for creating such an application, it was decided to go with Android platform, because of its availability and wide spreaded usage. By looking at the available tehnologies for creating such an application it was decided to go with the latest tehnology available, because of what it has to offer and for comparing it with other tehnologies. The application was made, and it cooperates with a web page which shares its data trough a database on a server. In writing the code for the application, good practices of programming were used, and state management was handled nicely with the goal of boosting performance. Comparison was made regarding performance, process of writing the application, looks of it and user experience while using it. It has been determined that this tehnology gives good performance and very nice user experience when done correctly and that it has potential to be a serious competator.

Keywords: user experience, performance, latest tehnology

ŽIVOTOPIS

Ivan Štajcer rođen je 26. Prosinca 1998. Godine u Zagrebu. Osnovnu školu je upisao i završio u Slavonskom Brodu. Nakon završetka osnovne škole upisuje srednju Tehničku školu u Slavonskom Brodu, smjer elektrotehničar, koju završava 2017. godine. Tokom iste godine upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.