RAZVOJ SUSTAVA ZA PREGLED NOVOGRADNJE U HRVATSKOJ

Janeš, Goran

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

RAZVOJ SUSTAVA ZA PREGLED NOVOGRADNJE U HRVATSKOJ

Goran Janeš

Zagreb, veljača 2023.

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada“.

U Zagrebu, 11.02.2023.

Predgovor

Ovom prilikom se zahvaljujem profesorima i asistentima Visokog učilišta Algebra na prenesenom znanju za uspješan završetak studija, mom mentoru Danielu Beleu na pomoći, savjetima i uputama prilikom odabira teme i izradi ovog rada te djevojci Nataši na razumijevanju i podršci koju mi je pružala tijekom čitavog studija.

Sažetak

U ovom završnom radu predstavljeno je programsko rješenje koje korisnicima pomaže u potrazi za novom nekretninom. Različita mjesta na kojima je moguće pretraživati i pregledavati projekte novogradnje kupcima otežavaju prikupljanje svih relevantnih informacija koje bi im mogle pomoći prilikom izbora prave nekretnine, a investitorima kompliciraju objavljivanje i ažuriranje podataka o projektima.

Programsko rješenje zamišljeno je i implementirano kao sustav koji se sastoji od mobilne Android aplikacije, poslužiteljskog web servisa i dvije baze podataka. Prilikom izrade komponenti sustava korištene su provjerene prakse i oblikovni obrasci za izradu modularnih i lako proširivih programskih rješenja.

Ključne riječi: projekti novogradnje, nekretnine, mobilna aplikacija, REST servis, sustav.

Abstract

This final paper presents a new software solution that helps users in their search for a new property. All the currently available places where it is possible to search and view properties make it difficult for buyers to find all the relevant information that could help them choose their ideal property and make publishing or updating data about the projects for investors even more complex.

This software solution was conceived and implemented as a system consisting of a mobile Android application, a web service, and two databases. While developing the system components, proven practices and appropriate design patterns were used to create a modular and easily upgradable software solution.

Keywords: new housing developments, real estate, mobile application, REST service, system.

Sadržaj

1. Uvod

Potraga za novom nekretninom koja bi zadovoljila želje i potrebe kupaca može predstavljati dugotrajnu i zahtjevnu zadaću. Potrebno je odrediti lokaciju, veličinu stana i raspored, karakteristike stana i zgrade u kojoj se nalazi, promisliti o sadržajima koji bi se trebali nalaziti u blizini te ono najvažnije – budžet. Kako bi potraga završila pronalaskom idealne nekretnine, potrebno je pretražiti sva mjesta na kojima je moguće pronaći projekte novogradnje. Kupci informacije o projektima novogradnje mogu dobiti direktno od investitora, putem različitih agencija za posredovanje pri kupovini nekretnina, pretraživanjem web oglasnika i putem mobilnih aplikacija.

Cilj ovog rada je predstavljanje jedinstvenog mjesta koje bi kupcima omogućilo pregled svih dostupnih projekata novogradnje u Republici Hrvatskoj, a investitorima pružilo mogućnost predstavljanja svojih projekata najširem mogućem broju korisnika. Ovakav skupni prikaz podataka pružao bi sve potrebne informacije kupcima na jednom mjestu te samim time bi im pojednostavio i olakšao potragu za novim nekretninama.

Rad se sastoji od pet poglavlja. U ovom uvodnom dijelu opisuje se tema i struktura rada. Problemi s kojima se susreću kupci prilikom potrage za novom nekretninom opisuju se u drugom poglavlju. Predstavljena su neka postojeća rješenja, njihove prednosti i nedostaci te se predlaže novo programsko rješenja u obliku sustava za pregled novogradnje.

U trećem poglavlju opisana je arhitektura sustava i komponenti, dok četvrto poglavlje donosi detalje o načinima na koji je sustav implementiran i pregled tehnoloških rješenja koja su korištena prilikom implementacije.

U završnom poglavlju predstavljena je analiza uspješnosti programskog rješenja. Analizom se utvrđuje u kojoj mjeri sustav zadovoljava početne korisničke zahtjeve, opisani su načini na koje je rješenje testirano te moguće nadogradnje i poboljšanja.

2. Problem preglednosti projekata novogradnje

Tržište novih nekretnina na teritoriju Republike Hrvatske u posljednjih nekoliko godina u stalnom je porastu. Neki od razloga za ovakav rast potražnje za nekretninama su: relativno stabilna ekonomska situacija i rast primanja, višegodišnji rast turističkog sektora, povećano zanimanje stranih državljana za kupnjom nekretnina u obalnim dijelovima zemlje, potreba za izgradnjom novog stambenog fonda i novih stambenih jedinica u potresom pogođenim gradovima i mjestima, rekordno mali prinosi na štedne uloge te dostupnost programa subvencioniranja stambenih kredita za mlade obitelji. [1]

Pandemija koronavirusa potaknula je globalnu ekonomsku krizu koja je prouzrokovala nedostatak radne snage i građevinskih materijala na tržištu, a mala ponuda kvalitetne novogradnje ne zadovoljava sve potrebe kupaca. Dolazi do rasta cijena nekretnina na tržištu koji se nastavlja sve do danas. [2]

2.1. Opis problema pregleda projekata novogradnje u Hrvatskoj

Potraga za novom nekretninom dugotrajan je i složen proces koji donosi različite izazove i probleme. Prilikom potrage kupci trebaju pronaći idealnu lokaciju koja zadovoljava sve njihove želje i potrebe i u čijoj se blizini nalaze prikladni sadržaji koji podižu kvalitetu boravka i stanovanja. Jednom kada je lokacija određena, izazov predstavlja pronalaženje nekretnine unutar određenog cjenovnog ranga i pronalaženje pravih informacija o samoj nekretnini, investitoru koji je gradi, rokovima, rasporedu prostorija, kvaliteti gradnje, materijalima i tome slično.

Nepostojanje centraliziranog mjesta koje bi pružilo uvid u projekte novogradnje na teritoriju Republike Hrvatske predstavlja problem za mnoge kupce koji traže informacije o investiciji u nekretnine. Jedan od načina na koji kupac u potrazi za idealnom novom nekretninom može doći do traženih informacija je pretraživanje oglasnika. Oglasnici u većini slučajeva nisu isključivo namijenjeni oglašavanju nekretnina, a informacije koje pružaju o nekretninama, investitorima i sadržajima koji se nalazi u široj okolici često su nepregledne, nepotpune ili neažurirane. Mnogi oglasnici za iste projekte i nekretnine donose različite podatke i upravo

zbog svih tih razloga potraga putem oglasnika kupcima i svim drugim zainteresiranim osobama može oduzeti mnogo vremena i truda.

Osim oglasnika, kupcima u potrazi za projektima novogradnje na raspolaganju stoje mnoge specijalizirane agencije za nekretnine. Korištenje njihovih usluga uključuje dodatni financijski trošak, a ponuda im je često ograničena na projekte određenih investitora, lokacija i gradova. Zbog mnoštva agencija koje posluju na tržištu, nedostatka transparentnosti i oštre konkurencije, ponekad je teško dobiti točne informacije o realnoj vrijednosti nekretnina koje one sadrže u svojoj ponudi.

Također, kupci mogu pregledavati projekte izravno kod investitora na njihovim web stranicama. Ondje mogu dobiti najpreciznije informacije o projektima, a problem kupcima predstavlja pronalazak svih investitora koji posluju na tržištu.

Objavljivanje informacija o projektima na različitim mjestima investitorima može predstavljati veliki problem. Dostupnost informacija o projektima bez korištenja usluga promocije i marketinga ograničena je samo na korisnike koji su s njihovim uslugama već ranije upoznati, a korištenje marketinških usluga povećava troškove. Projekte objavljene na različitim mjestima teško je stalno nadopunjavati novim informacijama kad se dogodi neka promjena. Nepravodobne, nepotpune i netočne informacije mogu dovesti do gubitka povjerenja koje kupci imaju u investitore i negativno utjecati na mogućnost prodaje nekretnina.

2.2. Postojeća rješenja

Potencijalni kupci projekte novogradnje mogu pregledati na različitim mjestima. Najveći oglasnik *Njuškalo* sadrži posebnu rubriku za njihov pregled. Rezultate pretraživanja moguće je, između ostalog, filtrirati po županijama, gradovima i naseljima. Za svaki projekt prikazane su opće informacije, fotografije, cijene i opisi svih stanova od kojih se projekti sastoje, informacije o investitorima i dr. Oglasnik pruža mogućnosti slanja poruka osobama koje su oglase postavile, a lokacije projekata mogu se pogledati na karti. Na karti nisu označeni ostali sadržaji koji bi mogli utjecati na vrijednost projekta i stanova, kao što su, škole, vrtići, zdravstvene ustanove, trgovine, sportski objekti i dr.

Osim na oglasnicima, projekte novogradnje moguće je pretraživati i pregledavati na web stranci *Lankea*. Trenutna ponuda sastoji se od nekretnina koje se nalaze u gradu Zagrebu.

Korisnici mogu pregledati osnovne informacije o projektima i investitorima. Stranica ima mnoga ograničenja i nedostatke: ne podržava pregled stambenih jedinica od kojih se projekti sastoje, investitore je moguće kontaktirati isključivo putem elektronske pošte, ne sadrži kartu na kojoj se mogu pregledati lokacije projekata. Također, investitori svoje projekte ne mogu sami objavljivati na web stranici, već za objavu moraju kontaktirati vlasnika stranice. Mnoge njezine funkcionalisti su nedovršene ili nisu implementirane.

Na tržištu mobilnih aplikacija ne postoji specijalizirana aplikacija za pretraživanje i pregledavanje projekata novogradnje za teritorij Hrvatske, već postoje različite aplikacije za pretraživanje oglasa za prodaju i najam nekretnina. Primjer jedne od njih je aplikacija *Evomedoma*. Ova mađarska inačica mobilne aplikacije za hrvatsko tržište omogućuje pretraživanje oglasa određivanjem grada, površine i cijene nekretnina. Temeljem odabranih kriterija, aplikacije će pretražiti web oglasnike i prezentirati rezultate pomoću ugrađenog web preglednika. Rezultate je moguće označiti i spremiti, sortirati te dodatno filtrirati. Aplikacija će u određenim vremenskim intervalima obavijestiti korisnika o novim rezultatima koji zadovoljavaju kriterije spremljenog pretraživanja i ne može se koristiti bez izrade korisničkog računa. Aplikacija ne pruža mogućnost uz pomoć koje bi investitori mogli dodavati svoje nekretnine niti postoji karta za pregled lokacija i okolnih sadržaja.

2.3. Prijedlog novog rješenja

Zbog nepostojanja prikladnog programskog rješenja na Android mobilnoj platformi koje bi kupcima pomoglo u njihovoj potrazi za idealnom novom nekretninom, a investitore jednostavnije i brže povezalo s kupcima, predlažem novo programsko rješenju u obliku sustava za pregled novogradnje u Hrvatskoj. Cilj mog rješenja ovog problema je na jednom mjestu skupiti i predstaviti sve dostupne i buduće projekte novogradnje te sudionicima olakšati proces kupovine i prodaje nekretnina.

Uz pomoć karte korisnici bi mogli pregledati sve lokacije na kojima se nalaze stambeni projekti i svi zanimljivi objekti, ustanove i sadržaji koji bi im mogli pomoći prilikom odabira nekretnine. Činjenica je da su lokacija i svi njezini popratni sadržaji najvažniji faktor koji određuje cijenu nekretnine. Zbog toga kupci bi mogli vidjeti nalazi li se u blizini nekog projekta škola, vrtić ili neka druga obrazovna ustanova. Mogli bi se pregledati parkovi i ostale zelene površine, sportski objekti, trgovine, restorani, kafići, financijske i zdravstvene ustanove te ostali objekti i lokacije koje kupcima mogu jasnije odrediti stvarnu vrijednost

nekretnine i pomoći prilikom ugovaranja cijene. Postojeća slična programska rješenja ne sadrže mogućnost pregleda popratnih sadržaja koji se nalaze u neposrednoj blizini nekretnina već se od kupaca očekuje da su unaprijed upoznati s mikrolokacijama i naseljima koja ih zanimaju.

Kupci bi o svim dostupnim projektima mogli dobiti detaljne informacije pomoću kojih bi mogli odrediti zadovoljava li nekretnina njihove želje i potrebe. Svaki projekt sastojao bi se od općih informacija kao što su naziv projekta, lokacija i adresa, opis projekta, statusa u kojoj se fazi nalazi projekt i kada je planirani vremenski rok za useljenje. Svi stanovi detaljno bi bili opisani te bi se za svaki stan moglo saznati da li je slobodan ili ne. Uz općenite informacije o nekretnini, svi objekti sadržavali bi i fotografije. Za projekte i stanove kupci bi mogli dobiti uvid u njihovu popularnost tako što bi mogli saznati koliko su puta pregledani i koliko je osoba već za njih zainteresirano. Uvid u te podatke mogao bi im pomoći prilikom odluke o izboru nekretnina.

Mogućnost direktne komunikacije kupaca s investitorima koju bi sustav omogućio pružila bi mnoge prednosti i olakšala proces odabira idealne nekretnine. Tako bi kupci izravno od investitora mogli saznati sve dodatne informacije koje ih zanimaju i dogovoriti termin za razgledavanje. Ako dođe do promjena na projektima, investitori bi zainteresirane kupce o tome mogli brzo i jednostavno obavijestiti.

Investitori bi dobili centralizirano mjesto na kojem bi mogli objavljivati i oglašavati svoje projekte pa bi se time smanjila potreba za oglašavanjem na različitim platformama, optimizirali troškovi te povećala vidljivost i dostupnost projekata.

Pretpostavka je da bi ovakvo programsko rješenje moglo doprinijeti kvaliteti stanovanja i olakšati kupcima odabir buduće nekretnine, a investitorima ponuditi brži i jednostavniji način kako bi postali dostupniji novim klijentima.

Predstavljeno rješenje sastojalo bi se od mobilne Android aplikacije, poslužiteljskog dijela te dvije baze podataka. Način na koji je zamišljena arhitektura sustava, kao i sva tehnološka rješenja koja su potrebna da bi se on razvio te detalji implementacije svake pojedine komponente objašnjeni su u nastavku rada.

3. Arhitektura programskog rješenja

U ovom poglavlju opisana je arhitektura programskog rješenja za sustav pregleda novogradnje. Arhitektura predstavlja cjelokupnu strukturu i organizaciju nekog programskog rješenja, prikazuje elemente od kojih se programsko rješenje sastoji, njihova obilježja i međusobne odnose. Kako bi programsko rješenje zadovoljilo sve korisničke zahtjeve i bilo efikasno, proširivo i jednostavno za održavanje, potrebno je prije same implementacije dobro dizajnirati arhitekturu sustava.

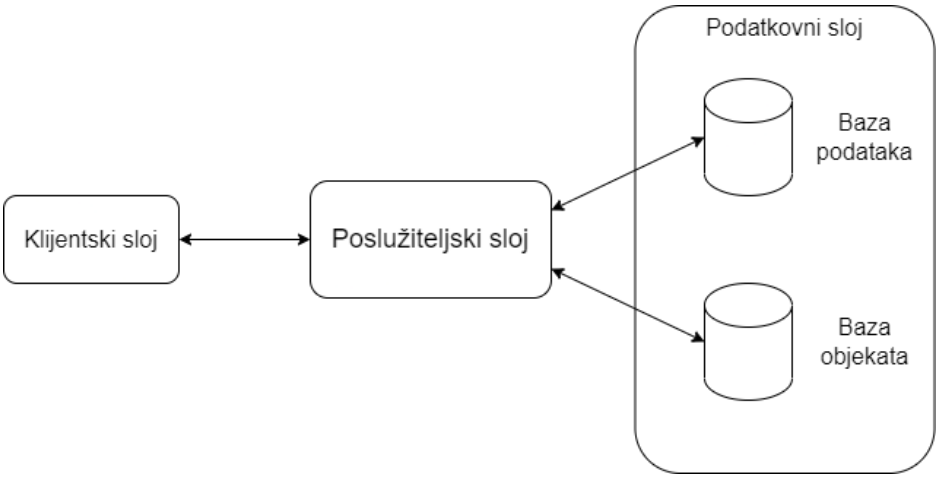
3.1. Model i opis arhitekture sustava

Postoje različiti načini ili stilovi na koje je moguće dizajnirati arhitekturu sustava, a najrašireniji zbog svoje jednostavnosti je stil koji sustav dijeli na slojeve. Broj slojeva nije ograničen, a svaki sloj, ovisno o potrebi, može se sastojati od dodatnih podslojeva.

Glavna prednost višeslojne arhitekture nalazi se u podjeli odgovornosti (engl. *separation of concerns*) među slojevima. Svaki sloj unutar arhitekture odgovoran je za određen set funkcionalnosti, a načini na koje su implementirane funkcionalnosti ostalih slojeva nisu bitne. Slojevi međusobno razmjenjuju podatke putem definiranih sučelja (engl. *interface*) tako da promjene unutar jednog sloja ne utječu na ostale slojeve. [3]

Ostale prednosti višeslojne arhitekture su njezina modularnost, fleksibilnost, jednostavnost testiranja i implementacije. Stil je dobro poznat i odražava se na organizacijske strukture razvojnih kompanija i timova. Vezu između organizacijskih struktura i načina na koji se razvija programska potpora te sustavi opisao je američki računalni znanstvenik Melvin Conway. On je primijetio da arhitekture programskih sustava nalikuju organizacijskim strukturama timova koji su ih razvili. [3]

Zbog svih ovih prednosti, sustav za pregled novogradnje sastojat će od troslojne arhitekture, a njegovi glavni slojevi su: klijentski, poslužiteljski i podatkovni sloj. Na slici 3.1 prikazan je dijagram sustava.



Slika 3.1 Dijagram sustava

3.2. Klijentski sloj

U troslojnoj arhitekturi klijentski sloj predstavlja način uz pomoć kojeg korisnik dolazi u interakciju sa sustavom. Prikaz podataka korisniku i prijenos korisničkih podataka ostalim dijelovima sustava predstavlja glavnu zadaću klijentskog sloja.

Klijentski sloj sustava za prikaz novogradnje predstavljen je u obliku Android mobilne aplikacije. Nakon registracije i prijave u sustav, korisnici pomoću nje mogu dobiti pregled dostupnih projekata novogradnje i pristupiti ostalim funkcionalnostima sustava. Detaljne informacije o korisničkim funkcionalnostima sustava objašnjene su u ostatku rada.

Mobilna aplikacija prima i šalje podatke poslužiteljskom sloju putem HTTP-a. HTTP, tj. protokol za prijenos hiperteksta predstavlja temeljni komunikacijski protokol na Webu i služi za razmjenu podataka i dokumenata između klijenata i poslužitelja. Da bi došao do podataka, klijent poslužitelju prvo šalje zahtjev (engl. *request*), a poslužitelj će nakon provjere i obrade zahtjeva, klijentu nazad poslati odgovor (engl. *response*). HTTP je jednostavan protokol za razmjenu podataka, lako je proširiv i ne čuva stanje (engl. *stateless*). Zbog njegove jednostavnosti i proširivosti zaslužan je za brz razvoj modernog Web-a. [4]

Na poslužiteljskom sloju definirane su krajnje točke (engl. *endpoints*) na koje klijent šalje zahtjeve za podacima i putem kojih povratno prima podatke i statusne poruke.

3.3. Poslužiteljski sloj

Poslužiteljski sloj odgovoran je za provođenje poslovne logike i komunikaciju s korisničkim slojem na jednoj strani i podatkovnim slojem na drugoj strani sustava. Time on predstavlja sponu između korisnika i podataka. Prilikom provođenja poslovne logike, poslužiteljski sloj izvršava korisničke zahtjeve, obrađuje korisničke podatke i vodi računa o njihovoj ispravnosti te izvršava sve ostale operacije.

Osim poslovne logike, poslužiteljski sloj zadužen je i za autentifikaciju i autorizaciju korisnika. Autentifikacijom se provjerava identitet korisnika i odlučuje hoće li mu se dopustiti pristup sustavu ili ne. Nakon uspješne autentifikacije, poslužiteljski sloj za svakog prijavljenog korisnika određuje razinu pristupa i na temelju te odluke određuje koje će mu funkcionalnosti biti dostupne.

Poslužiteljski sloj sustava za pregled novogradnje sastoji se od REST API servisa. REST predstavlja stil arhitekture i skup principa koji definira kako se pristupa resursima na nekom API servisu, dok API određuje skup protokola i pravila pomoću kojih klijenti mogu doći do resursa. [5] U četvrtom poglavlju detaljnije je opisan način na koji klijenti, da bi dohvatili resurse, komuniciraju s poslužiteljima.

Servis definira krajnje točke koje klijentskom sloju omogućuju slanje i dohvaćanje podataka iz podatkovnog sloja, a za slanje i dohvaćanje podataka koristi se komunikacijski protokol HTTP.

Prije dohvaćanje podataka, korisniku se provjerava identitet i njegova razina pristupa. Servis i pristup podacima dostupan je samo prijavljenim korisnicima uz važeći token. Klijenti uz svaki zahtjev za podacima šalju i token koji u šifriranom obliku sadrži osnovne korisničke podatke pomoću kojih servis određuje hoće li korisniku dopustiti pristup podacima ili ne.

Svi korisnici imaju mogućnosti izrade korisničkog profila putem registracije na sustav.

Servis za dohvaćanje podataka iz podatkovnog sloja koristi pomoć perzistentnog sloja (engl. *persistence layer*). Perzistentni sloj sastavni je dio servisa, a glavna mu je zadaća dohvaćanje i spremanje podataka iz podatkovnog sloja. Sve pojedinosti koje se tiču spajanja na relacijsku bazu podataka unutar podatkovnog sloja i izvršavanja upita nad bazom isključiva je odgovornost perzistentnog sloja. Zbog toga što poslovna logika servisa ne ovisi o

specifičnoj implementaciji baze podataka, pojednostavljuje se i omogućava nadogradnja i zamjena baze podataka bez većih promjena unutar same poslovne logike servisa.

3.4. Podatkovni sloj

Glavna zadaća podatkovnog sloja je prijenos podataka iz pripadajućih baza u sve ostale dijelove sustava. Podatkovni sloj pruža mogućnost za spajanje na baze podataka, dohvaćanje, spremanje, promjenu i brisanje podataka. Sloj se sastoji se od relacijske SQL baze podataka i baze objekata. SQL je jezik uz pomoć kojeg se dohvaćaju, spremaju i obrađuju podaci u relacijskim bazama podataka. Relacijska baza izabrana je kao sredstvo pohrane zbog načina na koji su organizirani povezani podaci. Podaci se pohranjuju u tablice, a spajanjem tablica opisuju se njihovi međusobni odnosi. Uz pomoć SQL-a podaci se lako mogu filtrirati i transformirati, a tehnologija na kojoj se relacijske baze podataka temelje je dobro poznata, široko rasprostranjena, izdržljiva i stabilna. [6]

Baza objekata predstavlja tip baze podataka za pohranu nestrukturiranih podataka. Svaki podatak unutar baze sprema se kao objekt s pripadajućim opisnim podacima. Svaki objekt sadrži identifikator pomoću kojeg mu se kasnije može pristupiti. Ovaj oblik pohrane podataka najčešće se koristi za pohranu slika, videa, dokumenata i svih onih objekata koji zbog svoje veličine nisu prikladni za pohranu u relacijske baze podataka. Baza objekata u sustavu za pregled novogradnje koristi se za pohranu fotografija.

4. Implementacija programskog rješenja

U poglavlju predstavlja se način na koji je implementiran sustav za pregled novogradnje u Hrvatskoj. Detaljno se opisuju tehnološka rješenja, programski okviri (engl. *frameworks*) i alati, jezici i razvojne okoline korištene prilikom implementacije sustava. Uz opis, predstavljaju se njihove prednosti, nedostaci i razlozi zbog kojih su odabrane za implementaciju sustava.

Nadalje, opisuju se koraci razvoja svake pojedine komponente sustava. Predstavljaju se slojevi od kojih se komponente sastoje, glavni oblikovni obrasci (engl. *design patterns*) korišteni prilikom njihove izrade i način na koji su komponente integrirane u sustav kao cjelinu. Na kraju poglavlja nalazi se opis osnovnih funkcionalnosti sustava.

4.1. Korištena tehnološka rješenja

Zbog velike količine različitih dostupnih tehnoloških rješenja i mnoštva faktora koje treba uzeti u obzir, izbor pravih tehnologija može predstavljati velik izazov. Za izradu sustava za pregled novogradnje glavni faktori koji su odlučivali o odabiru tehnologija su sljedeći:

- stabilnost i raširenost tehnologije
- prethodno iskustvo u radu s tehnologijom
- programska potpora otvorenog koda (engl. *open-source software*)
- dostupnost dokumentacije i kvalitetna podrška
- mogućnost integracije s ostalim tehnologijama

Bilo je iznimno važno da je tehnologija stabilna i zrela te da je već u širokoj upotrebi. Izbjegavale su se tehnologije koje su u vrlo visokoj fazi razvoja i koje nemaju dostupne stabilne inačice (engl. *versions*). Da bi se skratilo vrijeme potrebno za implementaciju sustava, izbjegle greške i problemi te smanjio rizik od nedovršenih funkcionalisti prednost su imale tehnologije koje su otprije poznate.

Kad god je to bilo moguće, koristile su se dobro dokumentirane tehnologije temeljene na otvorenom kodu. Dostupnost kvalitetne dokumentacije omogućila je na jednom mjestu pregled svih funkcionalnosti, pokazala najbolje načine na koje se tehnologija može koristiti i smanjila je vrijeme potrebno za pronalaženje rješenja za probleme nastale prilikom njene

upotrebe. Također, bilo je bitno da se izabrana tehnologija može jednostavno i brzo integrirati s ostalim tehnologijama koju su odabrane za primjenu unutar sustava.

Zbog svih ovih faktora za komponentu podatkovnog sloja u koju se spremaju podaci izabran je sustav za upravljanje relacijskom bazom podataka (engl. *relational database management system*, skraćeno RDBMS) PostgreSQL. Za bazu objekata izabran je sustav MinIO Object Storage tvrtke MinIO, Inc.

Na poslužiteljskom sloju REST API implementiran je upotrebom Spring Boot razvojnog okvira (engl. *framework*), tvrtke VMware, Inc. Za pristup relacijskoj bazi podataka koristi se modul Spring Data JPA.

JPA predstavlja specifikaciju na Java platformi koja definira kako će izgledati posebni Java objekti koje sustav treba spremiti u bazu podataka i načine na koji se spremanje u bazu izvršava. Najpoznatije implementacije JPA specifikacije su Hibernate i EclipseLink. [7]

Klijentski sloj sastoji se od Android mobilne aplikacije. Na aplikaciji za prikaz karte koristi se programska platforma Mapbox tvrtke Mapbox, Inc. Spajanje na REST API omogućeno je uz pomoć Retrofit HTTP klijenta tvrtke Square, Inc., a za formatiranje i prikazivanje fotografija koristi se biblioteka (engl. *library*) Glide autora Sama Judda.

Programski jezici u kojima je sustav razvijen su Java i Kotlin. Za razvoj REST servisa koristi se Java, a za razvoj mobilne aplikacije Kotlin.

Mobilna aplikacija razvijena je u službenom integriranom razvojnom okruženju (engl. *integrated development environment*, skraćeno IDE) Android Studio, a REST API u IntelliJ IDEA okruženju.

4.1.1. Podatkovni sloj

Podatkovni sloj sastoji se od dvije glavne komponente, i to od relacijske baze podataka na koju se pohranjuju svi povezani podaci sustava i baze objekata koja služi za pohranu fotografija.

4.1.1.1 Relacijska baza podataka

Za upravljanje relacijskom bazom podataka izabran je PostgreSQL sustav, i to inačica 14. PostgreSQL je najpopularniji slobodan i otvoren RDBMS na svijetu. Razvijen je kao dio projekta POSTGRES 1986. godine u SAD-u na sveučilištu Berkeley u saveznoj državi

Kaliforniji i otad se neprekidno razvija i koristi. PostgreSQL je iznimno moćan i nadogradiv sustav koji je moguće instalirati na većinu operativnih sustava i koji podržava velik broj programskih jezika. Korisnici sustav lako mogu nadograditi novim podatkovnim tipovima, funkcijama, operatorima i dr. [8]

Sustav je kompatibilan s mnogim verzijama SQL standarda, kao što su SQL-92, SQL:1999, SQL:2003, SQL:2006 itd. PostgreSQL podržava većinu obveznih funkcionalnosti zadnje verzije SQL standarda iz 2016. godine. Osim obveznih, sustav sadrži i dodatne funkcionalnosti kojih nema u ostalim RDBMS-ovima. [9]

Izdržljivost baza podataka može se testirati pomoću ACID karakteristika. PostgreSQL je usklađen s ACID karakteristikama koje se primjenjuju na transakcije. Četiri karakteristike predstavljene akronimom ACID su: atomarnost (engl. *atomicity*), konzistentnost (engl. *consistency*), izolacija (engl. *isolation*) i izdržljivost (engl. *durability*). [10]

Atomarnost zahtjeva da se transakcija obavi u cijelosti ili da se uopće ne obavi. Garantira se nedjeljivost transakcije kad se ona sastoji od više operacija. Ako se slučajno pojavila greška prilikom izvršavanja transakcije, sve prethodne promjene će se poništiti, a baza će se vratiti u prijašnje stanje. Konzistentnost omogućava da se baza uvijek nalazi u konzistentnom stanju, tj. osigurava se integritet njezinih podataka. Isto kao i kod atomarnosti, ako se dogodi da jedan dio transakcije nije uspješno proveden, sve promjene koje je ta transakcija napravila će se poništiti i baza će se vratiti u prethodno stanje. Izolacijom se transakcije odvajaju i osigurava se da paralelno pokrenute transakcije ne ometaju jedna drugu, a izdržljivost omogućava da se promjene koje je transakcija napravila sačuvaju i onda kada je došlo do greške u radu sustava. Nakon uspješnog oporavka, sustav će nastaviti izvršavanje transakcije na mjestu gdje je greškom prekinuta. [10]

Sustav se temelji na višeprocesnoj arhitekturi (engl. *multi-process architecture*). Koordinacijsku ulogu u sustavu ima *postmaster* proces čija je zadaća postavljanje sustava u ispravno početno stanje, pokretanje pozadinskih procesa, gašenje sustava, autentifikacija i upravljanje klijentskim konekcijama. Klijenti upite šalju pozadinskom procesu i od njega primaju rezultate. Pozadinski proces odgovoran je za sintaksnu analizu (engl. *parsing*), optimizaciju, izvršavanje upita i vraćanje rezultata. Za svaku aktivnu klijentsku konekciju, pokreće se zaseban pozadinski proces. Na sustavu se nalaze i ostali procesi koji obavljaju poslove vezane za ispravan rad PostgreSQL sustava. [11]

4.1.1.2 Baza objekata

Uz relacijsku bazu podataka, podatkovni sloj sadrži i bazu objekata. MinIO je programsko rješenje otvorenog koda za pohranjivanje objekata koje podržava sve funkcionalnosti Amazon S3 servisa, jednostavno se instalira, sadrži mnoštvo mogućnosti za dodatna proširenja i kompatibilan je sa S3 API-jem. Za potrebe testiranja i razvoja, može se pokrenuti na lokalnom računalu ili u privatnom oblaku. Upravo zbog tih razloga MinIO predstavlja najbolje tehnološko rješenje za pohranu fotografija za sustav pregleda novogradnje. [12]

4.1.2. Poslužiteljski sloj

Na poslužiteljskom sloju nalazi se REST API servis koji je implementiran pomoću Spring Boot razvojne okoline. REST API prima HTTP zahtjeve klijenata i prosljeđuje ih podatkovnom sloju za dohvat podataka. Između poslužiteljskog i podatkovnog sloja nalazi se perzistentni sloj implementiran uz pomoć Spring Data JPA modula.

4.1.2.1 REST API

Kratica REST predstavlja stil arhitekture temeljen na klijentsko-poslužiteljskom modelu za izradu servisa. Termin je definirao Roy Thomas Fielding, jedan od autora HTTP specifikacije, u doktorskoj disertaciji 2000. godine. [13]

REST predstavlja set principa pomoću kojih se pristupa resursima u mrežnoj okolini. REST nije standard, niti pravilo već donosi smjernice za efikasniju komunikaciju između mrežnih servisa. Najpoznatiji primjer REST arhitekture predstavlja WWW. Svaka informacija ili podatak koji se nalazi na mreži predstavlja resurs kojemu je moguće pristupiti. Resurs na mreži identificira se pomoću ujednačenog identifikatora resursa (engl. *Uniform Resource Identifier*, skraćeno URI), a pristupa mu se pomoću nekog komunikacijskog protokola, najčešće HTTP-a. [14]

REST definira sljedeće principe:

- ujednačenost sučelja (engl. *Uniform Interface*)
- klijentsko-poslužiteljska arhitektura (engl. *Client-Server*)
- bez stanja (engl. *Stateless*)
- pohrana u priručnu memoriju (engl. *Cacheable*)
- slojevitost sustava (engl. *Layered System*)

- kod na zahtjev (engl. *Code on Demand*) [15]

Ujednačenost sučelja najvažniji je princip REST stila arhitekture. Uz pomoć njega usluge koje servisi pružaju odvajaju se od implementacija. U slučaju REST API-a klijent ne mora znati na koji je način implementirana usluga koju pruža određeni servis, već je za dohvaćanje i upravljanje resursom najvažnije da postoji jedinstveno sučelje. Da bi se ujednačenost postigla, potrebna su četiri ograničenja: identifikacija resursa (engl. *identification of resources*), rukovanje resursima pomoću prikaza (engl. *manipulation of resources through representations*), samoopisne poruke (engl. *self-descriptive messages*) i HATEOAS. [15]

Svaki resurs mora imati identifikator u obliku URI-a. Klijent resursu pristupa uz pomoću identifikatora. Klijentu se resurs prikazuje i on njime upravlja pomoću različitih formata za prikaz podataka. Na primjeru REST API-a resurs se može prikazati u JSON ili XML standardima.

JSON i XML su dva otvorena standarda koja olakšavaju razmjenu podataka između različitih informacijskih sustava. XML se koristi za označavanje i opisivanje podataka. Podaci se opisuju pomoću oznaka (engl. *tags*) koje nisu unaprijed definirane, već ih korisnik sam mora odrediti. Proširivost i fleksibilnost koje XML podržava učinili su ga popularnim standardom za prijenos podataka. Unazad nekoliko godina, XML počinje gubiti svoju popularnost, a na njegovo mjesto dolazi novi standard – JSON. JSON je dizajniran kao jednostavan i lako čitljiv format za prijenos podataka. Dio je JavaScript jezika i za razliku od XML-a, može se koristiti bez upotrebe dodatnih biblioteka. Zbog svoje jednostavnosti i čitljivosti, počeo se koristiti izvan JavaScript okruženja i danas predstavlja najpopularniji format za prijenos podataka. [16]

Poruke koje servis razmjenjuje s klijentom moraju sadržavati dovoljno informacija da bi se one mogle obraditi. Uz poruke, API treba pružiti veze prema ostalim resursima koji se na njemu nalaze.

Klijentsko-poslužiteljskom arhitekturom odvajaju se odgovornosti i jasno je definirano sučelje koje se nalazi među njima. Odvajanjem se pojednostavljuju komponente sustava, omogućuje se naknadna proširivost, a svaka komponenta zasebno se može mijenjati i razvijati.

Da bi se zahtjev za podacima mogao provesti, u njemu moraju biti navedene sve potrebne informacije. Između zahtjeva poslužitelj ne pamti stanje, već je obveza čuvanja stanja

isključivo na klijentu. Time se povećava broj zahtjeva koje REST API može primiti, omogućava se razmještaj API-a na više različitih poslužitelja i pojednostavljuje se cjelokupni njegov dizajn.

Dopušteno je spremanje podataka u priručnu memoriju da bi se smanjio broj zahtjeva prema REST API-u. Spremanje podataka može se obavljati na klijentu ili poslužitelju s time da se prije mora navesti smiju li se podaci spremati ili ne.

Neki dijelovi sustava mogu se podijeliti u slojeve. Time se ograničavaju odgovornosti koje ti slojevi imaju, a svaki sloj komunicira samo sa slojem koje se nalazi do njega. Zbog podjele odgovornosti ukupna složenost sustava se smanjuje.

Kod na zahtjev nije obvezan princip i predstavlja mogućnost zahvaljujući kojoj je moguće proširiti funkcionalnosti klijenta slanjem nadogradnji u obliku programskog koda ili skripti.

Servis koji podržava sve REST principe nazivamo RESTful. U praksi najčešće nailazimo na REST API servise koji za komunikaciju koristi HTTP. Takvi servisi za pristup resursima koriste standardne HTTP metode kao što su: DELETE, GET, PATCH, POST i PUT. Na takvim servisima resursi se identificiraju pomoću ujednačenog lokatora resursa (engl. *Uniform Resource Locator*, skraćeno URL).

4.1.2.2 Spring Boot

Spring Boot je alat za izradu web aplikacija i mikroservisa (engl. *microservices*). Dio je popularne razvojne okoline otvorenog koda Spring koja se koristi za izradu poslovnih aplikacija. Spring je napisan u Java programskom jeziku i uz Javu, podržava i ostale JVM kompatibilne jezike kao što su Groovy i Kotlin. Nastao je kao alternativa razvoju poslovnih aplikacija na Java Enterprise Edition platformi i imao je za cilj pojednostaviti i olakšati njihov razvoj. [17]

Glavne mogućnosti koje su dovele do jednostavnijeg razvoja aplikacija su:

- inverzija kontrole (engl. *inversion of control*, skraćeno IoC)
- pojednostavljeno testiranje
- pristup podacima pomoću ORM-a
- mogućnost izrada web aplikacija i dr. [17]

ORM je kratica za objektno-relacijsko mapiranje koja predstavlja proces pomoću kojega se Java objekti spremaju u tablice unutar baza podataka i dio je JPA specifikacije.

Inverzijom kontrole Spring se brine za kreiranje objekata, njihovu konfiguraciju, uništavanje te upravljanje ovisnostima o ostalim objektima. Spring informacije o objektima dobiva iz konfiguracija, pomoću anotacija ili direktno iz koda. Jednom kad je objekt stvoren, sve ovisnosti ubacuju se u objekt, što je suprotno od principa u kojem objekt sam kreira i upravlja svojim ovisnostima. Princip inverzije kontrole maksimalno olakšava testiranje aplikacija koje su izrađene pomoću Spring razvoje okoline. Podrška za jedinično (engl. *unit*) i integracijsko testiranje (engl. *integration testing*) sastavni je dio okoline. [17]

Spring pruža jedinstveni transakcijski model za pristup podacima. Model podržava popularne API-e kao što su JDBC, JPA, Hibernate i dr. Podrška za sve te sustave glavni je razlog Spring-ove popularnost među Java razvojnim okolinama. [17]

Spring podržava različite tehnologije i razvojne okoline za izradu web aplikacija. Osnovna od njih je Spring MVC. Ona podržava *model-view-controller* paradigmu i izgrađena je na Servlet API-u koji je preuzet iz Java Enterprise Edition platforme. Mrežna komunikacija u realnom vremenu omogućena je pomoću WebSocket standarda. REST servisi jednostavno se mogu izraditi pomoću Spring Data REST tehnologije, a najnoviju podršku za izradu reaktivnih web aplikacija omogućuje Spring WebFlux. [17]

Zbog mnogih tehnologija i razvojnih okolina koje Spring podržava, izrada i konfiguracija aplikacija na Sprint platformi može biti iznimno zahtjevna i složena, a rješenje za tu složenost nalazi se u Spring Boot-u. Spring Boot nije nova razvojna okolina, već predstavlja set pristupa i alata koji mijenjanju način na koji se postavlja i pokreće Spring projekt. [18]

Spring Boot podržava automatsku konfiguraciju svih ovisnosti koje su potrebne za rad aplikacije. Prije pokretanje aplikacije, Spring Boot će potražiti sve dostupne ovisnosti i automatski ih podesiti prema konfiguracijskim postavkama. [18]

Spring se sastoji od mnoštva funkcionalnosti i može se konfigurirati za različite vrste aplikacija. Svaka od tih funkcionalnosti sa sobom povlači i dodatne pakete. Odabir ispravnih paketa i njihovih inačica korisnicima može oduzeti mnogo vremena. Što je projekt veći, to je teže upravljati svim paketima koji su potrebni za njegov pravilan rad. Spring Boot olakšava upravljanje ovisnostima tako što pakete grupira u različite početne ovisnosti, tzv. *Spring Boot Starters*. Za web aplikaciju dovoljno je izabrati web grupu paketa *spring-boot-starter-web*, a Spring Boot će se pobrinuti za odabir ispravnih pojedinačnih paketa i inačica. [18]

Aktuator (engl. *Actuator*) je alat pomoću koje možemo pratiti stanje Spring Boot aplikacija. Mogu se doznati koje sve objekte pokrenuta aplikacija koristi, na koji način je konfigurirana, trenutno stanje aplikacije, njezinih internih i vanjskih komponenti, dretvi, priručnih memorija, popis pristiglih web zahtjeva i dr. [18]

Spring Boot sadrži mnoge prednosti. Olakšava uporabu Spring razvojnog okvira, a razvoj web aplikacija i servisa čini efikasnijima i jednostavnijima. S obzirom na to da dolazi integriran s web poslužiteljem, testiranje i pokretanje web aplikacije može se odraditi i na lokalnom računalu.

4.1.2.3 Spring Data JPA

Spring Data JPA dio je većeg Spring Data projekta koji pojednostavljuje rad s baza podataka koristeći Java Persistence API. Podržane su sve popularne baze, a neke od njih su SQL, Redis, Cassandra, MongoDB itd. [19]

Spring Data JPA za implementaciju JPA-a koristi Hibernate. JPA donosi jedinstven model za pristup različitim relacijskim bazama podataka i omogućava pokretanje SQL upita na bilo kojoj bazi bez dodatnih promjena u kodu. Hibernate je najpopularnija implementacija JPA specifikacije. [20]

Pristup podacima omogućen je uz pomoć različitih sučelja, a ta sučelja ili repozitoriji implementiraju osnovne CRUD metode za izvođenje operacija nad bazama podataka, poput operacija za stvaranje, brisanje, ažuriranje ili pretraživanje podataka.

4.1.3. Klijentski sloj

Glavni dio klijentskog sloja predstavlja Android aplikacija. Mobilna aplikacija izvršava se na operativnom sustavu za mobilne uređaje Android tvrtke Google. Android operativni sustav temeljen je na Linux jezgri i primarno je dizajniran za uporabu na mobilnim uređajima i tabletima. Sustav je otvorenog koda, besplatan za upotrebu i nadogradnju i upravo zbog tih osobina, najpopularnija je mobilna platforma za razvoj aplikacija. [21] Razvija ga tvrtka Google u suradnji s programerima iz cijelog svijeta te računalnim i telekomunikacijskim tvrtkama ujedinjenim u Open Handset Alliance inicijativi. [22]

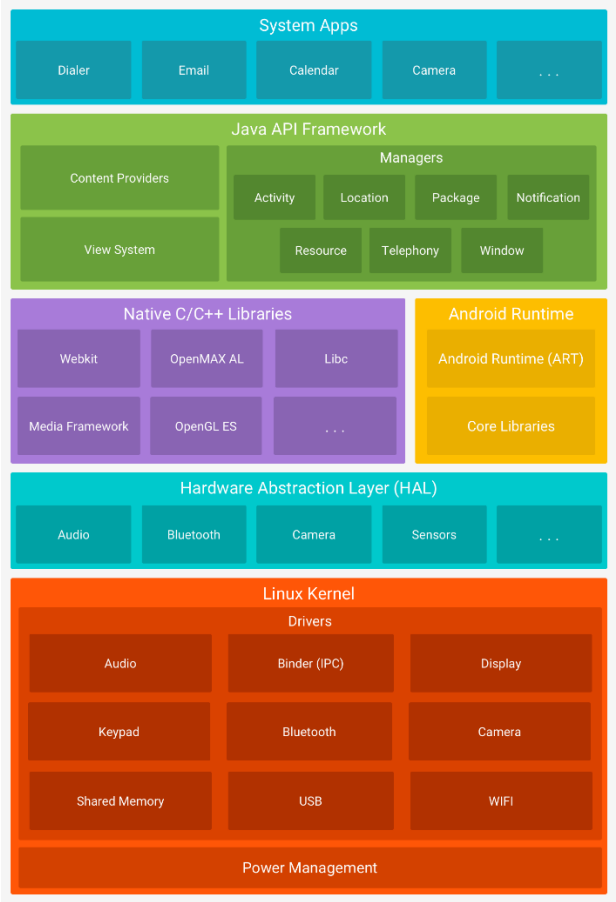
Osim samog operativnog sustava, uz Android dolazi cijeli niz programa, biblioteka, alata i dokumentacije koji čine platformu za razvoj mobilnih aplikacija. Jezici podržani za njihov

razvoj su Java i Kotlin. Android programska platforma sastoji se od niza komponenti koje kada su posložene u slojeve čine arhitekturu sustava (Slika 4.1). [23]

Prvi sloj sustava sastoji se od Linux jezgre. Jezgra je zadužena za upravljanje hardverskim komponentama, memorijom i procesima. Između jezgre i operativnog sustava nalazi se Hardware Abstraction Layer. Taj sloj sastoji se od niza biblioteka koje implementiraju sučelja za pristup različitim hardverskim komponentama mobilnog uređaja.

Android Runtime predstavlja virtualnu okolinu koja pokreće aplikacije na Android sustavu. Prilikom instalacije, Java kod aplikacije prevodi se u strojni kod za hardversku platformu na kojoj će se aplikacija izvršavati u svrhu poboljšanja performansi. Mnoge komponente Android sustava koriste biblioteke koje nisu napisane u Java programskom jeziku, a Android omogućava aplikacijama da ih koriste pomoću dodatnih API-a.

Sve funkcionalnosti koje sustav podržava dostupne su aplikacijama na korištenje pomoću bogatog seta Java API-a. Java API pomaže pri stvaranju grafičkih korisničkih sučelja, olakšava pristup podacima, dopušta upotrebu lokacijskih i telefonskih usluga itd. Na najvišem sloju nalaze se korisničke aplikacije napisane u Java ili Kotlin programskom jeziku.



Slika 4.1 Pregled slojeva Android arhitekture. Preuzeto iz [23].

Android aplikacija za pristup podacima spaja se na REST API uz pomoć Retrofit HTTP biblioteke. Retrofit je najpopularnije rješenje za pristup web API-jima na Androidu, a razvila ga je tvrtka Square, Inc. Retrofit krajnje točke REST API-a pretvara u jednostavna anotirana sučelja (engl. *annotated interfaces*) i olakšava slanje zahtjeva te zaprimanje odgovora.

Anotirana sučelja glavna su sastavnica Retrofit biblioteke uz pomoć kojih se zahtjevi šalju prema REST servisima i služe za dodatno opisivanje zahtjeva. Njima se određuju metode, lokacije na koje se zahtjevi šalju (URL) i parametri koje će zahtjevi sadržavati. Primljene odgovore Retrofit će pomoću različitih pretvarača (engl. *converters*) automatski pretvoriti u objekte. Retrofit podržava gotovo sve popularne biblioteke za pretvaranje JSON ili XML zapisa u objekte i obrnuto. [24]

Učitavanje i manipulacija fotografijama u Android aplikaciji obavlja se pomoću Glide biblioteke. Glide podržava dohvaćanje slika s mreže, njihovo prikazivanje, različite transformacije nad slikama i druge operacije. [25]

Prikazivanje karti moguće je zahvaljujući Mapbox platformi. Mapbox platforma nudi veliki izbor različitih karti, bogat API za razvoj na Android-u, IOS-u i Webu. Karte se mogu jednostavno integrirati u aplikacije, a pomoću različitih funkcionalnosti mogu se koristiti i za navigaciju, pretraživanje zanimljivosti, analizu podataka i dr. [26]

4.1.4. Programski jezici i razvojne okoline

Komponente sustava napisane su u programskim jezicima Java i Kotlin. Java je moderan objektno-orijentirani programski jezik nastao 1991. godine u tvrtki Sun Microsystems, Inc. koja je dio Oracle korporacije. Njegove glavne značajke su prenosivost (engl. *portability*), jednostavnost, stabilnost i sigurnost. Aplikacije napisane u Java programskom jeziku mogu se izvršavati bez promjene na bilo kojem računalu, što je glavni razlog zbog kojeg je Java jedan od najpopularnijih jezika na svijetu. Java predstavlja idealan izbor za izradu poslovnih aplikacija i sustava, mobilnih aplikacija na Android platformi i web aplikacija. [27]

Uz sam programski jezik, Java razvojna platforma uključuje bogat API te niz alata i programa koji olakšavaju izradu aplikacija. Java Standard Edition predstavlja glavni API platforme i koristi se za izradu svih vrsta aplikacija. Java Standard Edition sastoji se od temeljnih klasa i sučelja koje tvore Java jezik, biblioteka za rad s nizovima i kolekcijama, ulazno-izlaznim sučeljima, različitim strukturama podataka itd. API uključuje izvrsnu podršku za rad s mrežama i mrežnim tehnologijama, podršku za XML, izradu web servisa i aplikacija, rad s bazama podataka, korisničkim grafičkim sučeljima i mnoštvo drugih biblioteka. [27]

Od svojih početaka pa sve do danas, Java se neprestano nastavila razvijati. Prva verzija JDK 1.0 izašla je davne 1996. godine, a zadnja Java 18 2022. godine. [28]

JDK je kratica za Java razvojnu okolinu i predstavlja programski paket koji uključuje set alata koji su potrebni za razvoj aplikacija na Java platformi. U tom paketu dolazi Java jezični prevoditelj (engl. *compiler*) i sve standardne biblioteke. [29]

Jezik je godinama nadograđivan čitavim nizom novih funkcionalnosti. Dodana je podrška za refleksiju, novo okruženje za rad s kolekcijama, novo grafičko sučelje, podrška za regularne izraze (engl. *regular expressions*), nove mrežne protokole, anotacije, enumeracije i općenite tipove (engl. *generics*), funkcionalna sučelja (engl. functional interfaces), lambda izraze (engl. *lambda expressions*), okruženje za kolekcijske tokove (engl. *streams*). [28]

Tvrtka JetBrains 2011. godine razvila je novi programski jezik Kotlin. Novi jezik ima za cilj modernizirati Javu uvođenjem novih programskih paradigmi, poboljšati preglednost koda i sigurnost jezika te zadržati kompatibilnost s Java razvojnom platformom. Programi napisani u Kotlin jeziku mogu se pokretati u Java virtualnoj okolini i svaki Java kod jednostavno se može pretvoriti u Kotlin kod. [30]

Kotlin podržava tipove podataka bez vrijednosti (engl. *nullable types*). Osim ako se drugačije ne naznači, sve varijable moraju sadržavati vrijednosti. Time se umanjuje mogućnost za pojavljivanje grešaka prilikom pristupanja vrijednostima koje ne postoje (engl. *null pointer exceptions*). [30]

Novost koju Kotlin donosi je automatsko prepoznavanje tipova (engl. *type inference*). Ovisno o kontekstu, prevoditelj će automatski odrediti kojeg je tipa podataka bez da se on svaki put eksplicitno navodi. Kotlin u jezik uvodi paradigmu funkcijskog programiranja. Podržane su funkcije višeg reda (engl. *higher order functions*), lambda izraz i nepromjenjivost (engl. *immutability*) objekata. [30]

Komponente sustava napisane u programskom jeziku Java razvijane su u integriranom razvojnom okuženju IntelliJ IDEA tvrtke JetBrains. Najpopularnije je to razvojno okruženje za razvoj aplikacija za JVM platformu. Može se koristi za izradu poslovnih, web, desktop, mobilnih i drugih aplikacija. Sadrži integriranu podršku za mnoge razvojne okoline, kao što su Spring, Spring Boot, Jakarta EE i druge. Podržava sve popularne sustave za kontrolu inačica koda (engl. *version control system*), inteligentnu provjeru koda i dovršetak koda, podršku za restrukturiranje (engl. *refactoring*) i promjenu koda, a sve dodatne funkcionalnosti mogu se instalirati pomoću različitih dodataka (engl. *plugins*). [31]

Android aplikacija razvijala se u službenom razvoj okruženju za razvoj Android aplikacija Android Studio tvrtke Google i temelji se na IntelliJ IDEA razvojnom okruženju te uključuje specifične funkcionalnosti za izradu mobilnih aplikacija.

4.2. Razvoj komponenata

Poglavlje donosi detaljne korake razvoja svih komponenti sustava za pregled novogradnje, opise i načine na koji su iskorištene tehnologije, pregled slojeva arhitekture pojedinih komponenti i opis korištenih oblikovnih obrazaca. Opisuju se dvije baze podataka, relacijska baza i tablice od kojih se ona sastoji, njihove veze i odnosi te baza objekata na koju se

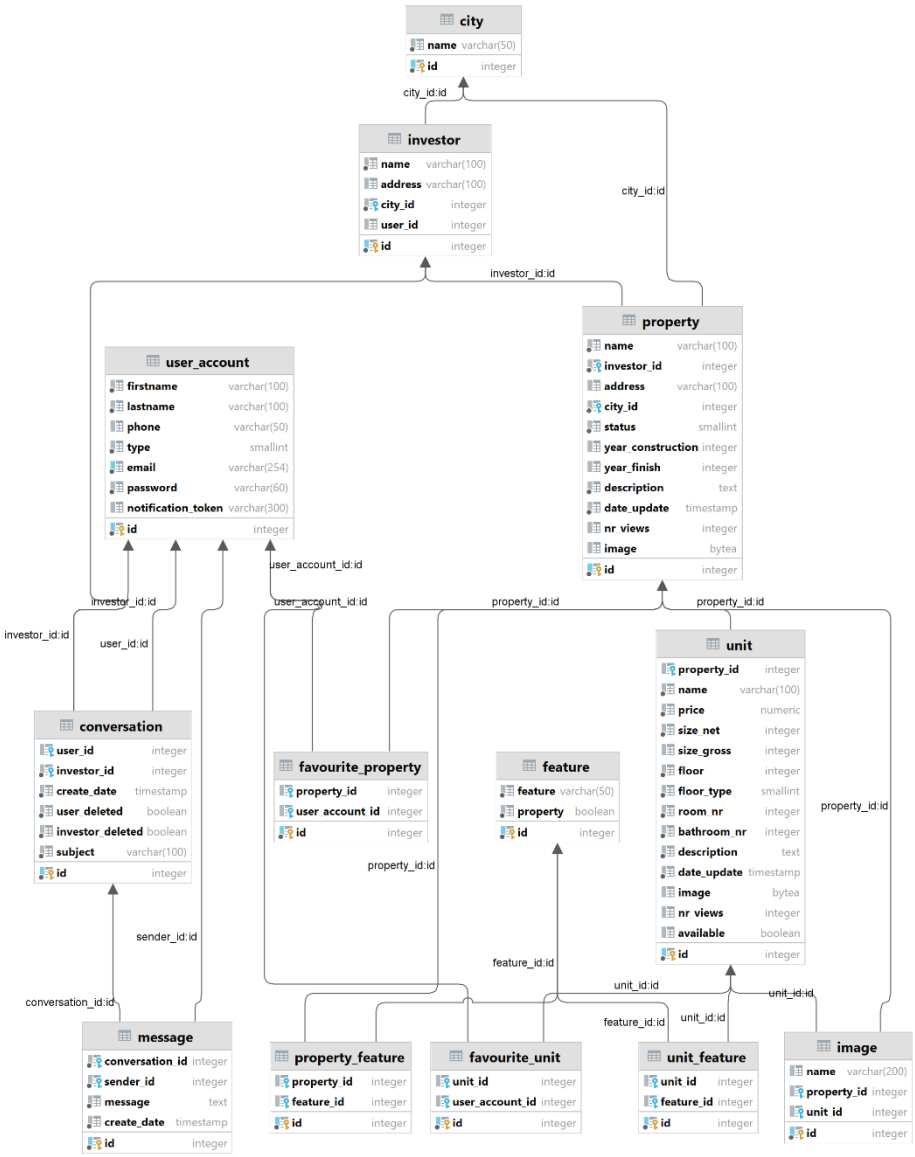
spremaju fotografije. Potom se opisuje web servis koji klijentima omogućava pristup podacima, način na koji je implementirana autentifikacija i druge funkcionalnosti. Na kraju poglavlja prikazuje se implementacija klijenta u obliku mobilne aplikacije.

4.2.1. Izrada baza podataka

Podatke potrebne za rad nekog programskog sustava potrebno je negdje pohraniti da bi im se kasnije moglo pristupiti i koristiti ih. Najbolje mjesto za pohranu strukturiranih podataka svakako je relacijska baza.

4.2.1.1 Relacijska baza podataka

Podaci sustava za pregled novogradnje spremaju se u relacijsku bazu podataka koja se temelji na PostgreSQL tehnologiji. Klijent u bilo kojem trenutku može pristupiti, dodavati, mijenjati ili brisati postojeće podatke.



Slika 4.2 Relacijski dijagram baze podataka

Baza podataka sadrži 13 tablica, a neke od njih međusobno su povezane relacijama. Slika 4.2 prikazuje grafički prikaz baze sustava na kojoj se mogu vidjeti odnosi među tablicama i elementi od kojih se tablice sastoje. Svaka tablica ima svoj naziv, sadrži primarni ključ (engl. *primary key*) koji identificira vrijednosti upisane u retke, stupce s nazivima elemenata i tipove podataka koje tablica sadržava. Neke tablice sadrže i strane ključeve (engl. *foreign key*) kojima se podaci povezuju s onima iz drugih tablica.

Glavna tablica u koju se spremaju podaci o projektima je *property*. U nju se sprema naziv projekta, adresa i grad u kojem se projekt nalazi, podaci o investitoru, opis projekta, datum izgradnje, datum završetka i ostalo. Tablica *unit* sadrži podatke o stanovima od kojih se

projekti sastoje. Svaki stan ima svoj naziv, cijenu, veličinu, tip i druge opisne podatke. Podaci o gradovima nalaze se u tablici *city*, a podaci o investitoru koji je zadužen za projekt nalaze se u tablici *investor*. Sustav razlikuje dvije vrste korisnika. Običnog korisnika ili kupca te investitora.

Tablica *property* povezana je s tablicom *unit*, a veza između njih je jedan na više (engl. *one-to-many*, skraćeno 1:N). Projekt može sadržavati više stanova, a jedan stan nalazi se samo u jednom projektu. Stan ne može postojati bez projekta, a u slučaju brisanja projekta, doći će do brisanja svih njegovih stanova. Veze između tablica *property* - *city* i *property* - *investor* također su 1:N. [6]

Značajke koje stanovi i projekti imaju nalaze se u tablici *feature.* Te značajke povezane su s projektima i stanovima pomoću posebnih tablica i čine relacije više na više (engl. *many-to-many*, skraćeno M:N). Tablica *property_feature* sadrži značajke projekata, dok tablica *unit_feature* sadrži značajke stanova. Pomoćne tablice potrebne su prilikom relacija više na više. One sadrže strane ključeve (engl. *foreign key*) i tvore veze prema glavnim tablicama koje sadrže podatke. Primjerice, tablica *property_feature* sastoji se od tri elementa, primarnog ključa, stranog ključa *property_id* koji upućuje na određeni projekt unutar tablice *property* i stranog ključa *unit_id* koji upućuje na stan unutar tablice *unit*. Time je omogućeno da neki projekt može imati više značajki i da se svaka značajka može nalaziti na više projekata. [6]

Tablica *image* sadrži podatke o fotografijama stanova i projekata i u relaciji je jedan na više s tablicama *property* i *unit*. Svaki projekt ili stan može imati više fotografija, a jedna fotografija može pripadati samo jednom projektu ili stanu. Da bi se smanjila količina podataka u SQL bazi, u tablicu *image* osim reference, sprema se samo naziv fotografije, dok se stvarna fotografija nalazi u bazi objekata koja je primjerenija za pohranu takvih vrsta podataka.

Svaki korisnik može odrediti i spremiti svoje omiljene projekte i stanove. Podaci o omiljenim projektima nalaze se u pomoćnoj tablici *favourite_property*, a podaci o stanovima u tablici *favourite_unit*. Veza između korisnika i projekata te korisnika i stanova je više na više.

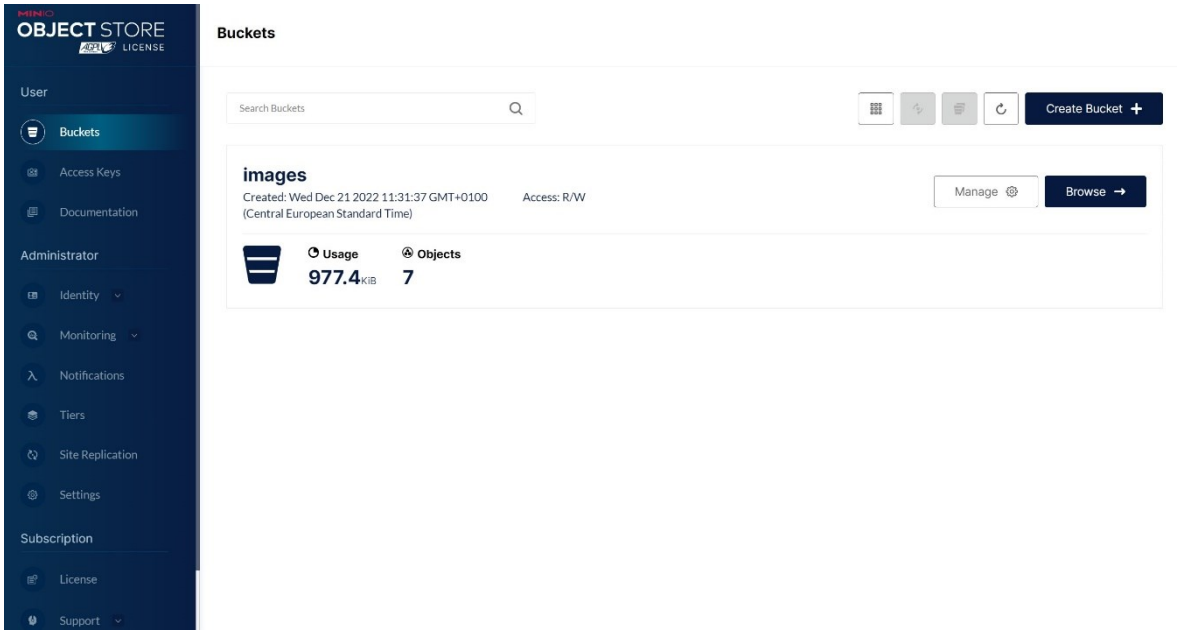
Podaci o korisnicima spremaju se u tablicu *user_account*, a osim korisničkih podataka, u tablicu je spremljena i korisnička lozinka u šifriranom obliku. Na ovakav način spremljene lozinke više nije moguće vratiti u originalni tekstualni oblik i time se jamči njihova sigurnost.

Korisnici se mogu s investitorima dopisivati i dogovarati oko stanova i projekata, a tablice pomoću kojih su te funkcionalnosti omogućene su *conversation* i *message*. Podaci o razgovorima spremaju se u tablicu *conversation*, a poruke u tablicu *message*. Razgovori su povezani s korisnikom i investitorom, a poruke s razgovorom pomoću jedan na više relacija.

4.2.1.2 Baza objekata

Spremanje velikih količina podataka u relacijske baze ne preporučuje se u praksi, već se za to koriste druga tehnološka rješenja. Jedno od njih za spremanje fotografija u sustavu novogradnje je baza objekata MinIO.

Nakon instalacije i početne konfiguracije administratorskog računa i mrežnih adresa, potrebno je stvoriti ključ za pristup servisu, odrediti korisnička prava i kreirati mjesto (engl. *bucket*) za spremanje podataka (Slika 4.3). Dohvaćanjem i spremanje podataka upravljat će se na poslužiteljskom sloju unutar REST API servisa.



Slika 4.3 Baza za pohranu objekata MinIO

4.2.2. Izrada REST API servisa

Središnje mjesto putem kojeg klijenti pristupaju podacima je REST API servis. Za njegovu implementaciju koristi se Spring Boot razvojna okolina i dodatne biblioteke. Početni Spring Boot projekt može se napraviti na više načina: primjenom komandno-linijskom sučelja, uz pomoć web aplikacije *Spring Initializr* ili unutar samog integriranog razvojnog okruženja.

Za potrebe ovog sustava početni projekt REST API-a stvoren je unutar integriranog razvojnog okruženja IntelliJ IDEA. Potom je konfiguriran projekt, izabrana ponuđena verzija Jave 17, određen JDK, alat za automatsko upravljanje ovisnostima – *Maven* i Spring Boot verzija *2.7.5*. Na kraju su izabrane potrebne početne ovisnosti (*Starter Web, Starter Data JPA, Starter Validation, Starter Security* itd.).

Maven je alat za automatsku izradu Java projekata i upravljanje njihovim ovisnostima. On pojednostavljuje proces na koji se Java projekti prevode, definiraju i distribuiraju. Maven definira centralizirana spremišta koja služe za pretraživanje i dohvaćanje biblioteka o kojima projekt ovisi. Dovoljno je upisati podatke o biblioteci koja je potrebna, a Maven će pronaći biblioteku na nekom centraliziranom spremištu, preuzeti i instalirati ju na mjesto gdje se ona može u projektu koristiti. Maven i ostali slični alati neizostavna su komponenta prilikom razvoja aplikacija na Java platformi. [32]

Strukturu Maven projekta i sve ovisnosti definira *pom.xml* datoteka. Ovisnosti su definirane unutar <*dependencies*> XML elementa, a primjer jedne ovisnosti predstavlja Kod 4.1.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

Kod 4.1 Struktura jedne ovisnosti unutar *pom.xml* datoteke.

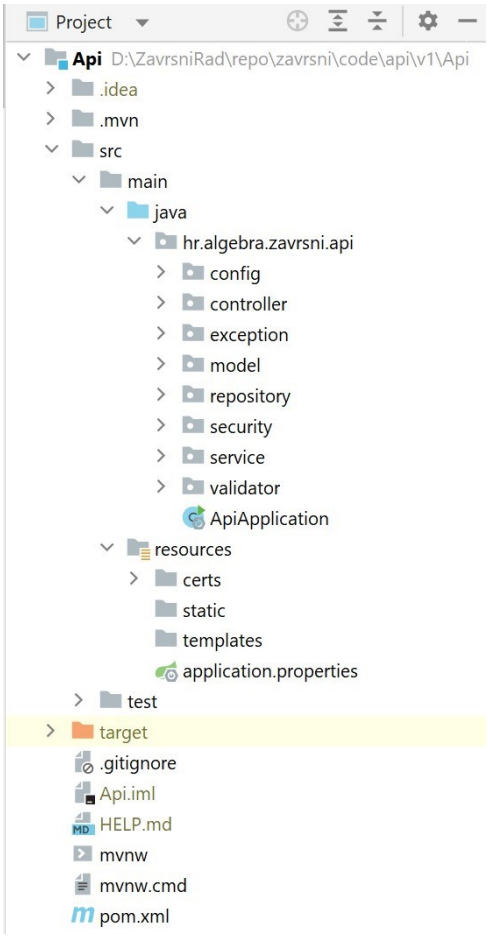
REST API Spring Boot projekt sastoji se od različitih mapa i datoteka. U */src/main/java* mapi nalazi se Java kod API servisa podijeljen u pakete, a mapa /src/main/resources sadrži sve datoteke potrebne za konfiguraciju Spring Boot projekta. Na slici 4.4 prikazana je struktura projekta.

Datoteka *ApiApplication* glavna je Spring Boot klasa uz pomoć koje se pokreće projekt. Ona sadrži anotaciju *@SpringBootApplication* koja se koristi za automatsko pretraživanje i konfiguraciju komponenti sustava i *main* metodu koja pokreće REST API servis. [33]

U datoteci *application .properties* nalaze se parametri zaduženi za konfiguraciju sustava. U njoj je definirana mrežna adresa, port i glavna putanja na kojoj će se servis nalaziti, konfiguracijski podaci potrebni za spajanje na SQL bazu podataka i bazu objekata, parametri potrebni za konfiguraciju JPA servisa i dr.

Java programski kod komponenti API-a organiziran je u pakete. Unutar *config* paketa nalaze se konfiguracijske klase koje dodatno konfiguriraju komponente sustava. U njemu je

konfiguriran MinIO klijent zadužen za pristup i spremanje fotografija u bazu objekata. U njemu su također definirane i sve sigurnosne postavke koje ograničavaju pristup servisu te sve konstante koje se koriste unutar čitavog sustava.



Slika 4.4 Struktura REST API projekta

Paket *expection* sadrži klase koje iznimke pretvaraju u prikladne HTTP statusne odgovore. Ako se prilikom obrade korisničkog zahtjeva dogodi greška, REST API će korisniku vratiti statusni kod koji određuje vrstu greške. Kod 4.2 donosi primjer jedne takve klase koja klijentu vraća HTTP statusni kod 409 onda kada korisnik pokušava dodati značajku koja već postoji.

```
@ResponseStatus(HttpStatus.CONFLICT)
public class ConflictException extends RuntimeException {}
```

Kod 4.2 Klasa *ConflictException* vraća HTTP statusni kod 409 za konflikt

Paket *validation* sadržava klase i sučelja koje određuju načine na koje se provjeravaju neki objekti sustava. Primjerice, *CityValidator* provjerava je li grad unesen i je li njegov

identifikator ispravna brojčana vrijednost veća od nule. Za provjeru podataka koriste se Spring Bean Validation paketi.

Unutar paketa *service* nalaze se klase koje definiraju funkcionalnosti vanjskih servisa na koje se REST API spaja. Funkcionalnosti uz pomoć kojih je moguće dohvatiti i spremiti fotografiju u bazu objekata nalaze se u *MinioService* klasi. U klasi je definiran URL baze objekata i lokacija na koju će se spremati podaci. Također, u njoj su definirane dvije metode za dohvaćanje i spremanje fotografija. Kod 4.3 prikazuje implementaciju metode za spremanje fotografija.

```
public boolean addImage(MultipartFile file) {
      try {
            minioClient.putObject(PutObjectArgs.builder()
                  .bucket(bucketName)
                  .object(file.getOriginalFilename())
                  .stream(file.getInputStream(),
                        file.getSize(), -1)
                  .build());
            return true;
      } catch (Exception e) {
            return false;
      }
}
```

Kod 4.3 Metoda za spremanje fotografija unutar MinIO baze objekata

Paket *model* sadrži Java klase koje opisuju objekte koji se nalaze unutar sustava za pregled novogradnje. Definirani su: projekt (*Property*), stan (*Unit*), grad (*City*), razgovor (*Conversation*), značajka (*Feature*), status projekta (*PropertyStatus*), korisnik (*User*), tip korisnika (*UserType*) i dr. Uz atribute, konstruktore i metode, klase sadrže i specijalne anotacije koje dodatno konfiguriraju objekte za pohranu unutar baze podataka. Kod 4.4 donosi dio klase koja predstavlja poruku.

```
@Entity
public class Message {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE,
generator = "message_generator")
    @SequenceGenerator(name = "message_generator",
sequenceName = "message_id_seq", allocationSize = 1)
    @Column(name = "id", updatable = false, nullable = false)
```

```
private Integer id;

@OneToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "sender_id")
private User sender;

@NotNull
private String message;

@Column(name = "create_date")
@JsonProperty(access = JsonProperty.Access.READ_ONLY)
private LocalDateTime dateCreated;

@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "conversation_id")
@JsonIgnore
private Conversation conversation;
```

Kod 4.4 Dio entitetske klase *Message*

REST API za pristup podacima unutar relacijske baze koristi modul Spring Data JPA. Da bi se klasa i njezini podaci mogli preslikati u tablicu unutar baze podataka treba dodati *@Entity* anotaciju i odrediti identifikator pomoću *@Id* anotacije. Svaki atribut klase, osim ako nije drugačije naznačeno, preslikava se na stupac unutar tablice u bazi podataka. [33]

Klasa poruka (*Message*) označena je kao entitet i preslikava se na tablicu poruka (*message*) u bazi podataka. Ima definiran atribut *id* koji u bazi predstavlja primarni ključ. Pomoću anotacije *@GeneratedValue* vrijednosti se automatski generiraju pomoću PostgreSQL generatora sljedova (engl. *sequence generator*).

Atribut *sender* označava osobu koja je poslala poruku i označen je dvjema anotacijama. *@OneToOne* označava odnos među klasama *User* - *Message* i njihovim tablicama u bazi podataka i znači da poruka može imati samo jednog pošiljatelja, a *FetchType.LAZY* argument predstavlja strategiju kojom će se podaci dohvatiti. Postoje dva načina na koji se povezani podaci mogu dohvatiti iz baze: odmah (engl. *eagerly*) ili na zahtjev (engl. *lazily*). Na zahtjev sprječava nepotrebno učitavanje povezanih podataka. Anotacija *@JoinColumn* određuje stupac koji upućuje na drugu tablicu gdje se nalaze podaci. [34]

Nadalje, klasa *Message* sadrži tekst poruke koji zbog *@NotNull* anotacije mora biti unesen. Datum kad je poruka kreirana sadrži dvije anotacije, *@Column* koja određuje naziv tablice

u bazi i *@JsonProperty* koja korisnicima dopušta samo čitanje podatka. Atribut *conversation* označava razgovor kojem poruka pripada. Anotacijom *@ManyToOne* definirana je jedna strana jedan na više odnosa između klasa *Message* i *Conversation*. Druga strana definirana je u klasi *Conversation* i ona sadrži listu svih poruka koje joj pripadaju. Anotacija *@JsonIgnore* jamči da će se povezani objekt ignorirati prilikom pretvaranja poruke u JSON format i obrnuto.

Ostale klase unutar *model* paketa definirane su na slične načine kao i klasa poruka.

Paket *repository* sadrži kolekciju sučelja koja nasljeđuju JpaRepository sučelje iz Spring Data JPA modula. Spring Data dio Spring razvojne okoline predstavlja posebnu apstrakciju za spremište podataka (engl. *repository*) definiranu kao Repository sučelje. Osnovne CRUD operacije definirane su uz pomoć CrudRepository sučelja. Sortiranje i dohvaćanje podataka po stranicama definirano je PagingAndSortingRepository, a specifične JPA funkcionalnosti JpaRepository sučeljem. Prije upotrebe Repository sučelja nije potrebno posebno implementirati. Tijekom izvršavanja aplikacije, Spring Data JPA automatski će implementirati sve njihove metode. [20]

Kod 4.5 prikazuje sučelje koje definira spremište podataka za projekte.

```
public interface PropertyRepository extends
JpaRepository<Property, Integer> {
    Iterable<Property> findAllByPropertyFeaturesId(int id);
    Iterable<Property> findAllByUsersId(int id);
    boolean existsByIdAndUsersId(int propertyId, int userId);
    Iterable<Property> findAllByInvestorId(int id);
}
```

Kod 4.5 Sučelje za spremište projekata

Sučelje definira četiri dodatne metode uz pomoć specifičnog domenskog jezika (engl. *domain-specific language*, skraćeno DSL) za kreiranje upita prema bazi podataka. Spring Data JPA analizirat će naziv metode i na temelju ključnih riječi odrediti SQL upit. Metoda *findAllByPropertyFeaturesId* vratit će sve projekte koji sadržavaju određenu značajku. [33]

Paket *controller* sastoji se od upravitelja (engl. *controller*) koji korisnicima dostavljaju tražene podatke. Upravitelji su dio Spring MVC arhitekture koja odgovornosti sustava dijeli na tri dijela: model sadrži podatke, pogled (engl. *view*) ih prikazuje, a upravitelj je zadužen za njihovu obradu. Klijenti do podataka dolaze slanjem HTTP zahtjeva na krajnje točke

REST API-a, a upravitelj je taj koji definira koje su sve krajnje točke dostupne na REST API servisu. Unutar servisa upravitelji su grupirani po tipu podataka koje pružaju klijentima. Za podatke o projektima zadužen je *PropertyController*, za stanove *UnitController*, korisnike sustava *UserController* itd.

Svaki upravitelj označen je @RestController anotacijom koja određuje na koji se način korisniku šalje traženi podatak. Umjesto da se podatak proslijedi pogledu za prikaz, on će se vratiti unutar tijela HTTP odgovora. *@RequestMapping* anotacija određuje dio putanje od koje se grade lokacije krajnjih točaka i format podataka u kojem će se odgovori vratiti korisniku.

Svaka krajnja točka definirana u upravitelju sadrži anotaciju koja određuje vrstu HTTP metode i njezinu implementaciju. Metode za dohvaćanje podataka označene su *@GetMapping* anotacijom, za stvaranje *@PostMapping,* ažuriranje *@PutMapping* itd. Kod 4.6 predstavlja dvije metode unutar upravitelja za korisnike.

```
@GetMapping("/{id}")
public User getUser(@PathVariable int id) {
      return userRepository.findById(id)
      .orElseThrow(NotFoundException::new);
    }
@DeleteMapping("/{id}")
public void deleteUser(@PathVariable int id) {
      try {
            userRepository.deleteById(id);
      } catch (EmptyResultDataAccessException e) {
            throw new NotFoundException();
      }
}
```

Kod 4.6 Dvije HTTP metode iz upravitelja korisnika

Podatak o određenom korisniku može se dobiti slanjem HTTP GET zahtjeva na URL krajnje točke koji završava identifikatorom korisnika. Anotacija *@PathVariable* određuje da je identifikator dio URL adrese. Primjerice, za dohvaćanje korisnika pod redni brojem 3 potrebno je poslati zahtjev na URL adresu koja završava na */users/3*. Metoda *getUser* iz korisničkog će spremišta dohvatiti korisnika pod tim rednim brojem i vratiti ga kao odgovor u JSON formatu. Ako korisnika nema u sustavu, vraća se HTTP statusni odgovor *404 Not Found*. Metoda *deleteUser* korisnika briše iz baze podataka. Za brisanje korisnika potrebno

je poslati DELETE HTTP zahtjev na isti URL kao i kod prethodne metode. Lokacija resursa se ne mijenja, a operacija koja će se na tom resursu izvršiti određena je vrstom HTTP metode.

Paket *security* sastoji se od klasa i sučelja koje omogućavaju autentifikaciju korisnika uz pomoć JSON Web Token (skraćeno JWT) standarda. Za autentifikaciju koriste se funkcionalnosti Spring Security modula. *SecurityConfiguration* klasa određuje i konfigurira način na koji se šifriraju lozinke i omogućava provjera JWT tokena.

Prilikom prijave u sustav, provjeravaju se korisničko ime i lozinka. Ako su podaci ispravni, korisniku se generira novi JWT token. Korisnik token šalje sa svakim novim HTTP zahtjevom, a njegovu ispravnost provjerava REST API servis. U slučaju da je riječ o ispravnom tokenu, zahtjev se obrađuje i korisniku se dostavlja traženi podatak. U suprotnome, korisnik dobiva obavijest da resursu nije moguć pristup u obliku HTTP statusnog odgovora 401 Unauthorized.

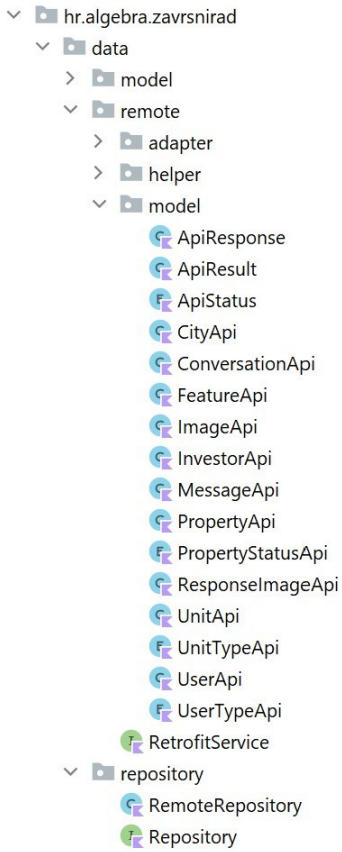
4.2.3. Izrada mobilne Android aplikacije

Mobilna aplikacija kojom korisnici pristupaju sustava za pregled novogradnje napravljena je u službenom integriranom razvojnom okruženju Android Studio. Izabran je novi projekt i predložak *Empty Activity*. Za programski jezik aplikacije izabran je Kotlin, a minimalna verzija Androida na kojoj se aplikacija može pokrenuti je 8.0 (API 26).

Arhitektura aplikacije sastoji se od dva sloja: podatkovnog sloja koji obrađuje podatke i korisničkog sloja koji te podatke prikazuje korisniku. [35]

4.2.3.1 Podatkovni sloj

Glavna zadaća podatkovnog sloja sastoji se u dohvaćanju, obradi i spremanju podataka. Unutar projekta stvoren je novi paket *data* i u njemu se nalaze Kotlin klase i paketi koji tvore podatkovni sloj. Paket *model* sadrži podatkovne klase za objekte koje se koriste unutar aplikacije, dok paket *remote* sadrži sve klase i pakete zadužene za dohvaćanje podataka putem udaljenog REST API-a. U *remote* paketu nalaze se: različiti adapteri koji objekte pretvaraju u JSON zapise, posebne pomoćne klase za obavljanje uobičajenih radnji koje se koristi kroz cijelu aplikaciju, podatkovne klase koje je moguće dohvatiti putem REST API-a, servis koji omogućuje dohvaćanje podatkovnih klasa i klase spremišta koje implementiraju metode za dohvat podatkovnih klasa. Slika 4.5 prikazuje strukturu podatkovnog sloja.



Slika 4.5 Podatkovni sloj mobilne aplikacije

Podatkovni sloj do podataka dolazi spajanjem na REST API servis uz pomoć Retrofit HTTP klijenta. U datoteci *RetrofitService* konfiguriran je Retrofit klijent i određene su sve metode prema krajnjim točkama REST API-a (Kod 4.7).

```
interface RetrofitService {
  companion object {
      private val moshi = Moshi.Builder()
        .add(KotlinJsonAdapterFactory())
        .add(PropertyStatusAdapter())
        .add(UnitTypeAdapter())
        .add(UserTypeAdapter())
        .add(NullPrimitiveAdapter())
        .add(LocalDateTime::class.java,
          LocalDateTimeAdapter().nullSafe())
        .add(ByteArray::class.java,
          Base64Adapter().nullSafe())
    .build()

  fun getInstance(token: String): RetrofitService {
```

```
        val instance: RetrofitService by lazy {
          val retrofit = Retrofit.Builder()
            .baseUrl(API_URL)
            .client(OkHttpClient.Builder()
              .addInterceptor(AuthInterceptor(token))
            .build())
            .addConverterFactory(
              MoshiConverterFactory.create(moshi)).build()
          retrofit.create(RetrofitService::class.java)
        }
        return instance
      }
    }
```

Kod 4.7 Dio Retrofit servisa za spajanje na REST API

Sučelje *RetrofitService* konfigurira dio servisa koji je zadužen za pretvaranje objekata u JSON zapise i JSON zapise nazad u objekte. Pretvaranje se provodi uz pomoć Moshi biblioteke. Zatim se konfigurira Retrofit HTTP klijent kojem se određuje URL na kojem se REST API nalazi, zaglavlje u kojem će se nalaziti token za autentifikaciju i Moshi pretvarač. Tako konfigurirani Retrofit klijent kreirat će se samo jednom prilikom prvog zahtjeva, a svaki naknadni zahtjev prema REST API-u koristit će taj isti klijent. Time se sprječava iznova bespotrebno stvaranje novog klijenta svaki puta kad je potrebno pristupiti udaljenim podacima.

Sučelje sadrži popis svih metoda za pristup krajnjim točkama udaljenog REST API-a. Primjer metode za dohvaćanje svih stanova definiranih unutar nekog projekta je kod 4.8.

```
    @GET("properties/{id}/units")
     suspend fun getPropertyUnits(@Path("id") id: Int?):
List<UnitApi>
```

Kod 4.8 Retrofit metoda za dohvaćanje stanova

Svaka metoda unutar Retrofit servisa definira HTTP zahtjev prema nekoj krajnjoj točki. Anotacijom je određena vrsta zahtjeva i definirana putanja. Za dohvaćanje svih stanova šalje se GET HTTP zahtjev prema REST API-u čiji URL završava sufiksom *properties/{id}/units*. Na mjesto *{id}* dolazi identifikator projekta čije stanove želimo dohvatiti. Retrofit podržava slanje pozadinskih zahtjeva uz pomoć Kotlin korutina (engl. *coroutines*), a privremeno zaustavljanje metoda omogućeno je pomoću ključne riječi *suspend*. [36]

Za pokretanje asinkronih operacija na Android platformi, službena dokumentacija preporučuje upotrebu Kotlin korutina. Korutine su nalik dretvama i podržavaju zaustavljanje operacija kojima ne blokiraju dretve. Jednom pokrenuta operacija može se zaustaviti na jednoj i ponovno pokrenuti na nekoj drugoj dretvi. Za razliku od dretvi, jednostavnije su za pokretanje i zauzimaju manje memorije. [37]

Unutar paketa *repository* nalaze se klase koje služe pristupanju podacima. Generičko sučelje Repository definira sve metode za dohvaćanje podataka iz REST API-a, a njihove implementacije nalaze se u *RemoteRepository* klasi. Trenutačno postoji samo jedna implementacija spremišta podataka, a u slučaju da se pojavi potreba za nekim novim spremištem, ono će se jednostavno moći integrirati u podatkovni sloj aplikacije.

U udaljenom spremištu podataka metode pozivaju HTTP metode iz Retrofit klijenta. Ti zahtjevi šalju se uz pomoć korutina na zasebnim pozadinskim dretvama koje služe dohvaćanju podataka putem mreže. Pozadinskim dohvaćanjem podataka ne blokira se glavna dretva korisničkog sučelja. Korutine se izvršavaju u različitim opsezima (engl. *scope*) koji određuju njihov životni ciklus, a za svaku operaciju određuje se kontekst ili skup dretvi u kojemu će se ona izvršavati. Za slanje i primanje podataka iz REST API-a spremište operacije izvršava u Dispatchers.IO kontekstu. [37]

Svi objekti koje aplikacija koristi nalaze se u paketu *model*. Nakon slanja HTTP zahtjeva prema REST API-u, spremište će vraćene objekte iz *model* paketa proslijediti korisničkom sloju za prezentaciju korisnicima.

4.2.3.2 Korisnički sloj

Korisnički sloj odijeljen je od podatkovnog sloja i dio je aplikacije koji je u interakciji s korisnikom. Njegova glavna zadaća je prikazivanje podataka korisnicima. Sastoji se od dvije komponente. Od različitih elemenata koji tvore grafičko-korisničko sučelje i ViewModel klasa koje čuvaju stanje podataka. [35]

Prvi prikaz koji korisnik vidi prilikom ulaska u Android mobilnu aplikaciju prikazan je pomoću *MainActivity* klase. Ona je zadužena za interakciju s korisnikom i prikazivanje grafičkog sučelja aplikacije na zaslonu uređaja. Svaka Android aplikacija mora sadržavati bar jednu klasu aktivnosti. Ova mobilna aplikacija sadrži samo jednu glavnu aktivnost (*MainActivity*), a svi ostali ekrani izmjenjivat će se unutar nje u obliku fragmenata (engl. *Fragments*).

Aktivnosti i fragmenti sastoje se od dva različita dijela: od XML datoteka koja definira izgled, elemente i raspored korisničkog sučelja te Java ili Kotlin pripadajuće klase koja upravlja tim grafičkim sučeljem i definira njegove funkcionalnosti. Sve XML datoteke nalaze se unutar projekta u posebnom direktoriju za pohranu resursa *res*. [36]

U glavnoj aktivnosti definirana su dva elementa grafičkog sučelja, element navigacije u donjem dijelu aplikacije koja korisniku služi za navigaciju - *BottomNavigationView* i glavni element u kojem će se izmjenjivati fragmenti aplikacije – *FragmentContainerView* (Kod 4.9).

```
<androidx.fragment.app.FragmentContainerView
  android:id="@+id/nav_host_fragment_activity_main"
  android:name="androidx.navigation.fragment.NavHostFragment"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  app:defaultNavHost="true"
  app:layout_constraintBottom_toTopOf="@id/nav_view"
  app:layout_constraintLeft_toLeftOf="parent"
  app:layout_constraintRight_toRightOf="parent"
  app:layout_constraintTop_toTopOf="parent"
  app:navGraph="@navigation/mobile_navigation" />
```

Kod 4.9 Navigacijski element glavne aktivnosti

Navigacijska komponenta sastoji se od pet glavnih fragmenata: karte, projekata, razgovora, korisničkog profila te dijela aplikacije koji služi za dodavanje novih projekata i stanova ili pregledavanje omiljenih projekata (ovisi o tipu korisnika koji je prijavljen u aplikaciji).

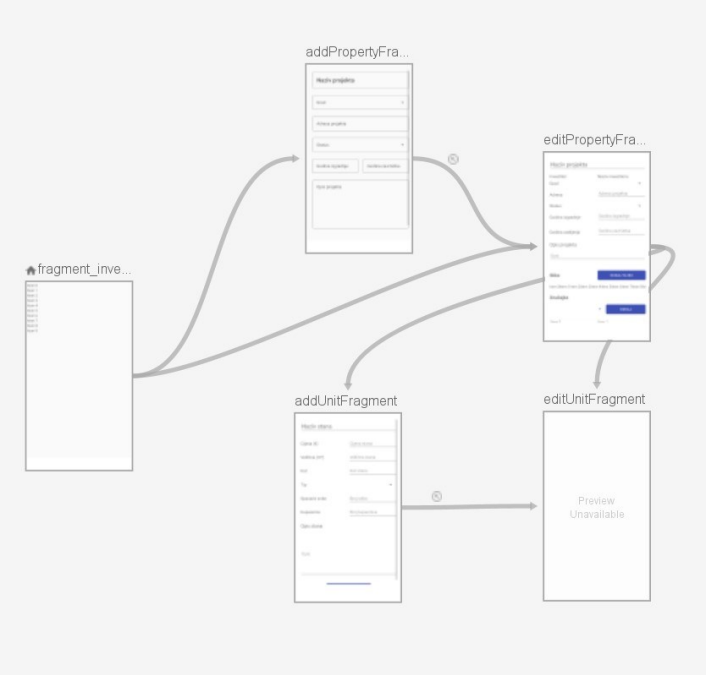
Uz pomoć navigacijske komponente korisnici se mogu kretati unutar aplikacije. Navigacijski dijelovi konfigurirani su u posebnoj datoteci *mobile_navigation.xml* koja se nalazi unutar direktorija s resursima. U njoj su definirani elementi navigacijskog grafa sa svim odredištima koja su dostupna korisnicima i svim putanjama uz pomoć kojih je moguće doći do određenih odredišta. Kod 4.10 prikazuje jedno odredište unutar navigacijskog grafa za investitore.

```
    <fragment
        android:id="@+id/fragment_investor"
android:name="hr.algebra.zavrsnirad.ui.investor.InvestorFragm
ent"
        android:label="Moji projekti"
        tools:layout="@layout/fragment_investor">
        <action
```

```
            android:id="@+id/action_investor_to_add_property"
            app:destination="@id/addPropertyFragment" />
        <action
            android:id="@+id/action_investor_to_edit_property"
            app:destination="@id/editPropertyFragment" />
    </fragment>
```

Kod 4.10 Odredište navigacijskog grafa za investitore

Definirano je odredište *InvestorFragment* kao početni ekran u kojem investitor može pregledati sve svoje projekte. Fragment ima svoj identifikator, naziv i putanju do XML datoteke u kojoj se nalaze svi grafički elementi ekrana. Određene su dvije akcije ili putanje koje vode prema drugim fragmentima. Akcija prema fragmentu u kojem investitor može dodati novi projekt i akcija prema fragmentu koji omogućava ažuriranje nekog postojećeg projekta. Na sličan način definirana su i sva ostala odredišta unutar aplikacije. Slika 4.6 prikazuje odredišta i putanje od kojih se sastoji navigacijski graf za investitore.



Slika 4.6 Dijagram navigacijskog grafa za investitore

Odredišta navigacijskog grafa predstavljaju fragmenti. Fragmenti su manji dijelovi grafičkog sučelja dizajnirani za višekratnu upotrebu. Usko su povezani s aktivnostima u kojima se nalaze i ne mogu se prikazivati izvan njih. Sastoje se od pogleda koji definiraju grafičke elemente i Java ili Kotlin klasa koje njima upravljaju. Svaki fragment sadrži svoj životni ciklus i povezan je sa životnim ciklusom aktivnosti u kojoj se nalazi. [36]

Mobilna aplikacije sastoji se od mnoštva fragmenata koji su podijeljeni u pakete prema svojim funkcionalnostima i unutar projekta nalaze se u *ui* paketu. Dio aplikacije *PropertiesFragment* prikazuje popis svih projekata i nalazi se u paketu *properties*.

Kroz svoj životni ciklus unutar glavne aktivnosti *PropertiesFragment* poziva različite metode. U metodi *onCreate()* koja služi za inicijalizaciju fragmenta provjeravaju se argumenti postavljeni prilikom navigacije na fragment (Kod 4.11). Iz Bundle klase izvlače se dva identifikatora i podatak je li prijašnja destinacija bila karta ili ne. Ako je prijašnja destinacija bila karta i ako je identifikator projekta vrijednost koja je veća od nule, prikazat će se novi fragment u kojemu se nalaze informacije o tome projektu. Bundle objekti služe za prijenos podataka između fragmenata i aktivnosti. Podaci su spremljeni unutar kolekcije ključ-vrijednost parova (engl. *key-value pair*). [36]

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    var fromMap = false
    var id = 0
    arguments?.let {
        favouriteId = it.getInt("favouriteId")
        fromMap = it.getBoolean("fromMap")
        id = it.getInt("propertyId")
    }
    if (fromMap && id > 0) {
        findNavController().navigate(
PropertiesFragmentDirections.actionPropertiesToProperty(id)
        )
    }
}
```

Kod 4.11 Provjera argumenata prije stvaranja fragmenta

Ako prijašnja destinacija nije bila karta, postavit će se izbornik unutar alatne trake koji omogućava pretragu liste projekata i njihovo filtriranje. Alatna traka jedan je od glavnih elemenata grafičkog sučelja aplikacije i njome se određuje naziv prikaza na kojem se korisnik nalazi i navigacijske komponente. Na nju se mogu dodati različiti izbornici koji fragment proširuju dodatnim mogućnostima.

Podaci o projektima dohvaćaju se pomoću Retrofit klijenta unutar podatkovnog sloja aplikacije, a za njihovo prikazivanje zadužena je RecyclerView komponenta grafičkog sučelja (Kod 4.12).

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/rv_properties"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layoutManager="LinearLayoutManager" />
```

Kod 4.12 Element grafičkog sučelja za prikazivanje liste projekata

RecyclerView komponenta koristi se za prikaz lista podataka. Za razliku od prijašnjih komponenti za prikazivanje lista, RecyclerView poboljšava performanse tako što reciklira one poglede (engl. *Views*) koji trenutno nisu prikazani. Umjesto da se za svaki objekt unutar liste stvori novi pogled, RecyclerView će ih stvoriti onoliko koliko je potrebno da se lista može prikazati na ekranu. Prilikom pomicanja liste, oni pogledi koji više nisu prikazani ponovno će se iskoristiti za prikaz novih podataka. Za prikazivanje podataka RecyclerView koristi tri pomoćne biblioteke, i to LayoutManager, ViewHolder i Adapter. LayoutManager određuje položaj podataka, odnosno hoće li se oni prikazati horizontalno ili vertikalno. ViewHolder predstavlja svaki element liste i sadrži referencu na pogled koji će se prikazati, a Adapter je komponenta koja podatke sprema u poglede. [36]

Podaci koje fragmenti koriste i prezentiraju korisnicima spremaju se u ViewModel klase. *PropertiesViewModel* zadužen je za spremanja podataka o projektima koji se prikazuju u *PropertiesFragment* fragmentu. Svaki fragment sadrži svoj ViewModel koji predstavlja središnje mjesto na kojem se čuvaju podaci. Kod 4.13 predstavlja metodu koja dohvaća projekte iz pripadajućeg spremišta podataka.

```
private fun getProperties() {
    _status.value = ApiStatus.LOADING
    viewModelScope.launch {
        when (val result = repository.getProperties()) {
            is ApiResult.Error -> {
                Log.e("ERROR", result.error)
                _originalProperties = listOf()
                _status.value = ApiStatus.ERROR
            }
            is ApiResult.Success -> {
                _originalProperties = result.value
```

```
                    filterProperties()
                    _status.value = ApiStatus.SUCCESS
                }
            }
        }
    }
```

Kod 4.13 Metoda za dohvat projekata unutar ViewModel klase

PropertiesViewModel za spremanje podataka koristi LiveData klase. U njih se spremaju podaci koji se mogu promatrati. Ako dođe do promjene podataka, dijelovi korisničkog sučelja o tome će dobiti obavijest i znak da mogu obaviti neku operaciju, odnosno prikazati nove podatke korisniku. [38]

U metodi *getProperties()* status odgovora pohranjuje se oblike *ApiStatus* enumeracije u status LiveData varijablu. Dok se podaci dohvaćaju, status je *LOADING*, dok se u trenutku pogreške on mijenja u *ERROR*, a kad je odgovor uspješno pročitan, status zahtjeva prelazi u *SUCCESS*.

Metoda za dohvaćanje projekata iz spremišta izvršava se uz pomoć korutine u posebnom opsegu predviđenom za pokretanje operacija unutar ViewModel klase - *viewModelScope*. On omogućava izvršavanje operacije samo onda kada je ViewModel aktivan. U trenutku njegove neaktivnosti, sve operacije će se automatski prekinuti. [36]

Lista projekata sprema se u *_originalProperties* LiveData varijablu i poziva se metoda koja filtrira projekte prema odabranim karakteristikama, a u slušaju pogreške sprema se prazna lista i zapisuje poruka o pogrešci.

U fragmentu promatra se varijabla koja sadrži projekte unutar *PropertiesViewModel*-a i u trenutku kada se oni prvi put pojave ili promijene, promijenit će se popis projekata prikazan korisnicima u RecyclerView-u. Prije toga stvara se adapter koji će postavljati vrijednosti u RecyclerView (Kod 4.14).

```
propertyAdapter = PropertyAdapter { property ->
adapterOnClick(property) }
        binding.rvProperties.adapter = propertyAdapter
```

Kod 4.14 Postavljanje adaptera za prikaz projekata

Adapter kao argument prima metodu *adapterOnClick* koja će otvoriti novi fragment s detaljima izabranog projekta.

Nakon postavljanja adaptera u RecyclerView, promatra se varijabla unutar ViewModel-a u kojoj se nalaze projekti (Kod 4.15).

```
viewModel.properties.observe(viewLifecycleOwner) {
    it?.let { properties ->
        propertyAdapter.updateList(properties)
        properties.forEach { property ->
            if (property.titleImage != null) {
                val file =
                    File(requireContext().filesDir, "prop-" +
property.id + "-titleimage")
                if (!file.exists()) {
                    file.outputStream().use { outStream ->
property.titleImage.inputStream().copyTo(outStream)
                    }
                }
            }
        }
    }
}
```

Kod 4.15 Promatranje varijable s projektima unutar ViewModel-a

Na Livedata *properties* varijabli postoji metoda *observe()* uz pomoć koje se mogu promatrati promjene na projektima. Ona prima dva parametra, metodu koja određuje u kojem dijelu životnog ciklusa će se promjene promatrati i metodu koja obavlja neku operaciju kada do promjena dođe. Promjene na projektima promatrat će se samo onda kada je fragment aktivan, a operacije su definirane pomoću anonimne lambda funkcije koja implementira *onChanged()* metodu Observer sučelja. [36]

Ako projekti postoje, ažurirat će se njihov adapter i prikazat će se nove promjene. Provjerit će se imaju li projekti na mobilnom uređaju spremljene naslovne fotografije te ako nemaju, stvorit će se nove datoteke s fotografijama.

4.3. Opis funkcionalnosti komponenata

U ovom poglavlju predstavljaju se glavne funkcionalnosti sustava za pregled novogradnje. Sustav korisnicima pruža mogućnost unosa i pregleda projekata novogradnje za gradove u Republici Hrvatskoj.

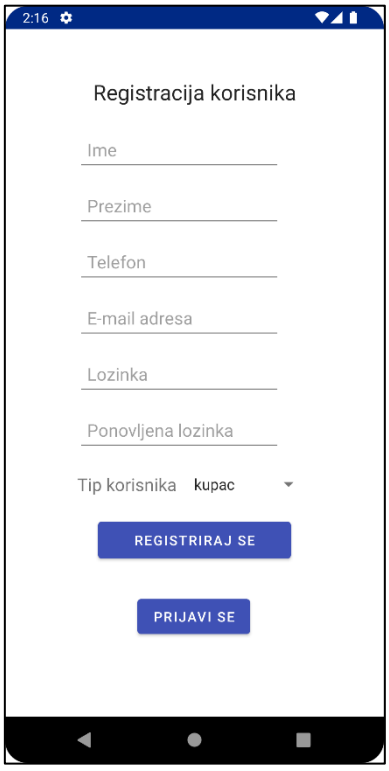
4.3.1. Funkcionalnosti registracije, prijave i korisničkog profila

Kako bi korisnici mogli koristiti sustav, prvo će morati izraditi korisnički račun. Njime dokazuju svoj identitet, a ovisno o vrsti korisnika, aplikacija će im omogućiti različite funkcionalnosti. Sustav razlikuje dvije vrste korisnika: kupca i investitora. Kupci mogu pretraživati projekte, a investitori ih mogu unositi u sustav.

Prilikom prvog pokretanja aplikacije korisnik se u sustav može prijaviti samo ako ima izrađen korisnički račun. U slučaju da ga nema, korisnički račun se može izraditi uz pomoć registracije.

4.3.1.1 Registracija korisnika

Registracija predstavlja postupak kojim se izrađuje korisnički račun (Slika 4.7). Za njegovu izradu potrebno je unijeti osobne podatke: ime, prezime, broj telefona, e-mail adresu, lozinku te odrediti tip korisnika. Kako bi se izbjegla mogućnost unosa krive ili neželjene lozinke, odabranu lozinku potrebno je potvrditi. Unos svih traženih podataka je obvezan i ako neki od podataka nije ispravno unesen, korisnik će o tome dobiti obavijest. Nakon uspješne registracije, aplikacija će korisnika preusmjeriti na prijavu u sustav.



Slika 4.7 Registracija korisnika

4.3.1.2 Prijava korisnika

Prijavom korisnik potvrđuje svoj identitet, što mu omogućuje ulazak u sustav i korištenje njegovih mogućnosti. Za prijavu je potrebno unijeti ispravnu e-mail adresu i lozinku. Sustav će unesene podatke provjeriti i, ako su ispravni, korisnik će biti preusmjeren na početni ekran aplikacije.

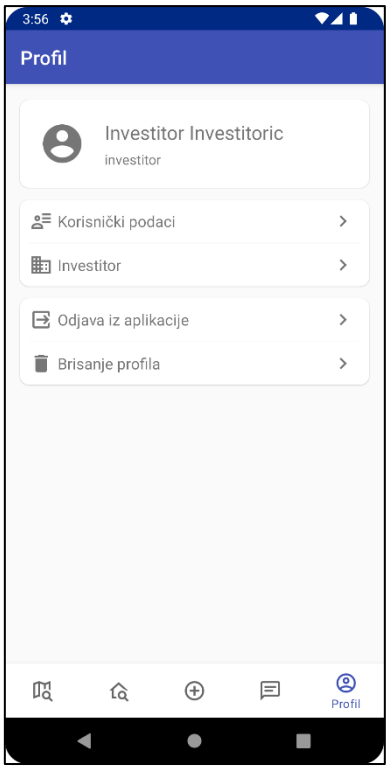
Uspješnu prijavu aplikacija će zapamtiti, a prilikom svakog novog pokretanja aplikacije, korisnika će se preusmjeriti direktno na početni ekran. Samo u slučaju odjave ili neispravnih podataka o prijavi, korisnik će se ponovno morati prijaviti u sustav.

4.3.1.3 Korisnički profil

Unutar korisničkog profila korisnici mogu pregledati svoje osobne podatke i vrstu korisnika. Korisnicima je omogućena promjena imena, prezimena, broj telefona i lozinke.

Korisnik se također može odjaviti iz aplikacije i obrisati svoj korisnički profil. Prije odjave ili brisanje profila, korisnik svoj izbor mora još jednom potvrditi. Nakon odjave ili brisanja korisničkog profila, korisnika se preusmjerava na ekran za prijavu.

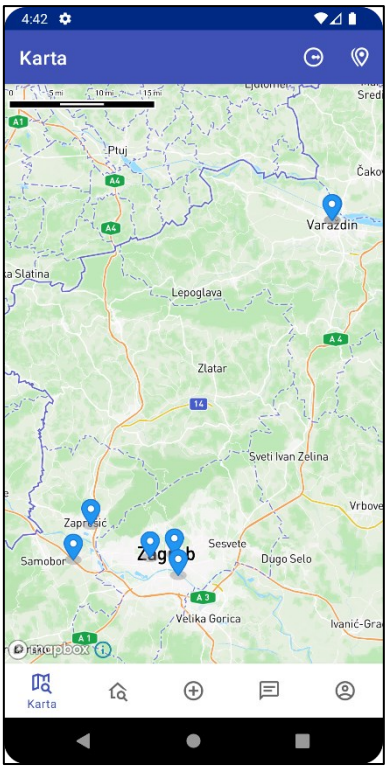
Ako je korisnik prijavljen u sustav kao investitor, u korisničkom profilu pruža mu se dodatna mogućnost pregleda i promjene podataka o investitoru (Slika 4.8). Investitoru se može promijeniti naziv, adresa i grad.



Slika 4.8 Korisnički profil investitora

4.3.2. Pregled karte projekata

Nakon otvaranja aplikacije, prvi ekran koji će se prikazati je karta sa svim projektima koji postoje u sustavu (Slika 4.9). Svaki projekt prikazan je uz pomoć plavog markera. Klikom na marker projekta, karta se uvećava i prikazuju se kratke informacije o tom projektu. Korisnik ovdje može vidjeti naziv grada, najmanju i najveću cijenu stanova, površine te godinu za kada je predviđeno useljenje. Za detaljnije informacije o projektu i stanovima, potrebno je kliknuti na detalje, prilikom čega će aplikacija preusmjeriti korisnika na ekran sa svim detaljima tog projekta.

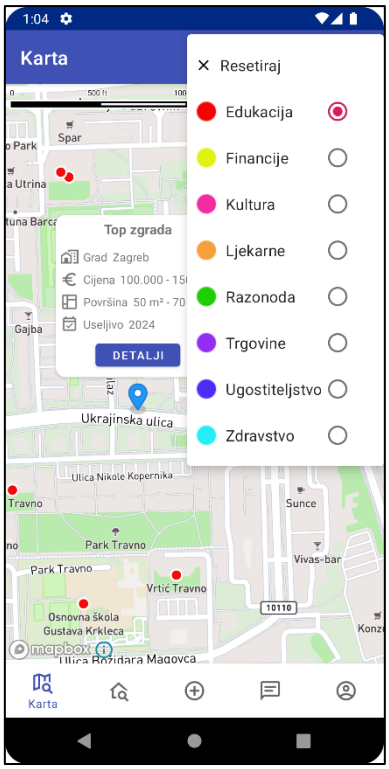


Slika 4.9 Karta projekata novogradnje

Osim lokacija projekata, korisnik može pregledati sve zanimljive popratne sadržaje koji se nalaze u blizini odabranog projekta. U alatnoj traci nalaze se dvije ikone. Prvom se određuje u kojoj udaljenosti od projekta će se prikazati sadržaji, a druga sadrži popis sadržaja raspoređenih u kategorije.

Zanimljivosti su grupirane u sljedeće kategorije: edukacija, financije, kultura, ljekarne, razonoda, trgovine, ugostiteljstvo i zdravstvo. Svaka grupa označena je svojom bojom. U grupi edukacija nalaze se vrtići, škole i ostale obrazovne i odgojne institucije; u financijama banke i druge financijske ustanove; u kulturi kazališta, koncertne dvorane, kina, knjižare i dr. U ostalim kategorijama nalaze se parkovi, sportske dvorane, igrališta, trgovine, kafići, barovi, restorani, bolnice i domovi zdravlja te ostali sadržaji.

Izborom neke kategorije i odabirom projekta, aplikacija će korisniku prikazati sve sadržaje u blizini tog projekta. Primjerice, ako korisnik želi saznati sadrži li neki projekt koji ga zanima u svojoj blizini edukacijske ustanove, potrebno je izabrati kategoriju *edukacija* i ako sadržaji postoje, prikazat će se kao točke označene bojom izabrane kategorije (Slika 4.10). Udaljenost sadržaja od projekta može se mijenjati na alatnoj traci, a za potvrdu potrebno je dulje kliknuti na marker projekta.

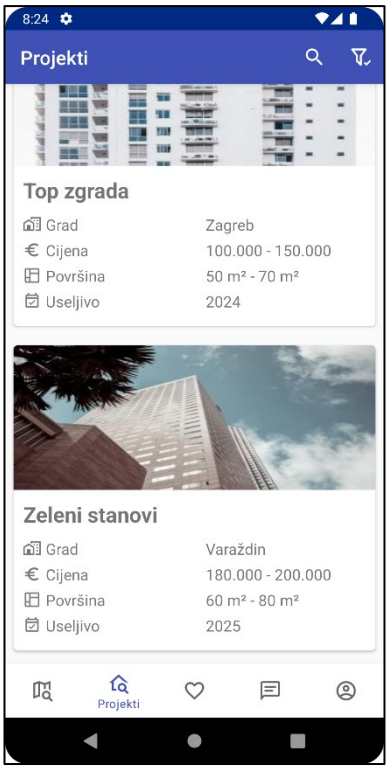


Slika 4.10 Najbliže obrazovne ustanove oko izabranog projekta

4.3.3. Pregled i pretraživanje projekata

Na ekranu s projektima korisnici mogu pregledati sve dostupne projekte. Projekti su prikazani u obliku liste koja sadrži naslovnu fotografiju, naziv, informaciju o minimalnoj i maksimalnoj cijeni stanova i površina od kojih se projekt sastoji te godinu useljenja. Na slici 4.11 prikazan je ekran s listom projekata.

Korisnici u alatnoj traci imaju mogućnosti pretraživanja i filtriranja liste projekata. Za pretraživanje potrebno je izabrati ikonu za traženje i upisati traženi pojam. Aplikacija će prikazati sve projekte koji sadrže taj pojam u svom nazivu. Projekti se mogu filtrirati i dodatno pretraživati po ostalim informacijama i karakteristikama, a za to je potrebno izabrati ikonu za filtriranje rezultata. Projekti se dodatno mogu pretraživati po investitoru, gradu, cijeni i površini stanova.



Slika 4.11 Pregled projekata

Slika 4.12 Detalji projekta

4.3.3.1 Pregled informacija o projektu

Izborom projekta s liste dostupnih projekata korisniku se prikazuju njegove detaljne informacije i popis svi stanova (Slika 4.12). Korisnik može saznati naziv projekta i investitora koji ga je objavio, adresu na kojoj se projekt nalazi, naziv grada, godinu kad je gradnja počela i godinu kad je planiran završetak te značajke od kojih se projekt sastoji i opis.

Broj i popis stanova prikazan je na donjem dijelu ekrana. Lista stanova prikazuje osnovne podatke o stanu i obavijest o tome da li je stan prodan ili ne. Na vrhu ekrana nalaze se sve objavljene fotografije projekta. Odabirom fotografije, korisniku se ona prikazuje u punoj veličini unutar galerije fotografija. Ispod fotografija prikazan je broj pregleda i broj osoba koje su zainteresirane za odabrani projekt.

Kupcima se pružaju dvije dodatne mogućnosti. U alatnoj traci nalaze se dvije ikone, ikona za slanje poruka i ikona za spremanje projekta. Kupci koji žele saznati više o nekom projektu, mogu poslati poruku investitoru, a sve poslane i primljene poruke mogu se pregledati u posebnom dijelu aplikacije za razgovore.

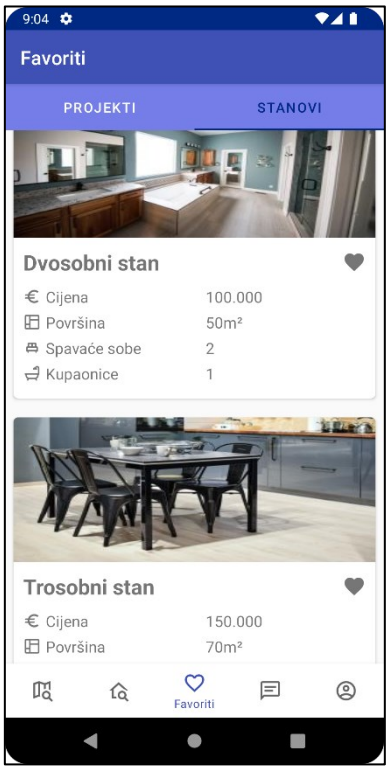
Projekte je također moguće označiti i spremiti kako bi ih se kasnije mogli jednostavnije i brže pronaći. Označavanjem korisnik pokazuju svoju zainteresiranost za projekt, a sve označene i spremljene projekte moguće je pregledati u posebnom dijelu aplikacije.

4.3.3.2 Pregled informacija o stanu

Korisnici mogu detaljnije pregledati informacije o svakom stanu unutar nekog projekta. Moguće je dobiti podatke o nazivu stana, njegovoj cijenu, veličini, tipu, dostupnosti, broju spavaćih soba i kupaonica. Svaki stan određen je svojim značajkama te sadrži opis i fotografije. Isto kao i za projekte, fotografije se mogu pregledati u punoj veličini unutar galerije. Za svaki stan također se prikazuje broj zainteresiranih osoba i broj osoba koje su stan pregledale.

4.3.4. Pregled spremljenih projekata

Označeni projekti i stanovi nalaze se u posebnom dijelu aplikacije pod nazivom favoriti. Ako se neki projekt ili stan želi maknuti iz liste spremljenih projekata ili stanova, potrebno je kliknuti na ikonu srca.



Slika 4.13 Spremljeni stanovi

4.3.5. Dodavanje i ažuriranje projekata

Glavne mogućnosti koje investitori imaju na raspolaganju u aplikaciji jesu dodavanje i ažuriranje projekata te stanova. Investitor je svaki korisnik koji je prilikom registracije odabrao tip korisnika investitor.

4.3.5.1 Unos podataka o investitoru

Prilikom prvog otvaranja aplikacije korisnik prije dodavanja novog projekta mora unijeti informacije o investitoru. Potrebno je unijeti naziv investitora te adresu i grad u kojem je prijavljen. Nakon uspješnog unosa podataka o investitoru, korisniku će se prikazati popis svih njegovih projekata.

4.3.5.2 Dodavanje novog projekta

Da bi se novi projekt dodao, potrebno je kliknuti na ikonu prikazanu znakom plus te na novom ekranu unijeti sve tražene podatke. Za svaki projekt potrebno je unijeti njegov naziv, grad i adresu, status projekta, godinu izgradnje, godinu završetka te njegov opis. Nakon uspješnog unosa osnovnih podataka o projektu, aplikacija će korisnika preusmjeriti na ekran s detaljima tog projekta. Na ovom mjestu korisnik može unijeti sve ostale podatke. Mogu se dodavati fotografije, odabrati značajke projekta i unijeti podaci o stanovima od kojih se projekt sastoji.

4.3.5.3 Uređivanje i brisanje projekta

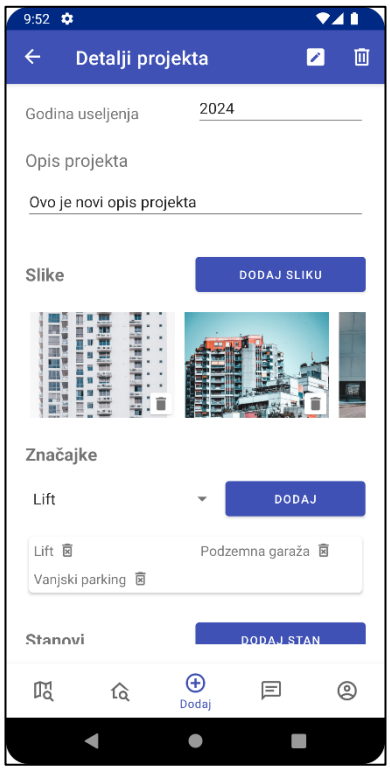
Za uređivanje podataka o projektu potrebnoj je kliknuti na ikonu za uređivanje, a aplikacija će omogućiti unos podataka. Slika 4.14 prikazuje ekran za uređivanje podataka o projektu. Da bi se projekt obrisao, potrebno je kliknuti na ikonu za brisanje i potvrditi radnju.

4.3.5.4 Dodavanje novog stana

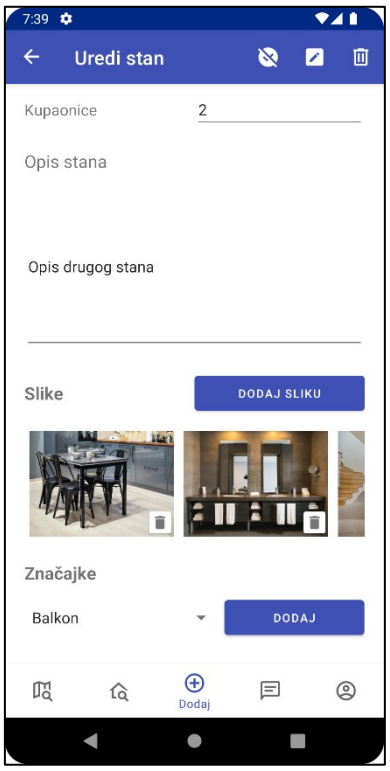
Stanovi u projekt dodaju se na način da se klikne na gumb dodaj stan nakon kojeg će aplikacija prikazati ekran za upisivanje podataka o stanovima. Za svaki stan potrebno je upisati njegov naziv, cijenu, veličinu, kat na kojem se u zgradi nalazi, tip stana, broj spavaćih soba i kupaonica te opis stana. Upisom osnovnih podataka otvara se dio aplikacije koji sadrži detalje upisanog stana na kojem je moguće dodavati fotografije i odrediti značajke stana. Uspješno upisan novi stan pojavit će na popisu stanova unutar detalja o projektu.

4.3.5.5 Uređivanje i brisanje stana

Isto kao i projekte, korisnik može uređivati i brisati stanove (Slika 4.15). Ikone za pokretanje radnji nalaze se na alatnoj traci u dijelu aplikacije koji prikazuje detalje o stanu. Također, u ovom dijelu aplikacije investitor može odrediti dostupnost stana. Ako stan više nije dostupan za prodaju, potrebno je kliknuti na prvu ikonu označenu kvačicom. Ako je radnja uspješno izvršena, ikona će se promijeniti novu ikonu označenu prekriženom kvačicom.



Slika 4.14 Uređivanje podataka o projektu



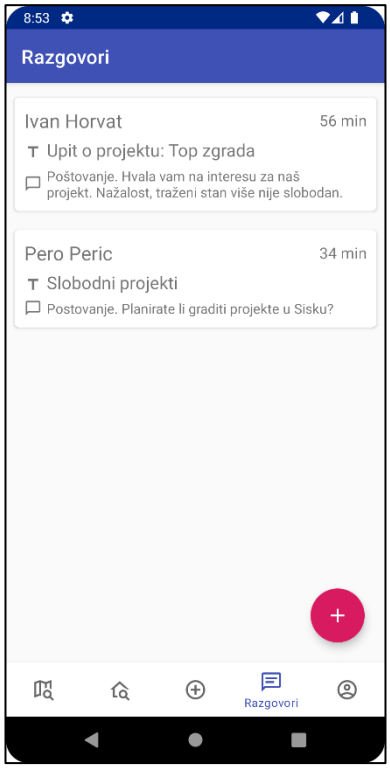
Slika 4.15 Uređivanje podataka o stanu

4.3.6. Funkcionalnost komunikacije porukama

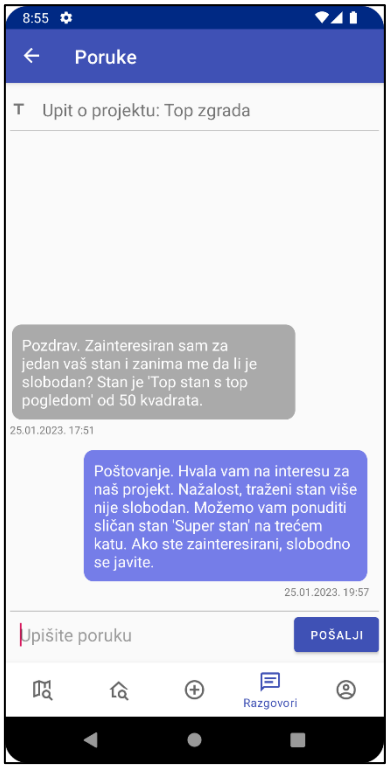
Svi korisnici aplikacije imaju mogućnost komunikacije putem poruka. Slanje i primanje poruka omogućeno je na posebnom dijelu aplikacije za razgovore.

Kupci zainteresirani za određeni projekt mogu poslati poruku investitoru i od njega dobiti odgovor. Poruke mogu poslati na dva načina. U detaljima projekta kupac može poslati samo jednu poruku koja je vezanu uz projekt, dok se sva ostala komunikacija obavlja u dijelu aplikacije predviđenoj za razgovore.

Ondje se nalazi popis svih razgovora koje korisnik vodi s ostalim korisnicima sustava. Razgovori su prikazani u obliku liste i svaki razgovor sastoji se od imena i prezimena sugovornika ili naziva investitora, naslova, dijela zadnje poslane poruke i vremena koje je prošlo od zadnje poslane poruke (Slika 4.16).



Slika 4.16 Lista razgovora



Slika 4.17 Poruke unutar razgovora

Odabirom razgovora otvara se novi ekran sa svim porukama koje se nalaze u tom razgovoru (Slika 4.17). Naslov razgovora prikazan je na vrhu ekrana, a poruke sudionika razlikuju se po boji i strani na kojoj su prikazane. Poruke osobe koja je prijavljena u aplikaciju nalaze se na desnoj strani i imaju plavičastu pozadinu, dok se na lijevoj strani nalaze poruke sugovornika. Ispod svake poruke prikazan je datum i vrijeme kad je poslana, a na dnu ekrana nalazi se polje za unos nove poruke.

Aplikacija nudi mogućnost brisanja razgovora, za što je potrebno dugo kliknuti na razgovor koji se želi obrisati, i potom, potvrditi radnju. Obrisani razgovor nestat će s liste razgovora i biti označen kao zatvoren. Sugovornik obrisanog razgovora i dalje će u svojoj listi imati prikazan razgovor, samo će biti označeno da je razgovor zatvoren, a slanje poruka će biti onemogućeno (Slika 4.18).

Slika 4.18 Obrisan i zatvoren razgovor

Za pokretanje novih razgovora i slanje poruka potrebno je kliknuti na kružić označen znakom plus. Nakon toga, u aplikaciji će se otvoriti novi ekran u kojem je potrebno izabrati kupca ili investitora s kojim se želi pokrenuti razgovor, odrediti naslov i tekst poruke. Bitno je napomenuti da kupci razgovore mogu pokretati samo s investitorima, a investitori samo s kupcima.

5. Testiranje i analiza uspješnosti programskog rješenja

Ovo poglavlje donosi kratak pregled načina na koji se testirao sustav za pregled novogradnje, njegove funkcionalnosti i daljnje mogućnosti za nadogradnju. Cilj sustava je ponuditi korisnicima programsko rješenje pomoću kojeg im se olakšava pretraživanje i uvid u projekte novogradnje na teritoriju Republike Hrvatske. Rješenje je predstavljeno u obliku mobilne aplikacije kojom se kupcima omogućava pregled, a investitorima predstavljanje projekata i stanova. Mobilna aplikacija je povezana s poslužiteljskim servisom koji preuzima podatke i sprema ih zatim u baze podataka.

Prvi korak prilikom izrade sustava bio je određivanje korisničkih zahtjeva kojima su definirane potrebe korisnika. Bilo je važno omogućiti kupcima da mogu pretraživati dostupne projekte i o njima dobiti detaljne informacije. Zahvaljujući karti, korisnicima je u aplikaciji omogućen pregled lokacija na kojima se projekti nalaze i sve zanimljive ustanove te objekti u blizini odabranih lokacija. Investitorima je omogućeno predstavljanje novih projekata i stanova te upravljanje njihovim podacima. Isto tako, omogućen je i način za jednostavniju i bržu komunikaciju između kupaca i investitora.

Potom se odredila arhitektura najprimjerenija za ovakav tip sustava. Korisnički zahtjevi i mogućnosti za buduće nadogradnje odredili su da će se sustav sastojati od troslojne arhitekture. I to od mobilne aplikacije, poslužiteljskog dijela i baza podataka. Prilikom dizajna mobilne aplikacije vodilo se računa o dobrom korisničkom iskustvu. Tema i boje su usklađene u svim dijelovima aplikacije, tamo gdje je to bilo prikladno dodavane su ikone da bi se vizualno bolje odredio sadržaj, a navigacija unutar aplikacije napravljena je na takav način da se maksimalno olakša snalaženje i pronalazak informacija.

Implementacijom svih komponenti sustava ispunjeni su korisnički zahtjevi i osnovne funkcionalnosti za njegovu upotrebu.

5.1. Testiranje programskog rješenja

Prilikom i nakon izrade sustava, zbog mnoštva funkcionalnosti i uštede vremena, komponente su testirane ručno bez upotrebe automatiziranih programa i alata. Poslužiteljski

dio testiran je upotrebom Swagger i Hoppscotch alata za testiranje i dokumentiranje REST API servisa. Krajnje točke REST API servisa pregledane su i testirane prije upotrebe u mobilnoj aplikaciji. Mobilna aplikacija testirana na Android Emulatoru koji se nalazi integriran u službenu razvojnu okolinu za izradu Android mobilnih aplikacija. Emulator omogućava testiranje aplikacija na različitim verzijama operativnog sustava i na različitim tipovima uređaja, veličinama ekrana i rezolucija, konfiguracijama i uvjetima rada.

Mobilna aplikacija testirana je na dvije različite verzije Android operativnog sustava i u različitim mrežnih uvjetima. Testirana je na verziji 12 i 13, te u uvjetima kada pristup mreži nije omogućen, na podatkovnog mreži slabijeg signala i na bežičnoj mreži. Aplikacija je također testirana na fizičkom Android uređaju i to na Android 12 verziji operativnog sustava.

Iako implementirani sustav ispunjava korisničke zahtjeve i donosi osnovne funkcionalnosti, za njegovu stvarnu primjenu potrebne su daljnje nadogradnje i poboljšanja.

5.2. Moguće nadogradnje i poboljšanja

U trenutnoj fazi sustav je testiran isključivo na lokalnom računalu putem lokalne mreže. Za primjenu u produkcijskoj okolini potrebno je njegove komponente prebaciti na programska rješenja u oblaku. Tijekom izbora tehnoloških rješenja pomoću kojih se sustav razvijao vodilo se računa o budućem postavljanju komponenata na rješenja u oblaku. Zbog toga izabrana je popularna i stabilna baza podataka PostgreSQL, a REST API implementiran je na Java Spring Boot razvojnoj okolini. Za bazu objekata izabrana je baza MinIO kompatibilna s najpopularnijim podatkovnim servisima za pohranu velikih količina podataka.

Određene krajnje točke REST API servisa mogle bi se promijeniti da odgovaraju RESTful pravilima. REST API ne podržava mogućnost navigacije putem linkova, pa klijenti za pristup resursima moraju unaprijed imati definirane lokacije.

U trenutnom obliku mobilna aplikacija za pristup podacima treba imati aktivnu mrežnu konekciju. U slučaju da ne postoji aktivna konekcija, korisnik aplikaciju neće biti u mogućnosti koristiti. U tom slučaju, nadogradnja bi se sastojala od dodatnog dijela podatkovnog sloja koji bi dohvaćene podatke spremao u lokalnu bazu podataka. Time bi se korisnicima omogućila upotreba aplikacije i onda kada nemaju aktivnu mrežnu konekciju, u

područjima slabijeg mrežnog signala… Način na koji je podatkovni sloj mobilne aplikacije implementiran omogućava dodavanje novih dijelova bez značajnijih promjena u kodu.

Osim ručnog testiranja komponenti sustava trebao bi uvesti automatizirano testiranje pomoću programa i alata, a testirati bi trebalo pojedinačne dijelove i sustav kao cjelinu.

Arhitektura sustava omogućava jednostavno dodavanje i integriranje komponenti, a dobrodošla nadogradnja sastojala bi se od klijentske web aplikacije kojoj bi mogli pristupiti i oni korisnici koji ne posjeduju Android mobilni uređaj.

Zaključak

Potreba da se kupcima olakša i pojednostavi proces pretraživanja svih mjesta na kojima se mogu pronaći nove nekretnine pružila je priliku za implementaciju novog programskog rješenja za njihov pregled u Hrvatskoj. Programsko rješenje prikazano u ovom radu nudi korisnicima jednostavniji i brži pregled novogradnje putem središnjeg sustava za njihov prikaz, a kojim se štedi vrijeme i trud koji bi bili potrebni u pronalasku idealne nekretnine.

Sustav se sastoji od mobilne Android aplikacije koja korisnicima omogućava pregled svih bitnih informacija o dostupnim projektima. Zahvaljujući aplikaciji, kupci mogu pregledati koji se sve sadržaji nalaze u neposrednoj blizini nekretnine za koju su zainteresirani kako bi dobili potpuniju sliku o njezinoj vrijednosti i kvaliteti života i stanovanja u mjestu u kojem se ona nalazi. Mobilna aplikacija pruža mogućnost direktne komunikacije s investitorima pomoću koje je moguće dobiti najrelevantnije informacije o projektima, dogovoriti razgledavanje te ubrzati i olakšati sam proces dogovora, pa naposljetku i - kupovine. Investitori objavljivanjem informacija o projektima na centraliziranom mjestu mogu brže i jednostavnije doći do kupaca te smanjiti vrijeme i trošak koji bi imali da informacije o svojim projektima objavljuju na različitim mjestima.

Pri izradi programskog rješenja koristile su se stabilne i pouzdane tehnologije temeljene na otvorenom kodu. Mobilna Android aplikacija napisana je u programskom jeziku Kotlin, koristeći najbolje prakse za izradu mobilnih aplikacija na Android platformi. Poslužiteljski dio sustava razvijen je uz pomoć programskog jezika Java koristeći dobro poznati razvojni okvir za izradu web servisa Spring Boot, dok je za bazu podataka zbog svoje stabilnosti odabran je PostgreSQL sustav. Pažnja je posvećena da, prilikom razvoja sustava, svaka njegova komponenta bude modularna i lako proširiva novim funkcionalnostima.

Ovako implementiran sustav predstavlja upotrebljivu cjelinu koja svojim osnovnim funkcionalnosti zadovoljava postavljene korisničke zahtjeve i kao takva može se koristiti. Prije toga potrebno je sve komponente sustava izdvojiti iz lokalnog razvojnog okruženja te postaviti na prikladne programske platforme u oblaku kao što su *Amazon Web Services, Heroku* i dr. Daljnjim nadogradnjama i poboljšanjima te dodatnim funkcionalnostima sustav bi mogao naći svoje mjesto na zahtjevnom tržištu mobilnih aplikacija koje se bave pretraživanjem, oglašavanjem i prodajom nekretnina.

Popis kratica

ACID	Atomicity, Consistency, Isolation, Durability	atomarnost, konzistentnost, izolacija, izdržljivost
API	application programming interface	aplikacijsko programsko sučelje
CRUD	create, read, update, and delete	stvori, pročitaj, ažuriraj, obriši
HATEOAS	Hypermedia As The Engine Of Application State	
HTTP	Hypertext Transfer Protocol	protokol za prijenos hiperteksta
IDE	integrated development environment	integrirano razvojno okruženje
JDBC	Java Database Connectivity	
JDK	Java Development Kit	Java razvojna distribucija
JPA	Java Persistence API	
JSON	JavaScript Object Notation	
JVM	Java virtual machine	Java virtualna okolina
JWT	JSON Web Token	
MVC	Model-View-Controller	
ORM	Object–relational mapping	objektno-relacijsko mapiranje
RDBMS	relational database management system	sustav za upravljanje relacijskim bazama podataka
REST	Representational State Transfer	
SQL	Structured Query Language	strukturni upitni jezik
URI	Uniform Resource Identifier	jedinstveni identifikator resursa
XML	Extensible Markup Language	proširivi jezik za označavanje podataka
WWW	World Wide Web	

Popis slika

Popis kôdova

Literatura

[1] BUTORAC, G. Građevinarstvo i nekretnine. Sektorske analize, Ekonomski institut, 11,97 (2022), 1-23.

[2] MUSTAĆ, J. Cjenovni balon na tržištu nekretnina u Republici Hrvatskoj, Svečilište u Zadru - Odjel za ekonomiju, Oeconomica Jadertina, 9,1 (2019), 78-88.

[3] RICHARDS, M., Software Architecture Patterns, https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html, siječanj. 2023.

[4] MOZILLA, An overview of HTTP, https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview, siječanj. 2023.

[5] RED HAT, What is a REST API?, https://www.redhat.com/en/topics/api/what-is-a-rest-api, siječanj. 2023.

[6] CHURCHER, C. Beginning Database Design: From Novice to Professional. New York: Apress, 2012.

[7] TYSON, M., What is JPA? Introduction to Java persistence, https://www.infoworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html, siječanj. 2023.

[8] POSTGRESQL GLOBAL DEVELOPMENT GROUP, About PostgreSQL, https://www.postgresql.org/about/, siječanj. 2023.

[9] POSTGRESQL GLOBAL DEVELOPMENT GROUP, PostgreSQL 14.6 Documentation, https://www.postgresql.org/docs/14/index.html, siječanj. 2023.

[10] JEPSON, B., PostgreSQL vs. MySQL: Building better databases, https://people.apache.org/~jim/NewArchitect/webtech/2001/09/jepson/index.html, siječanj. 2023.

[11] SUZUKI, H. The Internals of PostgreSQL, https://www.interdb.jp/pg/pgsql02.html, siječanj. 2023.

[12] MINIO, MinIO High Performance Object Storage, https://min.io/docs/minio/kubernetes/upstream/, siječanj. 2023.

[13] TILKOV, S. A Brief Introduction to REST, https://www.infoq.com/articles/rest-introduction, siječanj. 2023.

[14] KUMAR, D., Best Practises For Building Restful Web Services, https://www.infosys.com/digital/insights/documents/restful-web-services.pdf, siječanj. 2023.

[15] GUPTA, L., REST Architectural Constraints, https://restfulapi.net/rest-architectural-constraints, siječanj. 2023.

[16] OSMAN, J., JSON vs XML, https://appmaster.io/blog/json-vs-xml, siječanj. 2023.

[17] VMWARE, Spring Framework Documentation, https://docs.spring.io/spring-framework/docs/5.3.23/reference/html/index.html, siječanj. 2023.

[18] WALLS, C. Spring Boot in Action. New York: Manning Publications Co, 2016.

[19] VMWARE, Spring Data, https://spring.io/projects/spring-data, siječanj. 2023.

[20] VMWARE, Spring Data JPA - Reference Documentation, https://docs.spring.io/spring-data/jpa/docs/current/reference/html, siječanj. 2023.

[21] GOOGLE, Android Developer Fundamentals (Version 2) – Concepts, https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/index.html, siječanj. 2023.

[22] OPEN HANDSET ALLIANCE, Open Handset Alliance Overview, https://www.openhandsetalliance.com/oha_overview.html, siječanj. 2023.

[23] GOOGLE, Platform Architecture, https://developer.android.com/guide/platform/index.html, siječanj. 2023.

[24] SQUARE, Retrofit, https://square.github.io/retrofit, siječanj. 2023.

[25] GLIDE, About Glide, https://bumptech.github.io/glide, siječanj. 2023.

[26] MAPBOX, Maps SDK for Android, https://docs.mapbox.com/android/maps/guides, siječanj. 2023.

[27] IBM, What is Java?, https://www.ibm.com/topics/java, siječanj. 2023.

[28] TYSON, M., What is the JDK? Introduction to the Java Development Kit, https://www.infoworld.com/article/3296360/what-is-the-jdk-introduction-to-the-java-development-kit.html, siječanj. 2023.

[29] GUPTA, L., Java Versions and Features, https://howtodoinjava.com/series/java-versions-features, siječanj. 2023.

[30] JEMEROV, D., ISAKOVA, S. Kotlin in Action. New York: Manning Publications Co, 2017.

[31] JETBRAINS, What is IntelliJ IDEA?, https://www.jetbrains.com/idea/features, siječanj. 2023.

[32] TYSON, M., What is Apache Maven? Build and dependency management for Java, https://www.infoworld.com/article/3516426/what-is-apache-maven-build-and-dependency-management-for-java.html, siječanj. 2023.

[33] WALLS, C. Spring in Action, Sixth Edition. New York: Manning Publications Co, 2022.

[34] JANSSEN, T., Entity Mappings: Introduction to JPA FetchTypes, https://thorben-janssen.com/entity-mappings-introduction-jpa-fetchtypes, siječanj. 2023.

[35] GOOGLE, Guide to app architecture. https://developer.android.com/topic/architecture, siječanj. 2023.

[36] SILLS, B., GARDNER, B., MARSICANO, K., STEWART, C. Android Programming: The Big Nerd Ranch Guide. Indianapolis: Pearson Technology Group, 2022.

[37] GOOGLE, Kotlin coroutines on Android, https://developer.android.com/kotlin/coroutines, siječanj. 2023.

[38] GOOGLE, LiveData overview, https://developer.android.com/topic/libraries/architecture/livedata, siječanj. 2023.