

UNIVERZITET U ZENICI

Politehnički fakultet

Softversko inženjerstvo Web
dizajn

Kerim Nezo

**UPOTREBA LARAVEL I LIVEWIRE TEHNOLOGIJA U IZGRADNJI
SISTEMA ZA DIGITALIZACIJU PROCESA U AGENCIJAMA ZA
NEKRETNINE**

Diplomski rad

Mentor:

V. prof. dr. Elmir Babović

Zenica, 2025. godine

BIBLIOGRAFSKA KARTICA RADA

NAUČNO PODRUČJE RADA: Tehničke nauke

NAUČNO POLJE RADA: Računarstvo i informatike

NAUČNA GRANA RADA: Softversko inženjerstvo

USTANOVA U KOJOJ JE IZRAĐEN RAD: Univerzitet u Zenici, Politehnički fakultet
u Zenici

MENTOR RADA: V. prof. dr. Elmir Babović

DATUM ODBRANE RADA: 07.05.2025.

ČLANOVI KOMISIJE ZA ODBRANU:

1. V. prof. dr. Nevzudin Buzađija, Predsjednik Komisije
2. Prof. dr. Sabahudin Jašarević, član Komisije
3. V. prof. dr. Elmir Babović, Mentor i član komisije

IZJAVA

Izjavljujem da sam diplomski rad pod naslovom „Upotreba Laravel i Livewire tehnologija u izgradnji sistema za digitalizaciju procesa u agencijama za nekretnine” izradio samostalno pod mentorstvom V. prof. dr. Elmir Babović. U radu sam koristio literaturu koja je navedena na kraju diplomskog rada. Tuđe spoznaje, zaključke, stavove, teorije i zakonitosti koje sam izravno ili parafrazirajući naveo u diplomskom radu na uobičajen, standaran način sam povezao sa fusnotama i korištenim bibliografskim jedinicama.

Student:

Kerim Nezo

SAŽETAK

Projekti web razvoja igraju ključnu ulogu u modernom softverskom svijetu i postaju veoma bitne za poduzeća, organizacije i pojedince. U slučaju ovog diplomskog rada, kreirana je web aplikacija koja digitalizuje procese agencijama za nekretnine, upotrebom modernih tehnologija kao što su Livewire i Blade templating engine za Frontend, Laravel za Backend, MySQL baza podataka i mnogo drugih “*package-a*“. Dokumentovan je cijeli razvojni proces, u kojem su uključeni procesi kreiranja koncepta aplikacije, dizajna, implementacije i testiranja. Opisana je interakcija i način međusobnog rada mnogih tehnologija, ”*frameworka*”, biblioteka i alata uz pomoć kojih je kreirana efikasna, skalabilna i pouzdana aplikacija.

Ključne riječi: web razvoj, web aplikacija, Livewire, Blade, Laravel, MySQL, framework

ABSTRACT

Web development projects play a key role in the modern software world and have become essential for businesses, organizations and individuals. In the case of this thesis, a web application was developed to digitalise processes for real estate agencies using modern technologies such as Livewire and Blade templating engine for frontend, Laravel for backend, MySQL for database and many other packages. The entire development process has been documented, including the stages of creating the application concept, design, implementation and testing. The interaction and integration of various technologies, frameworks, libraries and tools used to create an efficient, scalable and reliable application are described in detail,

Keywords: web development, web application, Livewire, Blade, Laravel, MySQL, framework

SADRŽAJ

1.	UVOD.....	1
2.	WEB TEHNOLOGIJE.....	1
2.1	PHP i Laravel.....	1
2.2	Livewire.....	2
2.3	Blade templates.....	3
2.4	Visual Studio Code.....	3
3.	COMPOSER PAKETI.....	4
3.1	Spatie/laravel-permission.....	4
3.2	Spatie/laravel-medialibrary.....	5
3.3	Intervention/image-laravel.....	5
4.	TEHNOLOGIJE ZA BAZU PODATAKA.....	6
4.1	MySQL.....	6
4.2	DBeaver.....	7
5.	IMPLEMENTACIJA OGLEDNE WEB APLIKACIJE.....	8
5.1	Planiranje osnovne strukture i rasporeda.....	8
5.2	Use Case dijagrami.....	9
5.3	Dijagrami sekvenci.....	12
5.4	Klasni dijagram.....	15
5.5	ER dijagram.....	16
5.6	Dijagram komponenti.....	19
5.7	Kreiranje aplikacije.....	20
5.8	Inicijalizacija projekata.....	20
5.9	Konfiguracija baze podataka.....	21
5.10	Autentifikacija.....	23
5.11	Konfiguracija Spatie paketa.....	25
5.12	User testni podaci.....	27
5.13	Definisanje ostalih modela.....	30
5.14	Kreiranje Observer klase.....	34
5.15	Stilizovanje aplikacije.....	35
5.16	Kreiranje web stranica.....	36
5.17	Prikaz podataka u obliku grafova.....	55
5.18	Prikaz funkcionalnosti sistema preporuke.....	56
6.	ANALIZA REZULTATA.....	59
7.	ZAKLJUČAK.....	61

8. LITERATURA.....	62
--------------------	----

1. UVOD

Cilj ovog rada je prikaz procesa razvoja digitalnog rješenja namijenjenog agencijama za nekretnine, s fokusom na olakšavanje svakodnevnih poslovnih zadataka agenata i menadžera jedne agencije za nekretnine. Aplikacija je napravljenja uzimajući u obzir potrebe agencija za lakšim upravljanjem podataka o nekretninama, te se može koristiti za dodavanje i evidentiranje nekretnina i radnika agencije.

Sistem je kreiran u formi web aplikacije. Za backend tehnologije korišten je PHP-ov framework Laravel, na frontend dijelu su korišteni Laravel Livewire i Blade templating engine i baza podataka je u MySQL-u. Razlog za ove tehnologije jeste primarno korištenje open source tehnologije baziranih na PHP programskom jeziku, dok MySQL je dobro podržan za rad u Laravel aplikacijama.

Agentima agencije je omogućen pristup sistemu u kojem oni imaju mogućnosti pregleda svih dostupnih nekretnina kao i nekretnina u pripremi, te vođenja nekretnina koje su njima dodjeljene. Adminu sistema je napravljen zaseban pristup aplikaciji, gdje ima preglede svih agenata i svih nekretnina koje su bile u sistemu, te pristup admin komandnoj tabli koja prikazuje tabelu akcija izvršenih nad nekretninama, graf nekretnina te graf prodaje agencije. Običnom korisniku je omogućen pregled dostupnih nekretnina, detaljan prikaz jedne nekretnine sa svim podacima o istoj, te je prisutan i sistem za pregled i preporuku nekretnina.

2. WEB TEHNOLOGIJE

Za razvoj ove web aplikacije upotrijebljena je kombinacija modernih tehnologija i razvojnih alata kako bi se osiguralo efikasno i skalabilno rješenje. Ovaj dio rada će pružiti pregled ključnih tehnologija koje su korištene za Frontend i Backend dijelove prilikom razvoja web aplikacije, zajedno sa sistemom upravljanja bazom podataka. Izbor tehnologija bio je vođen njihovom sposobnošću da omoguće što brži razvoj, jednostavnu mogućnost održavanja i kompatibilnost sa trenutnim industrijskim standardima i zahtjevima.

2.1 PHP i Laravel

U ovom projektu, PHP je korišten kao osnovni programski jezik, dok je Laravel, open source PHP framework, odabran za izradu backend dijela aplikacije. Laravel prati MVC (Model-View-Controller) arhitekturni obrazac i nudi bogat set alata koji ubrzavaju razvoj, poput rutiranja, autentifikacije (Laravel Breezea starter kit), Eloquent ORM-a i migracija baze podataka.[7] Laravel je odabran zbog svoje elegantne sintakse, skalabilnih mogućnosti i besprijeckorne integracije sa PHP programskim jezikom. Pomoću Laravel Breeze starter kit-a, koji je jednostavna i minimalna implementacija svih Laravelovih autentifikacijskih feature-a (login, registracija, verifikacija, autentifikacija, reset i potvrda šifre računa), te ostalih ključnih backend funkcionalnosti. Jedna od prednosti korištenja Laravel radnog okruženja jest lagano postavljanje novog projekta sa svim funkcionalnostima koje većina web aplikacija ima. Kombinacija PHP-a i Laravel-a omogućila je brz, efikasan i skalabilan razvoj backend dijela aplikacije.

2.2 Livewire

Livewire je full-stack frontend radno okruženje za Laravel koji omogućava programerima izradu dinamičnih, reaktivnih interfejsa bez pisanja JavaScripta. Omogućava izradu interaktivnih komponenti koristeći PHP, što pojednostavljuje razvoj frontenda i osigurava čvrstu integraciju s Laravelom. Livewire je odabran za ovaj projekat kako bi se izbjegla moguća kompleksnost koja dolazi sa integracijom dodatnog JavaScript koda radi postizanja potrebne real-time interaktivnosti frontend dijela aplikacije. Nakon razmatranja mnogih opcija za postizanje željene interaktivnosti i efekata živih i interaktivnih komponenti na frontend dijelu aplikacije, odlučio sam se za korištenje Livewire open-source koji mi je omogućio da pravim dinamične UI (User Interfejs) komponente bez da napuštam PHP jezik i Blade templating engine, obe tehnologije koje su sastavni dio Laravel-a. Korišten je za izradu filteri pretrage, upload-aa i prikaz slika, izradu formi i validacije istih u realnom vremenu. Jedan od izazova bilo je razumijevanje Livewire-ovih koncepta kao što su Morphing, Hydration i Nesting. Morphing je inteligentni način na koji Livewire update-a DOM (Document Object Model) browsersa i način na koji prepoznaće koje dijelove komponenti treba ažurirati nakon requesta ka serveru. Hydration je proces ažuriranja komponenti nakon svakog zahtjeva komponente prema serveru. Pošto je svaki AJAX zahtjev koji Livewire napravi prema serveru “stateless” (što znači da nema neki long running backend proces koji pamti state komponente), Livewire mora ponovo

stvoriti posljednje poznat state komponente prije nego što uradi ikakvo ažuriranje. To radi tako što “uslika” PHP komponentu (napravi se JSON sa što više promjenjivog state-a komponente) nakon svakog update-a komponente na serveru tako da bi ona mogla biti ponovo napravljena na idućem requestu. Proces “slikanja” komponente se naziva “dehydration”, dok proces ponovnog kreiranja komponente od JSON state-a se naziva “hydration”. Livewire se pokazao kao skalabilan i moćan alat za poboljšanje korisničkog iskustva bez ugrožavanja efikasnosti razvoja.

2.3 Blade templates

Blade je Laravelov lightweight, ali robustan sistem za predloške, dizajniran za izradu jednostavnih, ponovo upotrebljivih i lagano integrisanih HTML komponenti koje odlično surađuju u PHP okruženju kao što je Laravel i Livewire. Blade engine, omogućava pisanje PHP koda unutar Blade file-ova, te nam pruža mogućnost korištenja PHP kontrolnih struktura, kao što su petlje i kondicionali, za prikaz html koda ili nekih drugih komponenti koristeći “direktive”. U ovom projektu, Blade je korišten za renderovanje korisničkog interfejsa, osiguravajući dosljedan izgled i dizajn u cijeloj aplikaciji. Integracija Blade-a sa Livewire-om omogućila je dinamično renderovanje sadržaja bez potrebe za dodatnim JavaScript frameworcima. Jednostavnost i fleksibilnost Blade komponenti koje su sastavni dio Livewire komponenti, učinili su odličan izbor tehnologija za pravljenje, ažuriranje, te preglednost frontend dijela projekta ove web aplikacije.

2.4 Visual Studio Code

Visual Studio Code, koji je se obično naziva VS Code, je uređivač izvornog koda koji je razvio Microsoft sa Electron Framework-om, za Windows, Linux i macOS.[8] Zasnovan je na Electron framework-u, koji se koristi za razvoj Node.js web aplikacija koje rade na Blink layout engine-u. Visual Studio Code uključuje osnovnu podršku za najčešće programske jezike. Ova osnovna podrška uključuje isticanje sintakse, podudaranje zagrada, konfigurable blokove koda itd. Podrška za dodatne jezike može biti preuzeta besplatno dostupnim ekstenzijama na VS Code Marketplace-u. Visual Studio Code se može proširiti preko ekstenzija koje su dostupne unutar centralnog „repositorya“ (engl. „spremište“). Ovo uključuje dodatke uređivaču koda i podršku za različite programske

jezike. „*Source control*“ je ugrađena karakteristika Visual Studio Code. Jedna od najvećih prednosti Visual Studio Code razvojnog okruženja jeste što ima namjensku karticu unutar meni-a gdje korisnici mogu pristupiti postavkama „*source control-a*“ funkcionalnosti i vidjeti promjene napravljene na trenutnom projektu. Da bi se mogla koristiti ova funkcija, Visual Studio Code mora biti povezan sa bilo kojim podržanim sistemom „*source control-a*“ (Git, Apache Subversion, Perforce, itd.), što je podržano od strane softvera, gdje jednostavnim klikom na instaliranje ekstenzije možemo omogućiti ove funkcionalnosti. Ovo omogućava korisnicima da kreiraju „*repository*“, kao i da prave „*push*“ i „*pull*“ zahtjeve direktno iz programa Visual Studio Code.

3. COMPOSER PAKETI

Composer je dependency manager (alat za upravljanje zavisnostima) za PHP programski jezik, koji omogućava programerima efikasno upravljanje i instalaciju open i close sourced paketa.[9] Pojednostavljuje proces integracije eksternog koda u projekat automatskim rješavanjem zavisnosti i osiguravanjem kompatibilnosti između paketa. Composer koristi “composer.json” fajl za definisanje zavisnosti u projektu i “composer.lock” fajl kako bi osigurao dosljedne instalacije u različitim okruženjima. U ovom projektu, Composer je igrao ključnu ulogu u upravljanju PHP zavisnostima (dependencies), uključujući komponente Laravel frameworka i dodatnim paketima koji su proširili funkcionalnost aplikacije.

Neki od paketa koje sam integrisao u projekat koristeći Composer radi poboljšanja funkcionalnosti softvera i unapređenja vremena razvoja istog su: “Spatie/laravel-permission”, “Spatie/laravel-medialibrary” i “Intervention/image-laravel”.

3.1 Spatie/laravel-permission

Ovaj paket pruža jednostavan i fleksibilan način za upravljanje ulogama (roles) i pravima (permissions) korisnika u Laravel aplikacijama.[10] On omogućava developerima da definišu uloge (npr. admin, urednik, korisnik, menadžer) i dodjele specifične dozvole svakoj ulozi. U ovom projektu, ovaj paket je korišten za implementaciju skalabilnog i pouzdanog sistema kontrole pristupa među autentificiranim korisnicima. Na primjer:

- Kreirane su uloge “admin” i “agent”, svaka sa specifičnim dozvolama (npr. “kreiranje nekretnine”, “brisanje nekretnine”, “uređivanje nekretnine”, “uređivanje agenta” itd..).
- Korišten je middleware za ograničavanje pristupa određenim rutama na osnovu korisničkih uloga i dozvola.
- Korištene su blade “if” kondicionalne direktive za ograničavanje prikaza nekih komponenti na stranici, najčešće buttona, na osnovu korisnikovih uloga i dozvola.
- Besprijekorna intergracija paketa sa Laravel-ovim sistemom autentifikacije olakšala je upravljanje i provođenje pravila pristupa cijeloj aplikaciji.

3.2 Spatie/laravel-medialibrary

Ovaj paket pojednostavljuje proces upravljanja medijskim datotekama (npr. slike, videi i dokumenti) u Laravel aplikacijama.[11] Pruža alate za upload, organizaciju i povezivanje medijskih datoteka sa modelima u bazi podataka. U ovom projektu, ovaj paket je korišten za rukovanje datotekama koje su korisnici uploadovali, kao što su profilne slike i slike nekretnina. Ključne karakteristike ovog paketa uključuju:

- Povezivanje medijskih datoteka sa specifičnim modelima (npr. povezivanje slike sa korisnikom, povezivanje slika sa nekretninom).
- Generisanje thumbnail-a i promjena meta podataka slika prilikom upload-a.
- Pohranjivanje medijskih datoteka na struktuiran efikasan način po modelima i kolekcijama, bilo lokalno ili na neke cloud storage “bucket-e” ili service poput AWS S3.

3.3 Intervention/image-laravel

Ovaj paket je moćna open-source biblioteka za rukovanje i izmjenu slika u PHP okruženjima, integrirana sa direktnom podrškom za Laravel projekte.[12] Pruža jednostavan API za manipulaciju slikama, kao što su promjena veličine, izrezivanje slika i dodavanje vodenog žiga. U ovom projektu, ovaj paket je korišten za obradu i optimizaciju slika koje su korisnici uploadovali. Na primjer:

- Profilne slike su automatski smanjenje kako bi odgovarale specifičnim dimenzijama definisanim na frontendu.
- Generisani su thumbnail-i za brži prikaz slika na frontend-u
- Slike su kompresirane i promijenjene u tip file-a “.webp” kako bi im se smanjila

veličina i kako bi se brže učitavale na stranicu bet gubitka kvalitete.

Korištenje Composera i third-party paketa značajno je ubrzalo razvoj i poboljšalo funkcionalnosti i potencijal za skalabilnost aplikacije. Korištenjem regularno održavanih i široko korištenih paketa, izbjegnut je proces samostalnog rješavanja problema koji su developeri već imali i istovremeno su osigurana visokokvalitetna i testirana rješenja za uobičajene zadatke. Osim toga, Composer-ovo upravljanje zavisnostima osiguralo je da svi paketi rade zajedno, što je smanjilo rizik od konflikata i potencijalnih problema sa kompatibilnošću.

4. TEHNOLOGIJE ZA BAZU PODATAKA

4.1 MySQL

MySQL je open-source sistem za upravljanje relacionim bazama podataka. Njegova uloga jeste skladištenje i upravljanje struktuiranim podacima u SQL baziranim relacionim bazama podataka. Kao relaciona baza podataka, MySQL skladišti podatke u tabele redova i kolona koje su organizovane po šemama. Šeme definišu kako su podaci organizovani, skladišteni i opisuje relacije između raznih tabela.[13] Sistem za upravljanje relacionim bazama podataka kao MySQL radi zajedno sa operativnim sistemom da implementira relacione baze podataka u sistem za skladištenje podataka računara, upravlja korisnicima, dozvoljava mrežni pristup i testira integritet baze podataka te radi na kreiranju, ažuriranju i skladištenju backup-a. MySQL ima samostalne klijente koji dozvoljavaju korisnicima da imaju direktnu interakciju sa MySQL bazom podataka koristeći jezik SQL, ali najčešće MySQL se koristi sa drugim programima (SQLite, MySQL Workbench, DBeaver) da se implementira u aplikacije kojima trebaju sposobnosti relacione baze podataka.

U ovom projektu, MySQL je korišten kao glavna baza podataka zbog:

- Brzine i efikasnosti - Indekstiranje i optimizacija upita omogućava brze operacije nad podacima.
- Sigurnosti - MySQL podržava različite metode autentifikacije i enkripcije podataka.
- Podrške za PHP/Laravel - Laravel ima nativnu podršku za MySQL kroz Eloquent ORM (Object-Relational-Mapper) i Query Buider, što je omogućilo jednostavno i efikasno upravljanje i komunikaciju sa bazom podataka.

4.2 DBeaver

DBeaver je besplatan i open-source alat za baze podataka, dizajniran za developere, programere, administratore baza i analitičare podataka.[14] Pruža korisnički prijateljsko sučelje za upravljanje i interakciju (GUI) s različitim sistemima podataka, uključujući MySQL, PostgreSQL, SQLite, Oracle i mnoge druge. DBeaver podržava širok spektar funkcija, poput SQL uređivanja, pregleda podataka, upravljanja šemama i izvršavanja upita, što ga čini odličnim alatom za razvoj i održavanje baza podataka.

DBeaver je odabran jer je jedan od najpopularnijih i regularno održavanih open-source projekata ovog tipa, koji je jednostavan za korištenje te ima bogat skup funkcionalnosti. Ključne prednosti su:

- Podrška mnogim OS - DBeaver je dostupan za Windows, MacOS i Linux osiguravajući kompatibilnost u različitim razvojnim okruženjima.
- Prijatno korisničko sučelje - Intuitivni interfejs pojednostavljuje zadatke na softveru, poput pregleda tabela, uređivanja podataka, i pisanje SQL upita.
- Napredne funkcionalnosti - DBeaver nudi napredne mogućnosti poput poređenja šema, izvoza/uvoza podataka i generisanja ER dijagrama, što je ključno za razvoj baza podataka.

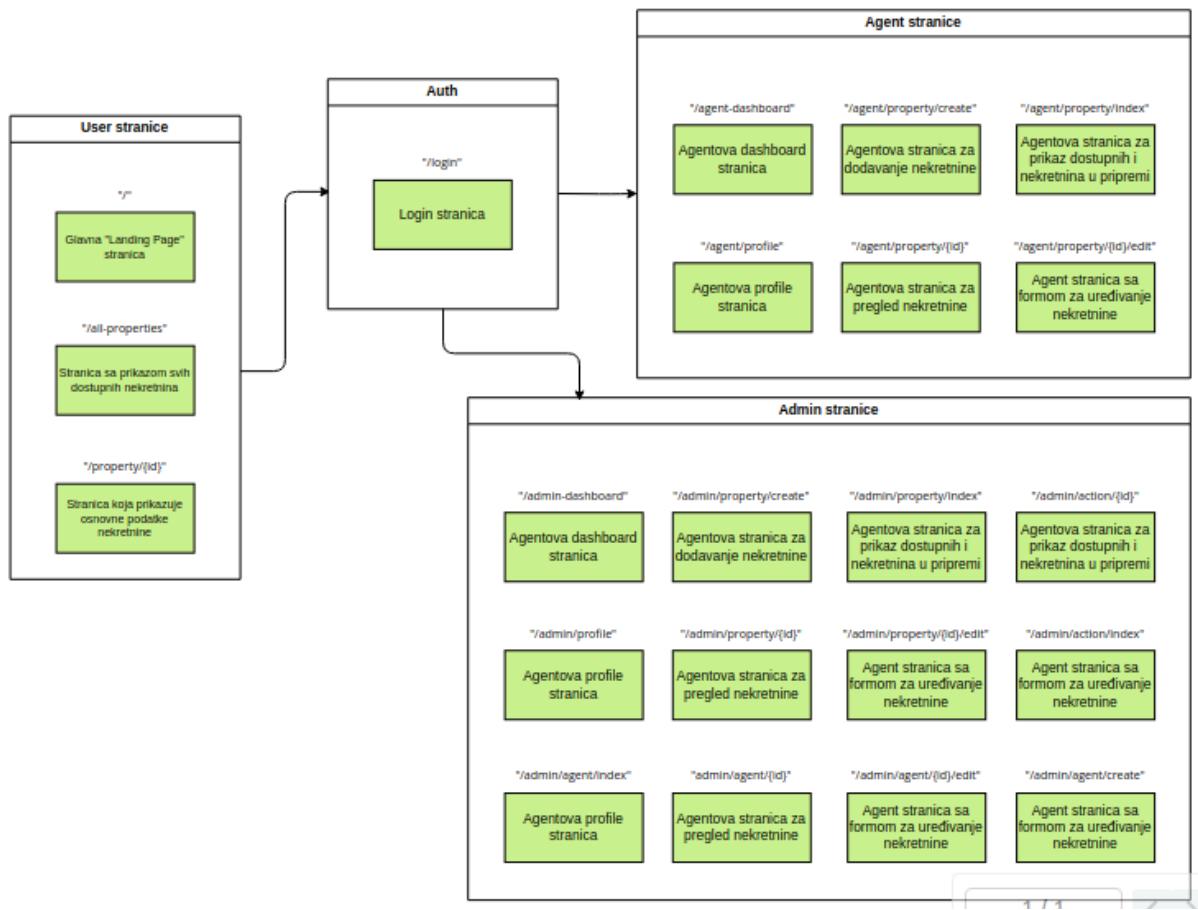
DBeaver se pokazao kao ključni alat za upravljanje i razvoj baze podataka u ovom projektu. Njegova svestranost, jednostavnost korištenja i moćne funkcionalosti omogućile su pojednostavljivanje zadataka vezanih uz baze podataka i doprinijele efikasnosti cijelog razvojnog procesa.

5. IMPLEMENTACIJA OGLEDNE WEB APLIKACIJE

U ovom dijelu diplomskog rada detaljno ćemo opisati cijelokupni proces implementacije web aplikacije za digitalizaciju procesa u agenciji za nekretnine. Cilj je bio razviti potpuno funkcionalnu web aplikaciju koristeći moderne tehnologije kako bi se osigurala što veća skalabilnost, performanse i mogućnost što jednostavnijeg održavanja. Objasnit ćemo način implementacije određenih funkcionalnosti na Backend-u i Frontend-u. Implementacije je podijeljena u nekoliko faza gdje su najvažnije podijeljene na: kreiranje strukture aplikacije uz pomoć dijagrama, kreiranje baze podataka, konfiguracija autentifikacije, konfiguracija i osnovna postavka backenda, konfiguracija i osnovna postavka frontenda, razvoj funkcionalnosti na backendu, razvoj korisničkog sučelja na frontendu.

5.1 Planiranje osnovne strukture i rasporeda

Prilikom izrade web aplikacije, osnovna struktura se sastoji iz različitih stranica s kojima će korisnici komunicirati. Svaka stranica služi određenoj svrsi, vodeći korisnike kroz funkcionalnosti aplikacije u zavisnosti da li su oni autenticirani korisnici (uposlenici agencije) ili potencijalni klijenti. Proces planiranja počinje određivanjem kakvo korisničko iskustvo želimo da obezbjedimo korisnicima i identifikacijom osnovnih radnji koje će korisnici morati izvršiti i biti u mogućnosti da izvrše u zavisnosti od njihovih uloga u našoj aplikaciji. Odlučeno je da osnovna struktura se sastoji iz stranica za autentifikaciju koje mogu biti stranica za login i registraciju, glavna stranica koja će služiti kao "*landing page*" stranica za ovu web aplikaciju koja će biti pretežno usmjerena potencijalnim klijentima, te stranica za administratore i registrovane korisnike (agente) koju nakon uspješnog logina čini "*dashboard page*". Stranica za administratore je stranica koja će administratorima pružati pregled svih agenata sa određenim funkcionalnostima koje će se moći primjeniti na te korisnike, te će administrator imati pregled i pristup svim nekretninama koje su pod agencijom te će imati mogućnosti izvršiti određene radnje nad istim. Stranica za registrovane korisnike će biti podijeljena na nekoliko sporednih stranica. Na slici 1. su pokazane relacije između stranica zajedno sa planiranim putanjama za svaku stranicu. Ona predstavlja grafički prikaz strukture ove ogledne web aplikacije.

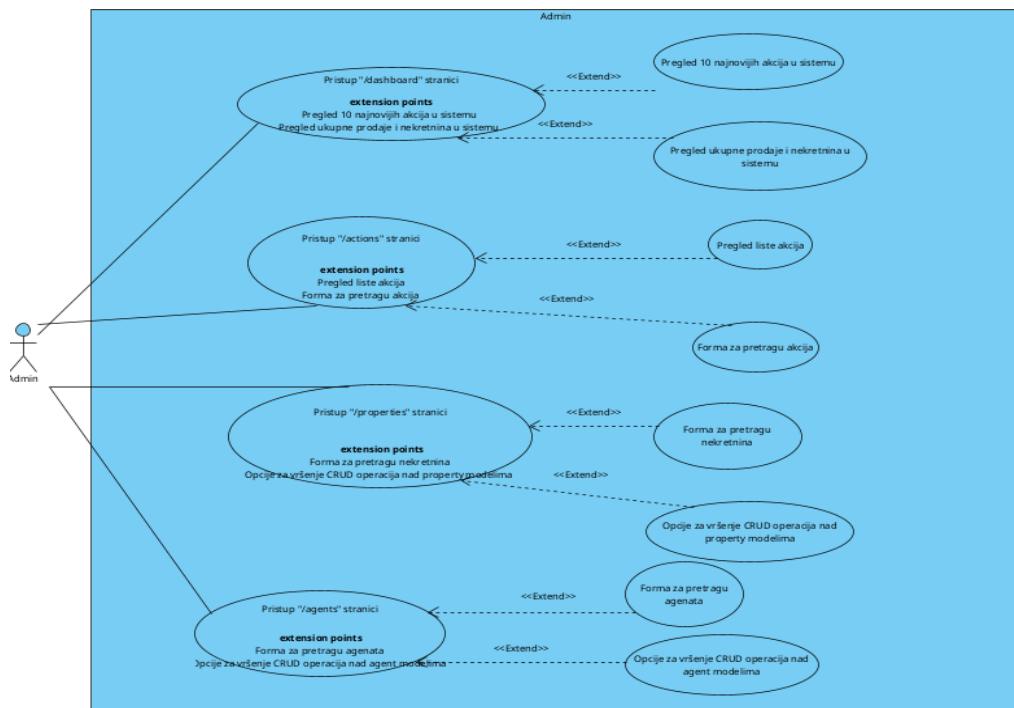


Slika 1. Prikaz osnovne strukture web stranice sa definisanim putanjama

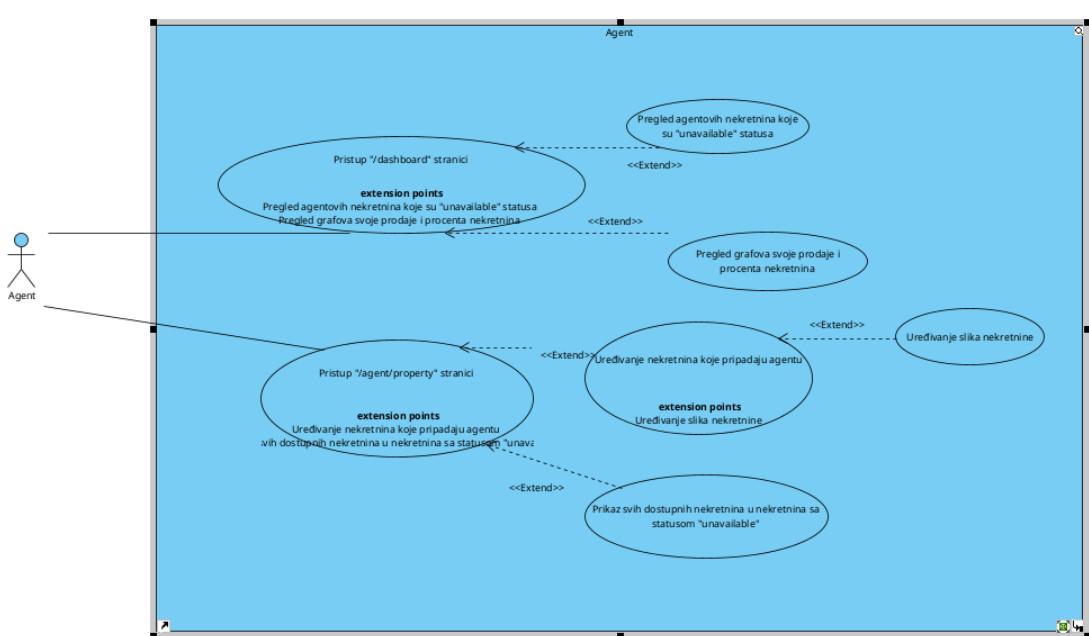
5.2 Use Case dijagrami

"Use Case" (engl. "slučaj upotrebe") dijagrami su dijagrami koji sadrže detalje korisnika naše aplikacije (definisanih kao akteri) i njihove interakcije sa funkcionalnostima aplikacije.[15] Koristi se za predstavljanje ciljeva interakcije korisnika sa sistemom, definiranje i organiziranje funkcionalnih zahtjeva u sistemu, određivanje konteksta akcija i zahtjeva sistema te modeliranje osnovnog toka događaja u sa strane aktera u aplikaciji. Za kreiranje "Use Case" dijagrama potrebno je koristiti skup specijaliziranih simbola i konektora a to su: 1. Slučajevi upotrebe - ovali koji predstavljaju različite funkcije koje korisnik može koristiti; 2. Akteri - figure za korisnike koji komuniciraju sa slučajevima upotrebe; 3. Asocijacije - linije koje povezuju aktere i slučajeve upotrebe, prikazujući interakcije; 4. Granica sistema - okvir koji definiše opseg sistema, sve izvan je van dometa. 5. Paketi - grupacije elemenata, prikazane kao fascikle datoteka, za organizovanje komponenti. Efikasan dijagram može pomoći u:

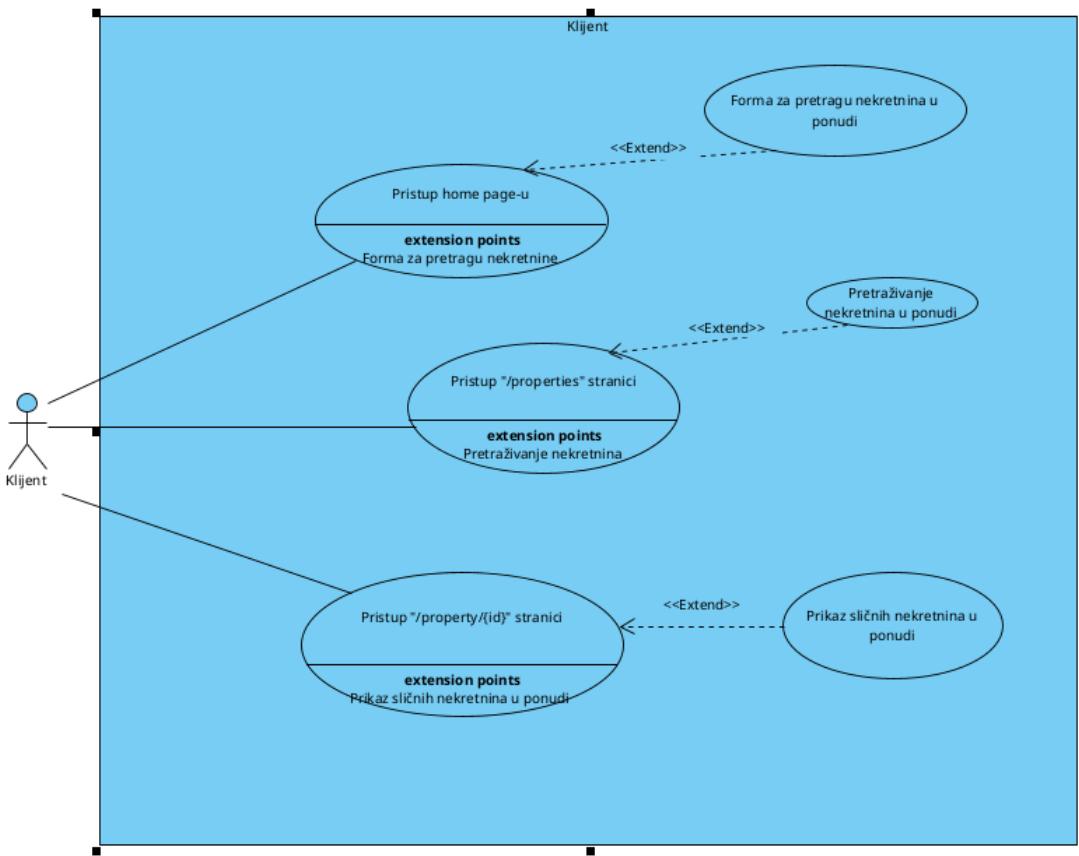
- Scenarijima u kojima naša aplikacija komunicira sa ljudima, organizacijama, eksternim aplikacijama ili korisnicima
- Ciljevima kojim naš sistem pomaže korisnicima (akterima) da ih postignu
- Definisanju opsega naše aplikacije



Slika 2. Prikaz use case dijagrama za administratora



Slika 3. Prikaz use case dijagrama za agente prvi dio



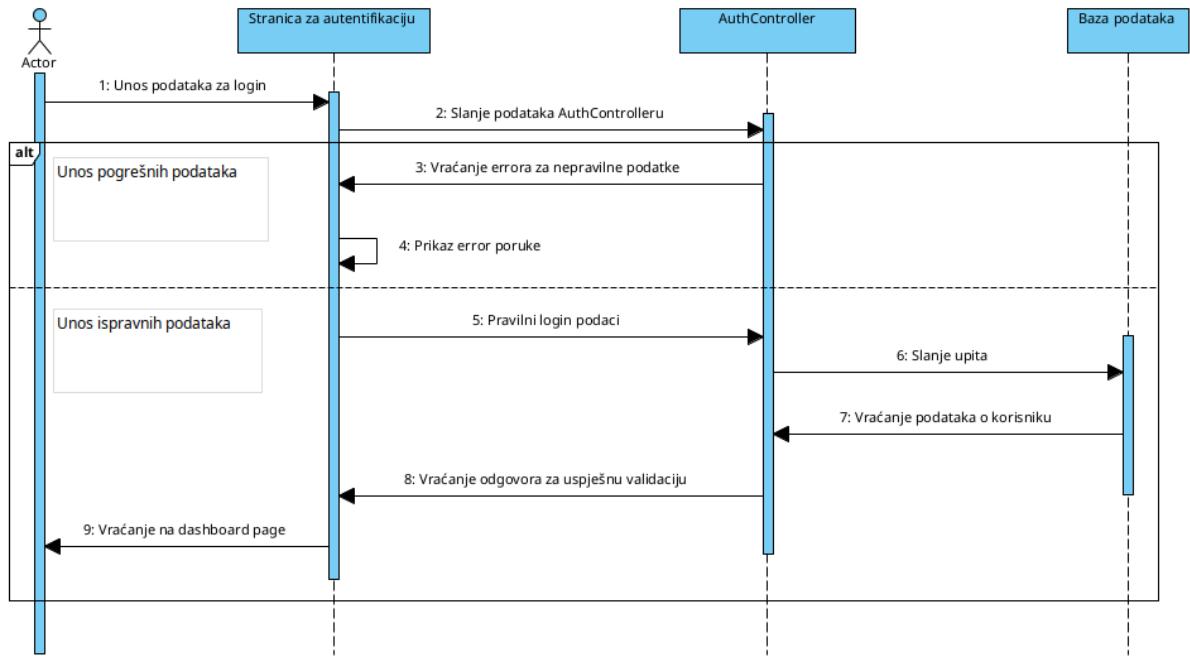
Slika 4. Prikaz use case dijagrama za klijente

Na Use Case dijagramima su navedene glavne funkcionalnosti ogledne web aplikacije. Dijagrami su rasčlanjeni na tri dijela gdje jedan dio prikazuje use case za administratora aplikacije (slika 2.), drugi dio prikazuje use case za agente (slika 3.) i zadnji dio prikazuje use case dijagram za potencijalne klijente (slika 4.). Zajednički use case za administratore i korisnike su Login (za koji se koriste email i šifra) jer se koristi ista poslovna logika na oba case-a.

5.3 Dijagrami sekvenci

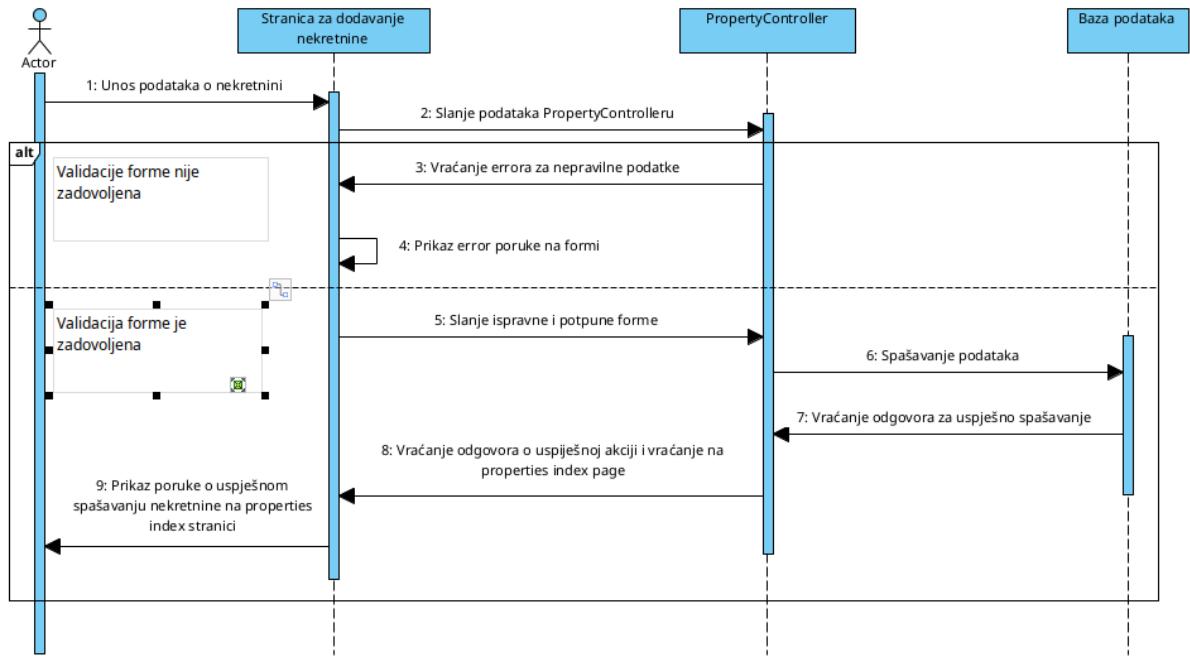
Dijagram sekvence je vrsta dijagrama interakcije jer opisuje kako i kojim redoslijedom grupa objekata radi zajedno.[16] Ove dijagrame koriste programeri softvera i poslovni profesionalci da razumiju zahtjeve za novi sistem ili da dokumentuju postojeći proces. Dijagrami sekvenci su ponekad poznati kao dijagrami događaja. Prednosti dijagrama sekvence su sljedeće: 1. Predstavlja detalje slučaja upotrebe UML-a; 2. Modelira logiku sofisticirane procedure, funkcije ili operacije; 3. Pruža pogled kako objekti i komponente međusobno djeluju kako bi izvršili proces; 4. Planiranje i razumijevanje detaljne funkcionalnosti postojećeg ili budućeg scenarija. Sekvencijalni dijagrami su idealni za sljedeće scenarije: 1. Scenarij upotrebe - dijagrami kako se sistem može koristiti za provjeru logike za svaki scenarij; 2. Logika metode - pomaže u istraživanju logike u bilo kojoj funkciji, proceduri ili složenom procesu; 3.

Logika usluge - preslikava metode visokog nivoa koje koriste različiti klijenti. Dijagram sekvenci koristi skup specijaliziranih simbola, strelica i poruka. Specijalizirani simboli su: 1. Simbol objekta - predstavlja klasu ili ponašanje objekta, bez atributa klase; 2. Okvir za aktivaciju - označava trajanje zadatka za objekat, s dužinom koja pokazuje potrebno vrijeme; 3. Simbol aktera: - predstavlja entitete koji su u interakciji sa sistemom; 4. Simbol paketa - sadrži interaktivne elemente, označen oblik pravougaonika; 5. Simbol linije života - prikazuje protok vremena i sekvencijalne događaje za objekat, proteže prema dole; 6. Simbol petlje opcije - Modelira uslovne scenarije "ako/onda"; 7. Alternativni simbol - predstavlja izvore između međusobno isključivih sekvenci poruka. Strelice i poruke služe za pokazivanje načina na koji se informacije razmjenjuju između objekata. Strelice i poruke su: 1. Sinhrona poruka - puna linija sa punim vrhom strelice, pošiljalac čeka odgovor. Prikazuje poziv i odgovor; 2. Asinhrona poruka - puna linija sa iscrtanim vrhom strelice, nije potreban odgovor, prikazuje samo poziv; 3. Asinhrona povratna poruka - isprekidana linija sa iscrtanim vrhom strelice; 4. Poruka o asinhronom kreiranju - isprekidana linija sa iscrtanim vrhom strelice, kreira novi objekat; 5. Poruka odgovora - isprekidana linija sa iscrtanim vrhom strelice, odgovara na poziv; 6. Brisanje poruke - Puna linija sa punom strelicom, koja se završava sa X, uništava objekat.



Slika 5. Prikaz sekvencijskog dijagrama za proces autentifikacije (login)

Proces cijelokupne autentifikacije koja je implementirana za ovu oglednu aplikaciju je prikazan na slici 5. Autentifikacija ogledne web aplikacije je implementirana preko Laravel Breeze starter kit-a. Starter kit su novi projekti koje Laravel nudi koji se kreiraju sa već definisanim funkcijama i konfiguracijom. Laravel Breeze je starter kit koji dolazi sa predefinisanom autentifikacijskom konfiguracijom koristeći Laravelove Auth i Session fasade. Laravel kao framework dolazi sa alatima koji olakšavaju implementiranje autentifikacije bez sigurno i lagano. U njegovoj srži, Laravelovi autentifikacijski objekti su definisani od strane "guard-ova" i "provider-a". Guardovi definišu kako se useri autentificiraju na svakom requestu, npr. Laravel dolazi sa "session" guard-om koji održava state korisnika koristeći session storage i cookies, dok provideri definišu kako se useri preuzimaju iz naše baze podataka. Autentifikacijska konfiguracija novog projekta je locirana u 'config/auth.php' file-u. U ovom file-u se nalazi nekoliko dobro dokumentovanih opcija za ažuriranje i mijenjanje "ponašanja" Laravelovog Auth servisa. Proses autentifikacije ide redoslijedom da korisnik unosi tražene podatke u formu za autentifikaciju, te Laravelovi servisi provjeravaju prisustvo emaila u bazi, te onda unesenu šifru za profil sa tim emailom. Ako ne postoji profil sa tim email-om u bazi ili unesena šifra za uneseni email ne odgovara, sistem vraća odgovarajuću poruku.

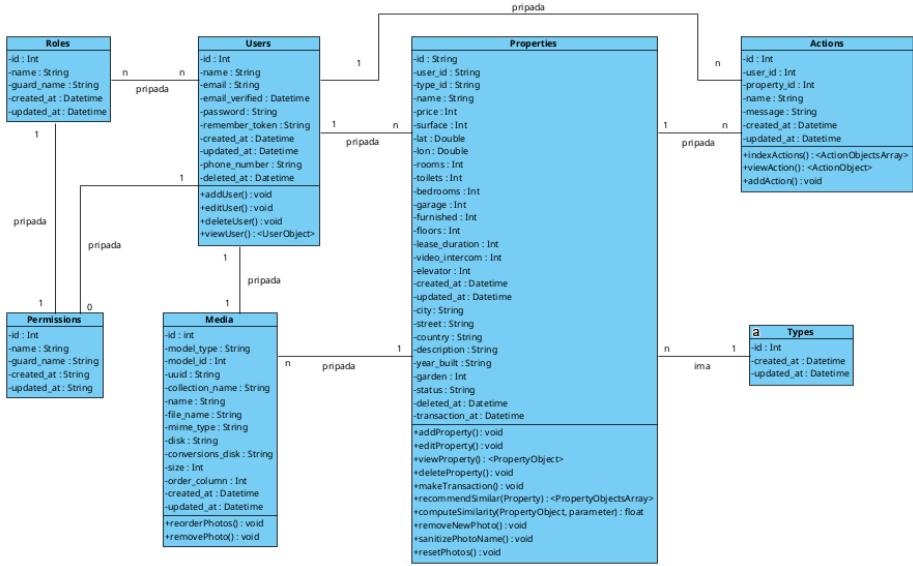


Slika 6. Prikaz sekvencijalnog dijagrama za proces kreiranja i spašavanja nove nekretnine u sistem

Na slici 6. možemo vidjeti sekvencijalni dijagram koji definiše proces kreiranja, validacije i spašavanje nove nekretnine u sistemu. Odlaskom autentificiranog korisnika na stranicu za dodavanje nekretnine u sistem, korisniku se prikazuje forma za unos podataka o nekretnini koja se sastoji od unosnih, da-ne i izbornih polja, mape za odabir lokacije nekretnine i sekcije za slike, gdje korisnik može unositi slike nekretnine. Nakon unosa željenih podataka i klikom na dugme “Create Property”, vrši se validacija unesenih podataka. Imamo dvije situacije, da su uneseni podaci validni i da nisu validni. U slučaju da podaci nisu validni, kontroler vraća error i ispisuje odgovarajuće poruke za ona unosna polja čiji unosni podaci nisu validi. Nakon što se isprave pogrešno uneseni podaci i ponovo klikne dugme za kreiranje novog agenta, ponovo se vrši validacija podataka, nakon što podaci prođu validacijski sistem, kreira se novi Property objekat koji se ispunjava sa podacima iz forme, te nakon toga se spašava u bazi podataka. Nakon uspješnog spašavanja podataka, kontroler vraća success message, te se korisnik re-routea na “properties-index” sa success porukom koja mu se prikazuje na ekranu, da je nova nekretnina uspješno kreirana.

5.4 Klasni dijagram

Dijagrami klase su jedan od najkorisnijih tipova dijagrama jer jasno prikazuju strukturu određene aplikacije modeliranjem njenih klasa, atributa, operacija i odnosa između objekata.[17] Oblik klase sastoji se od pravougaonika sa tri reda. Gornji red sadrži ime klase koji je uvijek obavezan, srednji red sadrži atribute klase, a donji dio sadrži metode ili operacije koje klasa može koristiti. Klase i podklase su grupisane zajedno da pokažu statički odnos između svakog objekta. Dijagram klasa se koristi za prikaz modela podataka za informacione sisteme, bez obzira koliko jednostavni ili složeni, bolje razumijevanje općeg pregleda šeme aplikacije, vizuelno izražavanje svake specifične potrebe aplikacije, kreiranje detaljnih grafikona koji naglašavaju bilo koji specifični kod koji je potreban za programiranje i implementaciju u opisanu strukturu, navođenje opisa tipova koji se koriste u aplikaciji koji se kasnije prenose između njenih komponenti, neovisno o implementaciji. Dodatne komponente klasnog dijagrama su: 1. Signali - jednosmjerna, asinhrona komunikacija između aktivnih objekata; 2. Tipovi podataka - definira vrijednosti podataka, uključujući primitivne tipove i nabranja; 3. Paketi - organizuje povezane elemente, simbolizirane kao pravougaonik s karticama; 4. Interfejsi - definira operacije i atribute, koji predstavljaju skup ponašanja; 5. Enumeracije - korisnički definirani tipovi podataka sa specifičnim vrijednostima; 6. Objekti - instance klase koje predstavljaju stvarne ili prototipne entitete; 7. Artefakti - predstavljaju komponente kao što su datoteke, baze podataka i softver. Interakcije kod klasnog dijagrama predstavljaju veze koje mogu postojati u klasama i dijagramima objekta. Ključne interakcije su: 1. Nasljeđivanje - podklasa nasljeđuje atribute i metode od superklase, prikazane punom linijom i zatvorenom šupljom strelicom; 2. Dvosmjerna asocijacija - obje klase su svjesne jedna druge, predstavljene ravnom linijom između njih; 3. Jednosmjerna asocijacija - jedna klasa je svjesna druge, predstavljena linijom sa otvorenom strelicom koja pokazuje od jedne do poznate klase.



Slika 7. Prikaz klasnog dijagrama

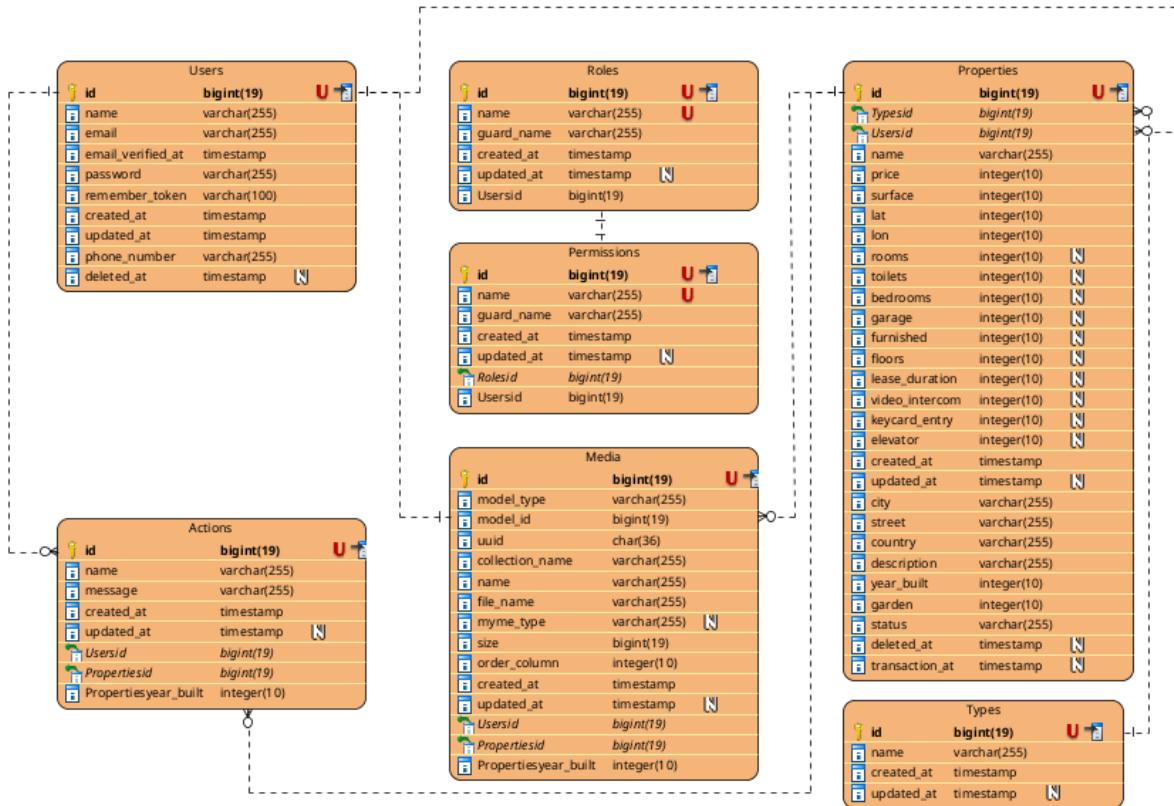
Ova struktura klasnog dijagrama, prikazana na slici 7., pokazuje objektno-orientisani strukturu ogledne web aplikacije i prikazuje detalje interakcije između klasa. Svaka klasa se sastoji od imena, atributa i operacija. Svi atributi za svaku klasu su označeni kao privatni "(-)" dok sve operacije za svaku klasu su označene kao javne "(+)". Sve klase su međusobno povezane sa vezom asocijacije na kojoj je postavljen kardinalitet.

5.5 ER dijagram

Dijagram odnosa entiteta (ER - Entity Relationship) je tip dijagrama toka koji prikazuje kako se „entiteti” kao što su ljudi, objekti ili koncepti međusobno odnose unutar sistema.[18] ER dijagrami se najčešće koriste za dizajniranje ili otklanjanje grešaka u relacionim bazama podataka u oblastima softverskog inženjeringu, poslovnih informacionih sistema, obrazovanja i istraživanja. Poznati i kao ER ili ER modeli, oni koriste definirani skup simbola kao što su pravougaonici, rombovi, ovali i linije povezivanja kako bi opisali međusobnu povezanost entiteta, odnosa i njihovih atributa. Oni odražavaju gramatičku strukturu, sa entitetima kao imenicama i odnosima kao glagolima. Najviše se koriste za: 1. Dizajn baze podataka - ER dijagrami modeliraju relacijske baze podataka, i logički i fizički, često služeći kao prvi korak u dizajnu softvera i baze podataka; 2. Rješavanje problema - koristi se za analizu i rješavanje problema logike baze podataka ili implementacije; 3. Poslovni informacioni sistemi - pomoći u dizajniranju baza podataka za poslovne procese, poboljšanje efikasnosti i dostupnosti

informacija; 4. Reinženjering poslovnih procesa (BPR) - pomaže u analizi i postavljanju baza podataka za reinženjering procesa; 5. Obrazovanje - korisno za planiranje obrazovnih struktura podataka za skladištenje i pronalaženje; 6. Istraživanje - ključ za kreiranje baze podataka za podršku strukturiranim analizama podataka u istraživanju. Glavne komponente ER dijagrama su entiteti, odnosi, atributi i kardinaliteti. Entitet je odrediva imenica (npr. kupac, automobil) sa pohranjenim podacima, prikazana kao pravougaonik. Tip entiteta - grupa sličnih entiteta (npr. studenti, proizvodi). Skup entiteta - grupa definirana u određeno vrijeme (npr. studenti upisani prvog dana). Kategorije entiteta: jake (definirane vlastitim atributima), slabe (potrebne su druge) i asocijativne (povezuju entitete). Ključevi entiteta: Atributi koji jedinstveno identificiraju entitete. Vrste ključeva: Super ključ - jedan ili više atributa koji definiraju entitet, Ključ kandidata - minimalni super ključ, Primarni ključ - odabrani ključ kandidata za jedinstvenu identifikaciju, Strani ključ - povezuje entitete kroz odnose. Odnos opisuje način na koji entiteti komuniciraju, kao što je student koji se registruje za kurs, prikazan u obliku dijamanta ili oznake na povezujućim linijama. Rekursivni odnos je slučaj kada se entitet više puta povezuje sa sobom. Atribut je svojstvo entiteta, obično prikazano kao oval ili krug. Deskriptivni atribut je karakteristika veze, a ne entiteta. Postoje kategorije atributa: Jednostavni - atomske vrijednosti (npr. broj telefona), Kompozitni - Sadrži podattribute, Izvedeni - izračunato iz drugog atributa (npr. starost od datuma rođenja), Više vrijednosni - više od jedne vrijednosti (npr. više telefonskih brojeva), Jedna vrijednost - samo jedna vrijednost. Tipovi se mogu kombinovati (npr. jednostavni jednostruki). Kardinalnost definira numeričke attribute odnosa između entiteta. Glavne vrste su: 1. Jedan na jedan - primjer: Jedan učenik ima jednu poštansku adresu; 2. Jedan-prema-više - primjer: Jedan student se registruje za više kurseva; 3. Mnogi-prema-mnogima - primjer: Studenti su povezani sa više članova fakulteta, i obrnuto.

Prikazi kardinalnosti prikazuju se kao pogled preko ili na istoj strani na osnovu postavljanja simbola. Postoje i ograničenja kardinalnosti koja mogu biti minimalni ili maksimalni brojevi u vezi.

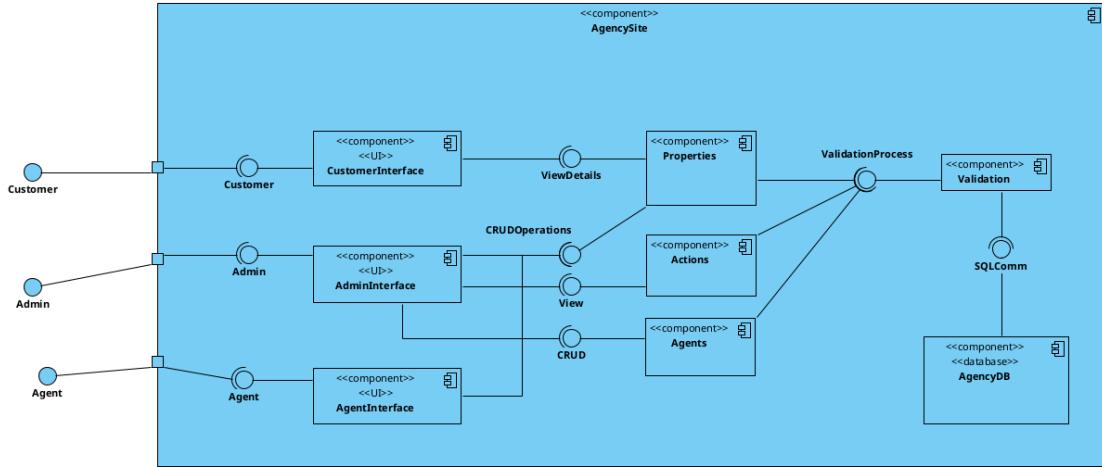


Slika 8. Prikaz ER dijagrama

Na slici 8. koja prikazuje cijelokupni ER dijagram, definisani su entiteti i relacije koji učestvuju u realizaciji baze podataka ove ogledne web aplikacije. Struktura baze podataka je kreirana da upravljanje i održavanje bude što jednostavnije, spremanje podataka efikasnije i da obezbjedi skalabilnost ukoliko bude potrebno u budućnosti. Svaki entitet sadrži primarni ključ "id", atribut "created_at" i "updated_at". Atribut "id" je generisan automatski jer je u migration file-u definisan kao auto-increment, što znači da će se pri kreiranju novog zapisa kreirati auto-incrementni "big int", što će osigurati da vrijednost atributa "id" unutar entiteta bude uvijek unikatna. Atributi "created_at" i "updated_at" se koriste za označavanje kada određena instanca u bazi bude kreirana i ažurirana. Entitet "Types" je povezan sa entitetom "Properties" vezom jedan-na-jedan(nekretnina može imati samo jednu tip, tj. može biti samo jednog tipa), dok entitet "Users" je povezan sa entitetima "Properties", "Actions" vezom jedan-prema-više(agent može imati više nekretnina i može napraviti više akcija). Tabele "Permissions" i "Roles" su tabele koja nisu povezane sa bilo kojom drugom tabelom. Važno je napomenuti da šifre korisničkih računa koje će biti spremljene unutar baze podataka će biti hash-irane jer smo u User

modelu definisali casts metodu koja automatski hashira password prije nego što ga store-a u bazu u Users tabelu.

5.6 Dijagram komponenti



Slika 9. Prikaz dijagrama komponenti

Na slici 9. je prikazan dijagram komponenti koji opisuje arhitekturu sistema aplikacije, koji modelira web aplikaciju agencije. Sistem je organizovan u više komponenti koje predstavljaju korisničke interfejsе i logiku poslovanja, kao i bazu podataka. U dijagramu su prikazane glavne uloge (akteri) te njihove interakcije sa sistemom. Vidimo da svaka vrsta aktera (korisnika) ima pristup zasebnom korisničkom sučelju preko kojih imaju pristup aplikaciji. Svako sučelje ima ograničeni set funkcionalnosti koje su dostupne korisnicima u zavisnosti od njihove uloge u sistemu. Preko operacija koje su u vezi sa interfejsima i sistemskim komponentama Agents, Actions i Properties, vidimo funkcionalnosti koje su omogućene akterima da vrše nad spomenutim komponentama. Nakon pozivanja operacija, sistemske komponente komuniciraju sa Validation komponentom putem porta ValidationProcess koji vrši validaciju akcije nad sistemskim komponentama koju zahtjeva akter. Uspješna validacija pozivom definisanih SQL komandi preko SQLComm porta, komunicira sa bazom podataka preko AgencyDB porta koji je naša baza.

5.7 Kreiranje aplikacije

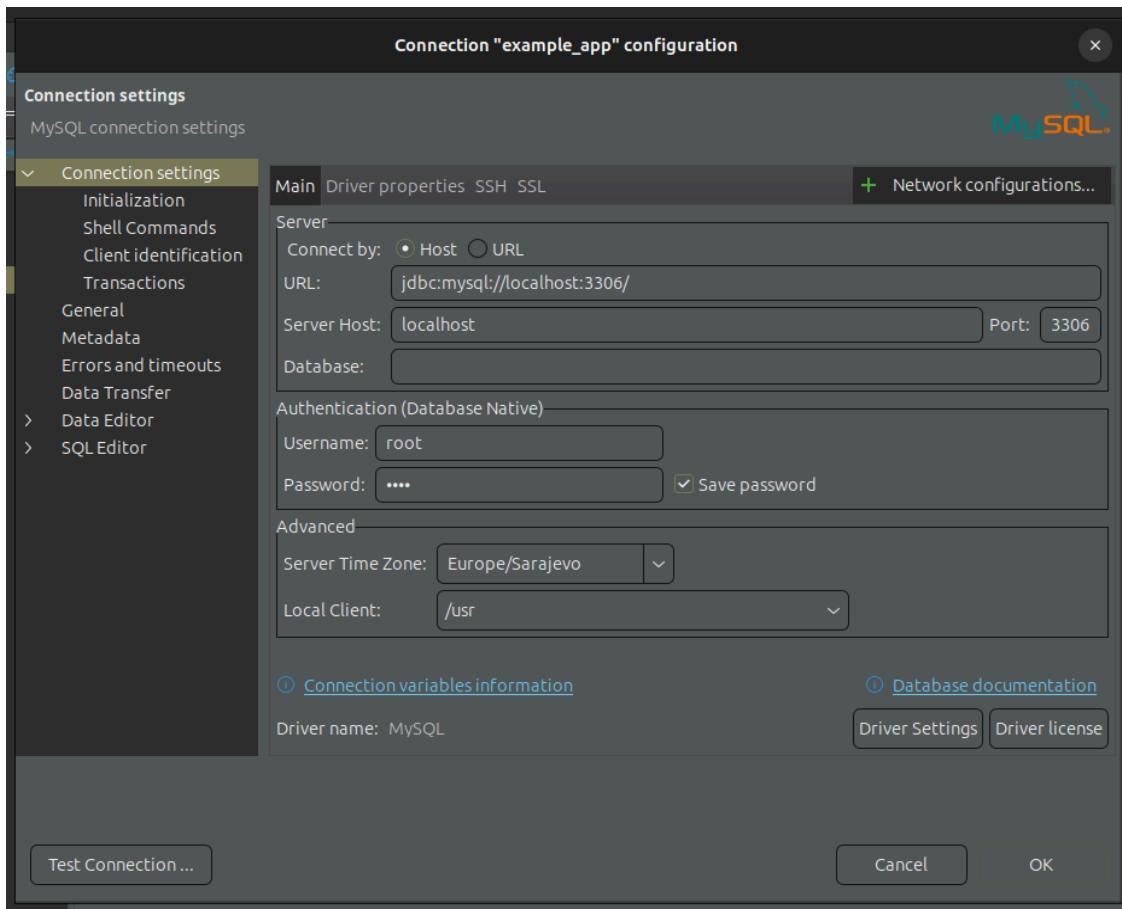
U sljedećim dijelovima ovog diplomskog rada, proći ćemo i objasniti proces kreiranja ogledne web aplikacije upotrebom Laravel i Livewire tehnologija. Prije nego što počnemo sa kodiranjem aplikacije, potrebno je odrediti kako i gdje će se čuvati izvorni kod aplikacije. Za skladištenje izvornog koda aplikacije, odabrana je popularna platforma GitHub koja pruža usluge hostinga Git repozitorija. Git je široko prihvaćeni distribuirani sistem kontrole verzija koji će omogućiti efikasno upravljanje promjenama koda i fajlova. Ovi alati osiguravaju da baza izvornog kodova ostane organizirana i dostupna za buduće potrebe razvoja. Za ovu aplikaciju napravili smo jedan projekat u kojem se nalazi backend kod sa controllerima, rutama, modelima i frontend view-evi sa komponentama, public resursima, layout file-ovima i livewire componentama, pod nazivom realestate-app, dok smo kreirao GitHub repozitorij pod nazivom “real-estate-app”.

5.8 Inicijalizacija projekata

Proces kreiranja aplikacije zahtijeva kreiranje projekata na lokalnom nivou. Uz pomoć terminala, kreiran je lokalni Laravel projekat koristeći komandu „laravel new realestate-app“, gdje ćemo izabrati da naš projekat uključuje i Laravel Breeze starter kit. Odabrana je verzija 11 Laravel-a dok je za PHP odabrana verzija 8.3.16. Idući upit u terminalu za naš novi projekat jeste izbor vrste projekta, da li da sadrži starter kit ili bude generisan bez jednog. Izborom „Breeze Starter kit“ opcije, ponuđeni su nam stekovi u kojima želimo da bude generisan naš projekat i starter kit. Budući da ćemo projekat raditi u Livewire-u izabrana je opcija „Livewire and Blade“. Nakon toga možemo vidjeti izbor baze podataka. Za bazu podataka sam izabrao MySQL. Nakon izbora baze podataka, počinje instaliranje svih potrebnih paketa za razvoj projekta sa tehnologijama koje smo izabrali. Nakon završenog instaliranja svih paketa, prikazuje se poruka za uspješan završetak instaliranja paketa, te nam se nude dvije komande: `cd example-app` i `php artisan serve`. Prvo je bash komanda koja nam omogućava ulazak u direktorij “example-app” preko našeg terminala, dok komanda `php artisan serve`, je Laravel specifična artisan komanda koja pokreće php development server za našu Laravel aplikaciju.

5.9 Konfiguracija baze podataka

Kao što je i unaprijed definisano, za bazu podataka će se koristiti MySQL. Da bih mogao raditi sa bazom podataka, te pisati skripte za query-anje podataka te pregled istih, moram napraviti konekciju na bazu preko nekog alata za upravljanje bazama podataka. Za ovaj projekat sam izabrao DBeaver. Konekcija se kreira klikom na “New Connection” dugme, nakon koje nam se prikazuje forma u kojoj biramo bazu podataka prema kojoj pravimo konekciju, kao što je prikazano na slici 10.

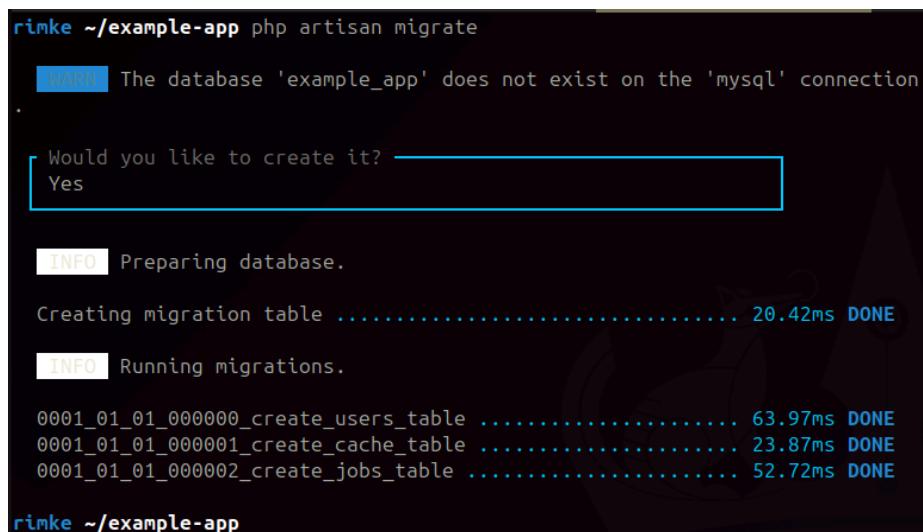


Slika 10. Postavke konekcije preko DBeaver aplikacije

Nakon izbora MySQL-a, moramo unijeti potrebne podatke i postavke pomoću kojih ćemo napraviti konekciju prema izabranoj bazi podataka, te moramo urediti podatke u “.env” file-u u našem projektu da se podudaraju sa podacima unesenim u konfiguracijama konekcije. Nakon što unošenja podataka u .env file, moramo kreirati bazu podataka sa tabelama. To radimo pomoću migracija. “Migration” je vrsta Laravel file-a u kojem se definiše “schema” tabele koju želimo kreirati unutar naše baze podataka, isto tako

migration file-ovi imaju ulogu historijata naše baze, jer se sve promjene prema bazi dešavaju preko migracija.

Unutar migration file-ova imamo “create” funkciju, pomoću koje kreiramo tabele, gdje definišemo retke, njihove tipove i sve ostale specifikacije koje su vezane i potrebne za rad tabele. Budući da migration file-ovi imaju ulogu i kontrolora verzije naše baze podataka, brisanje migration file-ova nije preporučljivo, te ako se žele i popraviti neke stvari koje smo prije definisali u našoj shemi, to možemo uraditi kreiranjem novog migration file-a čija će funkcija biti „edit“ ili „remove“ za specifični redak unutar već definisane sheme. Veoma je važno napomenuti da se ne preporučuje pokretanje artisan migration komandi na produkcijskom serveru, te da se prvo kreira dobar dizajn i plan za schemu baze.



```
rimke ~/example-app php artisan migrate
WARN The database 'example_app' does not exist on the 'mysql' connection
.
Would you like to create it?
Yes

INFO Preparing database.

Creating migration table ..... 20.42ms DONE
INFO Running migrations.

0001_01_01_000000_create_users_table ..... 63.97ms DONE
0001_01_01_000001_create_cache_table ..... 23.87ms DONE
0001_01_01_000002_create_jobs_table ..... 52.72ms DONE
rimke ~/example-app
```

Slika 11. Pokretanje “`php artisan migrate`” komande na novom projektu

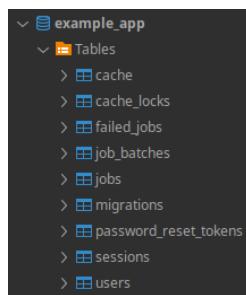
Pokretanjem `php artisan migrate` komande u našem terminalu, prikazuje nam se poruka da baza “example_app” ne postoji na našoj mysql konekciji. Nakon izbora da želimo kreirati tu bazu, ona se kreira, i pokreću se migration file-ovi koji se nalaze u našim database/migrations direktoriju. Pri svakom pokretanju `php artisan migrate` komande, slika 11., pokreću se samo migration file-ovi koji nisu pokrenuti do sada, da ne bi došlo do repeticije tabela, stupaca ili baza. Prikaz migracija koje su pokrenute i onih koje nisu možemo vidjeti koristeći komandu `php artisan migrate:status` u našem terminalu, kao što vidimo na slici 12.

```
rimke ~/example-app php artisan migrate:status

Migration name ..... Batch / Status
0001_01_01_000000_create_users_table ..... [1] Ran
0001_01_01_000001_create_cache_table ..... [1] Ran
0001_01_01_000002_create_jobs_table ..... [1] Ran
```

Slika 12. Prikaz statusa svih migration file-ova

I sada ako pogledamo našu bazu podataka preko DBeaver-a, na slici 13., možemo vidjeti da se u listi tabela nalaze sve tabele koje su definisane u “users”, “cache” i “jobs” migration file-ovima, te imamo još imamo i “migrations” tabelu koja se sastoji samo od osnovnih podataka migration file-ova u našoj aplikaciji tj. od id broja, naziva migracije i od broja serije u kojoj je pokrenuta ta migracija.

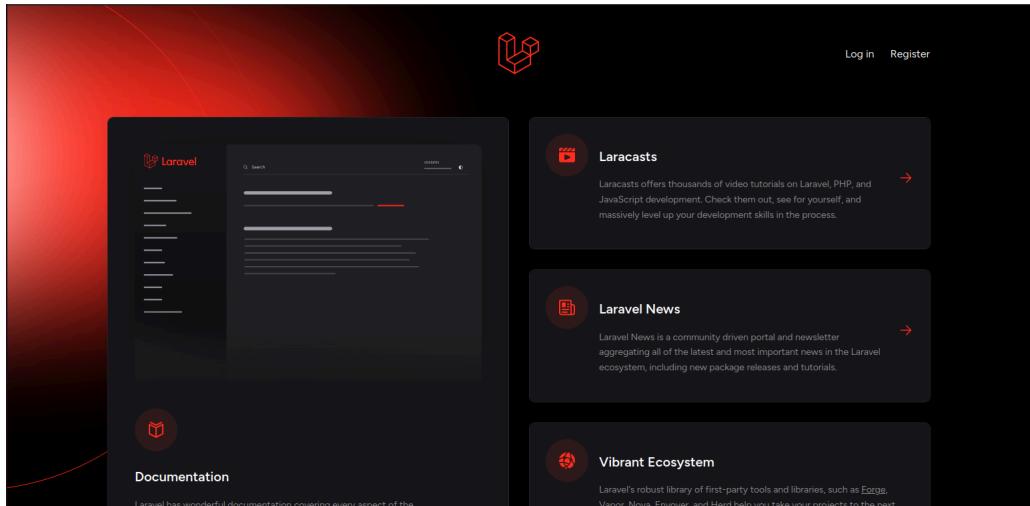


Slika 13. Prikaz tabela u example_app bazi podataka

5.10 Autentifikacija

Za autentifikaciju je korišten Laravelov ugrađeni autentifikacijski sistem kojem se pristupa preko „Auth“ i „Session“ fasada. Fasade (eng. Facade) pružaju statični interfejs klasama koje su dostupne u aplikacijskom „service provider-u“. Instalacija Laravela dolazi sa mnogim fasadama koje pružaju pristup skoro svim funkcionalnostima framework-a.[19] Auth i Session fasade pružaju funkcionalnosti autentifikacije koje su bazirane na kolačićima (eng. cookies) za zahtjeve koji su inicijalizirani preko pretraživača. U mnogim modernim tehnologijama, za korištenje i postavljanje ovakvih funkcionalnosti potrebno je dosta vremena i pažnje, dok korištenjem Laravelovih servisa, ovaj proces je znatno olakšan, da bi se developer ili product owner mogao fokusirati na razvoj biznis logike svoje aplikacije. Kreiranjem ovog projekta sa Breeze starter kit-om znatno je olakšano postavljane autentifikacijskih funkcionalnosti i dovoljno je samo

pokretanje migration komande da se postavi funkcionalno i skalabilno rješenje za autentifikaciju korisnika u našoj aplikaciji. Pokretanjem lokalnog servera pomoću komande `php artisan serve`, te odlaskom na `localhost:8000` adresu servera koji smo pokrenuli imamo početni prikaz naše aplikacije koji je prikazan na slici 14.



Slika 14. Prikaz prve stranice novog Laravel projekta

Na vrhu stranice imamo i dva linka za “Log in” i “Register” stranice koje su također dio Breeze starter kit-a koji nam pruža log in i register funkcionalnosti za naše korisnike koji su kreirani pomoću users migracije. Izgled login forme vidimo na slici 15.

A screenshot of a login form. It features a large 'Laravel' logo at the top. Below it is a light gray box containing fields for 'Email' (with a placeholder 'Email') and 'Password' (with a placeholder 'Password'). There is also a 'Remember me' checkbox. At the bottom of the box are two buttons: a blue 'Forgot your password?' link and a dark blue 'LOG IN' button.

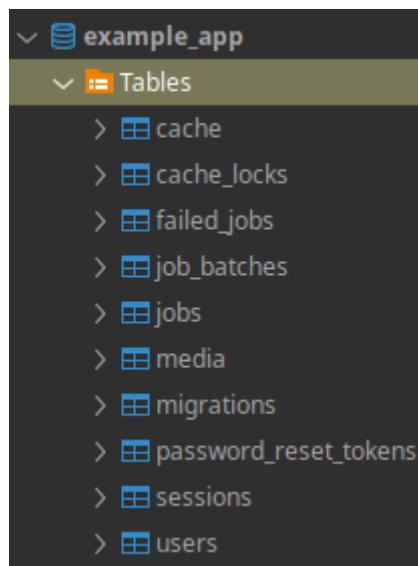
Slika 15. Prikaz login forme novog Laravel projekta

U našoj aplikaciji nije predviđena mogućnost registracije korisnika na sistem, već samo dodavanje novih korisnika putem admin panela. Rute, kontroleri i pogledi korišteni za

register dio aplikacije su obrisani code base-a.

5.11 Konfiguracija Spatie paketa

Za rad naše aplikacije, potrebna je instalacija dodatnih paketa. Za lakši rad sa slikama i manipulisanje istih, instalirat ćemo “spatie/laravel-medialibrary” paket, koji je kreiran i aktivno održavan od kompanije Spatie. Media Library je paket koji asocira različite vrste file-ova sa eloquent modelima. Paket možemo instalirati preko terminala pomoću composer package manager komande *composer require spatie/laravel-medialibrary*, te nakon pokretanja komande u našem *composer.json* file-u možemo pronaći naš paket i vidjeti da je instalirana verzija 11.12. Nakon uspješnog instaliranja paketa, moramo “objaviti” migracije da kreiramo “media” tabelu u našoj bazi, te nakon što pokrenemo komandu za objavljivanje migracija pojavit će nam se migracije koje moramo pokrenuti da bi se kreirale tabele u kojima se spašavaju podaci o našim datotekama i nakon toga ponovnim pokretanjem *php artisan migrate* komande, kreirat ćemo dodatne tabele u našoj bazi, te će naš folder tabela izgledati kao na slici 16.



Slika 16. Prikaz media tabele u našoj bazi podataka

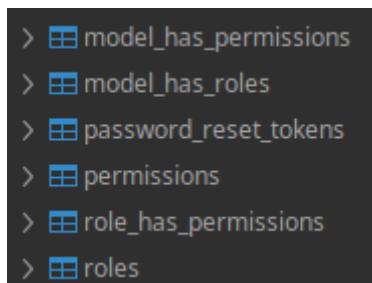
Nakon uspješne instalacije i konfiguracije paketa, moramo urediti modele sa kojima želimo asocirati određene datoteke u našem projektu. Za sada, to je samo *User* model, ali ćemo kasnije konfigurisati i *Properties* model kada ga kreirali. Da bi sa modelom mogli asocirati određene datoteke, u model file, koji se nalazi u *app/Models* folder-u, moramo dodati određene zavisnosti koje su instalirane zajedno sa paketom. Prvo dodajemo *HasMedia* i *InteractsWithMedia* zavisnosti, te nakon toga ih moramo primijeniti u našem

modelu.

```
use Spatie\MediaLibrary\HasMedia;
use Spatie\MediaLibrary\InteractsWithMedia;

class User extends Authenticatable implements HasMedia
{
    use HasFactory, InteractsWithMedia, Notifiable, SoftDeletes;
```

Naš model implementira HasMedia zavinost, te unutar definisane User class-e dodajemo zavisnost InteractsWithMedia, te one našem modelu daju pristup funkcijama preko kojih manipulišemo sa Media. Sada možemo na instance User model-a u našoj aplikaciji asocirati određene file-ove koji će biti vezani samo za te instance modela. Nakon uspješnog instaliranja paketa, dodavanja svih zavisnosti i uređenja modela sa čijim objektima želimo da vezujemo slike, sada možemo dodati posljednji Spatijeov paket. Iako Laravel pri instalaciji i kreiranju novog Breeze projekta, nudi mnoge mogućnosti i rješenja za problem autorizacije, pristup rutama, putem guardov-a i middleware-a koji definišemo u route file-ovima, nekada želimo da imamo moguće dinamično dodjeljivanje rola i pristupnih prava korisnicima. Takav sistem nam omogućuje Spatijeov paket „spatie/laravel-permission“. Laravel-permission je paket koji nam omogućava upravljanje korisničkih permisija i rola u bazi podataka. Ako naša aplikacija ima već ugrađen sistem koji koristi više Guard-ova, moguće je da svakom Guard-u postavimo njegov set korisničkih rola i permisija. Budući da su sve permisije registrovane na Laravel-ovom gate-u, možemo provjeriti da li korisnik ima permisiju koristeći Laravel-ovu ugrađenu *can()* funkciju. Paket možemo instalirati preko terminala pomoću composer package manager-a koristeći komandu *composer require spatie/laravel permission*, te nakon pokretanja komande možemo vidjeti u našem *composer.json* da nam je instalirana 6.15. verzija paketa.



```
> └─ model_has_permissions
> └─ model_has_roles
> └─ password_reset_tokens
> └─ permissions
> └─ role_has_permissions
> └─ roles
```

Slika 17. Prikaz generisanih tabela u bazi

Nakon uspješnog instaliranja, moramo “objaviti” migracije da bismo kreirali ‘permissions’ i ‘roles’ tabele, sve pivot tabele koje su potrebne za normalan rad paketa. To ćemo uraditi koristeći predloženu komandu nakom čijeg pokretanja će nam se pojavit migracije koje moramo pokrenuti da bi se kreirale. Nakon kreiranja tabela, slika 17., moramo dodati HasRoles zavisnost koja omogućuje “vezivanje” naših User objekata sa permisijama.

```
use Spatie\Permission\Traits\HasRoles;

class User extends Authenticatable implements HasMedia
{
    use HasFactory, HasRoles, InteractsWithMedia, Notifiable, SoftDeletes;
```

Nakon dodavanja ove zavisnosti, omogućeno nam je korištenje Spatie-ovog “laravel-permissions” paketa i svih njegovih funkcionalnosti. Sada možemo na lagan i skalabilan način upravljati dodjeli i provjeri autorizacijskih prava i mogućnosti svih korisnika u našoj aplikaciji, te nam je omogućeno da na osnovu korisničkih zahtjeva definišemo vrste role-a koje su potrebne.

5.12 User testni podaci

Za optimalno testiranje naših funkcionalnosti tokom razvojnog procesa naše aplikacije, potrebni su testni podaci koji ispunjavaju određene kriterije. Za generisanje potrebnih testnih podataka, koristit ćemo Laravel-ove Factory-e i Seed-ere. Jedna od razlika između Seed i Factory file-ova jeste što se Seed file-ovi koriste za “mass assignment” podataka u našoj bazi podataka, dok se Factory file-ovi koriste da se unese par zapisa u tabelu. Na kraju, veoma bitna odlika Factory file-ova jeste što se oni mogu koristiti i kao template-i ili nacrti objekata koje želimo generisati putem Seed file-ova. U našim Seed datotekama možemo pozvati Factory file-ove te na taj način generisati određeni broj objekata koji nam je potreban za testiranje našeg softvera.

```
public function definition(): array
{
    return [
        'name' => fake()->name,
        'email' => fake()->unique()->email,
        'email_verified_at' => now(),
        'phone_number' => fake()->phoneNumber,
        'password' => Hash::make('judaspriest'),
        'remember_token' => Str::random(10),
```

```
    ];
}
```

U “definition” funkciji našeg UserFactory-a definišemo početno stanje i izgled našeg User modela, te koristeći pomoćne funkcije možemo generisati realistične podatke za svaku od kolona u našem modelu. Sada smo definisali template prema kojem će se kreirati objekat User modela prilikom poziva UserFactory klase.

```
{
    $agentPermissions = [
        'create property',
        'edit property',
        'delete property',
        'edit agent',
    ];

    $agentRole = Role::firstOrCreate(['name' => 'agent']);
    foreach ($agentPermissions as $permission) {
        $createdPermission = Permission::firstOrCreate(['name' =>
$permission]);
        $agentRole->givePermissionTo($createdPermission);
    }

    for ($x = 0; $x < 4; $x++) {
        $user = User::factory()->create();
        $user->assignRole($agentRole);
        $user->addMedia(public_path('photos/icons/realestateagent.png'))
            ->preservingOriginal()
            ->toMediaCollection('agent-pfps');
    }
}
```

Nakon što smo kreirali i definisali Factory file za User model, dalje moramo definisati Seeder file u kojem ćemo koristeći User::factory metodu, generisati 4 korisnika. Prije generisanja korisnika, definisat ćemo *\$agentPermissions* varijablu koja sadrži nazive permisija koje će biti dodjeljenje Agent roli. Nakon toga kreiramo *\$agentRole* objekat tipa Role. Onda imamo *foreach* petlju, u kojoj za svaku permisiju unutar *\$agentPermissions* liste, kreiramo *\$createdPermission* varijablu tipa Permission koja se naziva po iteraciji *\$agentPermissions* liste, te kreiranu permisiju dodjeljujemo *\$agentRole*. Nakon toga, imamo još jednu *for* petlju koja će kroz 4 iteracije kreirati pomoću User::factory funkcije 4 testna korisnika, dodijeliti svakome rolu *\$agentRole* te nakon toga, pomoću addMedia() funkcije, dodijeliti testnu profilnu sliku svakom od kreiranih agenata. Nakon što smo napisali UserSeeder file, koji je kreiran sa namjerom da seed-a samo user objekte, primarno agente agencije, nakon toga moramo iskoristiti UserSeeder model unutar

DatabaseSeeder file-a. DatabaseSeeder file je file pomoću kojeg se baza podataka seed-a nakon pokretanja *php artisan db:seed komande*. Isto tako možemo dodati “–class” na kraju komande, da navedemo specifičnu seeder klasu koju želimo da pokrenemo prilikom pokretanja artisan komande, te bi takva komanda izgledala ovako *php artisan db:seed –option=UserSeeder*. Nakon kreiranja i završetka UserSeeder file-a, možemo srediti i DatabaseSeeder file da koristi dosad kreirane klase isto kao i u UserSeeder-u za kreiranje Admin role i Admin korisnika koji će predstavljati vlasnika/super admina agencije, te nakon toga pozivamo UserSeeder klasu da generiše korisnike prema definisanim kriterijima. Nakon definisanja DatabaseSeeder file-a i pokretanja artisan komande u našem terminalu, u bazi podataka će se izgenerisati testni podaci kao što je definisano u seeder file-ovima, koje možemo vidjeti na slici 18.

	id	name	email	email_verified_at	password	remember_token
1	1	admin	kerim.nezo@gmail.com	2025-01-16 11:22:50	\$2y\$12\$nB/M1UUl4/aGiLyzuFhA9OAzbhRXWdbA43AT8xz. vKPLpHXYOGqVxejS1goGwUo5rlg	
2	2	Burley Bayer	okihn@gutkowsk...	2025-01-16 11:22:50	\$2y\$12\$W.GkXynFz6wMTFTI1o3iuD2gBVHUUOaMqVzkwy 1RvYQoGlu	
3	3	Oswald Lubowitz	katelyn.kling@sanford.com	2025-01-16 11:22:51	\$2y\$12\$KTfGIQs5LOdjjRsmjnIdNOUEheVWpyBj0Gzf33KQ L9ohhIWDEj7	
4	4	Ms. Thora Hermann I	vada29@johns.biz	2025-01-16 11:22:51	\$2y\$12\$il/9TRoJPG.ug3tLv6LCXOWOvmdUe8HISaEmATGibY. R36ucArbQu	
5	5	Prof. Gianni Rau	madisyn.murphy@jenkins.com	2025-01-16 11:22:51	\$2y\$12\$yxka/IVP420biNx8r3Qpe189IfFoOCZPwBM2yoIhyj XUo4cwji6D	

Slika 18. Prikaz podataka iz Users tabele koji su generisani putem seeder file-ova

Putem DBeaver-a možemo napisati SQL skripte kojima dobavljamo podatke iz tabela u našoj bazi. Možemo vidjeti da se automatski pravi i veza putem pivot tabela za modele kojima su dodjeljene role, te za role kojima su dodjeljenje permisije. U DatabaseSeeder file-u je kreiran korisnik sa roлом “admin” koji će imatu ulogu superadminga za ovaj sistem, te smo njegove pristupne podatke detaljno definisali iz razloga da kasnije može pristupiti sistemu kao admin, dok su useri kreirani sa roлом “agent” trenutno testni korisnici sa kojima ćemo kasnije testirati agent poglede.

5.13 Definisanje ostalih modela

Nakona detaljnog prikaza dodavanja jednog Eloquent Modela u naš sistem, da bi završili sa pripremom baze, moramo dodati ostatak modela i file-ova koji su potrebni za definisano funkcionisanje ove aplikacije, a to su: *Type*, *Property* i *Action*. Prvo ćemo dodati model koji predstavlja tip nekretnina koje ćemo kasnije definisati. Pomoću artisan komande *php artisan make:model Type -mfsc* kreirat ćemo sljedeće file-ove: *create_types_table.php*, *TypeFactory.php*, *TypeSeeder.php* i *TypeController.php* (-mfsc je pomoćna opcija kod artisan komande koja označava da za navedeni model želimo da generišemo migration, factory, seeder i controller file-ove). Prvo moramo urediti migration file, te nakon toga, pomoću korištene artisan komande, pokrenut ćemo definisani migraciju i s njom generisati Types tabelu u našoj bazi podataka. Nakon toga ćemo urediti Seeder file-ove kako bismo kasnije generisali tipove nekretnina kakvi ispunjavaju korisničke zahtjeve, te omogućavaju kasnije dodavanje novih tipova, ako se javi potreba za istim. Radi jednostavnosti Type modela i njegove uloge u projektu, korištenje i definisanje Factory i Controller klase za Type model nije potrebno, te ćemo sada raditi samo sa TypeSeeder.php file i kasnije na Type.php kada budemo definisali relacije između svih Eloquent Modela u našem sistemu.

```
{  
    $propertyTypes = [  
        'office',  
        'house',  
        'appartement',  
    ];  
  
    foreach ($propertyTypes as $type) {  
        Type::create([  
            'name' => $type,  
        ]);  
    }  
}
```

U TypeSeeder file-u možemo vidjeti listu *\$propertyTypes* koja sadrži sve tipove nekretnina koji su definisani u korisničkim zahtjevima, te za svaki kreiramo Type objekat koji nosi naziv jednog tipa nekretnine iz liste. Imamo jednostavan Seeder file, koji ćemo pozvati u našoj DatabaseSeeder klasi, tako da se generišu ovi tipovi nekretnina zajedno ostalim modelima.

Nakon implementiranja Type modela u projekat, sada dodajemo Action model, koji će imati specifičnu ulogu. U kasnijem dijelu rada detaljnije ćemo spomenuti ulogu admina u cijelom sistemu, jer se jedino može pristupiti podacima o Action objektima unutar admin panela. Naime, akcije smo definisali kao logging klasu, čija je glavna primjena u sistemu da prati događaje koji se izvrše nad Property modelom. Primarna uloga akcija jeste da nadgledaju evenete, koje mi definišemo, koji se izvrše nad Property Modelom. Radi ovako specifičnog use case-a jednog modela, jedini file koji će nam trebati jeste controller, te ćemo Action model kreirati pomoću sljedeće artisan komande *php artisan make:model Action -cm* koja će nam pored *Action.php* generisati, *ActionController.php* file i migraciju potrebnu za kreiranje “actions” tabele u našoj bazi. Nakon pokretanja artisan komande, definisat ćemo strukturu našeg modela koristeći dijagrame iz prethodnih poglavlja. Možemo vidjeti u našoj da smo definisali dva “foreignIdFor” objekta, jedan za User klasu, drugi za Property klasu, za user i property objekte kojima ta akcija bude pripala. Naime, stvarna uloga akcija je da se kreiraju Action objekti nakon što agent ili admin sistema, izvrše neki od evenata koji mi definišemo na Property objektima. Definisali smo još naziv akcije i poruku

```
{
    Schema::create('actions', function (Blueprint $table) {
        $table->id();

        $table->foreignIdFor(User::class)->constrained()->cascadeOnDelete();

        $table->foreignIdFor(Property::class)->constrained()->cascadeOnDelete();
            $table->string('name');
            $table->text('message');
            $table->timestamps();
        });
    }
}
```

Nakon što smo definisali željeni izgled naše Actions tabele prema korisničkim zahtjevima, prije nego što pokrenemo migracije, morat ćemo da kreiramo Property model, jer on sadrži property_id kolonu. Laravel će pri pokretanju migracije pokušati da poveže *foreignIdFor(Property::class)* kolonu sa Property klasom, koja trenutno ne postoji u našoj aplikaciji, te je stoga moramo kreirati. Nakon implementiranja Action modela u našu aplikaciju, sada možemo početi sa dodavanjem Property modela, koji je ujedno i centralni model naše aplikacije. Razlog zašto smo prvo kreirali Type i User modele jeste u tome što je planirano da se u Property objektu nalazi strani ključ i korisnikovog id-a (koji

predstavlja id agenta kojem je u sistemu dodjeljna nekretnina) i id tipa nekretnina (type_id, koji predstavlja id tipa nekretnina u našem sistemu). Generisat ćemo sve potrebne file-ove za Property model koristeći istu artisan komandu u našem terminalu *php artisan make:model Property -mfsc* koja će nam kreirati migration, factory, seeder i controller file-ove potrebne za rad Property modela. Prvo ćemo urediti property migration file, gdje ćemo definisati sve kolone koje će postojati unutar Property modela koje smo definisali prilikom korisničkih zahtjeva, te smo u kasniju migraciju dodali i kolonu *year_built* koja predstavlja godinu gradnje nekretnine. Iako smo sada završili sa pisanjem svih migracija, radi pravilnog rada naše aplikacije, prvo moramo definisati sve relacije unutar Model file-ova svih modela. Nakon toga ćemo urediti Property.php, gdje ćemo definisati relacije sa ostalim modelima. Na vrhu možemo vidjeti da Property model isto tako implementira „HasMedia“ i „InteractsWithMedia“ zavinosti koje nam omogućavaju da vezujemo media objekte za property model. Isto tako vidimo zavisnost „softDeletes“, koja nam generiše još jednu kolonu na Property modelu, *deleted_at* tipa timestamp, koja označava trenutak kada je nekretnina obrisana, ali njen zapis se još uvijek nalazi u našoj bazi podataka. Ako bismo željeli popravno brisanje nekretnine iz sistema, koristit ćemo forceDelete(). Dalje možemo vidjeti protected varijablu \$fillable, koja je samo lista svih atributa našeg modela kojima možemo dodjeliti vrijednost, kada je Property objekat kreirani unutar seeder-a.

```
class Property extends Model implements HasMedia
{
    use HasFactory, InteractsWithMedia, softDeletes;

    protected $fillable = [...];

    public function type(): BelongsTo
    {
        return $this->belongsTo(Type::class);
    }

    public function user(): BelongsTo
    {
        return $this->belongsTo(User::class);
    }

    public function actions(): HasMany
    {
        return $this->hasMany(Action::class);
    }
}
```

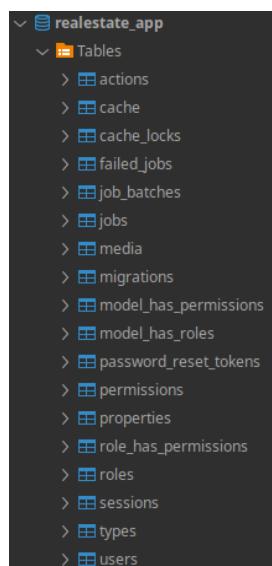
Na dnu naše klase, vidimo tri funkcije, koje predstavljaju relaciju Property Modela sa

Type, User i Action klasom. Vidimo da Property model pripada User i Type klasama, što označava da on u svom zapisu ima type_id i user_id kolone, te vidimo da ima HasMany relaciju sa Action klasom, koja predstavlja da se Property model nalazi u više Action objekata pod property_id kolonom. Nakon uređivanja Property modela, moramo urediti i ostale modele i njihove relacije sa modelima bi se ispunili korisnički zahtijevi.

```
public function properties(): HasMany
{
    return $this->hasMany(Property::class)->withTrashed();
}

public function actions(): HasMany
{
    return $this->hasMany(Action::class);
}
```

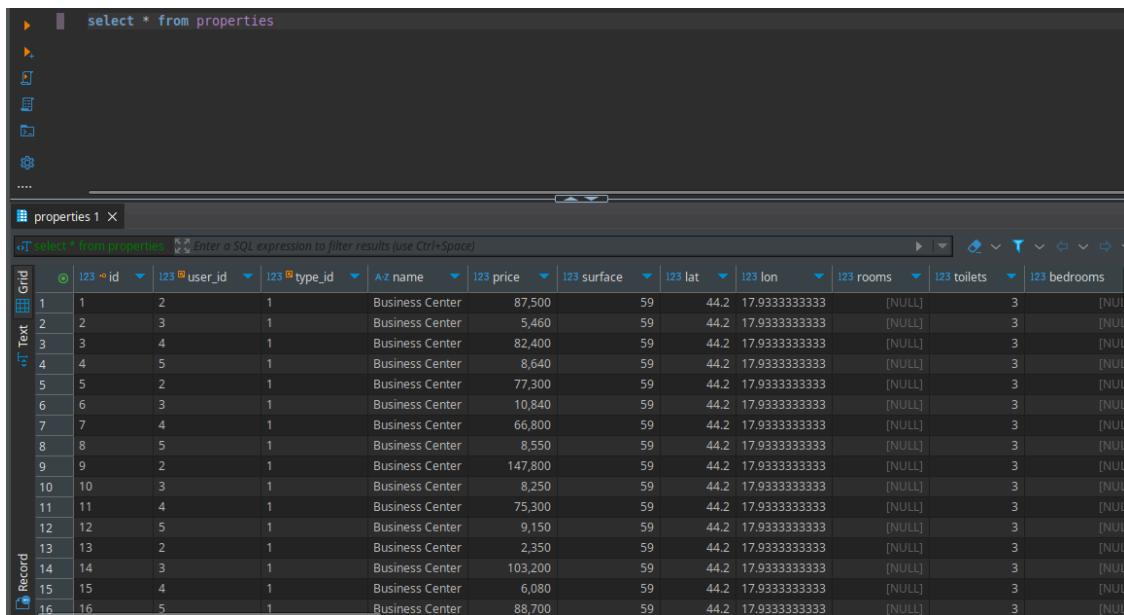
U narednom kodu možemo vidjeti HasMany relacije User modela sa Property i Actions modelima. Jedna od bitnih karakteristika jeste da u *properties()* relaciji vidimo dodatnu funkciju *->withTrashed()*, koja označava da se relacija ne gubi ako je Property objekat sa kojim je User u relaciji soft delete-an. Nakon definisanja svih relacija, možemo pokrenuti *php artisan migrate* komandu koja će kreirati tabele na osnovu migration file-ova koje smo dodali u sistem. Nakon pokretanja komande, naš Tables folder unutar Dbeaver-a će izgledati kao na slici 19.



Slika 19. Lista svih tabela u našoj bazi podataka

Nakon uspješnog generisanja tabela za sve modele, uredit ćemo PropertySeeder i PropertyFactory file-ove kako bismo dobili što raznovrsnije nekretnine da bismo imali što

bolji testni proces kasnije, pri kreiranju pogleda i front end dijela aplikacije. U PropertyFactory smo definisali default-no stanje onih kolona koje smo odredili da za sada budu iste svim nekretnina, radi testnih slučajeva. Nakon uređivanja PropertyFactory i PropertySeeder file-ova, te dodavanja PropertySeeder klase unutar DatabaseSeeder-a, možemo da pokrenemo `php artisan db:seed` komandu u terminalu kojom ćemo sada kreirati objekte definisane unutar DatabaseSeeder-a u kojem se pozivaju i Seeder klase od modela kojima smo ih definisali, te ćemo na taj način popuniti bazu podataka sa testnim podacima koje možemo vidjeti na slici 20. unutar našeg DBeaver randog okruženja.



The screenshot shows a DBeaver interface with a SQL editor at the top containing the query `select * from properties`. Below the editor is a results grid titled "properties 1". The results grid displays 16 rows of data from the properties table, each representing a business center. The columns are labeled: id, user_id, type_id, name, price, surface, lat, lon, rooms, toilets, and bedrooms. The data shows various business centers with names like "Business Center" and prices ranging from 2,350 to 147,800.

	id	user_id	type_id	A-Z name	price	surface	lat	lon	rooms	toilets	bedrooms
1	1	2	1	Business Center	87,500	59	44.2	17.9333333333	[NULL]	3	[NULL]
2	2	3	1	Business Center	5,460	59	44.2	17.9333333333	[NULL]	3	[NULL]
3	3	4	1	Business Center	82,400	59	44.2	17.9333333333	[NULL]	3	[NULL]
4	4	5	1	Business Center	8,640	59	44.2	17.9333333333	[NULL]	3	[NULL]
5	5	2	1	Business Center	77,300	59	44.2	17.9333333333	[NULL]	3	[NULL]
6	6	3	1	Business Center	10,840	59	44.2	17.9333333333	[NULL]	3	[NULL]
7	7	4	1	Business Center	66,800	59	44.2	17.9333333333	[NULL]	3	[NULL]
8	8	5	1	Business Center	8,550	59	44.2	17.9333333333	[NULL]	3	[NULL]
9	9	2	1	Business Center	147,800	59	44.2	17.9333333333	[NULL]	3	[NULL]
10	10	3	1	Business Center	8,250	59	44.2	17.9333333333	[NULL]	3	[NULL]
11	11	4	1	Business Center	75,300	59	44.2	17.9333333333	[NULL]	3	[NULL]
12	12	5	1	Business Center	9,150	59	44.2	17.9333333333	[NULL]	3	[NULL]
13	13	2	1	Business Center	2,350	59	44.2	17.9333333333	[NULL]	3	[NULL]
14	14	3	1	Business Center	103,200	59	44.2	17.9333333333	[NULL]	3	[NULL]
15	15	4	1	Business Center	6,080	59	44.2	17.9333333333	[NULL]	3	[NULL]
16	16	5	1	Business Center	88,700	59	44.2	17.9333333333	[NULL]	3	[NULL]

Slika 20. Rezultat upita nad properites tabelom

5.14 Kreiranje Observer klase

Za potpuno funkcionalisanje Action modela, koji smo ranije definisali da treba raditi kao event sistem svaki put kada se desi jedna od akcija koje mi navedemo nad Property objektom, moramo generisati još Observer nad Property modelom. Ako slušamo više događaja nad jednim modelom i hoćemo da ih sve grupišemo u jednu klasu u kojoj definišemo sve akcije za te događaje, možemo generisati Observer file nad tim modelom. Observer klase imaju metode čija imena reflektuju Eloquent događaje koje želimo da pokrenu određeni kod koji definišemo u tim akcijama.[20]

Komandom `php artisan make:observer PropertyObserver -model=Property` kreirat ćemo Observer klasu unutar `app/Observers` foldera. Unutar našeg observer file-a ćemo

definisati akcije čija imena odgovaraju radnjama koje se mogu desiti nad Eloquent modelima, a one su: *created*, *updated*, *deleted*. *Created()* funkcija će se pokrenuti kada se uspješno pohrani nekretnina u bazu podataka, *Updated()* funkcija se pokreće kada se ažurira već postojeći zapis u tabeli i *Deleted()* funkcija kada se uradi se nekretnina soft delete-a u bazi. Da bi Observer file funkcionalo kako treba, moramo ga registrovati nad odgovarajućim modelom koristeći `#ObservedBy` atribut i unutar njega deklarisati koju observer klasu dodjeljujemo tom modelu.

```
#[ObservedBy([PropertyObserver::class])]
class Property extends Model implements HasMedia
{
```

Nakon definisanih Observer akcija i ponovnog pokretanja komande za seed baze podataka, na slici 21. možemo vidjeti da se popunila i tabela Actions nakon što se kreiraju nekretnine putem DatabaseSeeder.php file-a.

Grid	1	1	1	created	Property was seeded to the database	2025-01-16 11:22:51	2025-01-16 11:22:51
Text	2	1	1	sold	Property was sold by agent: Burley Bayer	2025-01-16 11:22:51	2025-01-16 11:22:51
Record	3	1	2	created	Property was seeded to the database	2024-12-16 11:22:51	2025-01-16 11:22:51
Grid	4	1	2	rented	Property was rented by agent: Oswald Lubowitz	2024-12-16 11:22:51	2025-01-16 11:22:51
Text	5	1	3	created	Property was seeded to the database	2024-11-16 11:22:51	2025-01-16 11:22:51
Record	6	1	3	sold	Property was sold by agent: Ms. Thora Hermann I	2024-11-16 11:22:51	2025-01-16 11:22:52
Grid	7	1	4	created	Property was seeded to the database	2024-10-16 11:22:51	2025-01-16 11:22:52
Text	8	1	4	rented	Property was rented by agent: Prof. Gianni Rau	2024-10-16 11:22:51	2025-01-16 11:22:52
Record	9	1	5	created	Property was seeded to the database	2024-09-16 11:22:51	2025-01-16 11:22:52
Grid	10	1	5	sold	Property was sold by agent: Burley Bayer	2024-09-16 11:22:51	2025-01-16 11:22:52

Slika 21. Rezultat quarya nad actions tabelom

5.15 Stilizovanje aplikacije

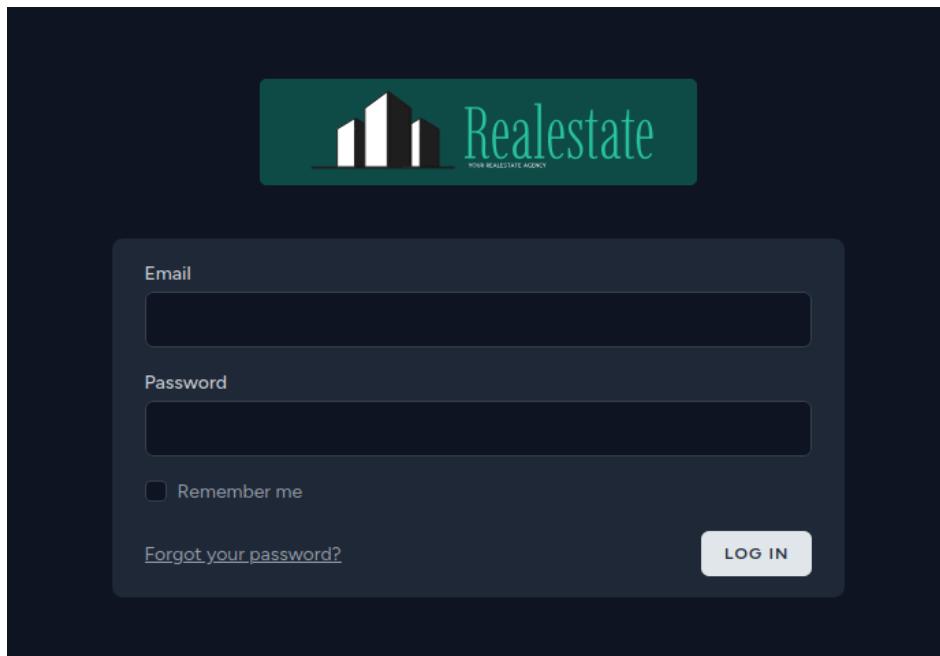
Za stilizovanje Frontend aplikacije i svih web stranica korišten je *framework* "Tailwind CSS". Tailwind CSS je CSS *framework* dizajniran da omogući korisnicima da brže i lakše kreiraju aplikacije[21] Mogu se koristiti uslužne klase za kontrolu izgleda, boje, razmaka, tipografije, sjenke i još mnoge opcije kako bi se kreirao potpuno prilagođeni dizajn komponente bez napuštanja HTML-a ili pisanja prilagođenog CSS-a. Za razliku od drugih CSS *framework-a* kao što su "*Bootstrap*" ili "*Materialize*", Tailwind CSS ne nudi potpuno

stilizovane komponente poput dugmadi, navigacijskih traka itd.

Kako bi se Tailwind CSS mogao koristiti potrebno je da se preuzmu odgovarajući paketi i podesi konfiguracija u našem projektu, ali Tailwind CSS je sastavna tehnologija svih Laravel-ovih starter kit-ova te dolazi instaliran i konfigurisan za rad. U našem "resources/css/app.css" file-u možemo naći tri Tailwind direktive: `@tailwind base;` `@tailwind components;` `@tailwind utilities;` za svaki sloj unutar "*index.css*" fajla od Vite projekta koji je integrisan u Laravel sa svojom Blade direktivom i može se naći definisan u layout file-ovima naše aplikacije. Vite se koristi radi lakšeg spašavanja naših asset-a za development i production razvojne faze našeg projekta.

5.16 Kreiranje web stranica

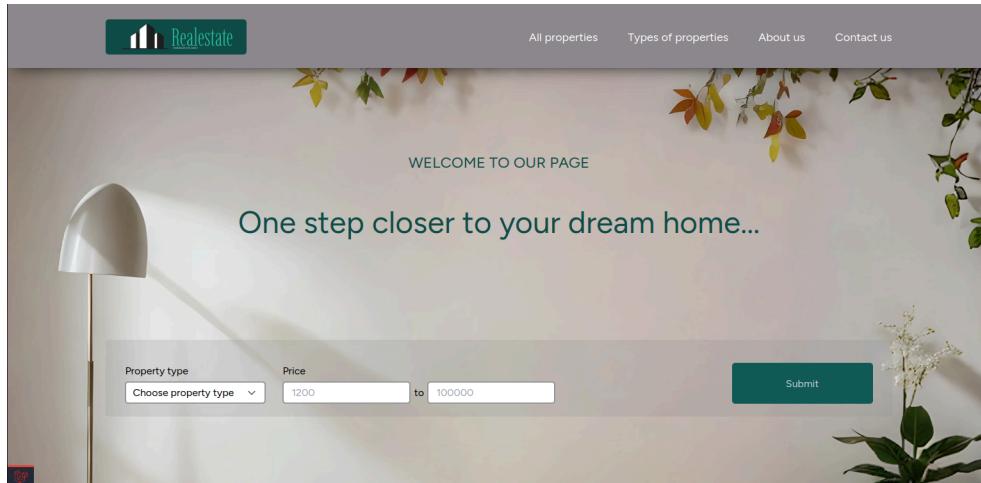
Prema unaprijed definisanim korisničkim zahtjevima, Frontend dio aplikacije će imati tri različite grupe stranica: agent, admin i guest stranice, s tim da će admin i agent stranice imati sličan raspored i strukturu ali sa različitim sadržajem koji ćemo ograničiti korištenjem rola i permisija. U narednom dijelu opisat ćemo strukturu, funkcionalnosti i dizajn svake web stranice. Važno je napomenuti da sve stranice su napravljene da budu "*responsive*" kako bi ovu aplikaciju korisnici mogli koristiti i na mobilnim uređajima, s tim da može postojati razlika u rasporedu ili izgledu jednog dijela. Za početak, prvo su editovane postojeće stranice za autentifikaciju koje dolaze kao dio Laravel Breeze starter kit-a, tačnije stranice za login i registraciju. Ovo su jednostavne stranice koje sadrže formu za popunjavanje podataka kako bi korisnik izvršio login i registraciju. Pošto u našem sistemu neće postojati klasična registracija, već će admin sistema kreirati agent profile, zasad smo izbrisali stranicu za registraciju, te samo uredili login stranicu.



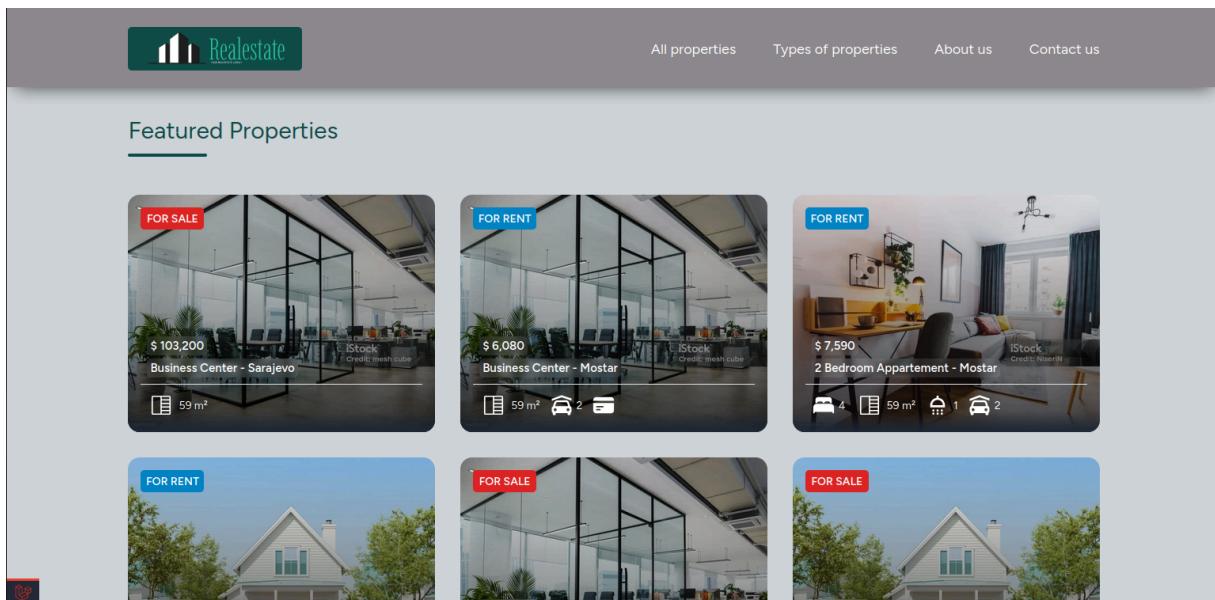
Slika 22. Sadržaj Login stranice

Ove stranica (slika 22.) koristi Breeze generisane funkcionalnosti za login i autentifikacijske feature. Klikom na login dugme, pokreće se “store” metoda koja kreira novi auth session token, nakon uspješne autentifikacije unesenih korisničkih podataka. Destroy funkcija, je funkcija koja handla logout request pri klikom na logout dugme, kada već postoji Auth sesija. Funkcije kao što su *"logout"* i *"guard"* su sastavni dio Auth Fasade, *"authenticate"* i *"session"* funkcije su dio Request objekta koje ubacujemo u naš kontroler preko *use Illuminate\Http\Request* zavisnosti. Nakon login page-a kojem se pristupa putem web pretraživača dodavanjem “/login” nastavka, na red dolazi “landing page” odnosno glavna početna stranica posjetiocima stranice agencije sa homepage rutom “/”. Glavna stranica se sastoji iz nekoliko dijelova kao što su *"header"* (engl. “zaglavje”) koji ima prikazanu poruku i malu formu koja pokreće *invokeable* SearchController klasu, *"topbar"* koji na mobilnim preglednicima postane “dropdown” koja sadrži dijelove landgin stranice kao što su tipovi nekretnina, “o nama” sekcija i “properties” link koji vodi na “/properties” rutu, dio koji prikazuje najnovije nekretnine dodane u naš sistem, dio sadržaja za vrstu nekretnina u ponudi, dio koji opisuje agenciju i footer-a. Važno je napomenuti da ukoliko se koristi uređaj čiji je ekran manjih dimenzija, onda u topbar dio postaje sidebar, te će se pojaviti dugme koje će otvoriti “dropdown” sa spomenutim linkovima. Glavni sadržaj stranice pruža osnovne informacije o samoj agenciji, vrsti nekretnina s kojom raspolaze i malim preglednikom. Na sljedećim slikama 23. i 24. je

prikazan dizajn header dijela stranice i dio stranice koji prikazuje najnovije nekretnine u našes sistemu.



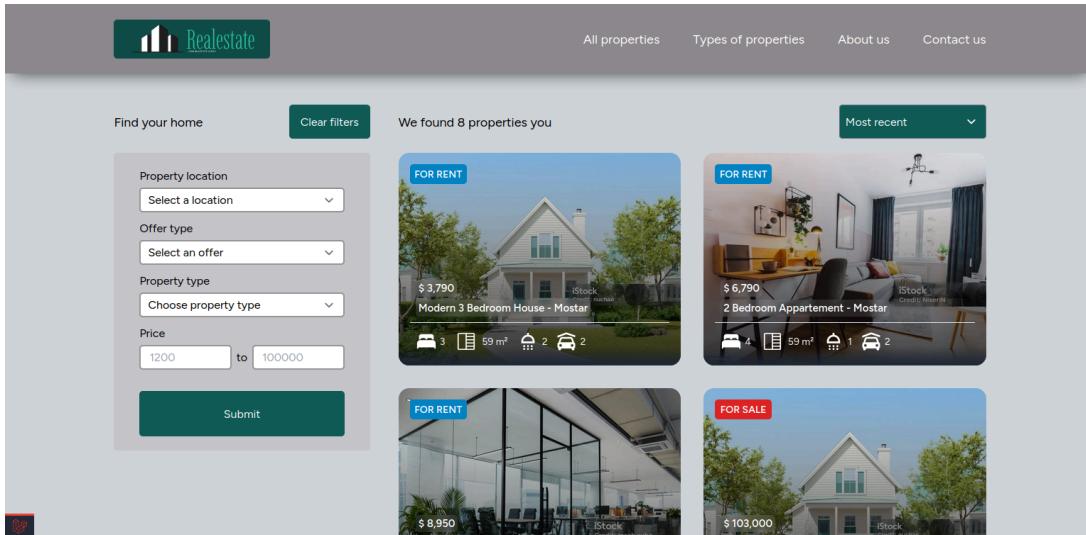
Slika 23. Prikaz glavnog dijela početne stranice



Slika 24. Prikaz najnovijih nekretnina u sistemu

Nakon završene početne i login stranice, slijede property stranice za potencijalne klijente agencije. To su stranice “index-property” i “property/{id}”. Ove stranice su inicijalno pravljene koristeći samo Blade file-ove, koji sami od sebe ne pružaju osjećaj “žive” stranice, već su server side renderovani html. Tu se pojavila potreba za reaktivnosću stranice. Pri dizajniranju frontend arhitekture ovog sistema planirali smo da koristimo Blade komponente, osim u slučaju kada određena komponenta treba da bude reaktivna,

tada bih koristili Livewire komponente. Ove stranice su kombinacije Livewire i Blade komponenti.



Slika 25. Prikaz “all-properties” page-a

Na “*index-property*” stranici (slika 25.) možemo vidjeti prikaz dvije glavne komponente ove stranice, formu pretrage sa lijeve i sekciju ponuđenih nekretnina sa desne strane. Obe komponente zahtjevaju reaktivnost jer klikom na “Submit” dugme, poslat će se request na server, koji će na osnovu inputa iz forme izvršiti upit nad bazom podataka, te nam poslati nazad listu Property objekata, koji su query-ani iz baze, te će restartovati “state” forme i onda “dispatchati” event da je forma podnesena, koju sluša akcija unutar Livewire komponente koja prikazuje nekretnine sa desne strane stranice, te se ona onda re-rendera sa novom listom nekretnina i na taj način se ažurira stranica sa dvije livewire komponente. Naredni isječak koda prikazuje *submitForm* akciju koja se pokrene klikom na Submit dugme.

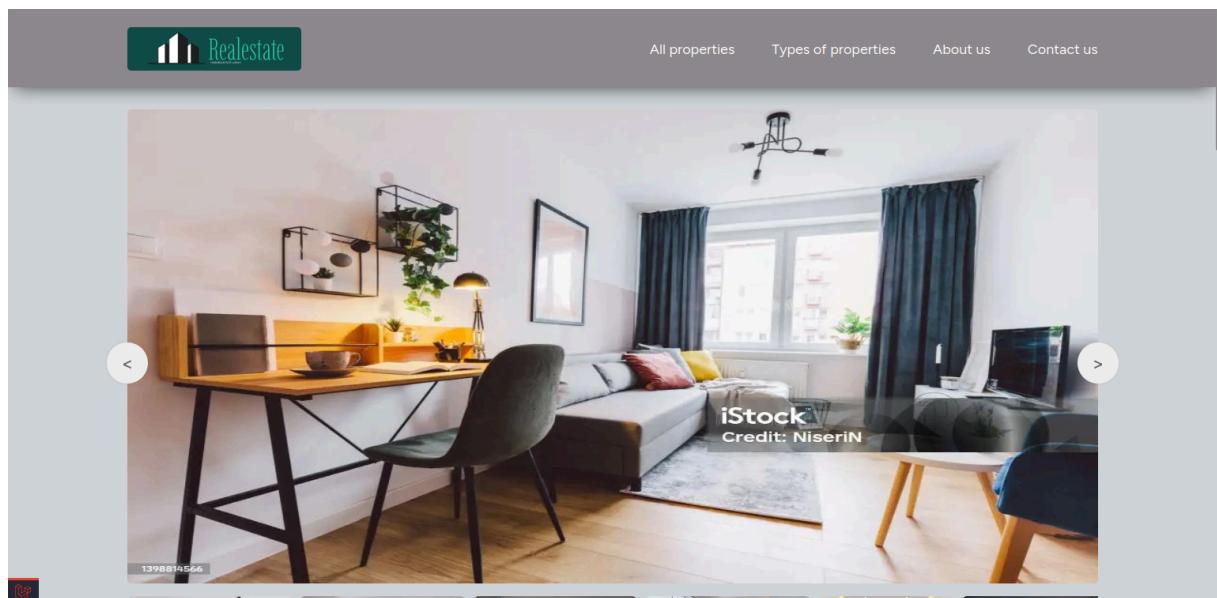
```
public function submitForm()
{
    $this->dispatch('form-submitted', filters: $this->filters);
    $this->reset('filters');
}
```

Ovaj proces nije asinhronan iz sigurnosnih razloga, jer želimo prvo da potvrdimo validnost podataka prenešenih kroz formu, te tek onda da pošaljemo event na komponentu sa nekretninama koja će da se rendera po novoj listi Property objekata. Način na koji Livewire raspoznaće i radi re-renderanje smo već objasnili, ali najbolja odlika kombinacije

Livewire i Laravel tehnologija jest što se renderaju na server strani. Trenutno taj proces traje malo duže, jer server radi na strani privatnog uređaja, a ne na pravom produkcijskom serveru. Pomoću spomenutih dispatch funkcija uspostavljena je komunikacija između dvije komponente koje se nalaze na istoj stranici, a razlog za korištenje dvije individualne komponente umjesto definisanja jedne komponente koja će predstavljati cijeli page, jeste što u slučaju sortiranja nekretnina koje su prikazane sa desne strane, koje možemo sortirati pomoću dugmeta u desnom dijelu komponente (na osnovu cijene ili datuma dodavanja u sistem), ne želimo ponovo renderati formu koja nema razloga da bude renderana. Sljedeći isječak koda je iz kontrolera komponente koja prikazuje nekretnine, te možemo vidjeti `#On` atribut koji označava da se ova funkcija pokreće kada se „*form-submitted*“ event dispatcha.

```
#[On('form-submitted')]
public function updateFilters($filters)
{
    $this->resetPage();
    $this->filters = $filters;
}
```

Posljednja stranica kojoj gosti imaju pristup jeste stranica koja prikazuje sve slike i detalje jedne nekretnine. Ruta za spomenutu stranicu je “`property/{id}`” gdje je varijabla “`id`” zamjenjena stvarnim id-om nekretnina koja je učitana iz baze i otvorena u našem pregledniku, te izgled stranice vidimo na slikama 26, 27, 28 i 29.



Slika 26. Otvaranje “single-property” stranice

Pri učitavanju “single-property” stranice prvo nam je prikazana velika galerija, sa svim slikama nekretnine koju smo izabrali za pregled. Sve slike su učitane iz baze unutar kontrolera, zajedno sa osnovim podacima pomoću jednog query-a koji smo napisali pomoću Laravel-ovog Query Builder-a koji nam pruža jednostavan i razumljiv interfejs za kreiranje i pokretanje upita nad našom bazom podataka.[22] Jedna od prednosti korištenja Eloquent modela unutar Laravel projekata jeste jednostavno dobavljanje podataka iz drugih tabela kada imamo prisutne strane ključeve unutar naših objekata koje dobavljamo iz baze podataka. U ovom slučaju, na veoma jednostavan način dobavljamo, slike iz “media” tabele i podatke o agentu nekretnine iz “users” tabele.

FOR RENT 2 Bedroom Appartement

Mostar, Bosnia & Herzegovina

Description

If you have older kids or frequent guests, this home is meant for you. A spacious loft at the top of the stairs adjoins two bedrooms with walk-in closets and a shared bath, providing just the right amount of privacy for everyone. With an open-concept layout in the great room, kitchen and casual dining, there's plenty of space and seating to bring the whole family together. Stream your favorite videos. Adjust the lighting. Check the front door without leaving your comfy seat. Our HomeSmart features are included and connect tech to your device.

Basic Details

SQM 59 m ²	Year built 1991	Garage No
Furnished No	Intercom Yes	Lease Yes

Contact Us

Price: \$6,790
Mostar, Bosnia & Herzegovina

Agent: Ms. Thora Hermann I
Contact us: 475.607.0701

Slika 27. Prikaz detalja o nekretnini i agentu

Location

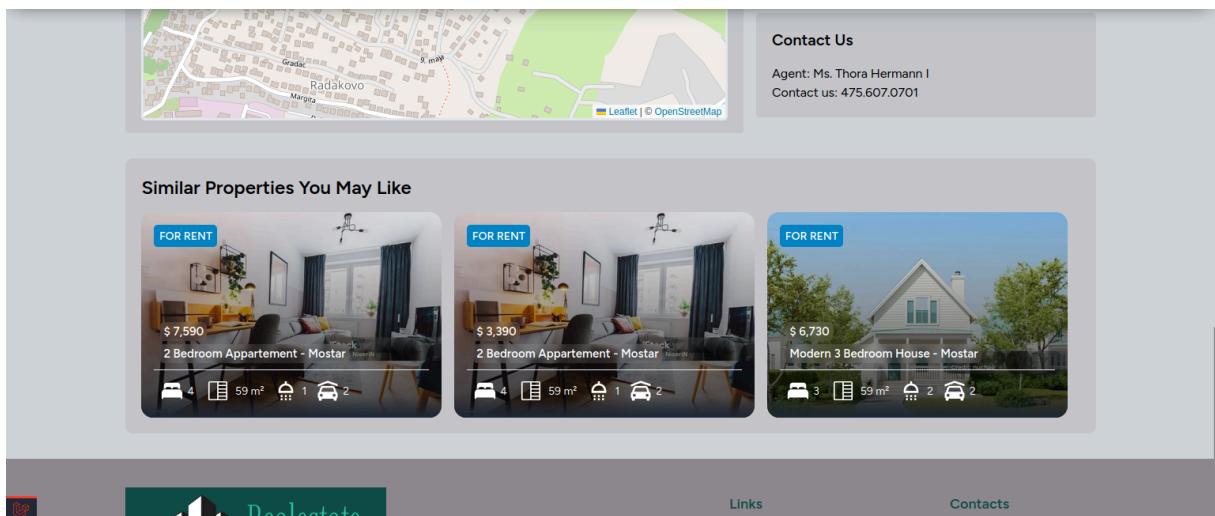
Modern 3 Bedroom House

Contact Us

Agent: Ms. Thora Hermann I
Contact us: 475.607.0701

Slika 28. Prikaz mape i lokacije nekretnine na mapi.

Postavljanje mape je omogućeno korištenjem “leaflet.js” JavaScript paketa. Leaflet je najkorištenija open-source JavaScript biblioteka za interaktivne mape koja je podržana i za mobilne aplikacije. Leaflet je dizajniran da bude jednostavan, dobrih performansi i lagan za implementirati u sve sisteme.[23]



Slika 29. Prikaz predloženih nekretnina koje bi se mogle svidjeti korisnicima

Ispod mape (slika 29.) vidimo sekciju stranice koji korisniku nudi nekoliko sličnih nekretnina koje bi ga mogle interesovati. Nekretnine koje su ponuđene u ovom dijelu su ponuđene na osnovu rezultata “item-to-item collaborative filtering” sistema koji je implementiran unutar ove aplikacije. U pitanju je sistem preporuke, koji uzima u obzir nekretninu, “nekretnina X”, koju je korisnik otvorio (pokazao interesovanje za nju) i na osnovu određenih podataka te nekretnine, sistem kreira upit za nekretnine koje imaju identičan set podataka (lokacija, vrsta ponude, tip nekretnine) te onda na osnovu nekih parametara nekretnine X, izdvaja nekretnine koje su najsličnije nekretnini X i preporučuje tri nekretnine koje imaju najviši rezultat sličnosti sa nekretninom X. Strukturu stranice možemo vidjeti na narednim slikama. Definisanim korisničkim zahtjevom da glavna slika u galeriji bude promjenjiva, ukazala se potreba za korištenjem Livewire framework-a i njegovih komponenti na ovoj stranici.

Nakon završenih stranica za klijente i stranica za autentifikaciju, slijede stranice za administratore i agente. Ove stranice imaju skoro identičan raspored i strukturu, osim što admin ima vidljive opcije na sidebar-u za funkcije nad agentima i aktivnostima (“actions” stranicama), dok su ti dijelovi agentima sakriveni, ali glavna razlika ovih stranica je u sadržaju koji je prikazan, odnosno podacima koji se nalaze na stranici u zavisnosti od role korisnika. Iako linkovi za aktivnosti i agent stranice nisu prisutni u agent dashboard-u, svaka ruta koja ima određeno autorizacijsko pravilo definisano u korisničkim zahtjevima, potrebno je da osiguramo i autorizaciju na backend strani. Postoji nekoliko načina na koje možemo ograničiti takav pristup koristeći Laravel-ove tehnologije i ugrađene metodologije, te se ovakav zahtijev može riješiti implementiranjem middleware pravila na rutama koje definišemo u routes folder-u.

```
Route::middleware(['role:admin', 'auth', 'verified'])->group(function () {  
    Route::get('admin/dashboard', function () {  
        return view('admin.dashboard');  
    })  
        ->name('dashboard');
```

Definisanjem middleware-a i korištenjem group() funkcije, definišemo middleware pravila nad svim rutama koje se nalaze unutar group() funkcije, a pravila koja smo definisali su: role:admin, auth i verified. Pravilo “*role:admin*” definiše da user-i koji imaju rolu admin, imaju pristup ovim rutama, pravilo “*auth*” definiše da ovoj ruti imaju pristup samo autentificirani korisnici i pravilo “*verified*” definiše da ovoj ruti imaju pristup samo korisnici koji su potvrdili svoju e-mail adresu. Od ovih najbitnije je “*role:admin*” pravilo. Sve stranice za autentifikovane korisnike koje su namjenjene administratorima i agentima imaju sa desne strane sidebar koji je stalno otvoren dok je na uređajima manje veličine sidebar uklonjen te se može prikazati klikom dugmeta koje se nalazi na vrhu stranice pored ikone.

Stranice koje su dostupne i agentima i administratorima jesu “dashboard” stranica i stranice koje se nalaze pod “properties” naslovom na sidebar-u. Te stranice su “add new”, “all properties”, “houses”, “appartements” i “offices”. Stranica “add new” je stranica koja

sadrži formu za kreiranje nove nekretnine i dodavanje iste u sistem, razlika između agent i admin forme je u tome, što admin kada kreira novu nekretninu ima dodatnu opciju da tu nekretninu dodijeli jednom od agenata iz sistema, dok agent kada kreira nekretninu, nema to polje i pri kreiranju nekretnine, ona se automatski njemu dodjeljuje. Na slikama 30. i 31. možemo vidjeti admin formu za dodavanje nekretnine sa svim unosnim poljima i poljem za odabir agenta.

The screenshot shows the 'Create New Property' form. It has two columns. The left column contains fields for Name (E.g. Two bedroom house), Price (E.g. 100000), Year Built (E.g. 1990), Offer Type (Choose offer type), Property Type (Choose property type), Choose agent (Choose an agent), Description, and Location. The right column contains fields for Number of Bedrooms, Number of Toilets, Number of Floors, Number of Rooms, Property surface, and checkboxes for Garden, Garage, Intercom, Furnished, Elevator, and Keypad entry.

Slika 30. Prikaz prvog dijela agent forme za dodavanje nove nekretnine

This screenshot shows the second part of the 'Create New Property' form. It includes a map of Bosnia and Herzegovina with city labels like Sarajevo, Zenica, Mostar, Trebinje, and Banja Luka. Below the map are fields for City and Street, with a note: 'Please make sure to double check the data entered by the map (City, Street)'. At the bottom are buttons for 'New photo', 'Reset photos', and 'Create Property'.

Slika 31. Prikaz drugog dijela agent forme za dodavanje nove nekretnine

Stranica “all properties” (slika 32.) je stranica koja sadrži tabelu svih nekretnina, malu formu u header dijelu tabele pomoću kojeg agent i admin mogu da pretraže sve nekretnine po određenim karakteristima i još ima dugme koje vodi na “add new property” stranicu

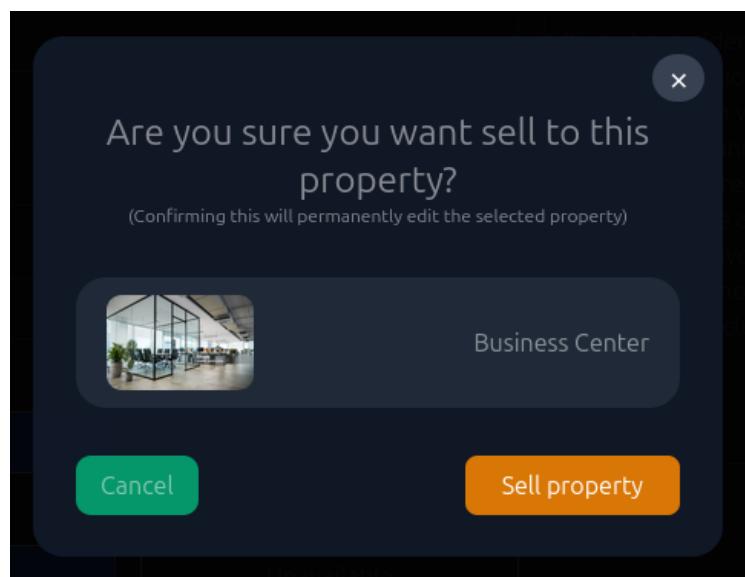
koju smo prethodno obradili. Jedina razlika između admin i agent dijelova ove stranice jeste u količini podataka koji su dostupni. Agenti na ovoj stranici vide sve nekretnine koje su njima dodjeljene i može da vidi sve nekretnine koje su statusa “Available” i “Unavailable”, te od izlistanih nekretnina svaku može detaljnije pogledati na “/agent/property/{id}” ruti, ali samo nekretnine koje su njemu dodjeljne, agent može urediti, to jest, može pristupiti ruti “/agent/property/{id}/edit” i urediti tu nekretninu. Na dnu tabele možemo vidjeti i da je implementiran sistem paginacije stranica tabele. Paginacija je veoma bitna jer utječe na postepeno učitavanje podataka sa backend strane, što olakšava pristup stranici.

Image	Agent	Price	City	Offer Type	Property Type	Year Built	Status
	Kerim Nezo	\$ 220,000	Zenica	FOR SALE	House	2024	Unavailable
	Burley Bayer	\$ 70,000	Vitez	FOR SALE	House	2020	Unavailable
	Burley Bayer	\$ 2,350	Zenica	FOR RENT	Office	2017	Unavailable
	Burley Bayer	\$ 8,940	Zenica	FOR RENT	Appartement	2013	Unavailable
	Burley Bayer	\$ 11,730	Zenica	FOR RENT	House	2019	Unavailable

Slika 32. Prikaz agent “properties/index” stranice

Admin stranica “admin/property/index” daje prikaz svih nekretnina iz sistema, te nudi dodatnu opciju, “delete” koja soft delete-a nekretninu iz sistema. I agent ima mogućnost da uređuje veći dio nekretnina, osim onih koje su statusa “sold”, “rented” i “removed” (nekretnine koje su soft obrisane) iz razloga što se te nekretnine ne smatraju u ponudi, ali njihovo prisustvo je još uvijek potrebno radi pravilnog računanja ukupne prodaje na “Sales” grafu na dashboard page-u. Stranice “offices”, “houses” i “appartements” su izvedene stranice od stranice “all properties” koje prilikom učitavanja nekretnina iz baze dodatno postave “where” klauzu da se dobiju nekretnine odgovarajućeg tipa, ako želimo od početka samo određeni tip nekretnine.

Nakon toga prikazat će mo “property/{id}/edit” stranice (slike 34. i 35.), gdje admin i agenci mogu da uređuju podatke o nekretninama, najčešće samo one koji su promjenjivi ili podaci brojnog tipa sa kojima može doći do pogreške. Glavna razlika između mogućnosti agenta i admina na ovoj stranici, jeste to što ako je nekretnina nema vrijednost u kolonu “transaction_at”, što predstavlja da nekretnina nije prodana ni iznajmljena, admin ima dodatnu opciju da proda ili iznajmi nekretninu u zavisnosti od njenog tipa ponude, dok agent nema tu opciju na edit formi. Klikom na dugme da se nekretnina proda ili iznajmi, pojavljuje se modal (popup forma) koja traži potvrdu od admina za prvočitnu akciju, čiji prikaz možemo vidjeti na slici 33.



Slika 33. Prikaz modal-a (popup-a) za prodaju nekretnine

Update Property Information

Property images:



[New photo](#) [Reset photos](#)

Agent: Oswald Lubowitz	Description: If you have older kids or frequent guests, this home is meant for you. A spacious loft at the top of the stairs adjoins two bedrooms with walk-in closets and a shared bath, providing just the right amount of privacy for everyone. With an open-concept layout in the great room, kitchen and casual dining, there's plenty of space and seating to bring the whole family together. Stream your favorite videos. Adjust the lighting. Check the front door without leaving your comfy seat. Our HomeSmart features are included and connect tech to your device.
Name: Business Center	
Price: 103200	
Offer Type: For Sale For Rent	

Slika 34. Prikaz forme za uređivanje nekretnine

Price: 103200	Description: If you have older kids or frequent guests, this home is meant for you. A spacious loft at the top of the stairs adjoins two bedrooms with walk-in closets and a shared bath, providing just the right amount of privacy for everyone. With an open-concept layout in the great room, kitchen and casual dining, there's plenty of space and seating to bring the whole family together. Stream your favorite videos. Adjust the lighting. Check the front door without leaving your comfy seat. Our HomeSmart features are included and connect tech to your device.
Offer Type: For Sale For Rent	
Status: Available Unavailable	
Update Property Sell property	

Powered by [Kerim Nezo](#)

Slika 35. Prikaz ostatka forme za uređivanje nekretnine

Te zadnja od stranica za nekretnine jeste “view” stranica, gdje agent i admin imaju identičan pregled svih detalja o nekretnini, jedina razlika je u pristupu stranici, gdje agenti mogu da pristupe samo nekretninama koje su im ponuđene preko index tabele, dok admini mogu pristupiti svim nekretninama. Rute kojima agenti pristupaju na poglede svih nekretnina imaju sloj autorizacije na početku akcije, gdje se provjerava da li agent može vidjeti podatke za taj tip nekretnine.

The screenshot displays a real estate listing interface. At the top, there is a section titled "Property Images" featuring six thumbnail images of various houses. Below this is a section titled "Agent information" containing a table with three rows: Name (Burley Bayer), Email (qkihn@gutkowski.com), and Phone Number ((980) 962-2787). The next section is "Property information", also containing a table with five rows: Name (Modern 3 Bedroom House), Type (House), Price (\$ 2,230), City (Zenica), and Street (Mejdandžik 11). To the right of the property information is a map titled "Property location" showing the house's position relative to streets like "Majmunska Put" and "Majmunska Plaz".

Slika 36. Prikaz view stranice nekretnine

Naredna stranica koja je dostupna agentima i administratorima jeste “dashboard” stranica. Jedan dio dashboard page-a je isti za agenta i admina a to je sekcija sa grafovima.

Na vrhu stranice, nalazi se sekcija sa dva grafa. Prvi je PieChart, koji, za admina, nudi procentualni prikaz svih nekretnina po statusu u sistemu, dok za agenta, nudi procentualni prikaz svih nekretnina po statusu od nekretnina koje su njemu dodjeljene. Naredni je LineChart, koji, za admina, nudi grafički prikaz ukupne prodaje i rente od svih nekretnina u određenom vremenskom periodu od 3, 6 ili 12 mjeseci, dok za agenta, nudi grafički prikaz ukupne prodaje i rente od nekretnina koje su njemu dodjeljene u određenom vremenskom periodu od 3, 6 ili 12 mjeseci. Na dnu stranice se nalazi tabela, koja, za admina, prikazuje 5 najnovijih aktivnosti koje su se desile u sistemu i nudi opciju odlaska na stranicu koja prikazuje tabelu svih aktivnosti, dok se agentu prikazuje tabela svih nekretnina koje su mu dodjeljene i statusa su “Unavailable”, koje predstavljaju nekretnine sa kojima nešto ”nije u redu”, te iz tog razloga ne mogu biti postavljene na stranicu na prodaju ili rentu, ali može im se pristupiti kao nekretnini koja će biti u ponudi u skorijoj budućnosti.

Activities						
ID	Agent	Activity	Property	Created	Options	
190	admin	SOLD	Modern 3 Bedroom House	1 week ago	🕒	
189	Kerim Nezo	EDITED	One bedroom house	1 month ago	🕒	
188	Kerim Nezo	CREATED	One bedroom house	1 month ago	🕒	
187	Kerim Nezo	EDITED	One bedroom house	1 month ago	🕒	
186	admin	REMOVED	2 Bedroom Appartement	1 month ago	🕒	

See all Actions

Powered by Kerim Nezo

Slika 37. Prikaz tabele aktivnosti

Your pending properties						
Image	Price	City	Offer Type	Property Type	Year Built	Status
	\$ 97,300	Banja Luka	FOR SALE	Office	1985	Unavailable
	\$ 111,000	Banja Luka	FOR SALE	House	2013	Unavailable
	\$ 66,300	Banja Luka	FOR SALE	House	2001	Unavailable
	\$ 84,000	Banja Luka	FOR SALE	House	2011	Unavailable
	\$ 139,400	Banja Luka	FOR SALE	Appartement	2010	Unavailable

« Previous

Showing 1-5 of 7 results

Next »

Slika 38. Prikaz tabele nekretnina u pripremi

Do sada smo obradili stranice koje su vidljive i agentima i adminu, gdje glavne razlike između njih jesu u količini podataka i opcijama koje su ponuđene tim stranicama u zavinosti od role koju ima autentificirani korisnik. Ostale stranice u sistemu su dostupne samo administratoru sistema. U sidebar-u administratora imamo dodatnu sekiju “Agents” pod kojom se nalaze “List” i “Add new” opcije, i dodatna “Activities” opcija pod “Dashboard” sekijom. Već smo spominjali tabelu aktivnosti koja se prikazuje Adminu na “dashboard” stranici i njenu opciju “See all actions”. Activities stranica je stranica na kojoj admin ima prikaz svih akcija i zapisanih aktivnosti koje su se desile nad nekretninama. Ona se sastoji od proširene tabele akcija koja je slična onoj na “dashboard” stranici i malo filter forme koja se nalazi na header dijelu tabele, pomoću koje možemo filtrirati akcije po njihovom naslovu i agentu koji je učinio promjenu nad nekretninom. Za prikaz akcija u tabeli implementiran je sistem paginacije, zbog veće količine podataka

koja će biti prisutna u bazi zbog prirode “Action” modela, te je ograničeno učitavanje po 10 akcija na listu tabele.

Activities table					
Id	Agent	Activity	Property	Created	Options
190	admin	SOLD	Modern 3 Bedroom House	2 weeks ago	(o)
189	Kerim Nezo	EDITED	One bedroom house	1 month ago	(o)
188	Kerim Nezo	CREATED	One bedroom house	1 month ago	(o)
187	Kerim Nezo	EDITED	One bedroom house	1 month ago	(o)
186	admin	REMOVED	2 Bedroom Appartement	1 month ago	(o)
185	admin	SOLD	Three bedroom house	1 month ago	(o)
184	Kerim Nezo	EDITED	One bedroom house	1 month ago	(o)
183	admin	CREATED	One bedroom house	1 month ago	(o)
182	admin	CREATED	Three bedroom house	1 month ago	(o)
181	admin	EDITED	2 Bedroom Appartement	1 month ago	(o)

Slika 39. Prikaz tabele aktivnosti u pripremi

Na slici 39. možemo detaljnije vidjeti dizajn tabele i njen sadržaj. Jedan od najbitnijih dijelova jeste raspored i dizajn naziva aktivnosti te opcije koje bi mogle biti ponuđene nad akcijama. Praćenjem korisničkih zahtjeva definisana je samo ogledna uloga akcija i njihovih podataka.

Za sve izlistane akcije u tabeli, ponuđena je samo view opcija, koja otvara novi page “admin/activity/{id}” koji nam prikazuje detaljnije sadržaj Action objekta kao što je prikazano na slici 40.

Activity information	
Key	Data
Property	One bedroom house
Property Id	110
Name	EDITED
Message	Property was edited by agent: Kerim Nezo
Created At	January 21, 2025 at 10:25 PM (1 month ago)

Slika 40. Sadržaj “admin/actions/{id}” stranice

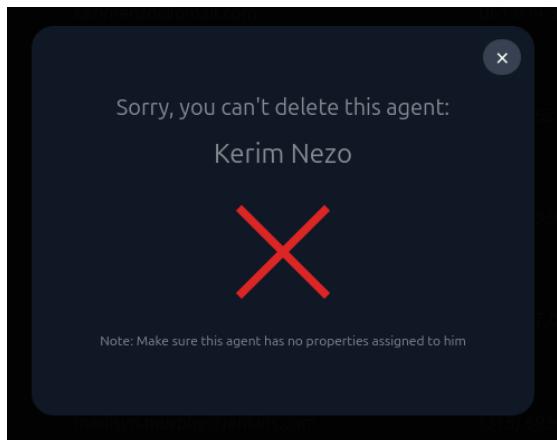
Posljednje stranice za admina koje imamo u sistemu jesu stranice koje se nalaze pod “Agents” sekcijom u sidebar-u. Od ponuđenih opcija na sidebar-u imamo „List” i „Add new”, prva od kojih nas vodi na “admin/agent/index” stranicu gdje imamo prikaz tabele sa svim agentima u našem sistemu, dok druga vodi na “admin/agent/create” rutu, koja otvara stranicu sa formom da dodamo novog agenta u sistem.

Agents table				
Avatar	Name	Email	Phone Number	Agent properties
	Kerim Nezo	kerimenzo@gmail.com	061 034 357	 View Edit Delete
	Burley Bayer	qkihn@gutkowski.com	(980) 962-2787	 View Edit Delete
	Oswald Lubowitz	katelyn.kling@sanford.com	810.389.9078	 View Edit Delete
	Ms. Thora Hermann I	vada29@johns.biz	475.607.0701	 View Edit Delete
	Prof. Gianni Rau	madiSyn.murphy@jenkins.com	(315) 693-0218	 View Edit Delete

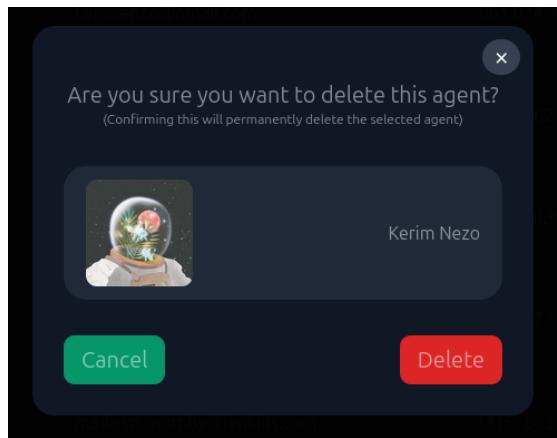
Slika 41. Prikaz Agent tabele sa “admin/agent/index” stranice

Na slici 41. možemo vidjeti izgled agent tabele, na kojoj imamo prikaz slike, naziva i kontakt informacija agenta. Pored toga vidimo i sliku jedne od nekretnina koja je dodjeljena agentu koja vodi na property sekciju “admin/agent/{id}” stranice, te na kraju imamo i ponuđene opcije za pregled, uređenje i brisanje agenta.

Brisanje je onesposobljeno u slučaju da agent ima zaduženih nekretnina i klikom na dugme za brisanje javlja se poruka (slika 42.) da agent ne može biti obrisan ili dodatna forma kojom trebamo potvrditi (slika 43.) brisanje agenta.

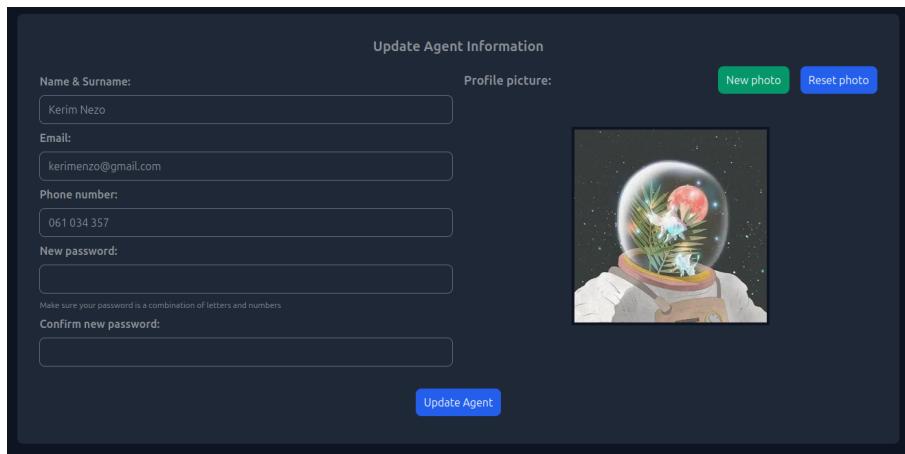


Slika 42. Prikaz modala upozorenja sa porukom pri pokušaja brisanja agenta koji ima nekretnine



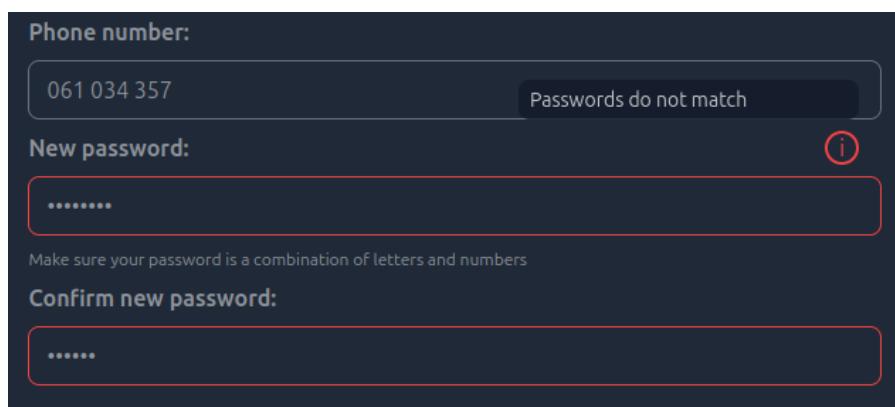
Slika 43. Prikaz modala potvrde sa porukom pri pokušaja brisanja agenta koji nema nekretnine

Pregled agenta nas vodi na “admin/agent/{id}” stranicu gdje imamo prikaz svih podataka o agentu i tabelu svih nekretnina koje su dodjeljene tom agentu. U header dijelu agentovih podataka imamo i dugme “Edit agent” koje nas vodi na “admin/agent/{id}/edit” stranicu gdje imamo formu za ažuriranje podataka o agentu. Tabela sa agentovim nekretninama je “agent-property-table” livewire komponenta, koja se koristi i na agentovom dashboard-u za prikaz nekretnina u pripermi. Ovo je jedan od benefita korištenja sistema komponenti u razvoju frontend dijela aplikacije, gdje možemo vidjeti korist “lomljjenja” stranica na manje, ponovo koristive komponente. Isto tako, na ovoj tabeli je implementiran sistem paginacije radi lakšeg učitavanja podataka.



Slika 44. Izgled “agent/{id}/edit” stranice sa formom za update podataka o agentu

Na slici 44. imamo prikaz Livewire komponente koje ima ulogu Edit forme za agentove podatke, gdje možemo promijeniti osnovne podatke o agentu, dodijeliti mu novu šifru i promijeniti profilnu sliku. Iznad trenutne slike agenta možemo vidjeti dva dugmeta “New photo” i “Reset photo”, gdje prvo otvara local storage iz kojeg možemo da uploadamo novu sliku koja će se pohraniti pri spašavanju forme ili ako smo dodali novu sliku a želimo ipak zadržati staru, klikom na “Reset photo” dugme, vraćamo staru sliku koja se nalazi u bazi podataka.



Slika 45 Prikaz validacije za dodjelu nove šifre agentu

Posljednja pregledna stranica za admina jeste “admin/agent/create” stranica, koja nam nudi mogućnost kreiranja i dodavanja novog agenta u sistem.

The screenshot shows the 'Create New Agent' form with the following data entered:

- Name & Surname: John Doe
- Email: johndoe@mail.com
- Phone number: (xxx) XXX XXXX
- New password: (empty field)

The 'Profile picture:' field contains a placeholder image of a person in a suit. A green button labeled 'New photo' is visible. The 'Create Agent' button at the bottom is blue.

Slika 46 Prikaz adminove forme za dodavanje agenta u sistem

Na lijevoj strani forme možemo vidjeti unosna polja za osnovne podatke o agentu kao što su ime i prezime, email, broj mobitela i lozinka koju unosi admin, dok na desnoj strani forme vidimo polje za unos profilne slike agenta. Nad svim poljima je implementiran sistem validacije pri popunjavanju polja, te postoji i validacija koja se pokreće nakon klika na "Create Agent" dugme. Na slici 47. možemo vidjeti promjene u formi ako validacija na nekim poljima nije zadovoljena.

The screenshot shows the same form as above, but with validation errors:

- Name & Surname: 'a' (highlighted with a red border)
- Email: 'johndoe' (highlighted with a red border)
- Phone number: '123' (highlighted with a red border)
- New password: '....' (highlighted with a red border)

The 'Profile picture:' field still contains the placeholder image, but a red message 'Please upload a photo' is displayed below it. The 'Create Agent' button is blue.

Slika 47. Prikaz validacijskih poruka pri unosu loših podataka

Ovakva mogućnost žive validacije je sposobnost Livewire komponenti, gdje unosna polja povezujemo sa property-ima komponentnih kontrolera koristeći “`wire:model`” direktivu. Na sljedećem isječku koda možemo vidjeti da na input tagu imamo “`wire:model`” direktivu sa “`.blur`” modifikatorom, koji šalje mrežni zahtjev sa ažuriranom verzijom property-a (sada property privremeno ima novu vrijednost) tek kada admin klikne van unosnog te se onda izvršava validacija u realnom vremenu [24].

```
<input type="file" id="file-upload" class="hidden" wire:model.blur="newPhoto"
/>
```

Validacijska pravila definišemo za svaki property naše komponente unutar kontrolera, gdje možemo definisati specifične vrijednosti ili samo pravila koja želimo da implementiramo. Pored definisanja pravila, možemo definisati i odgovarajuće error poruke koje će nam se vratiti u slučaju određene greške.

```
#[Validate]
public $newPhoto;
```

Na prethodnom isječku koda možemo vidjeti implementiran “[Validate]” atribut iznad definicije property-a “\$newPhoto” koji određuje da se mogu definisati validacijska pravila nad tim property-em. U narednom isječku koda možemo vidjeti funkciju *rules()* gdje definišemo validacijska pravila za naše property-e, da je on obavezan, da mora biti file ekstenzija “.jpg”, “.jpeg”, “.png”, “.webp” i da ne može biti veći od 1024KB, te u funkciji *messages()* definišemo error poruke koje se prikažu za odgovarajuće validacijsko pravilo ako ono nije zadovljeno. Klikom na dugme “*Create agent*” pokreće se *storeAgent()* akcija na kontroleru, koja na svome početku ima “*\$this->validate()*” funkciju koja pokreće i provjerava sva pravila definisana u “*rules()*” funkciji, te ako neko validacijskih pravila padne, funkcija “*storeAgent()*” se prekida i prikazuje nam se odgovarajuća validacijska poruka.

```
// Validation error rules
public function rules()
{
    return [
        'name' => 'required|min:3',
        'email' => 'required|email|unique:users,email',
        'phoneNumber' =>
'required|regex:/^(\?\d{3}\)?[-.\s]?\d{3}[-.\s]?\d{3,4}$/',
        'password' => [
            'required',
            'min:8',
            'regex:/^(?=.*[0-9])[a-zA-Z0-9]+$/',
        ],
        'newPhoto' =>
'required|mimetypes:image/jpg,image/jpeg,image/png,image/webp|max:1024',
    ];
}
```

5.17 Prikaz podataka u obliku grafova

Na vrhu admin i agent “*“dashboard”*” stranica možemo vidjeti podatke prikazane u obliku grafova. Upotrebnom biblioteke kao što su *“chartist”* i *“chartist.js”* omogućen je prikaz određenih statističkih podatka u vidu grafova.[25] Ove biblioteke uključuju već gotove komponente i potrebno je konfigurisati atribute koje možemo naći u dokumentaciji (dodijeliti nazive, boju) i proslijediti im podatke pomoću kojih će se generisati grafovi. Korisnik ima mogućnost biranja za koliko mjeseci (zadnja 3 mjeseca, pola godine ili zadnju godinu) želi da se prikažu podaci. Agent će imati prikaz istih grafova, gdje je jedina razlika što će agentovi grafovi biti kreirani koristeći set podataka samo onih nekretnina za koje je on zadužen, dok će adminovi grafovi biti kreirani koristeći set podataka svih nekretnina u sistemu. Primjer grafova možemo vidjeti na slici 48.



Slika 48. Prikaz dijela stranice sa grafovima

Unutar “*“@script”*” direktive, unutar kojih pišemo JavaScript kod, definišemo karakteristike

naših grafova prateći dokumentaciju. Za dobijanje ovih podataka definisani su upiti koji vraćaju set podataka u zavisnosti od role koju ima autentificirani korisnik koji su prikazani u narednom isječku koda.

```
// For admin we will fetch data of all agents, for agents only their data
if(Auth::user()->hasRole('admin')){
    $data = Property::query()
        ->withTrashed()
        ->select('type_id', DB::raw('count(*) as total'))
        ->where('status', $status)
        ->groupBy('type_id')
        ->get();
} else {
    $data = Property::query()
        ->withTrashed()
        ->select('type_id', DB::raw('count(*) as total'))
        ->where('status', $status)
        ->where('user_id', Auth::user()->id)
        ->groupBy('type_id')
        ->get();
}
```

Možemo vidjeti primjer upita prema bazi podataka koristeći Eloquent ORM model koji nam olakšava definisanje upita pomoću query builder metode. Isto tako možemo vidjeti korištenje hasRole funkcije unutar “*if*” kondicionala, koja nam u ovom slučaju služi kao autorizacijski metod jer provjeravamo rolu koja je dodjeljena autentificiranom korisniku čije podatke dobijamo pomoću Auth fasade, na rutu kojoj pristup imaju samo autorizovani korisnici. Ovdje imamo primjer korištenja rola, te se kroz projekat nigdje nije koristila provjera u kondicionalima po permisijama, jer su sve permisije vezane za role koje su dodjeljene svim korisnicima te se ne stvara potreba za takvom provjerom, a nema potrebe za ponavljanjem koda.

5.18 Prikaz funkcionalnosti sistema preporuke

Da bi omogućili dobro korisničko iskustvo na stranicama nekretnina za potencijalne kupce, potreban je funkcionalan sistem za preporuku nekretnina. U ovom poglavlju ćemo napraviti detaljan pregled arhitekture i rada sistema za preporuku nekretnina. Na osnovu korisničkih zahtjeva i poslovne logike aplikacije, definišu se karakteristike algoritma i pojam “sličnog objekta” odnosno *objekta X*. Sličan objekat je u definiciji objekat kakav želimo preporučiti korisniku koji sadrži podatke i kategorije prema kojima je prirodno

sličan objektu sa kojim se poredi i treba da predstavlja nešto što će privući pažnju potencijalnog klijenta. U ovom dijelu ćemo proći cijeli proces rada implementiranog algoritma i objasniti cilj iza svake akcije. Na početku `show()` akcije koja obrađuje zahtijev za pristup “`property/{id}`” ruti imamo “try” blok, prvo vidimo deklaraciju i inicijalizaciju `$property` varijable. U query builder-u definišemo da želimo sve atribute nekretnine, zajedno sa njenim podacima iz “`media`”, “`user`” i “`type`” tabele. Nakon uspješnog primanja podataka o nekretnini, definišemo varijablu `$similarProperties` kojoj dodjeljujemo rezultat funkcije `recommendSimilar()`. U definiciji akcije vidimo da ona prima dva parametra, jedan koji je predefinisan a to je brojnik `$count` koji je postavljen na tri i objekat `Property` koji predstavlja nekretninu koju je korisnik otvorio u svom web pregledniku. Na vrhu funkcije vidimo dobavljanje onih objekata, i dodjeljujemo ih `$properties` varijabli, koji su prirodno slični našem *objektu X* kojem želimo nači slične druge objekte. Prirodno je za nekretnine, odnosno za kupce nekretnina, u slučaju da ih interesuje “`kupovina`” “`stana`” u “`Sarajevu`”, da im sistem predlaže specifične nekretnine koje dijele osnovne i one najbitnije atribute sa tim ključnim pojmovima. Iz tog razloga definisan je još jedan upit na tabelu nekretnina za svim nekretninama koje dijele te identične ili slične ključne pojmove. Još jedan od razloga za ovakvim sistemom jeste olakšavanje obradnog procesa algoritma na serveru. Algoritam će ovako imati manji broj nekretnina za obraditi izračun sličnosti u odnosu da ga obrađuje za sve nekretnine u našoj tabeli. U sljedećem isječku koda možemo vidjeti definisani upit.

```
// Query all properties which share same city, offer and similar property_type
$properties = Property::query()
    ->where('id', '!=', $property->id)
    ->where('status', 'available')
    ->where('lease_duration', $property->lease_duration)
    ->where('city', $property->city)
    ->when($property->type_id === 1, function ($query) {
        $query->where('type_id', 1);
    }, function ($query) {
        $query->where('type_id', '!=', 1);
    })
->get();
```

Na kraju akcije definišemo `$recommendations` varijablu koja predstavlja mapu nekretnina zajedno sa njihovim rezultatom sličnosti sa objektom X. Na kraju vraćamo prve 3 nekretnine, u parametrima smo definisali da je `$count = 3`, koje smo prvo sortirali od najveće ka najmanje po njihovom rezultatu sličnosti i na kraju samo vratili te tri

nekretnine, bez rezultata sličnosti. Unutar `map()` funkcije nad `$properties` varijablom, izračunavamo rezultat sličnosti za svaku nekretninu unutar liste pomoću `computeSimilarity()` funkcije. Unutar ove funkcije izračunavamo rezultat sličnosti između dva objekta. Jedan od načina za izračunavanje sličnosti između objekata jeste korištenjem **Euklidove udaljenost**, gdje A i B predstavljaju vrijednost feature-a objekta, dok n predstavlja količinu feature-a:

$$d(A, B) = \sqrt{\sum_{i=1}^n (A_i - B_i)^2}$$

Pošto ovom formulom mjerimo udaljenost između dva objekta u višedimenzionalnom prostoru, sličnost, koja je pogodnija za sisteme preporuka, izražavamo preko ***inverza Euklidove udaljenosti***, koja nam osigurava da što su dva objekta sličnija jedan drugom (manja im je udaljenost), da je njihova vrijednost sličnosti veća, te ako im je udaljenost 0, onda im je sličnost 1 (ti objekti su identični):

$$\text{slučajnost}(A, B) = \frac{1}{1+d(A,B)}$$

Da bismo implementirali ovaj izračun, prvo definišemo “težine”, koje predstavljaju “feature” po kojima ćemo računati njihovu udaljenost, te ćemo dodatno povećati vrijednost izračuna na osnovu bitnosti tog feature-a kod objekta. Kao primjer uzmimo cijenu koja predstavlja jednu veoma važnu značajku pri kupovini nekretnine, što je prirodno, te je njen doprinos izračunu sličnosti dodatno povećan. Na kraju funkcije, nakon što dobijemo udaljenost nekretnina, ova funkcija vraća rezultat njihove sličnosti. Odabir atributa koji su bitni za naš objekat i definisanje njihove težine je ključno za dobar rad algoritma i kvalitet njegovih preporuka. Još jedna bitna karakteristika jeste i količina podataka koji su dostupni u našoj bazi. Ako ima manjak objekata koji imaju karakteristične sličnosti sa objektom X, moguća su dva scenarija: da algoritam ne preporučuje dovoljno slične objekte, što može predstavljati loše korisničko iskustvo ili da bude prazan prostor na stranici. Za rad i implementiranje ovakvih algoritama veoma su bitni korisnički zahtjevi, tip aplikacije i priroda objekata za koje radimo sistem preporuke.

6. ANALIZA REZULTATA

Razvoj web aplikacije za digitalizaciju poslovnih procesa agencije za nekretnine je projekat koji mi je omogućio da efikasno iskoristimo moderne tehnologije, što je omogućilo nastanak skalabilne, efikasne web aplikacije koja ispunjava korisničke zahtjeve. Kombinacijom Livewire i Blade komponenti za Frontend, Laravel za Backend i MySQL kao bazu podataka, uspio sam kreirati aplikaciju koja pojednostavljuje poslovne i menadžerske procese unutar jedne agencije za nekretnine. U ovom dijelu diplomskog rada, istaknut ćemo dostignuća i prednosti koje pružaju ove tehnologije, kao i njihove specifične snage u razvoju web aplikacija koje su ujedno bile od velikog značaja za ovaj projekat.

Livewire je igrao ključnu ulogu u implementaciji dinamičkih korisničkih interakcija u aplikaciji, omogućavajući reaktivno ažuriranje korisničkog interfejsa bez potrebe za pisanjem JavaScript koda. On je omogućio da se kompleksne funkcionalnosti, kao što su pretraga, filtriranje, paginacija i validacija, implementiraju na jednostavan i efikasan način. Jedna od ključnih funkcionalnosti implementiranih koristeći Livewire je dinamička pretraga i validacija podataka unešenih u forme. Korisnici mogu pretraživati i filtrirati nekretnine po različitim kriterijumima, poput cijene, lokacije, tipa nekretnina i tipu ponude. Paginacija podataka je još jedna funkcionalnost koja je implementirana koristeći Livewire. Prikaz velikog broja nekretnina podijeljen je na stranice kako bi se poboljšale performanse i korisničko iskustvo. Livewire komponenta upravlja paginacijom, omogućavajući korisnicima da pregledavaju podatke stranicu po stranicu. Podaci se učitavaju dinamički bez potrebe za ponovnim učitavanjem cijele stranice, što osigurava brzinu i fleksibilnost. Livewire omogućava implementaciju i korištenje PHP definisanih funkcionalnosti koristeći kontrolere koji su kreirani i "povezani" sa odgovarajućom komponentom. Pored ostalog, on je integriran sa Laravel-om, što mi je omogućilo korištenje Eloquent modela i funkcija ugrađenih u Laravel-ove Facade, te na taj način osiguravajući da ostanem unutar PHP radnog okruženja kroz cijeli razvoj frontend dijela aplikacije, te dodatno osigurava da se poslovna logika može jednostavno povezati s korisničkim interfejsom. Tailwind CSS je korišten za stiliziranje aplikacije, omogućavajući "responzivan" i dosljedan dizajn na svim uređajima. Njegove responzivne klase omogućile su komponentama da se prilagode različitim veličinama ekrana, stvarajući interfejs prilagođen mobilnim uređajima kao i tablet, laptop i desktop uređajima. Uz to, Tailwind-ov unaprijed definirani razmak, sintaksa i paleta boja osigurali

su da dizajn ostane uvezan, dok su prilagođene teme i varijante omogućile fleksibilnost stila. Ovaj pristup je rezultirao aplikacijom koja je vizuelno privlačna, responzivna i jednostavna za održavanje a ujedno uz što manji utjecaj na čitljivost koda.

Laravel je bio osnovni framework korišten za razvoj backend dijela aplikacije, pružajući snažnu podlogu za izgradnju skalabilnog, efikasnog i sigurnog sistema. Jedna od ključnih prednosti korištenja Laravel-a, bila je njegova MVC arhitektura. Ova arhitektura omogućila je jasnu separaciju logike, prezentacije i podataka, što je značajno olakšalo organizaciju koda i održavanje aplikacije. Laravel-ov Eloquent ORM (Object-Relational Mapping) igrao je ključnu ulogu u upravljanju podacima. Eloquent je omogućio jednostavno mapiranje tabele iz baze podataka na PHP objekte, što je uveliko pojednostavilo rad s bazom podataka. Umjesto pisanja kompleksnih SQL upita, koristili su se jednostavne Eloquent metode za dohvatanje, filtriranje i sortiranje podataka. Također, Eloquent je omogućio jednostavno upravljanje relacija između modela, poput veza između nekretnina i korisnika, te definisanja veza unutar modela. MySQL je korišten kao primarna baza podataka u aplikaciji, pružajući pouzdano i efikasno skladištenje podataka o nekretninama, korisnicima i akcijama. Kombinacija MySQL-a i Laravel-ovih migracija omogućila je strukturiran, organiziran i lako održiv sistem za upravljanje podacima. One igrale su ključnu ulogu u upravljanju strukturom baze podataka. Migracije su omogućile version control strukture baze. Svaka migracija predstavlja skup promjena koje se primjenjuju nad bazom podataka, poput kreiranja novih tabela, dodavanja kolona ili definiranja indeksa, te korištenjem migracija i Laravelovih Factory i Seeder servisa znatno je olakšano proces testiranja i sama provjera rada veza između modela koje smo definisali pomoću Eloquent veza u model file-ovima. Optimizacija performansi MySQL-a bila je ključna za osiguravanje brzog odziva aplikacije. Indeksiranje kolona koje se često koriste u upitima, poput “*location*” ili “*price*”, značajno je poboljšalo performanse pretrage i filtriranja.

Sveukupno izbor ovih tehnologija se pokazalo idealnim za izgradnju web aplikacije koja je namjenjena za digitalizaciju procesa u agencijama za nekretnine. Ove tehnologije ne samo da su obezbjedile kratko razvojno vrijeme, već su doprinijele u kreiranju aplikacije koja je skalabilna, lagana za održavanje i sposobna za upravljanje i rad sa mnogim uposlenicima i znatnom količinom podataka. Kombinacija modernih razvojnih tehnologija iz PHP radnog okruženja rezultiralo je projektu koji zadovoljava korisničke zahtjeve.

7. ZAKLJUČAK

Razvoj web aplikacije za digitalizaciju procesa agencije za nekretnine koristeći Laravel, MySQL, Blade i Livewire pokazao je snagu ovih tehnologija u stvaranju skalabilnih, efikasnih i korisnički orijentisanih rješenja. Ova aplikacija nije samo pojednostavila poslovne procese unutar agencije, već je i poboljšala korisničko iskustvo kroz dinamičke interakcije i brz odgovor sistema. Značaj ovog projekta ogleda se u njegovoj pristupačnosti i skalabilnosti. Korištenje Laravel-a omogućilo je brz razvoj backend sistema uz visok nivo sigurnosti i održivosti. Laravel-ov Eloquent ORM pojednostavio je upravljanje podacima, dok su migracije osigurale konzistentnu strukturu baze podataka u svim okruženjima. MySQL je pružio pouzdano i efikasno skladištenje podataka, dok je Livewire omogućio dinamičke korisničke interakcije bez potrebe za kompleksnim JavaScript kodom.

Sa porastom potrebe za digitalizacijom i sve većim oslanjanjem na kompjutersku obrazovanost, ovakve aplikacije postaju sve važnije. One ne samo da smanjuju potrebu za ručnim procesima, već i omogućavaju pristup podacima i uslugama sa bilo kojeg uređaja povezanog na internet. Ovo je posebno važno u kontekstu agencija za nekretnine, gdje su brzina, transparentnost, efikasnost i upravljanje podacima, te komunikacija ključni faktori za uspjeh. Na tržištu softverskog inženjeringu, tehnologije poput Laravel-a zauzimaju važno mjesto zbog svoje fleksibilnosti, performansi i aktivne zajednice koja ih podržava. Laravel ne samo da olakšava razvoj web aplikacija, već i omogućava programerima da se fokusiraju na stvaranje inovativnih rješenja i biznis specifičnih use case-ova umjesto na rješavanje tehničkih problema koji su univerzalni, poput autorizacije, autentifikacije i storage-a. Primjenom Laravel-a, Livewire-a i MySQL-a, uspješno je kreirana aplikacija koja ispunjava zahtjeve korisnika i postavlja odličan temelj za buduća unapređenja.

Proces razvoja nije bio bez izazova. Upravljanje velikim količinama podataka, optimizacija performansi, osiguravanje sigurnosti i integracija različitih tehnologija zahtijevali su pažljivo planiranje i kontinuirano učenje. Međutim, svaki od ovih izazova doprinio je profesionalnom rastu i boljem razumijevanju kompleksnosti razvoja web aplikacija. Kroz ovaj projekat, stekao sam vrijedna iskustva u radu sa modernim alatima i tehnologijama, kao i u rješavanju složenih problema. Ova iskustva pripremila su nas za buduće projekte i potakla na kontinuirano učenje i usavršavanje. U budućnosti, planiramo se fokusirati na daljnje unapređenje aplikacije, uključujući implementaciju naprednih

funkcionalnosti poput integracije sa eksternim servisima, poboljšanja korisničkog interfejsa i dodatnih alata za analizu podataka.

8. LITERATURA

- [1] Ultimate Laravel for Modern Web Development, Drishti Jain, Orange Education Pvt Ltd, 2024.
- [2] Battle Ready Laravel, Ashley Allen, Ashley Allen, 2022.
- [3] Programming PHP, Kevin Tatroe and Peter MacIntyre, O'Reilly Media, 2020.
- [4] MySQL Crash Course, Rick Silva, No Starch Press, 2023.
- [5] Pro PHP 8 MVC, Christopher Pitt, Apress , 2021.
- [6] Item-based Collaborative Filtering Recommendation Algorithms, Badrul Sarwar and George Karypis and Joseph Konstan and John Riedl, GroupLens Research Group/Army HPC Research Center Department of Computer Science and Engineering University of Minnesota, 2001
- [7] <https://laravel.com/docs/11.x#meet-laravel>
- [8] <https://visualstudio.microsoft.com>
- [9] <https://getcomposer.org/>
- [10] <https://spatie.be/docs/laravel-permission/v6/introduction>
- [11] <https://spatie.be/docs/laravel-medialibrary/v11/introduction>
- [12] <https://image.intervention.io/v3>
- [13] <https://www.oracle.com/mysql/what-is-mysql>
- [14] <https://dbeaver.io/about/>
- [15] <https://www.lucidchart.com/pages/uml-use-case-diagram>
- [16] <https://www.lucidchart.com/pages/uml-sequence-diagram>
- [17] <https://www.lucidchart.com/pages/uml-class-diagram>
- [18] <https://www.lucidchart.com/pages/er-diagrams>
- [19] <https://laravel.com/docs/11.x/facades#introduction>
- [20] <https://laravel.com/docs/11.x/eloquent#observers>
- [21] <https://blog.hubspot.com/website/what-is-tailwind-css>
- [22] <https://laravel.com/docs/11.x/queries#introduction>
- [23] <https://leafletjs.com/>
- [24] <https://livewire.laravel.com/docs/wire-model#updating-on-blur-event>
- [25] <https://chartist.defv/>

