

Programsko inženjerstvo

Ak. god. 2023./2024.

*Nestali ljubimci*

Dokumentacija, Rev. 2

Grupa: *For All the Dogs*

Voditelj: *Andrija Merlin*

Datum predaje: 19. 1. 2024.

Nastavnik: *Alan Jović*

# Sadržaj

<b>1 Dnevnik promjena dokumentacije</b>	<b>3</b>
<b>2 Opis projektnog zadatka</b>	<b>6</b>
2.1 Postojeća slična rješenja . . . . .	8
2.2 Moguće prilagodbe i nadogradnje rješenja . . . . .	10
<b>3 Specifikacija programske potpore</b>	<b>11</b>
3.1 Funkcionalni zahtjevi . . . . .	11
3.1.1 Obrasci uporabe . . . . .	13
3.1.2 Sekvencijski dijagrami . . . . .	18
3.2 Ostali zahtjevi . . . . .	21
<b>4 Arhitektura i dizajn sustava</b>	<b>22</b>
4.0.1 MVC stil arhitekture . . . . .	23
4.1 Programski jezici, razvojni okviri, alati i biblioteke koda . . . . .	24
4.1.1 Back-end i baza podataka . . . . .	24
4.1.2 Front-end . . . . .	24
4.2 Baza podataka . . . . .	25
4.2.1 Opis tablica . . . . .	26
4.2.2 Dijagram baze podataka . . . . .	30
4.3 Dijagram razreda . . . . .	32
4.4 Dijagram stanja . . . . .	40
4.5 Dijagram aktivnosti . . . . .	42
4.6 Dijagram komponenti . . . . .	44
<b>5 Implementacija i korisničko sučelje</b>	<b>45</b>
5.1 Korištene tehnologije i alati . . . . .	45
5.2 Ispitivanje programskog rješenja . . . . .	47
5.2.1 Ispitivanje komponenti . . . . .	47
5.2.2 Ispitivanje sustava . . . . .	55
5.3 Dijagram razmještaja . . . . .	64

5.4	Upute za puštanje u pogon . . . . .	65
5.4.1	Frontend . . . . .	65
5.4.2	Baza podataka . . . . .	68
5.4.3	Backend . . . . .	68
6	Zaključak i budući rad	72
	Popis literature	74
	Indeks slika i dijagrama	76
	Dodatak: Prikaz aktivnosti grupe	77

# 1. Dnevnik promjena dokumentacije

Rev.	Opis promjene/dodatka	Autori	Datum
0.1	Napravljen predložak	Božo Đerek	26.10.2023.
0.2	Opis projektnog zadatka Nefunkcionalni zahtjevi, dio funkcionalnih zahtjeva	Božo Đerek, Lucija Lovrić	27.10.2023.
0.3	Obrasci uporabe, uređivanje tablica	Božo Đerek	29.10.2023.
0.4	Arhitektura i dizajn sustava	Lucija Lovrić, Vedran Moškov, Borna Josipović	3.11.2023.
0.5	Baza podataka	Božo Đerek, Vedran Moškov, Lana Bartolović, Lucija Runjić	5.11.2023.
0.6	Sekvencijski dijagrami	Božo Đerek, Lucija Lovrić	6.11.2023.
0.7	Finaliziran dijagram obrazaca uporabe	Božo Đerek	13.11.2023.

Nastavljeno na idućoj stranici

Nastavljeno od prethodne stranice

Rev.	Opis promjene/dodataka	Autori	Datum
0.8	Opis baze podataka, dorade sekvencijskih dijagrama, dnevnik sastajanja	Lana Bartolović, Lucija Lovrić	15.11.2023.
0.9	Dijagram razreda	Božo Đerek, Lana Bartolović	16.11.2023.
<b>1.0</b>	Verzija samo s bitnim dijelovima za 1. ciklus	*	17.11.2023.
1.1	Dijagram aktivnosti	Božo Đerek	30.12.2023.
1.2	Korištene tehnologije i alati	Lucija Lovrić	5.1.2024.
1.3	Zaključak	Lucija Lovrić	12.1.2024.
1.4	Izmjena baze podataka	Božo Đerek, Lana Bartolović	13.1.2024.
1.5	Dijagram razmještaja	Božo Đerek	15.1.2024.
1.6	Upute za puštanje u pogon	Andrija Merlin, Božo Đerek	16.1.2024.
1.7	Dijagram komponenti	Božo Đerek	17.1.2024.

Nastavljeno na idućoj stranici

Nastavljeno od prethodne stranice

Rev.	Opis promjene/dodataka	Autori	Datum
1.8	Ispitivanje programskog rješenja	Lucija Lovrić, Lucija Ruđarić, Vedran Moškov, Borna Josipović	18.1.2024.
1.9	Dijagram stanja	Božo Đerek	19.1.2024.
2.0	Konačna verzija	*	19.1.2024.

## 2. Opis projektnog zadatka

Gubitak kućnog ljubimca može biti jedna od emocionalno najtežih stvari kroz koje vlasnik može proći. Ponekad se dogodi da ljubimac odluta od kuće zbog znatiželje, iznenadnog događaja koji je u njima probudio strah... U tom slučaju, vlasniku je prioritet brzo pronaći mezimca kako bi bio na sigurnom u svom domu.

Naša aplikacija je namijenjena onima koji su izgubili kućnog ljubimca, onima koji žele pomoći drugima u pronalasku svojih krvnenih prijatelja pa i skloništima koji pod svoje primaju odlutale prestrašene životinje. Svi zainteresirani za dobrobit ovih ljubimaca imaju direktni pristup svim informacijama o njima. Značaj aplikacije za zajednicu je da može doprinijeti smanjenju broja napuštenih ljubimaca i time olakšati rad skloništima za životinje te promovirati svijest o izgubljenim ljubimcima.

Ova korisna i jednostavna responzivna aplikacija pomoći će u rješavanju ovog problema mnogim korisnicima aplikacije.

Cilj ovog projekta je razviti programsku potporu za stvaranje web aplikacije "Nestali ljubimci". U opseg projektnog zadatka ulazi izrada web platforme koja podržava registraciju korisnika/skloništa, postavljanje, pretragu i ažuriranje postojećih oglasa te ima pridruženu bazu podataka koja pohranjuje korisne informacije. Sustav treba podržavati rad više korisnika u stvarnom vremenu. Manipuliranje podacima obavlja se kroz sučelje baze podataka tako da nije potreban administrator.

Web aplikacija je namijenjena za 3 vrste korisnika; neregistriranog korisnika, registriranog korisnika te skloništima za životinje (specijalni tip registriranog korisnika).

Neregistrirani korisnik ima mogućnost pregledavanja i pretraživanja nestalih kućnih ljubimaca. Klikom na sliku nestale životinje, znatiželjnog korisniku otvaraju se informacije o njoj: vrsta, ime, datum i sat nestanka, lokacija nestanka, boja, starost te tekstni opis. Uz to dostupne su do 3 slike te životinje kako bi ju tragač

lakše pronašao, a ako bi došlo do novih informacija ili pronađaka ljubimca dostupni su i kontakt podaci vlasnika. Neregistrirani korisnik bi se trebao registrirati ako bi poželio sudjelovati u potrazi i ostvariti komunikaciju s dotičnim vlasnikom. Za registraciju su potrebni sljedeći podaci:

- adresa e-pošte
- broj telefona
- korisničko ime
- lozinka
- ime i prezime/naziv skloništa

Registrirani korisnik ima širi spektar mogućnosti unutar aplikacije. On može, uz pregledavanje i pretraživanje, postaviti oglas o nestalom ljubimcu, izmijeniti i ukloniti ga te sudjelovati u komunikaciji s drugim registriranim korisnicima.

Postoje 4 kategorije oglasa:

- Za ljubimcem se traga (početna postavka oglasa)
- Ljubimac je sretno pronađen
- Ljubimac nije nađen, ali se za njim ne traga aktivno
- Ljubimac je pronađen pod nesretnim okolnostima

Skloništa za životinje su vrsta registriranih korisnika koji oglašavaju životinje koje su pronašli te se nalaze kod njih.

## 2.1 Postojeća slična rješenja

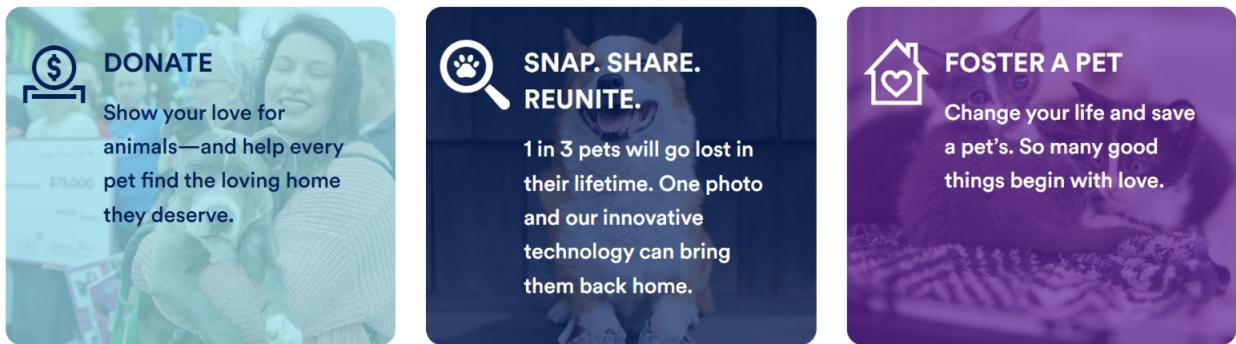
Model našeg projekta može se koristiti na globalnoj razini zbog svoje jednostavnosti i prilagodljivosti lokaciji. Već postoji nekoliko web stranica s kojima dijelimo zajednički cilj poput *PetFinder*, *LostMyDoggie.com*, *PawBoost* i *Petco Love*.

- *PetFinder* je naširoko poznata baza podataka za udomljavanje životinja, a imaju i odjeljak za tražene i pronađene kućne ljubimce.
- *LostMyDoggie.com* je web stranica koja je napravljena specijalno kako bi pomogla vlasnicima pronaći svoje kućne ljubimce.
- *PawBoost* je platforma koja omogućuje korisnicima da prijave nestanak ljubimca nakon čega stranica stvori oglas na Facebooku čime se širi vijest o nestanku ljubimca.
- *Petco Love* je web stranica na kojoj se može prijaviti nestanak, ali i pronačak kućnog ljubimca. Pri tome se šalju i slike ljubimca te se koristi *facial recognition technology* za identificiranje ljubimaca.

U našoj regiji se trenutno ipak nešto više komunicira preko raznih Facebook grupa, lokalnih skloništa za životinje, oglašavanja preko veterinara i lijepljenjem papira s oglasom po ulicama.

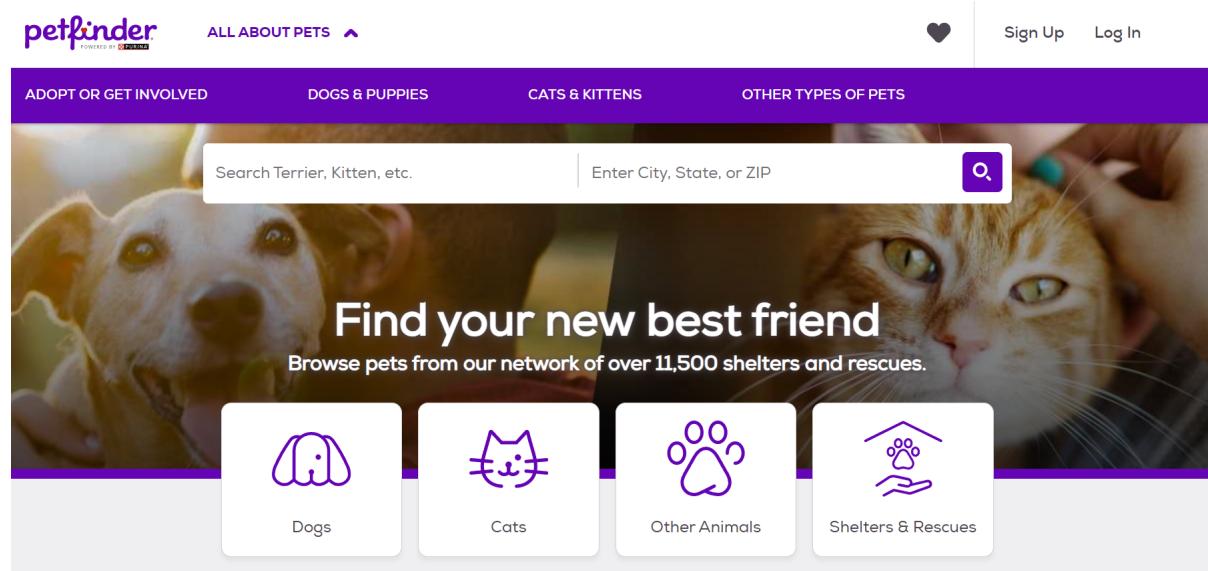


Slika 2.1: Oglas platforme PawBoost



**FREE PET VACCINES: Together, we can stop deadly, preventable diseases for pets in need. Petco Love leads the way with a commitment to providing two million free vaccines for family pets.**

Slika 2.2: Usluge koje nudi platforma PetcoLove



Slika 2.3: Web stranica platforme PetFinder

## 2.2 Moguće prilagodbe i nadogradnje rješenja

- Lokaliziranjem aplikacije ona bi postala dostupna i korisnicima u zemljama s drugim jezicima, zakonima i običajima.
- Osnovni princip naše aplikacije se može primijeniti i na razne izgubljene predmete. Naša aplikacija je svojevrsni *lost and found* primjenjen na ljubimce.
- Nakon početne implementacije, neke od mogućih projektnih nadogradnji uključuju i *real time chat* opciju. Korisnici bi mogli međusobno privatno komunicirati i dijeliti razne informacije o nestalim ljubimcima. Registrirani korisnik s informacijama koje mogu pomoći vlasniku izgubljenog ljubimca mogao bi privatno kontaktirati vlasnika koji je objavio oglas.
- Uvođenje naprednih algoritama za prepoznavanje životinja putem fotografija olakšalo bi i ubrzalo pronađazak.
- Povezivanje podataka aplikacije, skloništa i veterinarskih klinika dodatno bi poboljšalo potragu.
- Na web aplikaciji bi mogla postojati mogućnost donacije sredstava lokalnim skloništima za životinje.

## 3. Specifikacija programske potpore

### 3.1 Funkcionalni zahtjevi

Dionici:

1. Neregistrirani korisnik
2. Registrirani korisnik
3. Sklonište za životinje
4. Razvojni tim
5. Naručitelji

Aktori i njihovi funkcionalni zahtjevi:

1. Neregistrirani korisnik (inicijator) može:
  - (a) pregledavati i pretraživati oglašene nestale kućne ljubimce i skloništa za životinje
  - (b) odabrati nekog od kućnih ljubimaca, čime se otvara mogućnost detaljnijeg pregleda informacija o njemu kao i pregled komunikacije oko potrage za ljubimcem
  - (c) registrirati se, stvoriti novi korisnički račun za koji su mu potrebni e-pošta, broj telefona, korisničko ime, lozinka te optionalno (u slučaju registracije kao sklonište) naziv skloništa
2. Registrirani korisnik (inicijator) može:
  - (a) sve što može neregistrirani korisnik
  - (b) prijaviti se u sustav
  - (c) postaviti, izmijeniti i ukloniti oglas o nestalom kućnom ljubimcu
  - (d) sudjelovati u komunikaciji oko potrage za ljubimcem
  - (e) pretraživati neaktivne oglase
3. Sklonište (inicijator) može:
  - (a) sve što može registrirani korisnik

- (b) oglašavati pronađene životinje koje se nalazi u prostoru skloništa pomoću kategorije oglasa "*u skloništu*"

4. Baza podataka (sudionik):

- (a) pohranjuje sve podatke korisnika  
(b) pohranjuje sve podatke vezane uz oglase

### 3.1.1 Obrasci uporabe

#### Opis obrazaca uporabe

##### UC1 - Registracija

- **Glavni sudionik:** Neregistrirani korisnik
- **Cilj:** Stvoriti korisnički račun za pristup sustavu
- **Sudionici:** Baza podataka
- **Preduvjet:** -
- **Opis osnovnog tijeka:**
  1. Korisnik odabire opciju za registraciju
  2. Korisnik unosi potrebne korisničke podatke
  3. Korisnik prima obavijest o uspješnoj registraciji
- **Opis mogućih odstupanja:**
  - 2.a Odabir već zauzetog korisničkog imena i/ili e-maila, unos korisničkog podatka u nedozvoljenom formatu ili pružanje neispravnog e-maila
    1. Sustav obaveštava korisnika o neuspjelom upisu i vraća ga na stranicu za registraciju
    2. Korisnik mijenja potrebne podatke te završava unos ili odustaje od registracije

##### UC2 - Prijava u sustav

- **Glavni sudionik:** Neprijavljeni registrirani korisnik
- **Cilj:** Dobiti pristup mogućnostima registriranih korisnika
- **Sudionici:** Baza podataka
- **Preduvjet:** Registracija
- **Opis osnovnog tijeka:**
  1. Unos korisničkog imena i lozinke
  2. Provjera ispravnosti unesenih podataka
  3. Pristup korisničkim funkcijama
- **Opis mogućih odstupanja:**
  - 2.a Neispravno korisničko ime/lozinka
    1. Sustav obaveštava korisnika o neuspjeloj prijavi i vraća ga na stranicu za prijavu

##### UC3 - Pregled korisničkih podataka

- **Glavni sudionik:** Registrirani korisnik/sklonište za životinje

- **Cilj:** Pregledati korisničke podatke
- **Sudionici:** Baza podataka
- **Preduvjet:** Prijava u sustav
- **Opis osnovnog tijeka:**
  1. Korisnik odabire opciju za pregled korisničkih podataka
  2. Aplikacija prikazuje podatke korisnika

#### UC4 - Promjena korisničkih podataka

- **Glavni sudionik:** Registrirani korisnik/sklonište za životinje
- **Cilj:** Promijeniti korisničke podatke
- **Sudionici:** Baza podataka
- **Preduvjet:** Prijava u sustav
- **Opis osnovnog tijeka:**
  1. Korisnik pregledava korisničke podatke
  2. Korisnik odabire opciju za promjenu podataka
  3. Korisnik mijenja željene podatke i potvrđuje izmjenu
  4. Baza podataka se ažurira
- **Opis mogućih odstupanja:**
  - 3.a Korisnik je promijenio svoje podatke, ali ih je zaboravio spremiti
    1. Sustav obavještava korisnika o neuspjeloj promjeni podataka
    2. Korisnik sprema izmijenjene podatke

#### UC5 - Brisanje korisničkog računa

- **Glavni sudionik:** Registrirani korisnik/sklonište za životinje
- **Cilj:** Obrisati korisnički račun
- **Sudionici:** Baza podataka
- **Preduvjet:** Prijava u sustav
- **Opis osnovnog tijeka:**
  1. Korisnik pregledava korisničke podatke
  2. Korisnik odabire opciju za brisanje korisničkog računa
  3. Korisnik potvrđuje odabir
  4. Baza podataka se ažurira

#### UC6 - Pretraživanje i pregled oglasa o nestalim ljubimcima

- **Glavni sudionik:** Korisnik
- **Cilj:** Pregledati oglase nestalih ljubimaca

- **Sudionici:** Baza podataka
- **Preduvjet:** -
- **Opis osnovnog tijeka:**
  1. Korisniku se prikazuju oglasi o nestalim ljubimcima
  2. Oglasi se mogu filtrirati po relevantnim podacima, s tim da su neaktivni oglasi vidljivi samo registriranim korisnicima
  3. Prikaz filtriranih oglasa
- **Opis mogućih odstupanja:**
  - 2.a Ne postoji oglas koji odgovara postavljenom filtru
    1. Sustav korisniku prikazuje odgovarajuću poruku

### **UC7 - Postavljanje oglasa o nestalom ljubimcu**

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Postaviti oglas o nestalom ljubimcu
- **Sudionici:** Baza podataka
- **Preduvjet:** Prijava u sustav
- **Opis osnovnog tijeka:**
  1. Korisnik odabire opciju postavljanja oglasa
  2. Korisnik dobiva mogućnost unošenja sljedećih kategorija podataka o ljubimcu:
    - (a) vrsta
    - (b) ime na koje se odaziva
    - (c) datum i sat nestanka
    - (d) lokacija nestanka
    - (e) boja
    - (f) starost
    - (g) tekstni opis
    - (h) do 3 slike
  3. Ako je korisnik sklonište, postavlja kategoriju oglasa "*u skloništu*"
  4. Korisnik odabire opciju za objavljivanje i njegov oglas postaje vidljiv drugima

### **UC8 - Izmjena oglasa o nestalom ljubimcu**

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Izmijeniti oglas o nestalom ljubimcu
- **Sudionici:** Baza podataka

- **Preduvjet:** Prijava u sustav
- **Opis osnovnog tijeka:**
  1. Korisnik odabire opciju izmjene svog oglasa
  2. Korisnik mijenja željene podatke, dostupna mu je i promjena kategorije oglasa u neku od sljedećih:
    - (a) za ljubimcem se traga (*prepostavljen*)
    - (b) ljubimac je sretno pronađen
    - (c) ljubimac nije pronađen, ali se za njim više aktivno ne traga
    - (d) ljubimac je pronađen uz nesretne okolnosti
  3. Korisnik potvrđuje izmjene

#### UC9 - Uklanjanje oglasa o nestalom ljubimcu

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Ukloniti oglas o nestalom ljubimcu
- **Sudionici:** Baza podataka
- **Preduvjet:** Prijava u sustav
- **Opis osnovnog tijeka:**
  1. Korisnik odabire opciju uklanjanja svog oglasa
  2. Uklonjeni oglas i sva pripadna komunikacija nestaje iz popisa vidljivih oglasa, ali se ne briše iz baze podataka

#### UC10 - Ovlašavanje nestalih ljubimaca u skloništu

- **Glavni sudionik:** Sklonište za životinje
- **Cilj:** Postaviti oglas o nestalom ljubimcu u skloništu radi pronalaska vlasnika
- **Sudionici:** Baza podataka
- **Preduvjet:** Prijava u sustav
- **Opis osnovnog tijeka:**
  1. Sklonište za životinje postavlja oglas kategorije "*u skloništu*"
  2. Oglas se pohranjuje u bazu podataka
  3. Oglas se postavlja na web stranicu i vidljiv je drugim korisnicima

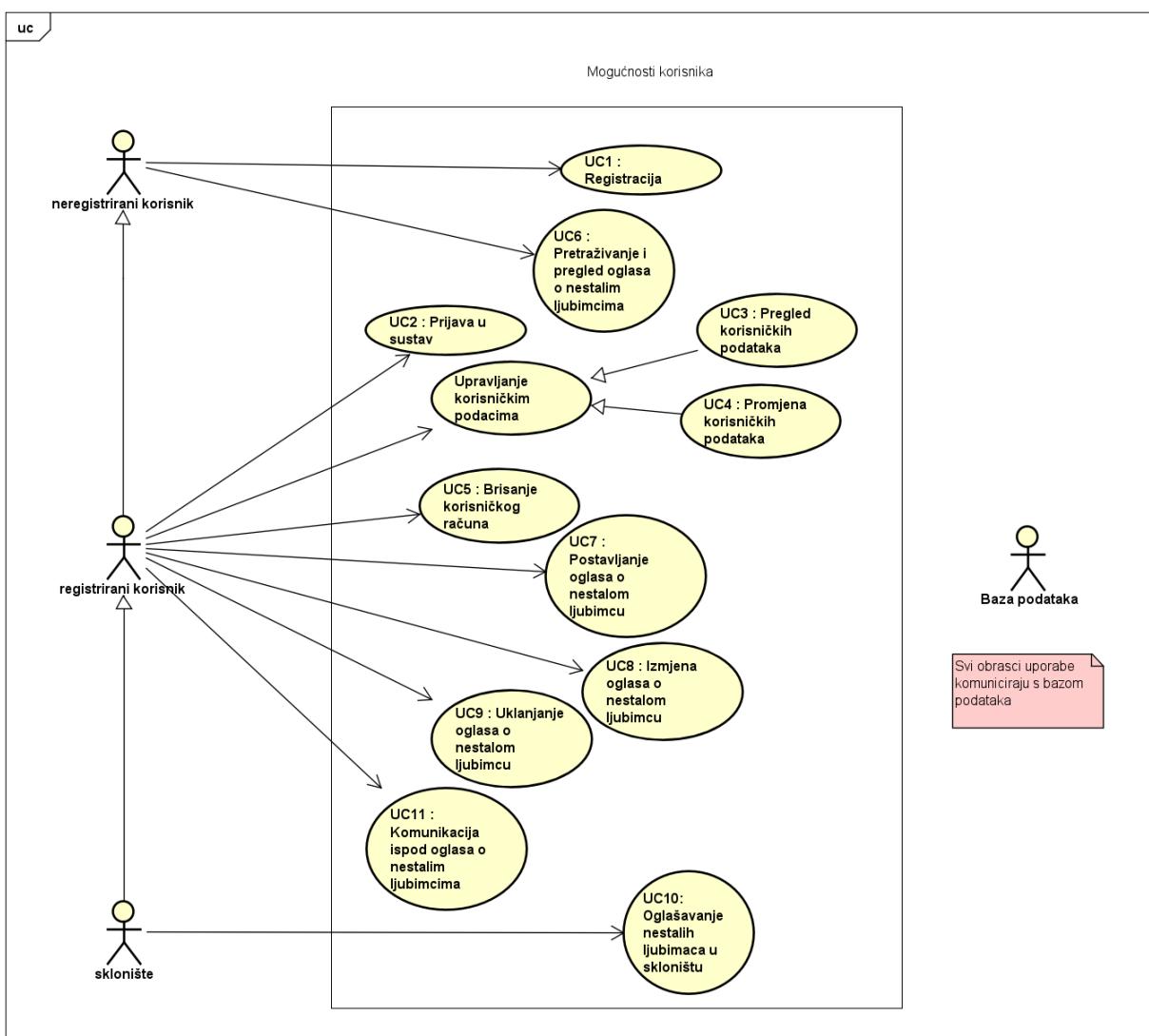
#### UC11 - Komunikacija ispod oglasa o nestalom ljubimcu

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Sudjelovati u komunikaciji oko potrage za ljubimcem
- **Sudionici:** Baza podataka
- **Preduvjet:** Prijava u sustav

- **Opis osnovnog tijeka:**

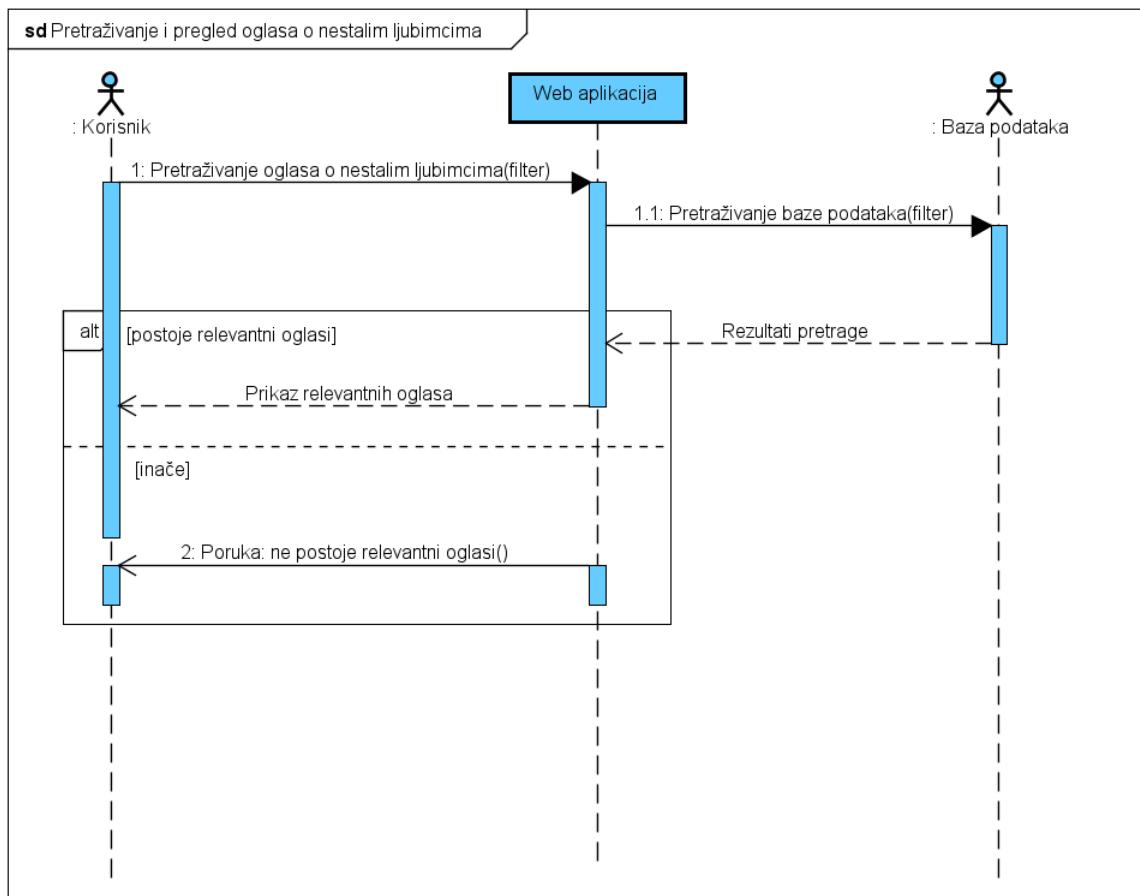
1. Korisnik odabire opciju komunikacije
2. Korisnik unosi poruku koja (uz kontakt podatke korisnika) može sadržavati:
  - (a) tekst
  - (b) sliku
  - (c) geolokaciju
3. Korisnik potvrđuje poruku koju želi ostaviti na oglasu
4. Poruka postaje vidljiva ostalim korisnicima

### Dijagrami obrazaca uporabe



Slika 3.1: Dijagram mogućnosti korisnika

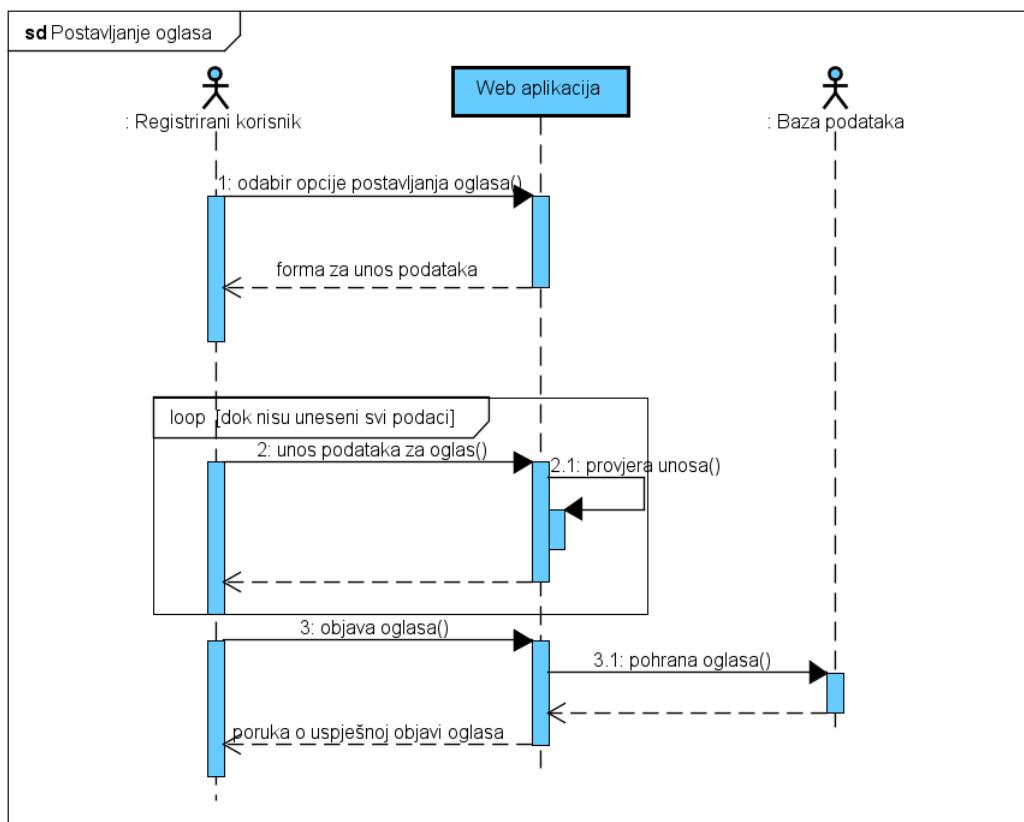
### 3.1.2 Sekvencijski dijagrami



Slika 3.2: Sekvencijski dijagram pretraživanja i pregleda oglasa

#### Obrazac uporabe UC6 - Pretraživanje i pregled oglasa o nestalim ljubimcima

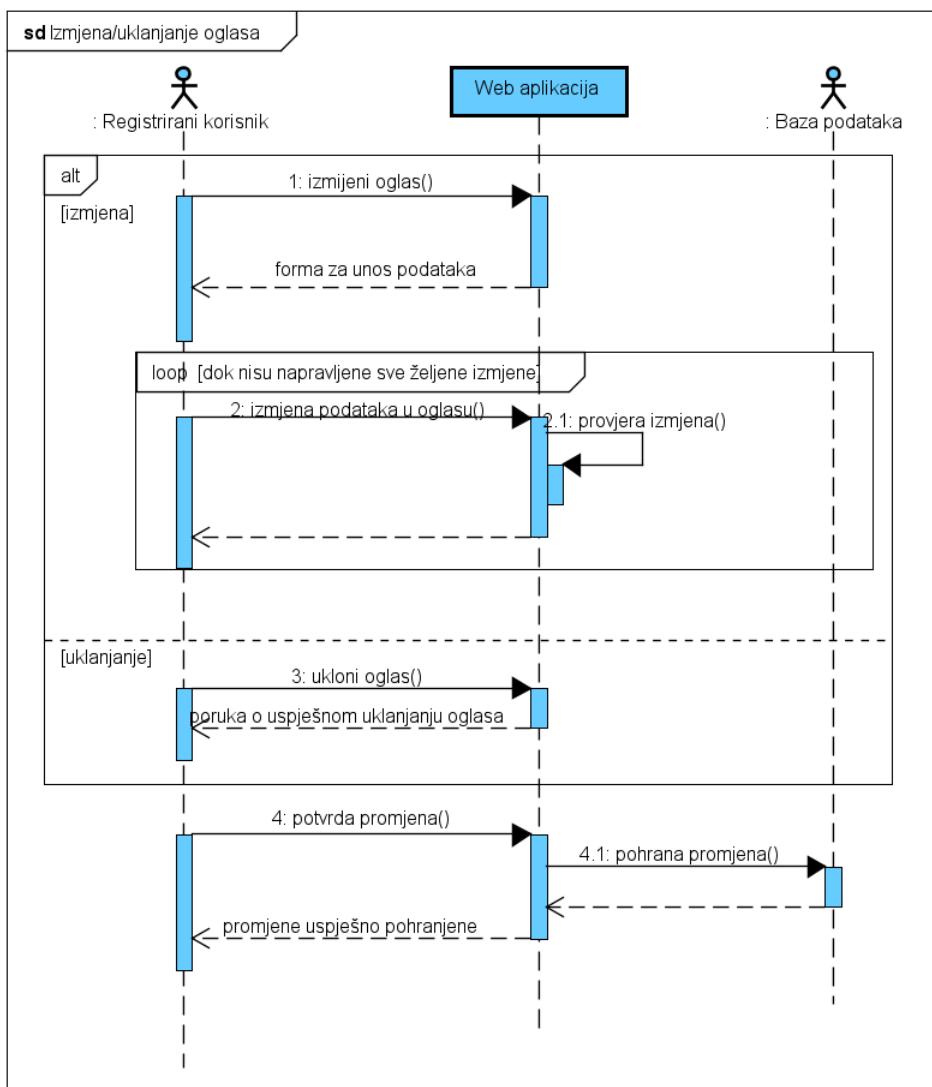
Korisnik (neregistriran ili registriran) u tražilicu unosi podatke o životinji koja ga zanima. Filter tražilice ispunjava raznim vrijednostima po kojima se životinje razlikuju poput imena, vrste, lokacije nestanka, boji i vremenskom rasponu nestanka. Nakon toga se u bazi podataka traži podudarnost s traženim upitom i vraća se rezultat korisniku ovisno o nađenom. Korisniku će se prikazati relevantni oglasi ili (u slučaju da nema nikakvih podudarnosti) poruka da nema rezultata.



Slika 3.3: Sekvencijski dijagram postavljanja oglasa

### Obrazac uporabe UC7 - Postavljanje oglasa o nestalom ljubimcu

Korisnik nakon registracije i/ili prijave u sustav ima, između ostalog, i mogućnost postavljanja oglasa. Korisnik na web stranici odabire opciju postavljanja oglasa čime mu se otvara forma za unos podataka kao što su vrsta, ime na koje se životinja odaziva, datum i sat nestanka, lokacija nestanka, boja, starost, tekstni opis te do 3 fotografije nestalog ljubimca. Ako korisnik nije unio sve potrebne podatke u aplikaciju za prijavu nestale životinje, sustav ga o tome obavještava i navodi na dio forme koji treba biti popunjena. Također, u slučaju da je registrirani korisnik sklonište za životinje, početna postavka kategorije oglasa je "U skloništu", a inače "Za ljubimcem se traga". Kad je korisnik spreman objaviti oglas mora stisnuti gumb "Objavi". Nakon što je stisnuo gumb za objavu, oglas se pohranjuje u bazu podataka, a korisniku pristiže poruka o uspješnom postavljanju oglasa.



Slika 3.4: Sekvencijski dijagram izmjenjivanja/brisanja oglasa

### Obrasci uporabe UC8 i UC9 - izmjena/uklanjanje oglasa o nestalom ljubimcu

Nakon što je objavio oglas, registrirani korisnik može upravljati oglasom – raditi izmjene na njemu ili ga ukloniti.

Ako registrirani korisnik odabere opciju izmjene oglasa otvara mu se forma za izmjenu podataka. Korisnik može mijenjati određene podatke, a može promijeniti i kategoriju oglasa iz "Za ljubimcem se traga" u "Ljubimac je sretno pronađen", "Ljubimac nije pronađen, ali se za njim više aktivno ne traga" ili "Ljubimac je pronađen uz nesretne okolnosti". Sustav nakon toga provjerava jesu li promijenjeni podaci adekvatni te šalje korisniku povratnu informaciju.

Ako korisnik odabere opciju za uklanjanje oglasa, sustav ga obavještava o uspješnom uklanjanju oglasa.

Nakon bilo koje od ovih radnji, korisnik potvrđuje promjenu u sustavu te se promjena pohranjuje u bazu podataka. Korisnik na kraju dobiva poruku kako je uspio izvršiti radnju.

## 3.2 Ostali zahtjevi

- Sustav treba omogućiti rad više korisnika u stvarnom vremenu
- Sustav treba funkcionirati ispravno neovisno o web pregledniku ili uređaju
- Korisničko sučelje i sustav moraju podržavati hrvatsku abecedu (dijakritičke znakove) pri unosu i prikazu tekstualnog sadržaja
- Učitavanje početne stranice ne smije trajati duže od nekoliko sekundi
- Izvršavanje dijela programa u kojem se pristupa bazi podataka ne smije trajati duže od nekoliko sekundi
- Sustav treba biti implementiran kao (responzivna) web aplikacija koristeći objektno-orientirane jezike
- Neispravno korištenje korisničkog sučelja ne smije narušiti funkcionalnost i rad sustava
- Nadogradnja sustava ne smije narušavati postojeće funkcionalnosti sustava
- Sustav treba biti jednostavan za korištenje, korisnici se moraju znati koristiti sučeljem bez opširnih uputa
- Veza s bazom mora biti kvalitetno zaštićena, brza i otporna na vanjske greške
- Pristup sustavu mora biti omogućen iz javne mreže pomoću HTTPS

## 4. Arhitektura i dizajn sustava

Arhitektura se može podijeliti na 4 podsustava:

- Web poslužitelj
- *Front-end*
- *Back-end*
- Baza podataka

*Web poslužitelj* je program koji omogućuje korisnicima pristupanje internet resursima putem zahtjeva poslanih poslužiteljima. Web preglednik je prevoditelj koji web stranicu pisanu u kodu interpretira i prikazuje u klijentu razumljivom obliku. Koristeći web preglednik klijent šalje zahtjeve web poslužitelju.

*Front-end* web aplikacija omogućuje korisniku interakciju s *back-end*-om na poslužitelju, koje se omogućuje kroz korisničko sučelje. *Front-end* aplikacija šalje zahtjeve *back-end* aplikaciji koja ima ulogu web poslužitelja. Njezin je osnovni zadatak pohrana, obrada i dostava web stranica klijentu te nam ona predstavlja centar za razmjenu informacija i pružanje usluga. Sam poslužitelj pokreće web aplikaciju i prosljeđuje joj klijentske zahtjeve.

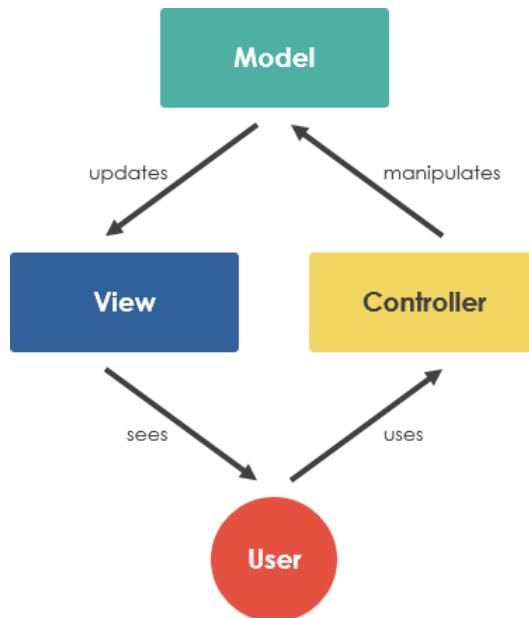
Prednosti odabrane arhitekture su te što se slojevi mogu oblikovati odvojeno te sve komponente mogu biti jednostavnije i razumljivije, ostvaruje se podjela brige (*separation of concerns*) time što svaki sloj brine o svojoj funkcionalnosti i ne mijesha se u brige nekog drugog sloja, a njihova međuvisnost ostvaruje se komunikacijom putem sučelja čija se implementacija može prilagoditi u određenom sloju.

#### 4.0.1 MVC stil arhitekture

Arhitektura razrađena u nastavku prati stil arhitekture MVC (model-view-controller).

Osnovna karakteristika ovog stila je nezavisan razvoj pojedinih dijelova aplikacije što pojednostavljuje testiranje i razvijanje dijelova sustava te njihove dorade. Korisničko sučelje je odvojeno od ostatka sustava, a kohezija elemenata postiže se kroz 3 sloja: 2 na serverskoj strani – model (*Model*) i upravitelj (*Controller*) te 1 na klijentskoj strani - pogled (*View*).

- *Model* - Predstavlja glavnu komponentu sustava koja sadrži dinamičke strukture podataka odnosno razrede koji opisuju domenu primjene te sadrže pravila i aplikacijsku logiku. Usko je vezan uz bazu podataka aplikacije.
- *View* - Sadrži komponente koje služe za prikaz podataka modela i interakciju kroz grafičko korisničko sučelje.
- *Controller* - Povezuje serversku i korisničku stranu: upravlja korisničkim zahjevima prema modelu i odgovorima modela nazad prema pogledima.



Slika 4.1: MVC stil arhitekture

## 4.1 Programski jezici, razvojni okviri, alati i biblioteke koda

### 4.1.1 Back-end i baza podataka

U okviru *back-end* aplikacije koriste se razni alati i tehnologije kako bi se postigla funkcionalnost web aplikacije.

Sama funkcionalnost *back-end-a* ostvarena je koristeći *Kotlin* i *Spring Boot*, popularne *frameworke* za Javu i Kotlin.

*Spring Boot* olakšava izradu web aplikacije pružajući razne automatske konfiguracije. To uvelike omogućava integraciju različitih dijelova aplikacije i pruža mnogo gotovih implementacija koje se koriste putem vrlo intuitivnih sučelja.

Baza podataka ostvarena je u *PostgreSQL*-u, a za njenu jasnu definiciju korišten je alat *Flyway*. *Flyway* omogućava precizno i upravljivo definiranje strukture baze podataka.

Za preslikavanje entiteta iz *PostgreSQL* baze podataka u klase *back-end* aplikacije korišten je *JPA (Jakarta Persistence API)*, koji značajno olakšava generiranje upita ovisno o pozivima metoda nad klasama entiteta, a za komunikaciju baze i same aplikacije koristi se *JDBC (Java Database Connectivity)*.

Za konfiguracijske datoteke aplikacije koristi se *YAML* format. Upravljanje bazom podataka odvija se preko *Datagrip-a*, a razvoj kompletne *back-end* aplikacije obavlja se u *IntelliJ IDEA*, popularnom alatu tvrtke *JetBrains*.

Za izgradnju cijele aplikacije koristi se *Gradle*, alat za automatizaciju izgradnje. *JWT (JSON Web Token)* standard korišten je za sigurnost aplikacije, generirajući tokene, koji istovremeno služe za autorizaciju i autentifikaciju korisnika.

### 4.1.2 Front-end

U izradi *front-end* dijela aplikacije koristimo niz tehnologija kako bismo postigli željene funkcionalnosti i estetski privlačan dizajn. Ključne tehnologije koje se koriste u razvoju uključuju *TypeScript*, *React*, *Bootstrap*, *Vite* i *IntelliJ*.

*React*, kao glavni okvir, omogućava olakšanu izradu web stranica i pruža širok spektar alata za navigaciju, dohvaćanje i prikazivanje podataka. Koristi se označni kod sličan *HTML*-u, obogaćen mogućnostima *TypeScript*-a za definiranje sadržaja stranica, dok se za definiranje stila i izgleda koristi *Bootstrap*.

Za efikasno upravljanje podacima u aplikaciji koristi se *React Query*. *Vite*, alat za brzu izgradnju aplikacija, osigurava optimiziran razvojni proces i ubrzanje vremena učitavanja stranica.

Konačno, za pisanje *TypeScript* koda koristi se *IntelliJ*, moćno razvojno okruženje koje omogućava precizno kodiranje i upravljanje projektom. Ovaj skup tehnologija omogućava nam izradu kvalitetne *front-end* aplikacije s visokom funkcionalnošću i atraktivnim dizajnom.

## 4.2 Baza podataka

Između više tipova baza podataka (relacijska, hijerarhijska, objektno-orientirana), za našu smo web aplikaciju odabrali koristiti relacijsku bazu podataka. Odlučili smo se za tu vrstu baze jer s njom imamo najviše iskustva, a jednostavna je, jasna i praktična za naše potrebe (upravljanje podacima o korisnicima koji postavljaju, izmjenjuju, brišu ili komentiraju oglase za izgubljene odnosno pronađene ljubimce) i implementaciju (klase u Springu su preslikane relacije iz baze) i omogućavaju jednostavan vizualni prikaz točnih međuovisnosti relacija, a time i veću razumljivost. Bazu podataka koristimo za efikasnu pohranu i dohvata podataka za obradu. Elementi baze podataka su relacije, odnosno tablice i njihovi vlastiti atributi. Baza podataka ove aplikacije sadrži 13 tablica:

- User
- UserType
- Ad
- Activity
- Pet
- Color
- Of\_Color
- Species
- Message
- Image
- City
- County
- Location

#### 4.2.1 Opis tablica

**APP\_USER** Predstavlja registriranog korisnika aplikacije, u `@ManyToOne` vezi s **USER\_TYPE** preko `userTypeId`.

APP_USER		
userId	BIGINT	jedinstveni ID korisnika
username	VARCHAR	korisničko ime (jedinstveno)
email	VARCHAR	korisnikova e-mail adresa (jedinstvena)
password	VARCHAR	korisnikova lozinka
name	VARCHAR	ime korisnika
telephone_number	VARCHAR	korisnikov broj telefona (jedinstven)
userTypeId	BIGINT	ID koji označava tip korisnika

**USER\_TYPE** Predstavlja popis tipova korisnika aplikacije (`userTypeId=1` odgovara osobi, `userTypeId=2` skloništu).

USER_TYPE		
userTypeId	BIGINT	jedinstveni ID tipa korisnika
name	VARCHAR	ime tipa korisnika

**MESSAGE** Poruka koju korisnici mogu ostavljati u komunikaciji ispod oglasa, u `@ManyToOne` vezi s **AD** preko `adId`, `@ManyToOne` vezi s **USER** preko `userId`, `@OneToOne` vezi s **LOCATION** preko `locationId`.

MESSAGE		
messageId	BIGINT	jedinstveni ID poruke
text	VARCHAR	sadržaj poruke
date	DATE	datum kada je poruka ostavljena ispod oglasa

Nastavljeno na idućoj stranici

Nastavljeno od prethodne stranice

MESSAGE		
adId	BIGINT	jedinstveni ID oglasa ispod kojeg je poruka ostavljena
locationId	BIGINT	jedinstveni ID lokacije
userId	BIGINT	jedinstveni ID korisnika koji je ostavio poruku

**IMAGE** Tablica u koju se spremaju slike koje se dohvaćaju preko URL.

IMAGE		
imageId	BIGINT	jedinstveni ID slike
imageUrl	VARCHAR	URL slike
adId	BIGINT	jedinstveni ID pripadajućeg oglasa
messageId	BIGINT	jedinstveni ID pripadajuće poruke

**ACTIVITY** Predstavlja kategoriju oglasa.

ACTIVITY		
activityId	BIGINT	jedinstveni ID kategorije
activityCategory	VARCHAR	naziv kategorije

**AD** Predstavlja oglas, u *@ManyToOne* vezi s **ACTIVITY**, *@ManyToOne* vezi s **USER** preko *userId*, *@OneToOne* vezi s **PET**.

AD		
adId	BIGINT	jedinstveni ID oglasa
inShelter	INT	1 ako je oglas od skloništa, inače 0
deleted	INT	1 ako je oglas obrisan, inače 0
activityId	BIGINT	jedinstveni ID kategorije oglasa
petId	BIGINT	jedinstveni ID ljubimca u oglasu

Nastavljeno na idućoj stranici

Nastavljeno od prethodne stranice

<b>AD</b>		
userId	BIGINT	jedinstveni ID korisnika koji je postavio oglas

**PET** Predstavlja ljubimca, u *@ManyToMany* vezi s **COLOR**, *@ManyToOne* vezi sa **SPECIES**, *@OneToOne* vezi s **LOCATION**.

<b>PET</b>		
petId	BIGINT	jedinstveni ID ljubimca
petName	VARCHAR	ime na koje se ljubimac odaziva
petAge	INT	starost ljubimca
dateTimeMissing	DATETIME	datum i vrijeme nestanka ljubimca
description	VARCHAR	opis ljubimca
speciesId	BIGINT	jedinstveni ID vrste ljubimca
locationId	BIGINT	ID lokacije nestanka ljubimca

**SPECIES** Tablica vrsta ljubimaca.

<b>SPECIES</b>		
speciesId	BIGINT	jedinstveni ID vrste ljubimca
speciesName	VARCHAR	naziv vrste ljubimca

**COLOR** Tablica s bojama ljubimaca, u *@ManyToMany* vezi s **PET**.

<b>COLOR</b>		
colorId	BIGINT	jedinstveni ID boje
colorName	VARCHAR	naziv boje

**OF\_COLOR** Tablica veze između **PET** i **COLOR**.

OF_COLOR		
colorId	BIGINT	jedinstveni ID boje
petId	BIGINT	jedinstveni ID ljubimca

**COUNTY** Predstavlja županije nestanka/pronalaska ljubimaca.

COUNTY		
countyId	BIGINT	jedinstveni ID županije
countyName	VARCHAR	naziv županije

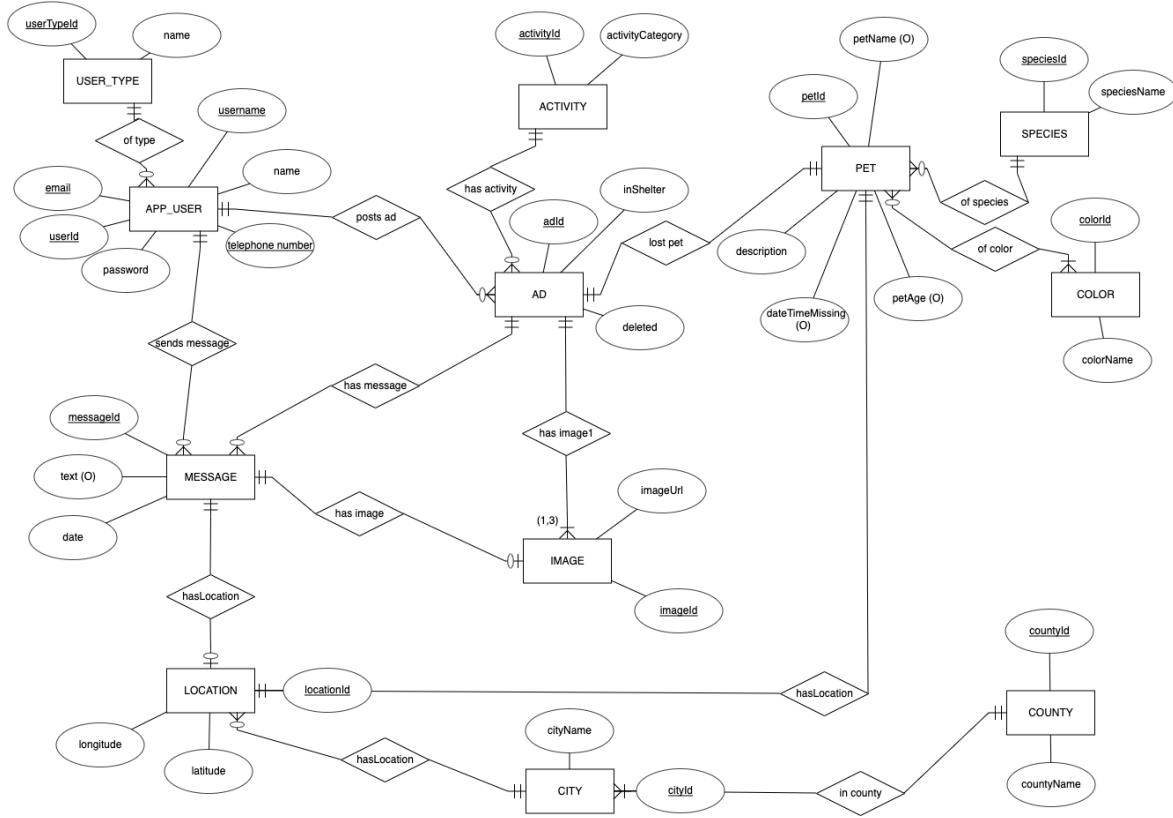
**CITY** Predstavlja gradove nestanka/pronalaska ljubimaca, u *@ManyToOne* vezi s **COUNTY**.

CITY		
cityId	BIGINT	jedinstveni ID grada
cityName	VARCHAR	naziv grada
countyId	BIGINT	jedinstveni ID županije

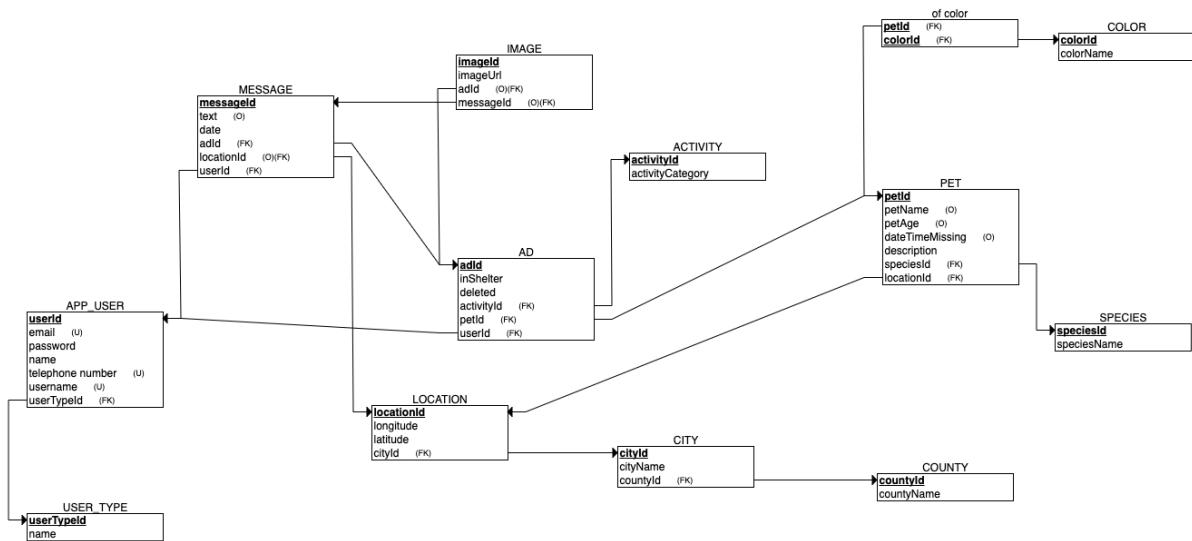
**LOCATION** Predstavlja točne lokacije, u *@ManyToOne* vezi sa **CITY**.

LOCATION		
locationId	BIGINT	jedinstveni ID lokacije
longitude	DOUBLE PRECISION	geografska dužina
latitude	DOUBLE PRECISION	geografska širina
cityId	BIGINT	jedinstveni ID grada

#### 4.2.2 Dijagram baze podataka



Slika 4.2: ER dijagram baze podataka

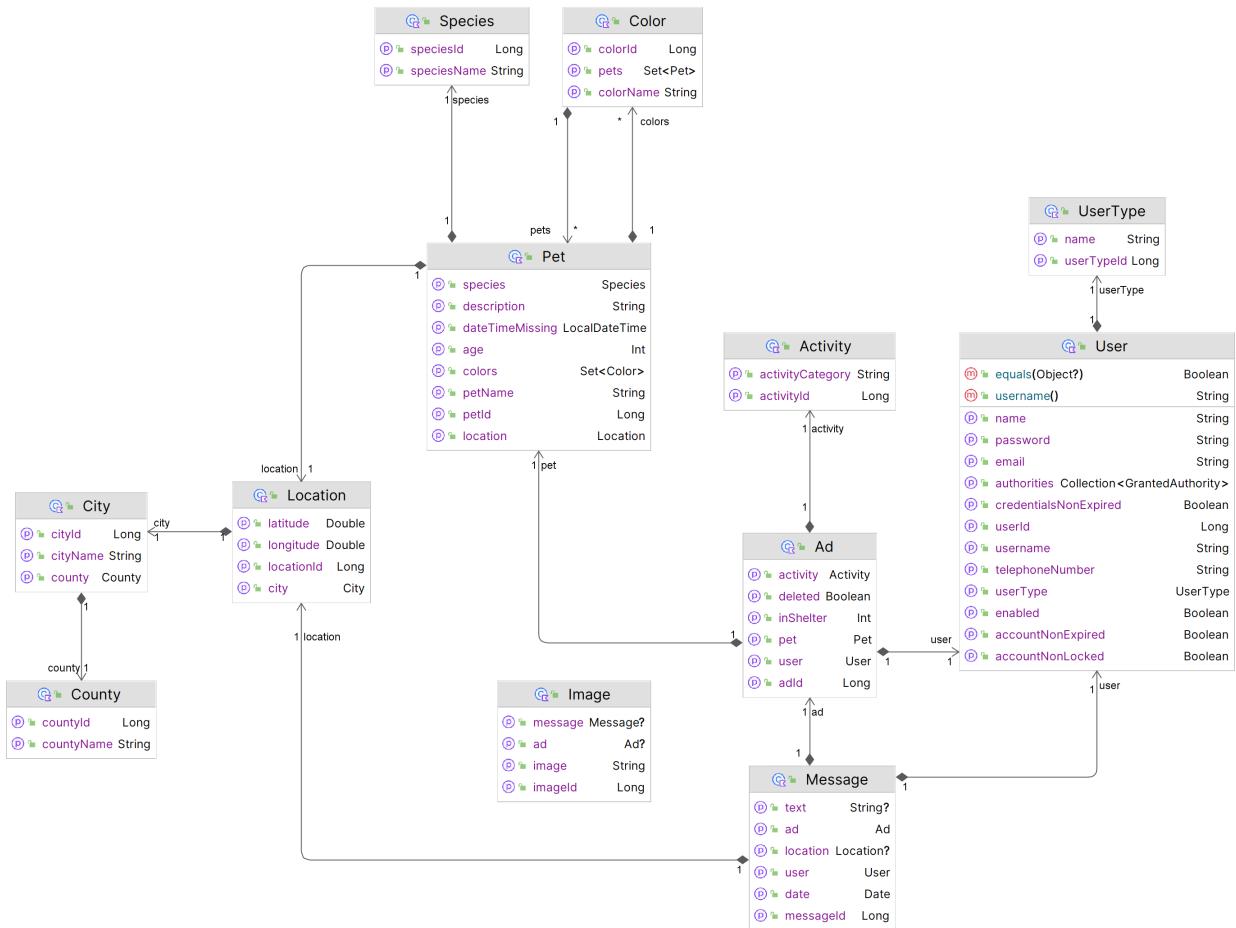


Slika 4.3: Relacijski dijagram baze podataka

## 4.3 Dijagram razreda

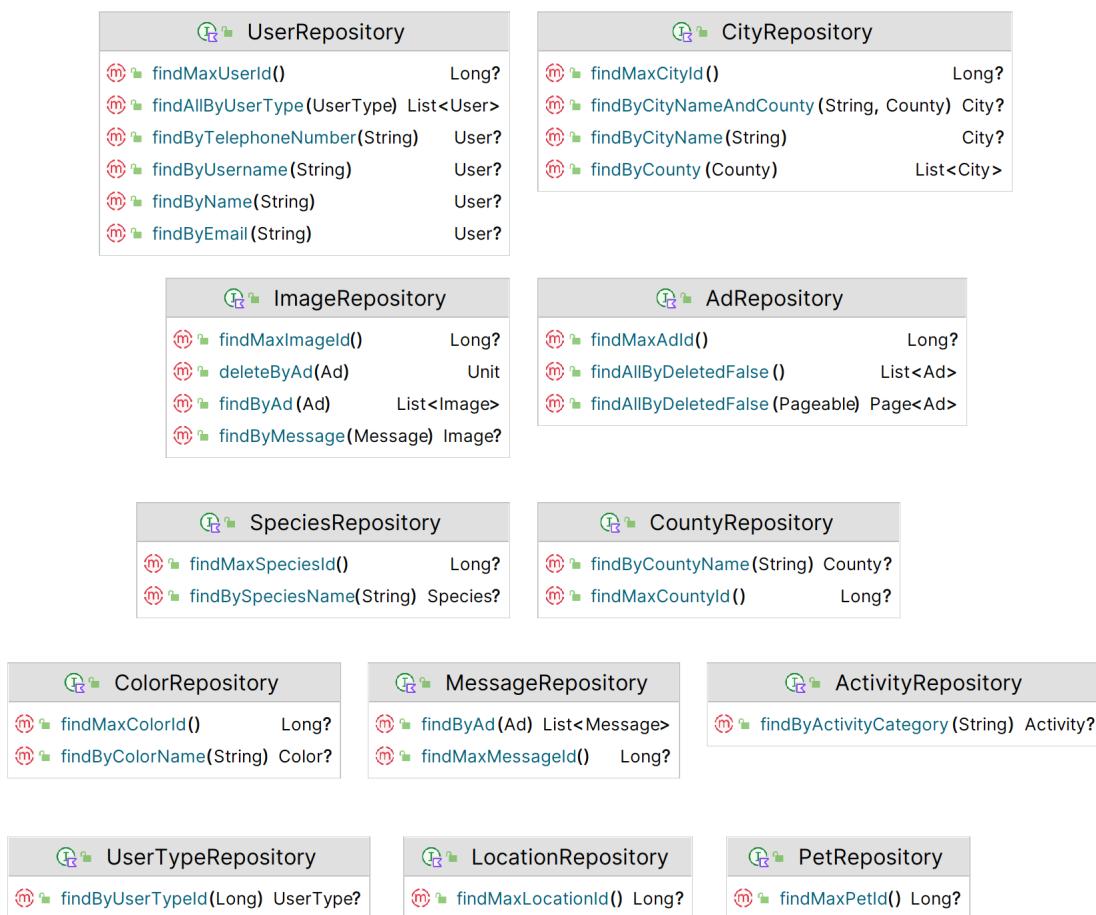
Na dijagramima su prikazani razredi vezani uz serversku stranu aplikacije.

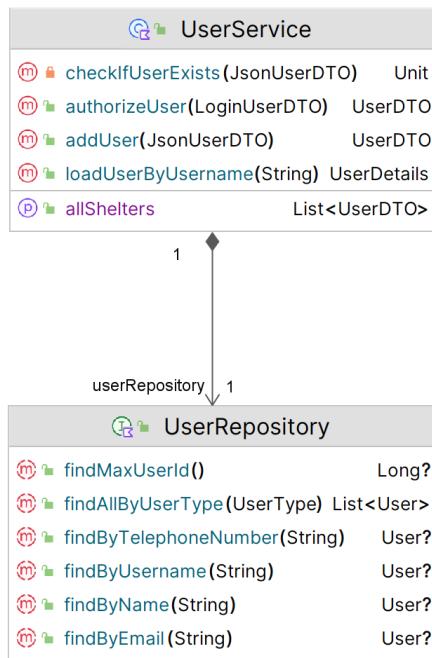
Dijagram na slici 4.4 prikazuje razrede entitete (modele) koji odgovaraju tablicama u bazi podataka.



Slika 4.4: Dijagram razreda entiteta

Na slici 4.5 prikazana su sučelja *Repository* (npr. `UserRepository`, `MessageRepository`), odgovorna za komuniciranje s bazom podataka. Radi preglednosti je na slici 4.6 prikazan samo odnos `UserService` i `UserRepository`, ostali se ponašaju analogno.

Slika 4.5: Dijagram razreda *Repository*

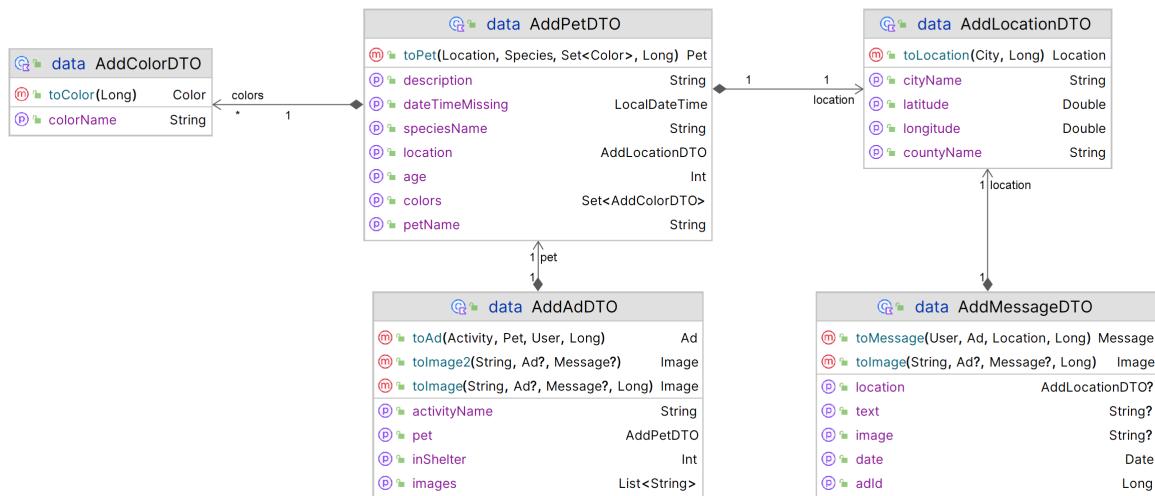


Slika 4.6: UserService i UserRepository

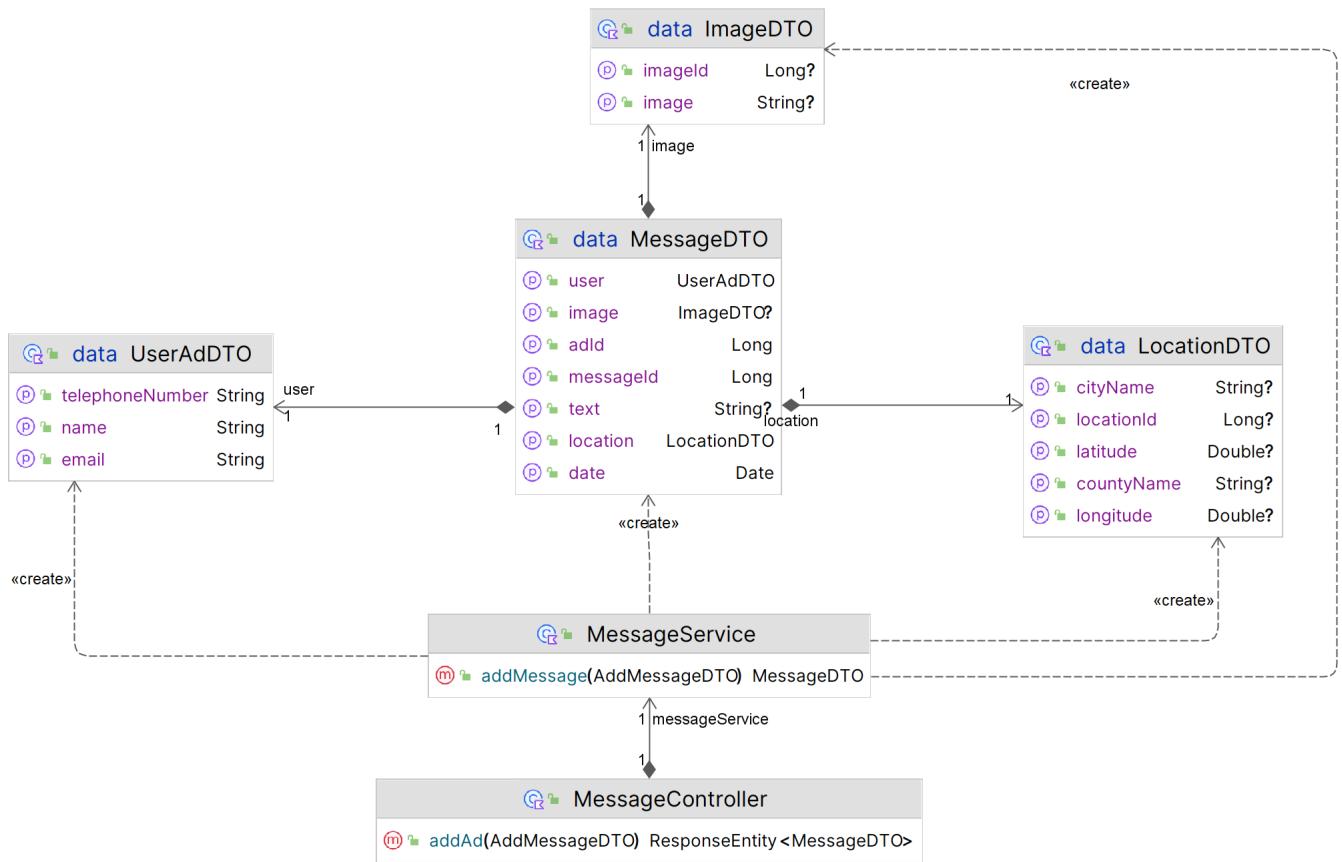
Dijagrami 4.7, 4.8, 4.9, 4.10, 4.11, 4.12 prikazuju odnose između *Controller*, *Service* i *Data Transfer Object* (DTO) razreda vezanih uz oglase, poruke, korisnike, gradove i županije te boje i vrste ljubimaca.



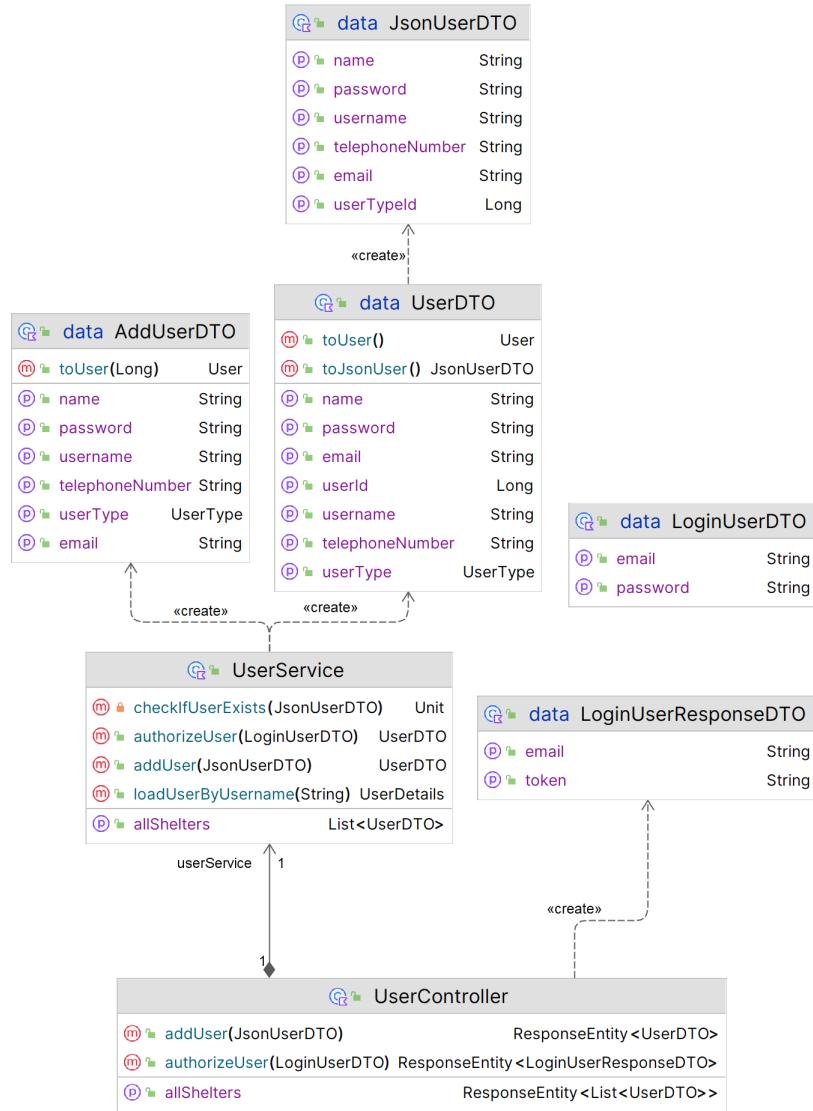
Slika 4.7: Razredi - oglasi



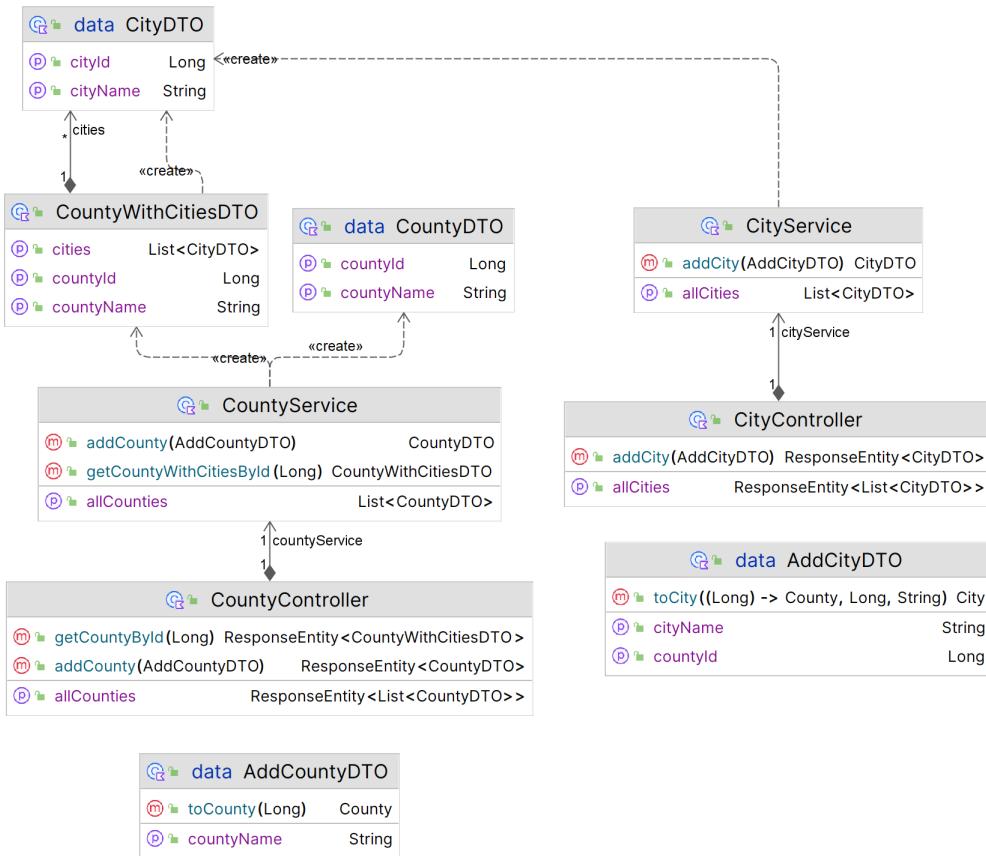
Slika 4.8: Razredi - dodavanje oglasa i poruka



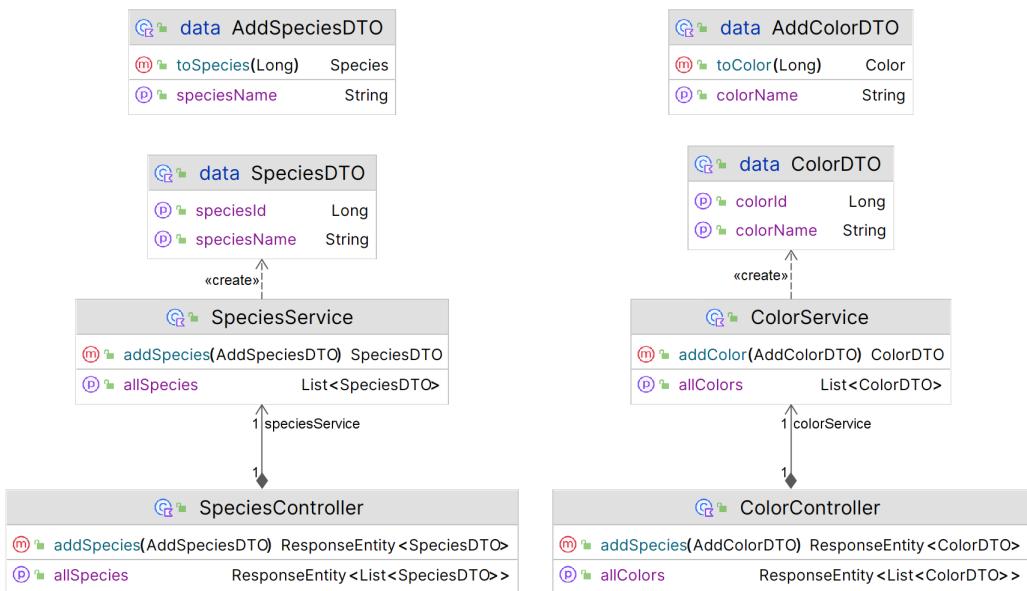
Slika 4.9: Razredi - poruke



Slika 4.10: Razredi - korisnici

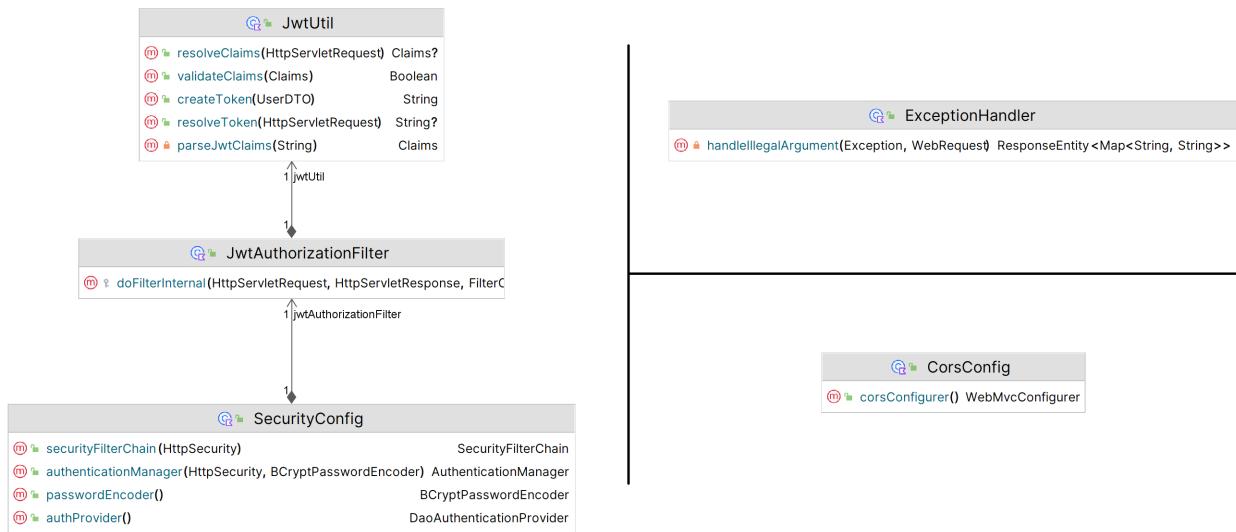


Slika 4.11: Razredi - gradovi i županije



Slika 4.12: Razredi - vrste i boje ljubimaca

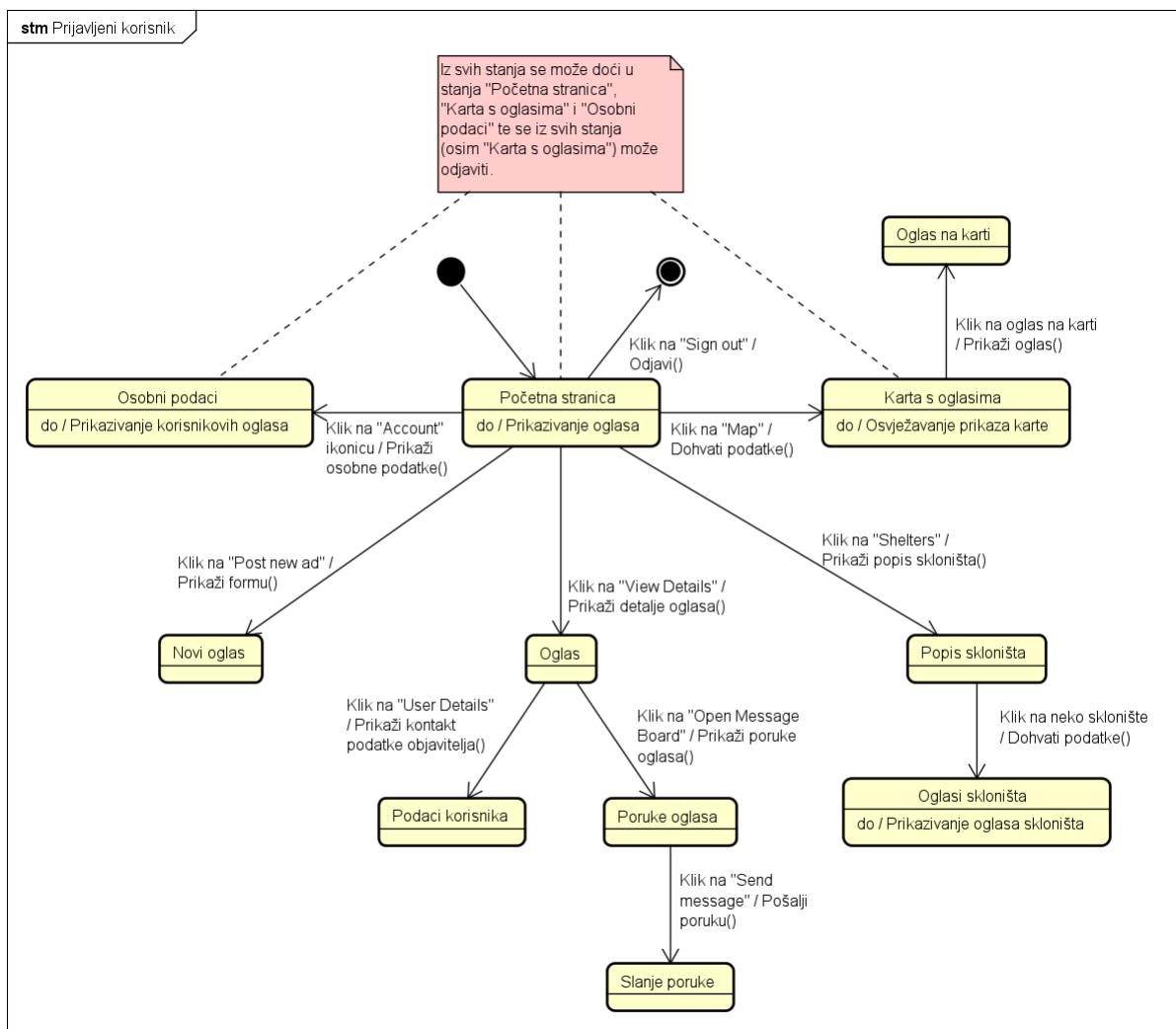
Na slici 4.13 prikazani su razredi odgovorni za autorizaciju, upravljanje iznimkama te *cross-origin resource sharing (CORS)*.



Slika 4.13: Autorizacija, upravljanje iznimkama i CORS

## 4.4 Dijagram stanja

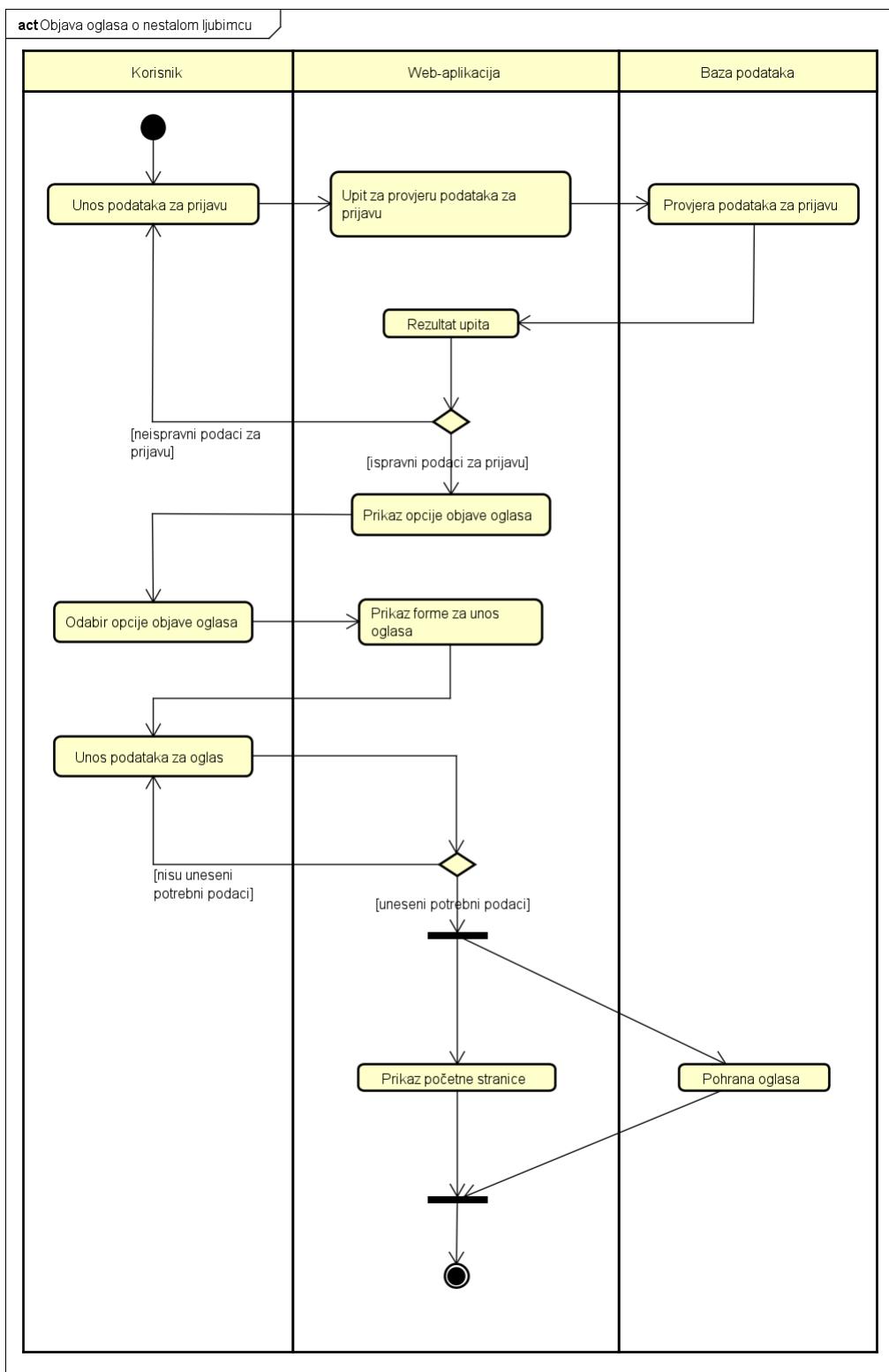
Slika 4.14 prikazuje dijagram stanja za prijavljenog korisnika. Nakon prijave, korisniku se prikazuje početna stranica na kojoj može vidjeti oglase o nestalim ljubimcima. Korisnik ima mogućnost pregledati detalje nekog oglasa, kao i kontakt podatke korisnika koji je objavio oglas te pregledati poruke i sudjelovati u komunikaciji slanjem vlastite poruke. Klik na "Shelters" vodi korisnika na prikaz popisa skloništa čije oglase zatim može pregledati (nazad se vraća klikom na "Posts"). Gumb s ikonicom "Account" prikazuje korisniku osobne podatke i postavljene oglase, koje korisnik može (klikom na "...") obrisati ili urediti (izmijeniti podatke, promijeniti kategoriju). Gumb "Post new ad" otvara formu za stvaranje novog oglasa. Klikom na gumb "Map" korisniku se prikazuje karta na kojoj su označene lokacije nestalih ljubimaca. Na njih se može kliknuti i vidjeti pojednostavljeni prikaz oglasa.



Slika 4.14: Dijagram stanja

## 4.5 Dijagram aktivnosti

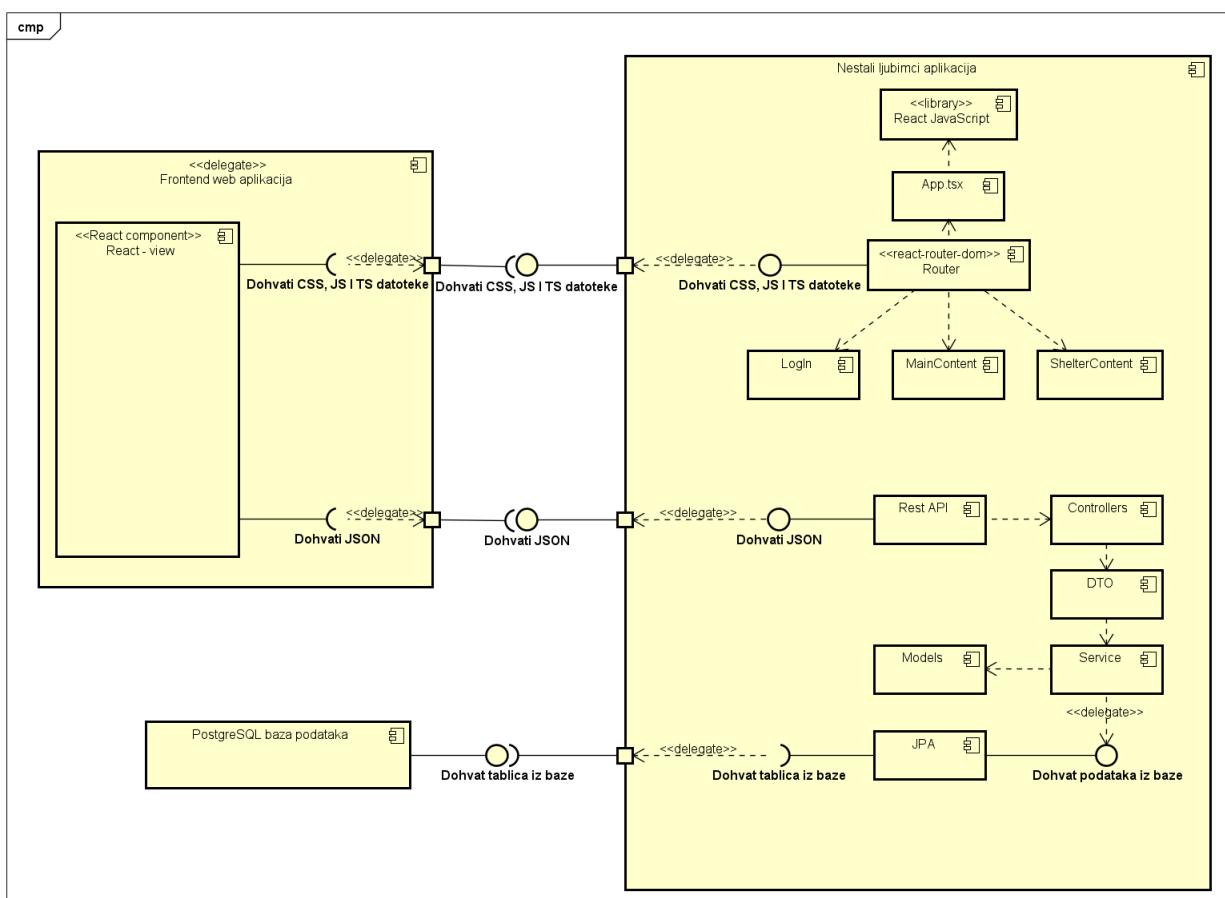
Dijagram aktivnosti na slici 4.15 prikazuje proces objave oglasa o nestalom ljubimcu. Korisnik se prijavi u sustav, odabere opciju stvaranja novog oglasa te ispunjava formu za oglas. Nakon što korisnik unese sve potrebne podatke i predaje formu, oglas se pohranjuje u bazu podataka te se korisnika vraća na početnu stranicu.



Slika 4.15: Dijagram aktivnosti

## 4.6 Dijagram komponenti

Dijagram komponenti na slici 4.16 prikazuje ustroj i međuvisnost komponenti, interne strukture i odnose prema okolini. Dva su sučelja putem kojih se pristupa sustavu. Sučelje za dohvat CSS, JS i TS datoteka odgovorno je za posluživanje datoteka vezanih uz *frontend*. Sve TS odnosno JS datoteke ovise o React biblioteci. Sučeljem za dohvat JSON podataka se pristupa REST API komponenti, odgovornoj za posluživanje podataka vezanih uz *backend*. Tablice se iz baze podataka dohvaćaju pomoću JPA (*Jakarta Persistence*). Podaci iz baze se prosljeđuju kao DTO (*data transfer object*).



Slika 4.16: Dijagram komponenti

# 5. Implementacija i korisničko sučelje

## 5.1 Korištene tehnologije i alati

Komunikacija u timu ostvarena je pomoću platforme Discord<sup>1</sup> koja nudi opcije tekstualnog dopisivanja, glasovnog poziva i videopoziva. U serveru našeg projektnog tima postojalo je nekoliko kanala koji su se bavili različitim temama (back-end, front-end, dokumentacija...) što nam je pomoglo u podjeli posla i filtriranju zadataka. Za izradu UML dijagrama koristili smo se aplikacijom Astah<sup>2</sup> pomoću koje smo vrlo jednostavno napravili željene dijagrame i time pokazali ideju projektnog zadatka, kao i način rada naše aplikacije. Za rad s LaTeX-om poslužio nam je Texmaker<sup>3</sup>. Kao sustav za upravljanje izvornim kodom koristili smo Git<sup>4</sup>. Udaljeni repozitorij projekta dostupan je na web platformi GitHub<sup>5</sup>.

Za razvojno okruženje koristili smo Visual Studio Code<sup>6</sup> i IntelliJ IDEA<sup>7</sup>. VSC nudi široku podršku za jezike i ekstenzije, integraciju s Git-om i različitim alatima i servisima (Docker<sup>8</sup>) te snimanje koraka za debuggiranje. Također, dolazi s ugrađenom podrškom za JavaScript i TypeScript. IntelliJ IDEA nudi naprednu podršku za pišanje koda (Code Assistance), odličan debugger, refaktoriranje koda te podršku za različite tehnologije (Spring Framework, JavaScript, HTML, CSS, TypeScript, React, SQL) i jezike (Java, Kotlin, Groovy).

Klijentska strana aplikacije (*front-end*) napisana je React<sup>9</sup>-om preko TypeScript<sup>10</sup>-a i Vite.js<sup>11</sup>-om. Material UI<sup>12</sup> je biblioteka React komponenata koja pruža implementaciju Google-ovog Material Design koncepta u React aplikacijama, a fokusiran je na stvaranje dosljednih i intuitivnih korisničkih sučelja.

---

<sup>1</sup><https://discord.com>

<sup>2</sup><https://astah.net>

<sup>3</sup><https://www.xmlmath.net/texmaker>

<sup>4</sup><https://git-scm.com>

<sup>5</sup><https://github.com>

<sup>6</sup><https://code.visualstudio.com>

<sup>7</sup><https://jetbrains.com/idea>

<sup>8</sup><https://docker.com>

<sup>9</sup><https://react.dev>

<sup>10</sup><https://typescriptlang.org>

<sup>11</sup><https://vitejs.dev>

<sup>12</sup><https://mui.com/material-ui>

Poslužiteljska strana aplikacije (*back-end*) napisana je u Kotlinu<sup>13</sup> koristeći Spring Framework<sup>14</sup>. Kotlin omogućuje interoperabilnost s jezikom Java<sup>15</sup> (može se koristiti zajedno s njim u sklopu istog projekta te koristiti njegove biblioteke). Spring Framework je open-source radni okvir za razvoj aplikacija baziranih na Javi. On implementira inverziju kontrole (IoC), uvođenje ovisnosti (DI), Model-View-Controller (MVC), pristup podacima, sigurnost i Spring Boot.

Pokretanje, izvršavanje i puštanje u pogon izvršavaju se preko 3 kontejnera platforme Docker koji pokrivaju najvažnije dijelove aplikacije: back-end, front-end i PostgreSQL bazu podataka. Pri deployment-u projekta koristimo radni okvir Express.js<sup>16</sup> za izgradnju API-ja s Node.js<sup>17</sup> radnim okruženjem.

---

<sup>13</sup><https://kotlinlang.org>

<sup>14</sup><https://spring.io/projects/spring-framework>

<sup>15</sup><https://java.com/en>

<sup>16</sup><https://expressjs.com>

<sup>17</sup><https://nodejs.org/en>

## 5.2 Ispitivanje programskog rješenja

### 5.2.1 Ispitivanje komponenti

Komponente programskog sustava ostvaruju određenu funkcionalnost te međusobno imaju usku komunikaciju preko raznih ponuđenih sučelja. Pomoću unit testova provjeravamo ispravnost i pravilno ponašanje pojedinačnih "jedinica" komponenti, najčešće funkcija ili metoda. To su najmanji dijelovi softverskog sustava koji se mogu testirati odvojeno, a provođenjem unit testiranja poboljšavamo kvalitetu koda tijekom vremena te identificiramo eventualne probleme.

Ispitivanje komponenti ostvareno je preko JUnit radnog okvira, a proveli smo 2 vrste testova: testiranje ponašanja sustava prilikom unosa ispravnih podataka te testiranje rubnih slučajeva odnosno ponašanja sustava kod unosa neispravnih podataka. Prilikom testiranja UserService metoda, korišteno je sučelje UserRepository uz klase User i DTO objekata koji su korišteni za entitet User. Prilikom testiranja AdService metode za dodavanje oglasa korištena su sučelja: ActivityRepository, CityRepository, AdRepository, ColorRepository, CountyRepository, ImageRepository, LocationRepository, PetRepository, SpeciesRepository, UserRepository te klase DTO objekata potrebne za kreiranje novog oglasa.

Proveli smo sveukupno 6 unit testova te u nastavku slijede njihovi opisi i slike kodova.

Ispitni slučaj 1: Najprije smo testirali funkcionalnost logiranja. Prvi test provjerava uspješnu prijavu s unaprijed zadanim emailom i lozinkom.

```
@Test
fun loginValidTest(){

    val newUser = User(
        userId = 1L,
        username = "username",
        email = "us.us@us.us",
        password = "ususus123",
        name = "user",
        telephoneNumber = "1234567899",
        userType = UserType( userTypeId: 1L, name: "Osoba")
    )

    val login = LoginUserDTO(
        newUser.email,
        newUser.password
    )

    `when`(userRepository.findByEmail(newUser.email)).thenReturn(newUser)

    val savedUser: User = userService.authorizeUser(login).toUser()

    assertNotNull(savedUser)
    assertEquals(newUser, savedUser)
}
```

Slika 5.1: Unit test 1 - *log-in*

**Ispitni slučaj 2: Drugi test provjerava prijavu s neispravnom lozinkom.**

```
@Test
fun loginInvalidTest(){

    val newUser = User(
        userId = 1L,
        username = "username",
        email = "us.us@us.us",
        password = "ususus123",
        name = "user",
        telephoneNumber = "1234567899",
        userType = UserType(userTypeId: 1L, name: "Osoba")
    )

    val login = LoginUserDTO(
        newUser.email,
        password: "krivi password"
    )

    `when` (userRepository.findByEmail(newUser.email)).thenReturn( value: null)

    assertThrows<IllegalArgumentException> {
        userService.authorizeUser(login)
    }
}
```

Slika 5.2: Unit test 2 - *log-in* s neispravnom lozinkom.

**Ispitni slučaj 3: Treći test provjerava uspješnu registraciju.**

```
@Test
fun registerValidTest(){
    val jsonUserDTO = JsonUserDTO(
        username = "username",
        email = "us.us@us.us",
        password = "ususus123",
        name = "user",
        telephoneNumber = "1234567899",
        userTypeId = 1L
    )

    val userType = UserType(userTypeId = 1L, name = "Osoba")

    `when` (userTypeRepository.findByUserTypeId(jsonUserDTO.userTypeId)).thenReturn(userType)
    `when` (userRepository.findMaxUserId()).thenReturn(value = 0L)

    val savedUser = User(
        userId = 1L,
        username = jsonUserDTO.username,
        email = jsonUserDTO.email,
        password = jsonUserDTO.password,
        name = jsonUserDTO.name,
        telephoneNumber = jsonUserDTO.telephoneNumber,
        userType = userType
    )

    `when` (userRepository.save(any(User::class.java))).thenReturn(savedUser)

    val result = userService.addUser(jsonUserDTO).toJsonUser()

    assertEquals(jsonUserDTO, result)

    // Verify that userRepository.save was called with a non-null argument
    verify(userRepository, times(wantedNumberOfInvocations = 1)).save(any(User::class.java))
}
```

Slika 5.3: Unit test 3 - registracija

Ispitni slučaj 4: U četvrtom testu testiramo pokušaj registracije s e-mail-om koji je već u uporabi.

```
@Test
fun registerInvalidTest(){

    val jsonUserDTO = JsonUserDTO(
        username = "username",
        email = "us.us@us.us",
        password = "ususus123",
        name = "user",
        telephoneNumber = "1234567899",
        userTypeId = 1L
    )

    //user that is already saved in database
    val savedUser = User(
        userId = 1L,
        username = jsonUserDTO.username,
        email = jsonUserDTO.email,
        password = jsonUserDTO.password,
        name = jsonUserDTO.name,
        telephoneNumber = jsonUserDTO.telephoneNumber,
        userType = UserType( userTypeId: 1L, name: "Osoba")
    )

    `when`(`userRepository.findByEmail(jsonUserDTO.email)`).thenReturn(savedUser)

    assertThrows<IllegalArgumentException> {
        userService.addUser(jsonUserDTO)
    }
}
```

Slika 5.4: Unit test 4 - registracija s već postojećim e-mail-om

**Ispitni slučaj 5: Peti test testira rutu dohvaćanja svih korisnika koji su prijavljeni kao sklonište odnosno "shelter".**

```
@Test
fun getAllSheltersTest(){
    val shelterUserType = UserType( typeId: 2L, name: "Sklonište")

    val shelter1 = User(
        userId = 1L,
        username = "username1",
        email = "email1",
        password = "password1",
        name = "name1",
        telephoneNumber = "telephoneNumber1",
        userType = shelterUserType
    )

    val shelter2 = User(
        userId = 2L,
        username = "username2",
        email = "email2",
        password = "password2",
        name = "name2",
        telephoneNumber = "telephoneNumber2",
        userType = shelterUserType
    )

    val shelter3 = User(
        userId = 3L,
        username = "username3",
        email = "email3",
        password = "password3",
        name = "name3",
        telephoneNumber = "telephoneNumber3",
        userType = shelterUserType
    )

    val shelters = listOf(shelter1, shelter2, shelter3)

    `when`(`userTypeRepository.findById(id: 2L)).thenReturn(shelterUserType)
    `when`(`userRepository.findAllByUserType(shelterUserType)).thenReturn(shelters)

    val result = userService.getAllShelters().map { it.toUser() }

    assertEquals(shelters, result)
    assertEquals(shelters, result)
}
```

Slika 5.5: Unit test 5 - ruta dohvaćanja svih korisnika

**Ispitni slučaj 6: U šestom testu testiramo dodavanje novog oglasa.**

```
@Test
fun addNewAd() {
    // mocking SecurityContextHolder and SecurityContext for spring security authentication
    val authentication = mock(Authentication::class.java)
    val securityContext = mock(SecurityContext::class.java)
    SecurityContextHolder.setContext(securityContext)

    // defining behavior of mocked repositories
    `when`(activityRepository.findByActivityCategory(activity1.activityCategory)).thenReturn(activity1)
    `when`(securityContext.authentication).thenReturn(authentication)
    `when`(authentication.name).thenReturn(user1.email)
    `when`(userRepository.findByEmail(authentication.name)).thenReturn(user1)
    `when`(countyRepository.findByCountyName(county1.countyName)).thenReturn(county1)
    `when`(cityRepository.findByCityName(city1.cityName)).thenReturn(city1)
    `when`(cityRepository.findByCityNameAndCounty(city1.cityName, county1)).thenReturn(city1)
    `when`(colorRepository.findByName(color1.colorName)).thenReturn(color1)
    `when`(speciesRepository.findByName(species1.speciesName)).thenReturn(species1)
    `when`(locationRepository.findMaxLocationId()).thenReturn( value: 0L)
    `when`(locationRepository.save(any<Location>())).thenReturn(location1)
    `when`(petRepository.findMaxPetId()).thenReturn( value: 0L)
    `when`(petRepository.save(any<Pet>())).thenReturn(pet1)
    `when`(adRepository.findMaxAdId()).thenReturn( value: 0L)
    `when`(adRepository.save(any<Ad>())).thenReturn(ad1)
    `when`(imageRepository.findMaxImageId()).thenReturn( value: 0L)
    `when`(imageRepository.save(image1)).thenReturn(image1)

    val result = adService.addAd(addAdDTO1)

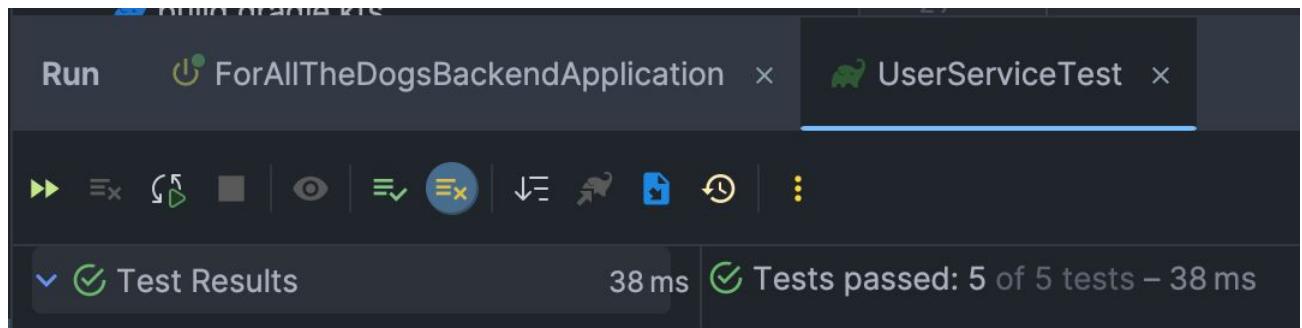
    assertNotNull(result)
    checkIfEqual(addAdDTO1, result)

    SecurityContextHolder.clearContext()
}
```

Slika 5.6: Unit test 6 - dodavanje novog oglasa

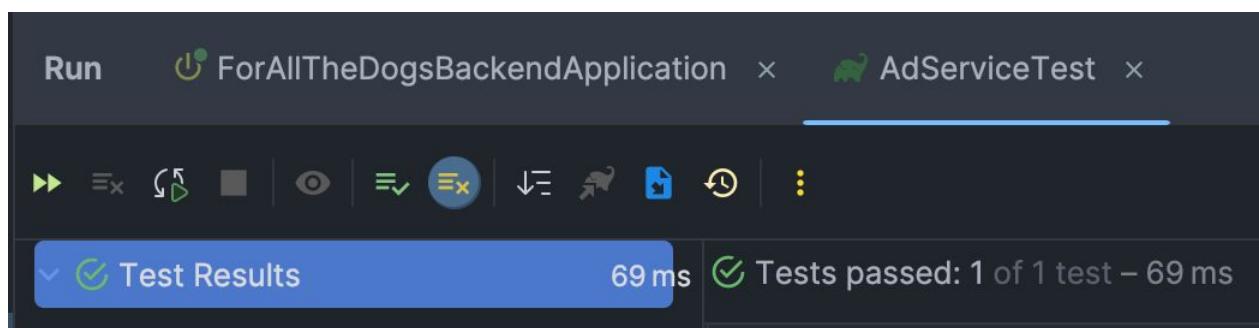
### Prikaz rezultata unit testova

Prvih 5 unit testova (loginValidTest, loginInvalidTest, registerValidTest, registerInvalidTest i getAllSheltersTest) služi za testiranje metoda iz UserService-a. Svi testovi su bili uspješni.



Slika 5.7: Prikaz prolaznosti unit testova (1. - 5.).

Zadnji test (addNewAd) služi za testiranje metode iz AdService-a koja omogućuje dodavanje novog oglasa. Test je također bio uspješan.



Slika 5.8: Prikaz prolaznosti 6. unit testa

### 5.2.2 Ispitivanje sustava

Ispitivanje sustava je vrsta softverskog testiranja koja se fokusira na ispitivanje i provjeru cjelokupnog sustava kako bi se omogućilo da svi njegovi dijelovi funkcioniraju zajedno prema specifikacijama i očekivanjima. Kako bismo to postigli u projektu smo koristili radni okvir *WebDriverIO* koji je baziran na Selenium WebDriver-u te pruža jednostavan i efikasan način za automatizaciju testova web stranica i aplikacija.

#### Ispitni slučaj 1: provjera ispravnosti registracije

Ovaj system test provjerava ispravnost sustava registracije. Korisnik odlazi na glavnu stranicu i čeka dok se ne pojavi gumb za sign-up. Tada ga klikne, ispuni podatke za registraciju (username, e-mail, password, ime i broj telefona) i označi tip korisnika, a zatim klikne gumb za predaju (submit). Na samom kraju se provjerava ispravnost dobivenog URL-a koji nam govori o uspješnosti registracije.

#### Ulaz:

1. Unos svih ispravnih podataka
2. Izostavljen unos username-a
3. Izostavljen unos e-mail-a

#### Očekivani rezultat:

1. Nije došlo do greške - svi su podaci uspješno uneseni.
2. Došlo je do greške. I dalje smo na sign-up dijelu dok se ne unesu svi podaci ispravno.
3. Došlo je do greške. I dalje smo na sign-up dijelu dok se ne unesu svi podaci ispravno.

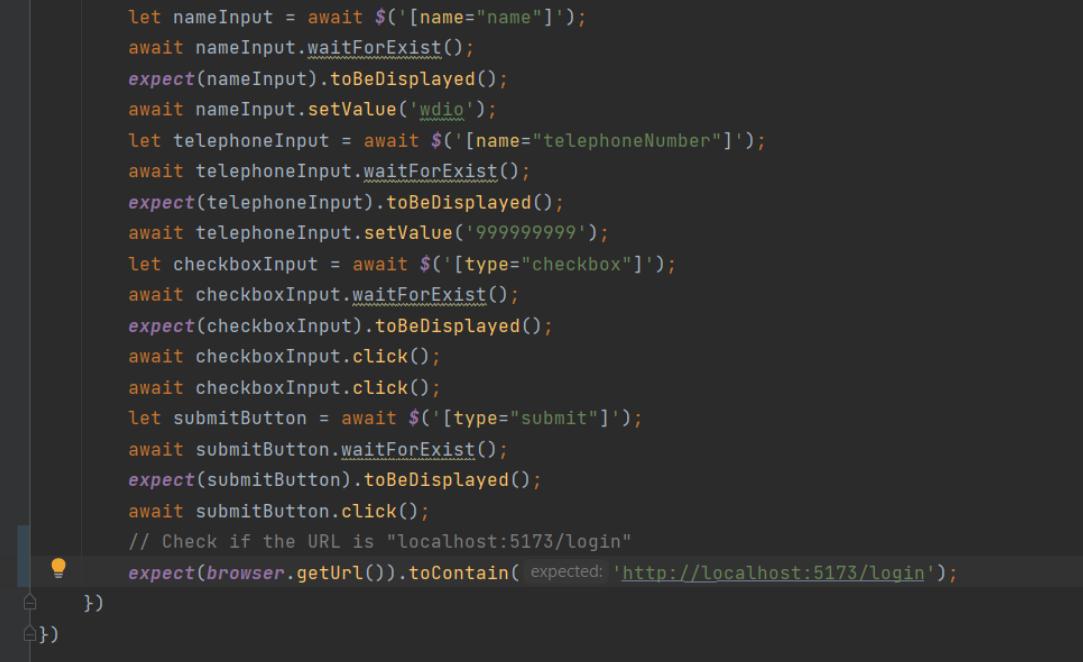


```

    ↳ Borna Josipović *
describe( title: 'Register test', fn: () => {
  it( title: 'should register valid credentials', fn: async () => {
    //load main page
    await browser.url( path: `http://localhost:5173/` );
    //click on signup
    const signUpButton = await $( '#signupButton' );
    await signUpButton.waitForExist();
    expect(signUpButton).toBeDisplayed();
    await signUpButton.click();
    //find and input username
    const usernameInput = await $( '[name="username"]' );
    expect(usernameInput).toBeDisplayed();
    await usernameInput.setValue('wdioTest');
    //find and input email
    let emailInput = await $( '[name="email"]' );
    expect(emailInput).toBeDisplayed();
    await emailInput.setValue('wdioTest@test.com');
    // Wait for the email input to be available
    let passwordInput = await $( '[name="password"]' );
    await passwordInput.waitForExist();
    expect(passwordInput).toBeDisplayed();
    await passwordInput.setValue('wdio');
    let nameInput = await $( '[name="name"]' );
    await nameInput.waitForExist();
    expect(nameInput).toBeDisplayed();
    await nameInput.setValue('wdio');
  });
});

```

Slika 5.9: System test: ispravan unos podataka pri registraciji (1.)



```

let nameInput = await $( '[name="name"]' );
await nameInput.waitForExist();
expect(nameInput).toBeDisplayed();
await nameInput.setValue('wdio');
let telephoneInput = await $( '[name="telephoneNumber"]' );
await telephoneInput.waitForExist();
expect(telephoneInput).toBeDisplayed();
await telephoneInput.setValue('999999999');
let checkboxInput = await $( '[type="checkbox"]' );
await checkboxInput.waitForExist();
expect(checkboxInput).toBeDisplayed();
await checkboxInput.click();
await checkboxInput.click();
let submitButton = await $( '[type="submit"]' );
await submitButton.waitForExist();
expect(submitButton).toBeDisplayed();
await submitButton.click();
// Check if the URL is "localhost:5173/login"
expect(browser.getUrl()).toContain( expected: 'http://localhost:5173/login' );
})
}

```

Slika 5.10: System test: ispravan unos podataka pri registraciji (2.)

**Rezultat:** Svi su rezultati testova jednaki očekivanim.

### Ispitni slučaj 2: provjera ispravnosti ulogiravanja

Ovaj system test provjerava ispravnost sustava ulogiravanja. Korisnik čeka dok se ne pojavi gumb za log-in, a kad se pojavi unosi svoj e-mail i password. Zatim klikne gumb za predaju te se odvija provjera ispravnosti dobivenog URL-a. Nakon toga se prepoznaje i dohvata username korisnika i ispisuje se poruka "Hello, *username*".

#### Ulaz:

1. Unos svih ispravnih podataka
2. Unos e-mail-a koji ne postoji u bazi podataka
3. Izostavljen unos e-mail-a

#### Očekivani rezultat:

1. Nije došlo do greške - svi su podaci uspješno uneseni.
2. Došlo je do greške i ispisa "Wrong Credentials!" pri predaji promjena (submit).
3. Došlo je do greške. I dalje smo na log-in dijelu dok se ne unesu svi podaci ispravno.

```

1  const { expect, browser, $ } = require('@wdio/globals')
2
3  new *
4  describe('Login test', () => {
5    it('should login with valid credentials', async () => {
6      await browser.url('http://localhost:5173/')
7      const loginButton = await $('#loginButton');
8      await loginButton.click();
9      const emailInput = await $('#email');
10     await emailInput.waitForExist();
11     expect(emailInput).toBeDisplayed();
12     await emailInput.setValue('wdioTest@test.com');
13     const passwordInput = await $('#password');
14     await passwordInput.waitForExist();
15     expect(passwordInput).toBeDisplayed();
16     await passwordInput.setValue('wdio');
17     const submitLogin = await $('#submitLogin');
18     await submitLogin.waitForExist();
19     expect(submitLogin).toBeDisplayed();
20     await submitLogin.click();
21     await browser.waitUntil(
22       condition: async () => (await browser.getUrl()) === 'http://localhost:5173/',
23       {timeout, interval, timeoutMsg}: {
24         timeout: 5000,
25         timeoutMsg: 'URL is not http://localhost:5173 after login',
26       }
27     );
28     // Wait for the showUsername element to be available

```

Slika 5.11: System test: ispravan unos podataka pri ulogiravanju (1.)

```

await browser.waitUntil(
  condition: async () => (await browser.getUrl()) === 'http://localhost:5173/',
  {timeout, interval, timeoutMsg}: {
    timeout: 5000,
    timeoutMsg: 'URL is not http://localhost:5173 after login',
  }
);
// Wait for the showUsername element to be available
const showUsername = await $('#showUsername');
await showUsername.waitForExist();
// Get the text content of the showUsername element
const usernameText = await showUsername.getText();
// Assert that the username is Drizzy drake
expect(usernameText).toContain(expected: 'Hello, wdioTest');
// Wait for the email input to be available
}
}

```

Slika 5.12: System test: ispravan unos podataka pri ulogiravanju (2.)

**Rezultat:** Svi su rezultati testova jednaki očekivanim.

**Ispitni slučaj 3: provjera ispravnosti izmjene oglasa**

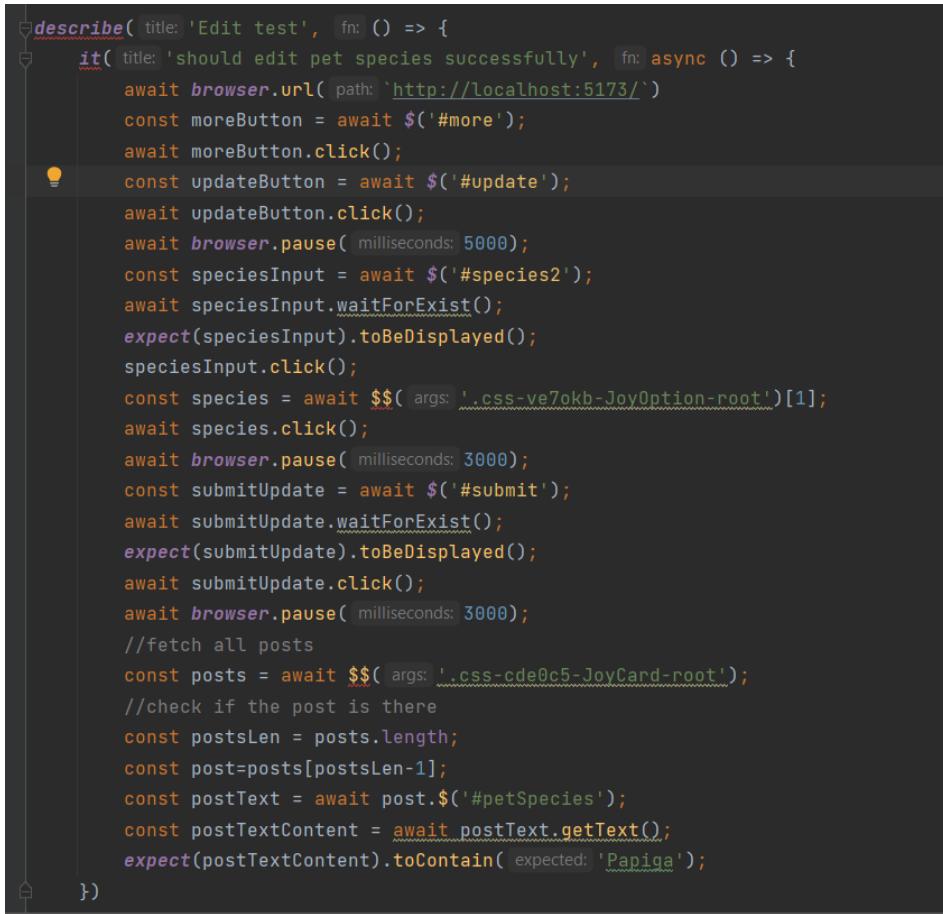
Ovaj system test provjerava ispravnost sustava izmjene oglasa. Pronalazi se post korisnika, a zatim se ulazi u ekran za izmjenu oglasa. Korisnik izvodi promjene u oglasu i klikne gumb za predaju. Uspoređuje se tekst iz oglasa na glavnoj stranici o izmijenjenom podatku s novom vrijednosti podatka. Ako su identični, izmjena je uspjela.

**Ulaz:**

- Promijenjena je vrsta životinje iz "Pas" u "Papiga" i pohranjuje se ta promjena
- Brisanje jedine slike unutar oglasa

**Očekivani rezultat:**

- Nije došlo do greške - promjena je uspjela.
- Došlo je do greške i ispisa "At least 1 image!" pri predaji promjena (submit).



```
describe('Edit test', fn: () => {
  it('should edit pet species successfully', fn: async () => {
    await browser.url(path: 'http://localhost:5173/')
    const moreButton = await $('#more');
    await moreButton.click();
    const updateButton = await $('#update');
    await updateButton.click();
    await browser.pause(milliseconds: 5000);
    const speciesInput = await $('#species2');
    await speciesInput.waitForExist();
    expect(speciesInput).toBeDisplayed();
    speciesInput.click();
    const species = await $$([ args: '.css-ve7okb-JoyOption-root' ])[1];
    await species.click();
    await browser.pause(milliseconds: 3000);
    const submitUpdate = await $('#submit');
    await submitUpdate.waitForExist();
    expect(submitUpdate).toBeDisplayed();
    await submitUpdate.click();
    await browser.pause(milliseconds: 3000);
    //fetch all posts
    const posts = await $$([ args: '.css-cde0c5-JoyCard-root' ]);
    //check if the post is there
    const postsLen = posts.length;
    const post=posts[postsLen-1];
    const postText = await post.$('#petSpecies');
    const postTextContent = await postText.getText();
    expect(postTextContent).toContain(expected: 'Papiga');
  })
})
```

Slika 5.13: System test: promjena vrste životinje iz "Pas" u "Papiga" na oglasu

```
new "Edit fail test"
describe('Edit fail test', () => {
  it('should try to edit pet but fail', async () => {
    await browser.url(`http://localhost:5173/`);
    const moreButton = await $('#more');
    await moreButton.click();
    const updateButton = await $('#update');
    await updateButton.click();
    const deleteImageButton = await $('#deleteImage');
    await deleteImageButton.click();
    const submitUpdate = await $('#submit');
    await submitUpdate.waitForExist();
    expect(submitUpdate).toBeDisplayed();
    await submitUpdate.click();
    const errorMessage = await $('.error-message');
    await errorMessage.waitForExist();
    expect(errorMessage).toBeDisplayed();
    const errorMessageText = await errorMessage.getText();
    expect(errorMessageText).toContain(expected: 'At least 1 image!');
    //fetch all posts
  })
})
```

Slika 5.14: System test: pokušaj brisanja jedine slike unutar oglasa

**Rezultat:** Svi su rezultati testova jednaki očekivanim.

#### Ispitni slučaj 4: provjera ispravnosti brisanja postojećeg oglasa

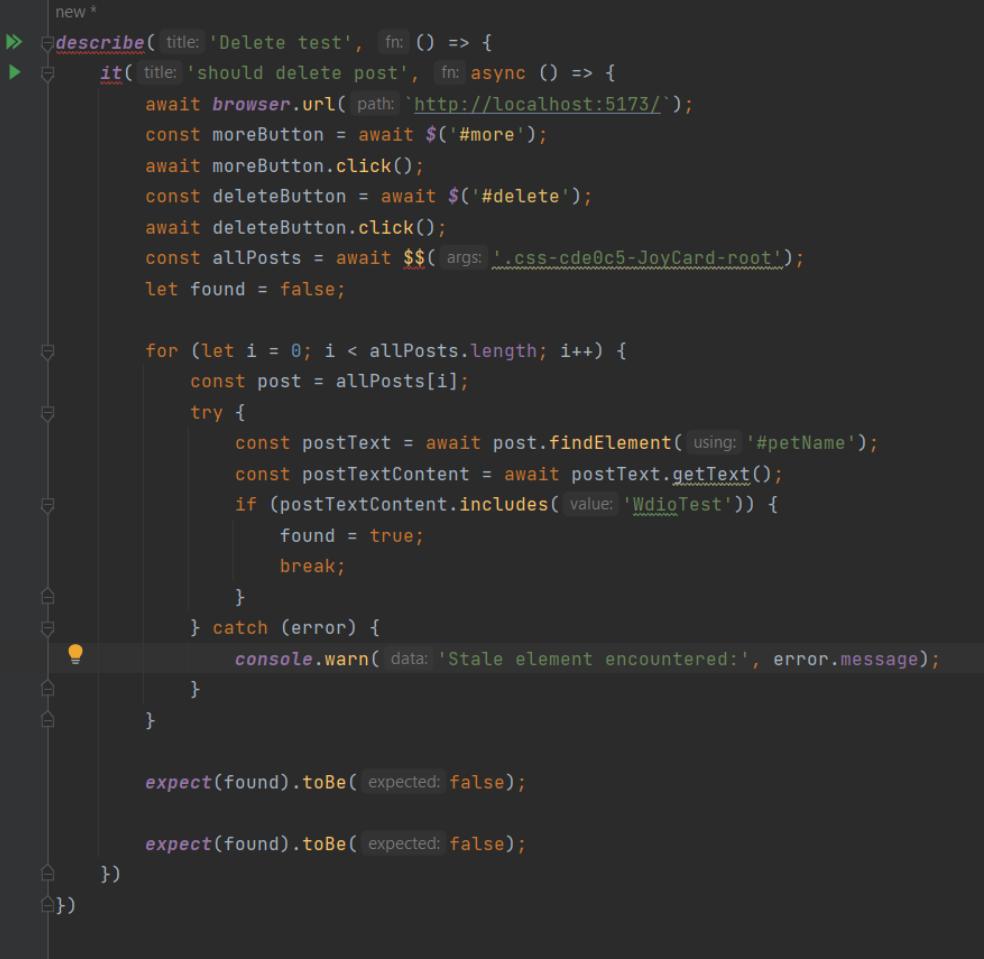
Ovaj system test provjerava ispravnost sustava brisanja oglasa. Korisnik pro-nalazi svoj oglas koji namjerava obrisati. Klikne na gumb "3 točke" te se otvaraju opcije manevriranja oglasom i izabire opciju "Delete". Pregledavaju se sva imena objava i ako ne postoji ime upravo obrisane objave među njima onda objava i je obrisana.

**Ulaz:**

1. Prikaz testiranja brisanja postojećeg oglasa

**Očekivani rezultat:**

1. Nije došlo do greške - oglas je uspješno obrisan.



```

new *

describe('Delete test', () => {
  it('should delete post', () => {
    await browser.url(`http://localhost:5173/`);
    const moreButton = await $('#more');
    await moreButton.click();
    const deleteButton = await $('#delete');
    await deleteButton.click();
    const allPosts = await $$({ args: '.css-cde0c5-JoyCard-root' });
    let found = false;

    for (let i = 0; i < allPosts.length; i++) {
      const post = allPosts[i];
      try {
        const postText = await post.findElement({ using: '#petName' });
        const postTextContent = await postText.getText();
        if (postTextContent.includes('WdigoTest')) {
          found = true;
          break;
        }
      } catch (error) {
        console.warn({ data: 'Stale element encountered:', error.message });
      }
    }

    expect(found).toBe(false);
    expect(found).toBe(false);
  })
})
}

```

Slika 5.15: System test: prikaz brisanja postojećeg oglasa

**Rezultat:** Rezultat testa je jednak očekivanom.

### Ispitni slučaj 5: provjera ispravnosti odjave korisničkog računa

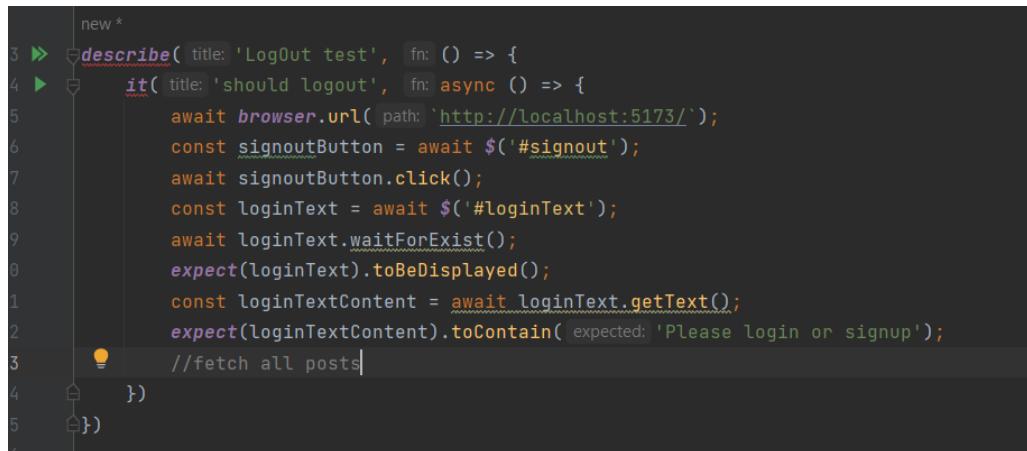
Ovaj system test provjerava ispravnost sustava odjave korisničkog računa. Korisnik klikne na gumb za odjavu (sign-out). Zatim se provjerava postojanje teksta koji se ispisuje na ekranu kada korisnik nije prijavljen ("Please login or signup") što potvrđuje da u tom trenutku nema prijavljenog korisnika.

#### Ulaz:

- Prikaz testiranja odjave korisničkog računa

#### Očekivani rezultat:

- Nije došlo do greške - korisnik se uspješno odjavio.



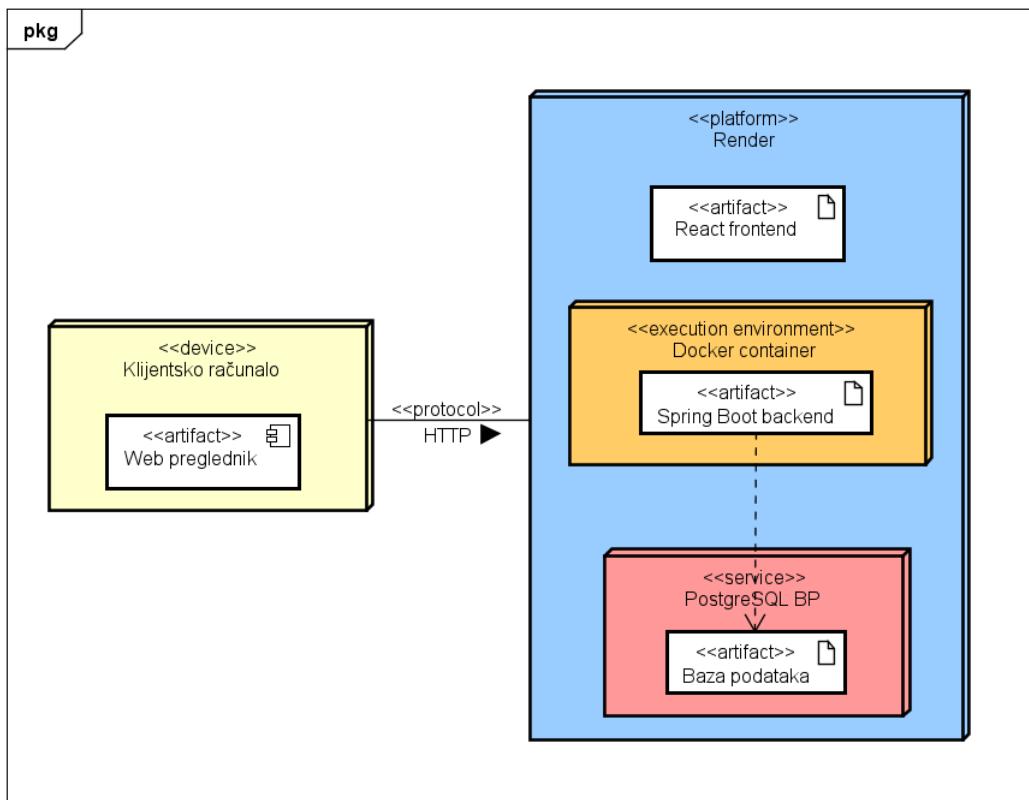
```
new *  
3 > 	describe( title: 'LogOut test', fn: () => {  
4 > 	> it( title: 'should logout', fn: async () => {  
5 	> 	> 	await browser.url( path: 'http://localhost:5173/' );  
6 	> 	const signoutButton = await $( '#signout' );  
7 	> 	await signoutButton.click();  
8 	> 	const loginText = await $( '#loginText' );  
9 	> 	await loginText.waitForExist();  
10 	> 	expect( loginText ).toBeDisplayed();  
11 	> 	const loginTextContent = await loginText.getText();  
12 	> 	expect( loginTextContent ).toContain( expected: 'Please login or signup' );  
13 	> 	//fetch all posts  
14 	> }  
15 };
```

Slika 5.16: System test: prikaz odjave postojećeg računa

**Rezultat:** Rezultat testa je jednak očekivanom.

### 5.3 Dijagram razmještaja

Sustav se zasniva na arhitekturi "klijent - poslužitelj". Klijenti koriste web preglednik za pristupanje web aplikaciji. Komunikacija klijenata i poslužitelja odvija se preko HTTP veze. Za posluživanje se koristi platforma Render.



Slika 5.17: Dijagram razmještaja

## 5.4 Upute za puštanje u pogon

Izvorni kod aplikacije postavljen je na GitHub repozitorij na adresi <https://github.com/Ljubimci-Grupa-1/ForAllTheDogs>. Puštanje u pogon obavit ćemo uz pomoć platforme Render<sup>18</sup>.

### 5.4.1 Frontend

U datoteku **package.json** dodajemo "dotenv", "express" i "http-proxy-middleware" ovisnosti (engl. *dependencies*) kao što je prikazano na slici 5.18.

```
17      "dependencies": {  
18          "@emotion/react": "^11.11.1",  
19          "@emotion/styled": "^11.11.0",  
20          "@fontsource/inter": "^5.0.15",  
21          "@mui/joy": "^5.0.0-beta.15",  
22          "bootstrap": "^5.3.2",  
23          "cors": "^2.8.5",  
24          "dotenv": "^16.3.1",  
25          "express": "^4.18.2",  
26          "http-proxy-middleware": "^2.0.6",|  
27          "react": "^18.2.0",  
28          "react-dom": "^18.2.0",  
29          "react-router-dom": "^6.17.0"  
30      },
```

Slika 5.18: "dependencies" u package.json

Datoteku **server.js**, prikazanu na slici 5.19, stavljamo u korijenski direktorij za frontend. Ona sadrži konfiguraciju za Express server koji koristimo za posluživanje frontenda.

<sup>18</sup><https://render.com>

```

1  const express = require('express');
2  const path = require('path');
3  const cors = require('cors');
4  const corsOptions = {
5      origin: 'https://forallthedogs-omw5.onrender.com',
6      methods: 'GET,HEAD,PUT,PATCH,POST,DELETE',
7      credentials: true,
8  };
9  const app = express();
10
11  app.use(cors(corsOptions));
12
13  app.use(express.static(path.join(__dirname, 'dist')));
14
15  app.get('*', (req, res) => {
16      res.sendFile(path.join(__dirname, 'dist', 'index.html'));
17  });
18
19  const port = process.env.PORT || 3000;
20  app.listen(port, () => {
21      console.log(`Server running on http://localhost:${port}`);
22  });

```

Slika 5.19: Sadržaj datoteke server.js

Zatim se vraćamo u datoteku **package.json** i ažuriramo "scripts" kao na slici 5.20 te obavimo *git push*.

```

9   "scripts": {
10     "dev": "vite",
11     "build": "npm install && npx tsc && vite build",
12     "start": "node server.js",
13     "start-prod": "node server.js",
14     "lint": "eslint . --ext ts,tsx --report-unused-disable-directives --max-warnings 0",
15     "preview": "vite preview"
16   },

```

Slika 5.20: "scripts" u package.json

Prijavljujemo se na <https://dashboard.render.com> svojim korisničkim podacima te odabiremo *New → Web Service*. Nakon odabira našeg GitHub repozitorija postavljamo željeno ime, regiju (EU Central), odgovarajuću granu repozitorija (main), te ime korijenskog direktorija za frontend dio aplikacije (direktorij **frontend**).

**You are deploying a web service for Ljubimci-Grupa-1/ForAllTheDogs.**

Name	for_all_the_dogs_frontend
Region	Frankfurt (EU Central)
Branch	main
Root Directory <small>Optional</small>	frontend

Slika 5.21: Postavljanje Web Service

Nadalje navodimo *Build* i *Start* naredbe koje će biti pokrenute u odabranom korijenskom direktoriju. Naredba *Build* (u našem slučaju *yarn build*) odgovorna je za instalaciju potrebnih programskih biblioteka i kompilaciju koda (uz automatski *redeployment* prilikom svake promjene izvornog koda na GitHub repozitoriju). Naredba *Start* (u našem slučaju *yarn start-prod*) služi za pokretanje aplikacije odnosno web poslužitelja.

**Runtime**

The runtime for your web service.

Node

**Build Command**

This command runs in the root directory of your repository when a new version of your code is pushed, or when you deploy manually. It is typically a script that installs libraries, runs migrations, or compiles resources needed by your app.

frontend/ \$ yarn build

**Start Command**

This command runs in the root directory of your app and is responsible for starting its processes. It is typically used to start a webserver for your app. It can access environment variables defined by you in Render.

frontend/ \$ yarn start-prod

Slika 5.22: Build i Start

Za kraj odabiremo opciju *Create Web Service*.

### 5.4.2 Baza podataka

Na Render Dashboard odabiremo *New → PostgreSQL*. Ispunjavamo željene postavke (ime, regija, PostgreSQL verzija (15) itd.) te završavamo s *Create Database*.

### 5.4.3 Backend

Za puštanje backenda u pogon ponovno odabiremo *New → Web Service* te kao korenjski direktorij odabiremo direktorij **backend**.

**You are deploying a web service for Ljubimci-Grupa-1/ForAllTheDogs.**

Name	for_all_the_dogs_backend
Region	Frankfurt (EU Central)
Branch	main
Root Directory <small>Optional</small>	backend
Runtime	Docker

Slika 5.23: Postavke za backend

Kao *Runtime* postavljamo Docker. Datoteka Dockerfile, pozicionirana u kori-jenskom direktoriju za backend, sadrži instrukcije za izgradnju i pakiranje (engl. *build-and-package*) aplikacije. Objasnimo sadržaj datoteke Dockerfile.

Prvo se definira "builder".

```
1 # Container za izgradnju (build) aplikacije
2 FROM openjdk:17-alpine AS builder
3
```

Slika 5.24: builder

Zatim se zadaje radni direktorij te u kontejner kopira konfiguracijske dato-ke alata Gradle (**build.gradle.kts** i **settings.gradle.kts**), skripte Gradle omotača (*Gradle Wrapper*) te izvorni kod aplikacije.

```
4      # Set up the working directory
5      WORKDIR /app
6
7      # Kopiranje Gradle Wrapper-a
8      COPY gradlew /app/
9      COPY gradle /app/gradle
10     COPY build.gradle.kts /app/
11     COPY settings.gradle.kts /app/
12     COPY src /app/src
13
```

Slika 5.25: Radni direktorij i Gradle

Naredbe na slici 5.26 (redom) služe za:

- davanje skripti Gradle omotača pravo izvršavanja (*execute permission*)
- preuzimanje *project dependencies*
- izgradnju aplikacije

```
14     # Dajemo izvršna prava Gradle Wrapperu
15     RUN chmod +x /app/gradlew
16
17     # Downloading dependencies
18     RUN /app/gradlew --version
19
20     # Pokretanje builda
21     RUN /app/gradlew assemble
22
```

Slika 5.26: Naredbe za izgradnju

Na kraju imamo naredbe vezane uz *staging*. Pokrećemo novi *stage* koji koristi istu osnovnu sliku za izvođenje aplikacije (*build stage* više nam nije potreban, ko-

piraju se samo potrebni artefakti). Izgrađena JAR datoteka (iz *build stage*) kopira se u trenutni *stage*. Zatim su navedene naredbe koje će biti izvedene pri pokretanju kontejnera. Aplikacija se pokreće kao izvršna datoteka (*executable*).

```
23      # Stvaranje containera u kojem ce se vrtiti aplikacija
24      FROM openjdk:17-alpine
25
26      # Kopiranje izvrsnog JAR-a iz build containera u izvrsni container
27      COPY --from=builder /app/build/libs/*.jar /app/app.jar
28
29      # Izlaganje porta
30      EXPOSE 8080
31
32      # Naredba kojom se pokrece aplikacija
33      ENTRYPOINT ["java","-jar","/app/app.jar"]
```

Slika 5.27: Naredbe za *staging*

Kao i prije, završavamo s *Create Web Service*.

## 6. Zaključak i budući rad

Zadatak naše grupe bio je napraviti web aplikaciju koja bi pomogla opet ujediniti vlasnike sa svojim nestalim ljubimcima. Aplikacija služi kao svojevrsni portal s oglasima o izgubljenim, ali i pronađenim životinjama.

Aplikaciju smo izrađivali tijekom oba nastavna ciklusa predmeta što je trajalo otprilike 14 tjedana. U svakom ciklusu smo imali zadatke koje smo morali ispuniti do unaprijed zadanog roka, a na kraju svakog ciklusa se ocjenjivao naš rad i uspjeh projektnog zadatka.

U prvoj reviziji projekta, naglasak je bio na dokumentaciji i na izradi temelja za programsku potporu naše web aplikacije. Na početku projekta znali smo da je važno imati dobru povezanost i komunikaciju unutar tima te je uslijedilo upoznavanje članova, podjela posla i rasprava o potencijalnim načinima i tehnologijama pomoću kojih bismo mogli provesti ovaj projekt. Posao smo podijelili po parovima za izradu klijentske i poslužiteljske strane aplikacije te izradu dokumentacije, a znalo se dogoditi i da si svi članovi međusobno pomažu čime smo ubrzali proces izrade aplikacije, ali i podijelili znanje. Krenuli smo izradom modela baze podataka i pisanjem dokumentacije, a kako smo se bližili kraju prve faze projekta krenuli smo i s implementacijom pojedinih obrazaca uporabe i povezivanjem klijentske i poslužiteljske strane. Ipak, izrada dokumentacije je imala glavnu riječ u ovoj fazi projekta. Shvatili smo da trebamo velik fokus staviti na dijagram baze podataka, smišljanje obrazaca uporabe, sekveničkih dijagrama i dijagram razreda jer će upravo oni biti podloga za razvoj programske potpore naše aplikacije. Pomoći su nam da unaprijed uvidimo gdje bi mogli izviriti problemi i koji su dijelovi ključni za dobru izradu i izvedbu našeg portala.

Za drugu reviziju naglasak se prebacio na samu implementaciju programske potpore za sve funkcionalnosti koje nisu uključene u izvedbu tijekom prve revizije. U ovom dijelu su članovi tima koji su radili *back-end* puno više komunicirali s onima koji su radili *front-end* jer su njihovi zadaci ovisili jedni od drugima. Dijagrami su u ovoj fazi prikazivali stanja objekata i njihove prijelaze, način funkcioniranja aplikacije (odnos između korisnika, web stranice i baze podataka) te organizaciju i međuovisnost komponenti.

Očekivali smo da ćemo projekt uspješno svladati, iako su svi članovi imali različita iskustva i znanja. To se i dogodilo, a svaki je član ponešto novo naučio i upoznao se s tehnologijom koju je koristio pri rješavanju svog dijela projektnog zadatka. Naravno, svi su članovi međusobno dijelili sve što su do nekog trenutka naučili. Na početku su možda iskusniji članovi uzeli inicijativu u svoje ruke, ali s rastom projekta i novim naučenim znanjem drugi su ih članovi dostigli i svi su jednako doprinosili projektu. Članovi koji su se već bili susreli s potencijalnim problemima znali su kako reagirati i po mogućnosti ih izbjegići. Kada bi jedan član završio s nekim dijelom zadatka, drugi član/ovi bi pregledali zadovoljava li napravljeno sve što je i trebalo i time bismo skratili vrijeme izrade projekta jer bi moguća greška bila na vrijeme uočena.

Komunikacija vezana uz projekt održana je preko Discord-a gdje su postojali kanali za različite dijelove projekta. Najčešće smo komunicirali preko poruka, a sastanci su se odvijali uživo i preko video-poziva.

Sudjelovanjem u ovom projektu svi su članovi nešto novo naučili, čak i oni koji su imali nešto više znanja od drugih. Možemo reći da se svi ponešto odvažnije osjećamo po pitanju ovakvog tipa projektnog zadatka. Na kraju projekta smo shvatili što se sve od nas očekuje pri izradi web aplikacije te kako sve zahtjeve implementirati i povezati tako da sustav bude samoodrživ. Također smo shvatili da svaki član mora dati sve od sebe za dio zadatka kojeg radi, proučiti literaturu i istražiti nešto novo kako bismo probali izbjegći zaostatke i imali kontinuirani rast i razvoj web aplikacije.

# Popis literature

1. Programsko inženjerstvo, FER ZEMRIS, <http://www.fer.hr/predmet/proinzh>
2. The Unified Modeling Language, <https://www.uml-diagrams.org/>
3. Astah Community, <http://astah.net/editions/uml-new>

# Indeks slika i dijagrama

2.1 Oglas platforme PawBoost . . . . .	8
2.2 Usluge koje nudi platforma PetcoLove . . . . .	9
2.3 Web stranica platforme PetFinder . . . . .	9
3.1 Dijagram mogućnosti korisnika . . . . .	17
3.2 Sekvencijski dijagram pretraživanja i pregleda oglasa . . . . .	18
3.3 Sekvencijski dijagram postavljanja oglasa . . . . .	19
3.4 Sekvencijski dijagram izmjenjivanja/brisanja oglasa . . . . .	20
4.1 MVC stil arhitekture . . . . .	23
4.2 ER dijagram baze podataka . . . . .	30
4.3 Relacijski dijagram baze podataka . . . . .	31
4.4 Dijagram razreda entiteta . . . . .	32
4.5 Dijagram razreda <i>Repository</i> . . . . .	33
4.6 UserService i UserRepository . . . . .	34
4.7 Razredi - oglasi . . . . .	35
4.8 Razredi - dodavanje oglasa i poruka . . . . .	35
4.9 Razredi - poruke . . . . .	36
4.10 Razredi - korisnici . . . . .	37
4.11 Razredi - gradovi i županije . . . . .	38
4.12 Razredi - vrste i boje ljubimaca . . . . .	38
4.13 Autorizacija, upravljanje iznimkama i CORS . . . . .	39
4.14 Dijagram stanja . . . . .	41
4.15 Dijagram aktivnosti . . . . .	43
4.16 Dijagram komponenti . . . . .	44
5.1 Unit test 1 - <i>log-in</i> . . . . .	48
5.2 Unit test 2 - <i>log-in</i> s neispravnom lozinkom. . . . .	49
5.3 Unit test 3 - registracija . . . . .	50
5.4 Unit test 4 - registracija s već postojećim <i>e-mail</i> -om . . . . .	51
5.5 Unit test 5 - ruta dohvaćanja svih korisnika . . . . .	52

5.6	Unit test 6 - dodavanje novog oglasa . . . . .	53
5.7	Prikaz prolaznosti unit testova (1. - 5.) . . . . .	54
5.8	Prikaz prolaznosti 6. unit testa . . . . .	54
5.9	System test: ispravan unos podataka pri registraciji (1.) . . . . .	56
5.10	System test: ispravan unos podataka pri registraciji (2.) . . . . .	56
5.11	System test: ispravan unos podataka pri ulogiravanju (1.) . . . . .	58
5.12	System test: ispravan unos podataka pri ulogiravanju (2.) . . . . .	58
5.13	System test: promjena vrste životinje iz "Pas" u "Papiga" na oglasu .	60
5.14	System test: pokušaj brisanja jedine slike unutar oglasa . . . . .	61
5.15	System test: prikaz brisanja postojećeg oglasa . . . . .	62
5.16	System test: prikaz odjave postojećeg računa . . . . .	63
5.17	Dijagram razmještaja . . . . .	64
5.18	"dependencies" u package.json . . . . .	65
5.19	Sadržaj datoteke server.js . . . . .	66
5.20	"scripts" u package.json . . . . .	66
5.21	Postavljanje Web Service . . . . .	67
5.22	Build i Start . . . . .	68
5.23	Postavke za backend . . . . .	69
5.24	builder . . . . .	69
5.25	Radni direktorij i Gradle . . . . .	70
5.26	Naredbe za izgradnju . . . . .	70
5.27	Naredbe za <i>staging</i> . . . . .	71
6.1	Pregled aktivnosti (1.) . . . . .	82
6.2	Pregled aktivnosti (2.) . . . . .	83

# Dodatak: Prikaz aktivnosti grupe

## Dnevnik sastajanja

### 1. sastanak

- Datum: 16. listopada 2023.
- Prisustvovali: svi članovi grupe
- Teme sastanka:
  - dogovor oko korištenih tehnologija
  - okvirna podjela na podtimove

### 2. sastanak

- Datum: 20. listopada 2023.
- Prisustvovali: Božo Đerek, Lucija Lovrić
- Teme sastanka:
  - razrada plana i podjela posla za *Opis projektnog zadatka*

### 3. sastanak

- Datum: 25. listopada 2023.
- Prisustvovali: svi članovi grupe
- Teme sastanka:
  - razmatranje osmišljenog plana za bazu podataka
  - izrada ER dijagrama

### 4. sastanak

- Datum: 30. listopada 2023.
- Prisustvovali: svi članovi grupe
- Teme sastanka:
  - razrada plana za *Specifikaciju programske potpore*
  - definiranje obrazaca uporabe
  - definiranje ostalih zahtjeva

**5. sastanak**

- Datum: 31. listopada 2023.
- Prisustvovali: Lucija Runjić, Vedran Moškov, Andrija Merlin, Borna Josipović, Lana Bartolović
- Teme sastanka:
  - dogovor oko povezivanja *back-end-a* s bazom podataka

**6. sastanak**

- Datum: 3. studenoga 2023.
- Prisustvovali: svi članovi grupe
- Teme sastanka:
  - dogovor oko *Arhitekture i dizajna sustava*

**7. sastanak**

- Datum: 8. studenoga 2023.
- Prisustvovali: Lucija Runjić, Vedran Moškov, Borna Josipović, Andrija Merlin
- Teme sastanka:
  - izrada *log-in* registra
  - ostvarivanje potpune funkcionalnosti ruta za *sign-up* i *log-in*

**8. sastanak**

- Datum: 17. studenoga 2023.
- Prisustvovali: svi članovi grupe
- Teme sastanka:
  - revizija napravljenog

**9. sastanak**

- Datum: 7. prosinca 2023.
- Prisustvovali: svi članovi grupe
- Teme sastanka:
  - raspodjela posla i dogovor oko prepravljanja i punjenja baze podataka

**10. sastanak**

- Datum: 20. prosinca 2023.
- Prisustvovali: Lucija Runjić, Vedran Moškov, Borna Josipović, Andrija Merlin, Lana Bartolović
- Teme sastanka:

- spajanje *back-end-a* i *front-end-a* za postavljanje novog oglasa i dodavanje komentara

#### 11. sastanak

- Datum: 9. siječnja 2024.
- Prisustvovali: Lucija Runjić, Vedran Moškov, Borna Josipović, Andrija Merlin
- Teme sastanka:
  - dogovor oko stavljanja aplikacije na javni poslužitelj

#### 12. sastanak

- Datum: 18. siječnja 2024.
- Prisustvovali: svi članovi grupe
- Teme sastanka:
  - revizija napravljenog

## Tablica aktivnosti

Napomena: Doprinose u aktivnostima treba navesti u satima po članovima grupe po aktivnosti.

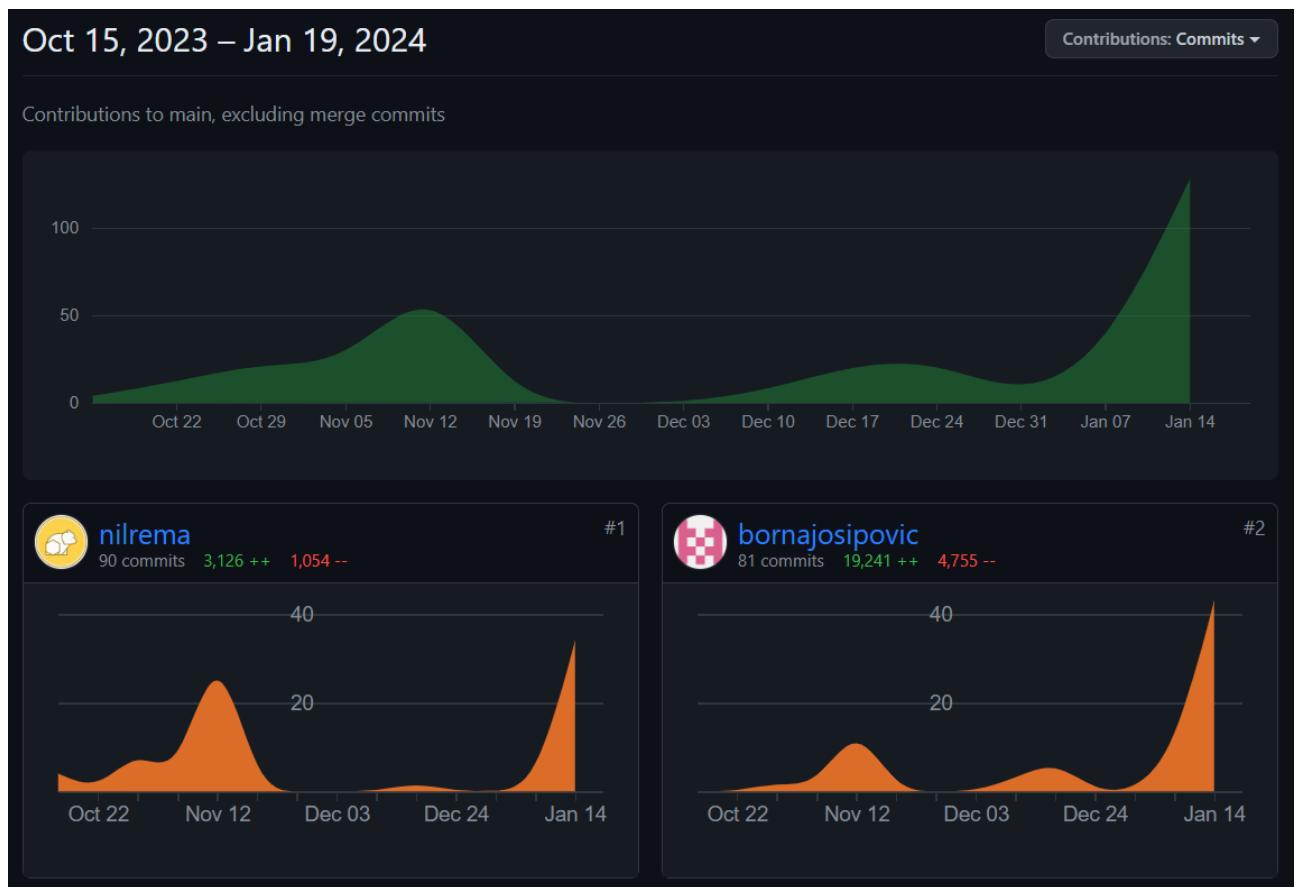
	Andrija Merlin	Božo Derek	Lana Bartolović	Lucija Lovrić	Vedran Moškov	Lucija Runjić	Borna Josipović
Upravljanje projektom	2h						
Opis projektnog zadatka		4h		6h			
Funkcionalni zahtjevi	4h	6h		6h			3h
Opis pojedinih obrazaca		5h		5h			
Dijagram obrazaca		4h		4h			
Sekvencijski dijagrami		5h		5h			
Opis ostalih zahtjeva		1h					
Arhitektura i dizajn sustava	2h	1h		2h	2h	1h	2h
Baza podataka		3h	4h			5h	
Dijagram razreda		11h	4h				
Dijagram stanja		4h					
Dijagram aktivnosti		2h					
Dijagram komponenti		2h					
Korištene tehnologije i alati		1h		3h			
Ispitivanje programskog rješenja				4h			
Dijagram razmještaja		1h					
Upute za puštanje u pogon	2h	2h					
Dnevnik sastajanja				1.5h			
Zaključak i budući rad				3h			

Nastavljeno na idućoj stranici

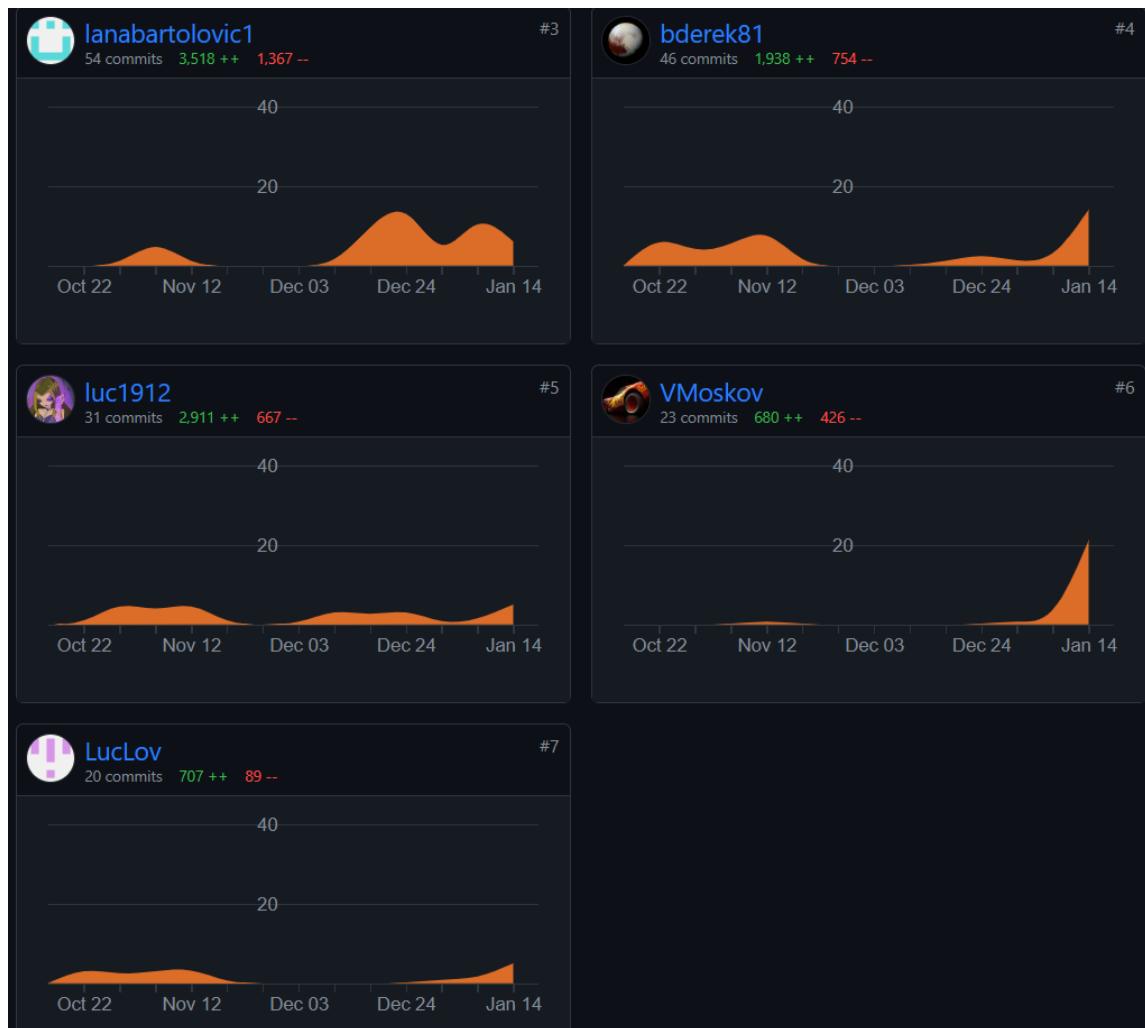
Nastavljeno od prethodne stranice

	Andrija Merlin	Božo Đerek	Lana Bartolović	Lucija Lovrić	Vedran Moškov	Lucija Runjić	Borna Josipović
Popis literature		0.5h					
<i>Frontend</i>	25h		74h			87h	
<i>Izrada modela baze podataka</i>			16h			3h	
<i>Spajanje s bazom podataka</i>					11h	7h	
<i>Backend</i>					56h	60h	
<i>Puštanje u pogon (deployment)</i>	18h				3h	8h	
<i>Testiranje rada aplikacije</i>	2h				8h	10h	7h

## Dijagrami pregleda promjena



Slika 6.1: Pregled aktivnosti (1.)



Slika 6.2: Pregled aktivnosti (2.)