Web aplikacija za vlasnike kućnih ljubimaca

Stjepanović, Nikolina

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

Sveučilišni diplomski studij Računarstva

WEB APLIKACIJA ZA VLASNIKE KUĆNIH LJUBIMACA

Diplomski rad

Nikolina Stjepanović

Osijek, 2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za ocjenu diplomskog rada na sveučilišnom diplomskom studiju

Ocjena diplomskog rada na sveučilišnom diplomskom studiju

Ime i prezime pristupnika:	Nikolina Stjepanović
Studij, smjer:	Sveučilišni diplomski studij Računarstvo
Mat. br. pristupnika, god.	D1325R, 07.10.2022.
JMBAG:	0165082505
Mentor:	prof. dr. sc. Krešimir Nenadić
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Mirko Köhler
Član Povjerenstva 1:	prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 2:	izv. prof. dr. sc. Ivica Lukić
Naslov diplomskog rada:	Web aplikacija za vlasnike kućnih ljubimaca
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	Opisati zahtjeve za izradu web aplikacije koja se može koristiti kao oblik društvene mreže za vlasnike kućnih ljubimaca. Dati opis sličnih postojećih rješenja s usporedbom izrađene web aplikacije i postojećih rješenja te naglasiti prednosti i nedostatke u odnosu na postojeća rješenja. Projektirati i izraditi bazu podataka koju će web aplikacija koristiti za pohranu svih relevantnih podataka te opisati postupak izrade baze. Web aplikacija treba imati mogućnost registracije novih korisnika. Registrirani korisnici trebaju imati mogućnost objave slika svojih ljubimaca kao i komentiranje
Datum ocjene pismenog dijela diplomskog rada od strane mentora:	20.09.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	01.10.2024.
Ocjena usmenog dijela diplomskog rada (obrane):	Izvrstan (5)
Ukupna ocjena diplomskog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:	15.10.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 15.10.2024.

Ime i prezime Pristupnika:	Nikolina Stjepanović
Studij:	Sveučilišni diplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D1325R, 07.10.2022.
Turnitin podudaranje [%]:	3

Ovom izjavom izjavljujem da je rad pod nazivom**: Web aplikacija za vlasnike kućnih ljubimaca**

izrađen pod vodstvom mentora prof. dr. sc. Krešimir Nenadić

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

SADRŽAJ

1. UVOD 1

1.1. Zadatak diplomskog rada 1

2. PREGLED PODRUČJA 3

2.1. YummyPets 3

2.2. PetLovers Chat 4

2.3. United Pets 5

2.4. Bauwow 6

2.5. Dokonoko 7

3. ZAHTJEVI ZA APLIKACIJU 8

3.1. Funkcionalni zahtjevi 8

3.2. Nefunkcionalni zahtjevi 9

3.3. Arhitekturalni dizajn aplikacije 10

3.3.1. Struktura baze podataka 10

4. IZRADA WEB APLIKACIJE 13

4.1. Struktura web aplikacije 13

4.2. Registracija i prijava korisnika 14

4.2.1. Registracija korisnika 14

4.2.2. Prijava korisnika 16

4.3. Korisnički profil 18

4.3.1. Slika profila 18

4.3.2. Dodavanje, uređivanje i brisanje ljubimca 20

4.3.3. Dodavanje i brisanje objava 22

4.4. Interakcija s drugim korisnicima 24

4.4.1. Pretraživanje i praćenje korisnika 24

4.4.2. Pregled pratitelja i praćenih osoba 28

4.4.3. Pregled objava 29

4.4.4. Označavanje objava sa sviđanjem 30

4.4.5. Komentiranje objava 31

4.4.6. Slanje poruka 33

4.5. Dodatne funkcionalnosti 35

4.5.1. Ocjenjivanje korisničkog iskustva 35

4.5.2. Statistička obrada rezultata koju obavlja administrator 36
4.6. Objavljivanje aplikacije 36
5. PRIKAZ RADA WEB APLIKACIJE 38
5.1. Prijava korisnika 38
5.2. Korisnički profil prijavljenog korisnika 40
5.2.1. Dodavanje slike profila 41
5.2.2. Dodavanje, uređivanje i brisanje ljubimca 42
5.2.3. Dodavanje i brisanje objave 45
5.2.4. Označavanje objava sa sviđanjem 46
5.2.5. Komentiranje objava 47
5.3. Korisnički profil ostalih korisnika 48
5.3.1. Praćenje korisnika 49
5.4. Slanje poruka 50
5.5. Obavijesti 53
5.6. Ocjenjivanje korisničkog iskustva i obrada rezultata 54
6. ZAKLJUČAK 57
LITERATURA 58
SAŽETAK 60
ABSTRACT 61
PRILOZI 62

1. UVOD

U današnjem digitalnom dobu, društvene mreže postale su središnje mjesto za dijeljenje svakodnevnih trenutaka, čega su i kućni ljubimci postali neizostavan dio. Mnogi ljudi svoje ljubimce doživljavaju kao članove obitelji te žele podijeliti njihove avanture i posebne trenutke s prijateljima, poznanicima, ali i drugima putem interneta. Ova tendencija potaknula je potražnju za specijaliziranim društvenim mrežama koje su posvećene isključivo ljubiteljima životinja.

Diplomski rad koji slijedi fokusira se na izradu web aplikacije koja služi kao društvena mreža za vlasnike kućnih ljubimaca. Glavni cilj aplikacije je omogućiti vlasnicima da dijele fotografije, međusobno se povezuju, te razmjenjuju iskustva i savjete. Aplikacija, osim standardnih funkcionalnosti poput objavljivanja sadržaja, pruža korisnicima mogućnost kreiranje profila, dodavanja ljubimaca, praćenje aktivnosti drugih korisnika, komentiranje i označavanje objava sa sviđanjem, kao i komunikaciju putem privatnih poruka. Na taj način stvara se digitalno okruženje koje potiče interakciju i izgradnju zajednice temeljene na zajedničkoj ljubavi prema životinjama.

U drugom poglavlju rada prikazano je nekoliko sličnih rješenja, te su identificirane njihove glavne mogućnosti, prednosti i nedostaci. U trećem poglavlju nabrojani su i opisani svi funkcionalni i nefunkcionalni zahtjevi aplikacije. Osim toga, navedene su korištene tehnologije te je prikazana i objašnjena shema baze podataka. Četvrto poglavlje prikazuje strukturu izrađene aplikacije te sadrži detaljno opisanu implementaciju svih glavnih funkcionalnosti aplikacije uz primjere koda. U petom poglavlju prikazan je i opisan način rada aplikacije, te su prikazane statistički obrađene ocjene korisnika vezane za korisničko iskustvo aplikacije.

1.1. Zadatak diplomskog rada

Opisati zahtjeve za izradu web aplikacije koja se može koristiti kao oblik društvene mreže za vlasnike kućnih ljubimaca. Dati opis sličnih postojećih rješenja s usporedbom izrađene web aplikacije i postojećih rješenja te naglasiti prednosti i nedostatke u odnosu na postojeća rješenja. Projektirati i izraditi bazu podataka koju će web aplikacija koristiti za pohranu svih relevantnih podataka te opisati postupak izrade baze. Web aplikacija treba imati mogućnost registracije novih korisnika. Registrirani korisnici trebaju imati mogućnost objave slika svojih ljubimaca kao i komentiranje ljubimaca drugih korisnika. Omogućiti praćenje poruka nekog korisnika kao i neki način komunikacije korisnika u obliku društvene mreže. Postupak izrade web aplikacije detaljno opisati kao i izrađene funkcionalnosti. U aplikaciju je potrebno ugraditi funkcionalnost kojom će

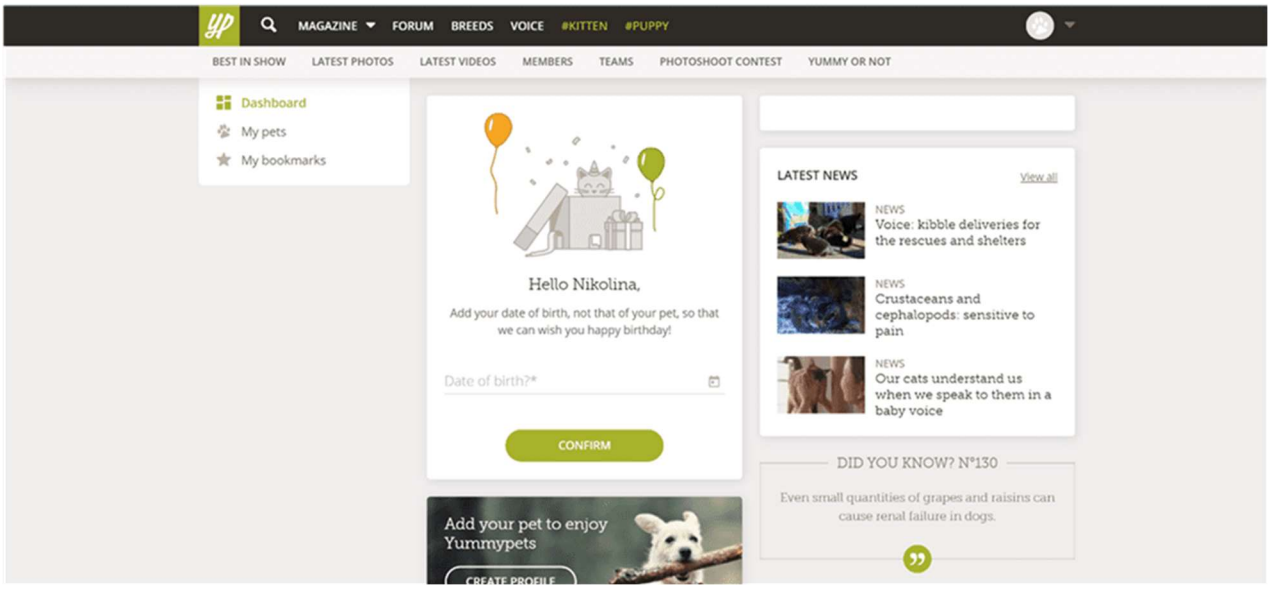
korisnici moći ocijeniti iskustvo korištenja izrađene web aplikacije. Za potrebe istraživanja potrebno je prikupiti određeni broj ocjena aplikacije te podatke statistički obraditi i komentirati rezultate.

2. PREGLED PODRUČJA

Postoje brojne aplikacije koje se mogu koristiti kao oblik društvene mreže za vlasnike kućnih ljubimaca, a većina ih je dostupna u obliku mobilne aplikacije. Za potrebe ovog rada, pronađeno je i analizirano četiri web i jedno mobilno rješenje, a cilj svake je omogućiti vlasnicima kućnih ljubimaca povezivanje, dijeljenje trenutka, te razmjenu savjeta i iskustava o zdravlju i brizi svojih kućnih ljubimaca.

2.1. YummyPets

Yummypets [1] je društvena mreža za vlasnike kućnih ljubimaca, koja osim u web obliku postoji i kao mobilna aplikacija. Na početnom zaslonu aplikacije nalazi se obrazac za prijavu, a neprijavljeni korisnik ima mogućnost čitanja članaka, pregledavanja posljednje dodanih fotografija i videozapisa, čitanja foruma, pregledavanja timova i članova aplikacije te pretraživanje ljubimaca, članaka, fotografija i drugog sadržaja. Također, može saznati više informacija o različitim pasminama kućnih ljubimaca poput porijekla, očekivanog životnog vijeka i ostalih karakteristika. Nakon prijave u aplikaciju korisniku se otvara zaslon kao na slici 2.1.



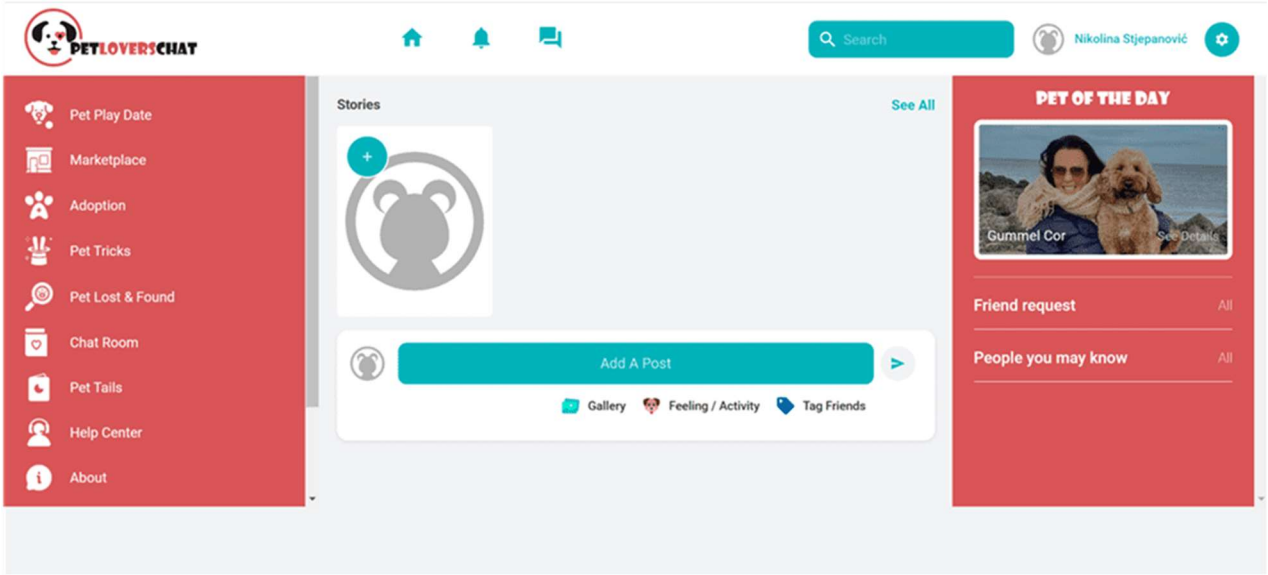
Sl. 2.1. Početni zaslon web aplikacije Yummypets

Korisnik sada može dodati svog kućnog ljubimca, pri čemu se traže informacije poput imena, pasmine i fotografije kućnog ljubimca, a nakon dodavanja može unijeti dodatne informacije poput opisa, najdraže hrane, najdraže igračke i drugih. Na profilu je moguće dodavati objave, kreirati

albume, napisati priču ljubimca te vidjeti popis praćenih korisnika i pridruženih timova. Osim praćenja korisnika, mogu se pregledavati njihove objave, označiti ih sa sviđanjem te ostavljati komentari. Moguće su i donacije te sudjelovanje na raznim natjecanjima. Yummypets ne sadrži izravnu komunikaciju poput chata, te je jedini oblik komunikacije putem foruma. Nasuprot tome, aplikacija razvijena u ovom radu implementira privatne poruke, što povećava povezanost korisnika i omogućuje bržu i učinkovitiju razmjenu informacija. Iako nudi manje mogućnosti u usporedbi s Yummypets, fokus razvijene aplikacije je na jednostavnosti korištenja i funkcionalnosti, a mogućnost ocjenjivanja korisničkog iskustva može pridonijeti daljnjem poboljšanju aplikacije.

2.2. PetLovers Chat

PetLovers Chat [2] je društvena mreža za vlasnike kućnih ljubimaca za koju je odmah potrebna prijava. Nakon prijave, korisniku se otvara zaslon na kojem može dodati sliku profila, naslovnu fotografiju te upisati osnovne informacije o svom kućnom ljubimcu. Na profilu korisnik može vidjeti svoje objave te popis prijatelja. Objave se dodaju na početnoj stranici koja je prikazana na slici 2.2.



Sl. 2.2. Početna stranica aplikacije Petloverschat

Prilikom dodavanja objave moguće je označiti prijatelje te dodati osjećaj, a ostali korisnici mogu označiti objavu sa sviđanjem, komentirati objavu te podijeliti istu. Postoji i mogućnost dodavanja priče, koja se može dodati kao fotografija ili videozapis uz koji se može dodati i opis. Svakog dana pojavljuje se novi ljubimac označen kao ljubimac dana, što je vidljivo na desnoj strani zaslona

prikazanog na slici 2.2., na čiji klik se otvara izvorna objava korisnika. Na lijevoj strani zaslona vidljive su brojne opcije koje nisu funkcionalne, kao ni poruke i obavijesti vidljive na navigacijskoj traci. Izrađena aplikacija sadrži funkcionalnu razmjenu poruka te primanje obavijesti, omogućujući korisnicima informiranje o ključnim događajima, poput novih pratitelja, komentara i oznaka sviđanja. Upravo zbog velikog broja nefunkcionalnog sadržaja unutar PetLoversChat, izrađena aplikacija omogućuje korisnicima bolje korisničko iskustvo.

2.3. United Pets

United Pets [3] je web aplikacija za vlasnike kućnih ljubimaca koja korisnicima pruža razne mogućnosti. Prije same prijave u aplikaciju, korisnik može vidjeti tko je na mreži, posljednje aktivnosti u aplikaciji, ljubimca tjedna. Može čitati blogove, forum, objave iz klubova, pregledavati fotografije i videozapise, rođendane, nove korisnike aplikacije kao i nedavno dodane ljubimce. Nakon prijave u aplikaciju korisniku se otvara zaslon prikazan na slici 2.3.

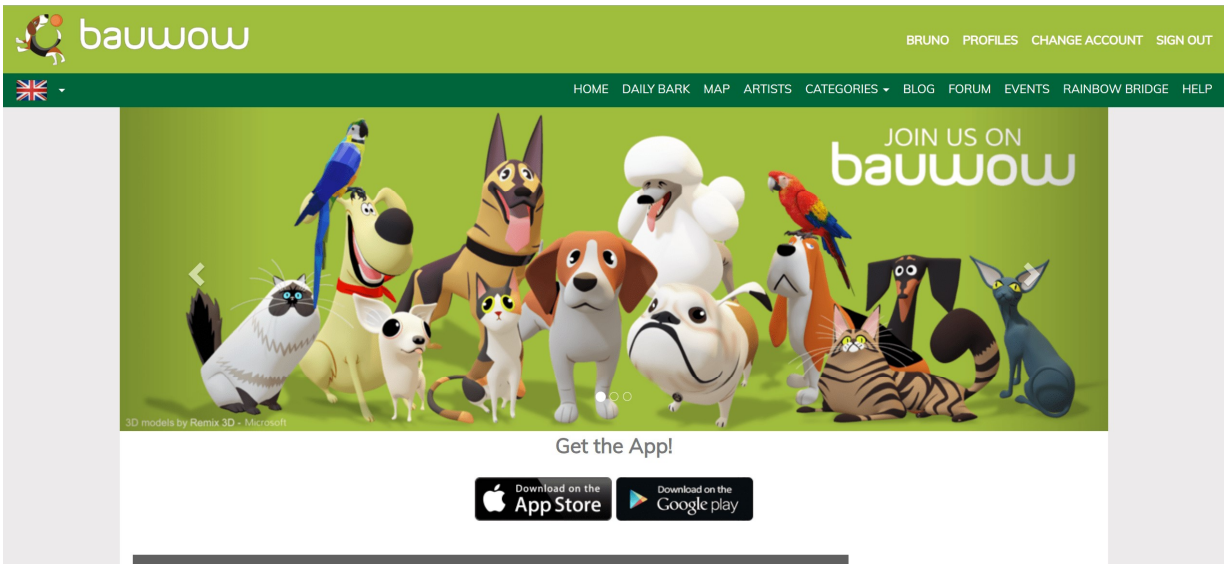
Sl. 2.3. Početni zaslon aplikacije United Pets

Osim navedenih funkcionalnosti, korisnik sada može pretraživati druge korisnike i ljubimce, dodavati objave, te komentirati objave drugih korisnika. Na profilu također može dodavati objave, dodati kućnog ljubimca, ali i kreirati blog, otvoriti klub, vidjeti popis svojih prijatelja i drugo. Moguće je i razmjenjivati poklone, a postoji i poseban dio posvećen preminulim ljubimcima. Komunikacija se odvija preko chata, a omogućeno je i slanje zahtjeva za prijateljstvo. United Pets, za razliku od izrađene aplikacije, ne sadrži personalizirane obavijesti, nego se obavijesti odnose

na sve korisnike unutar aplikacije. Osim toga, izrađena aplikacija jednostavnija je za korištenje, a unatoč tome što United Pets sadrži brojne korisne značajke, djeluje neatraktivno zbog previše sadržaja što može umanjiti korisničko iskustvo.

2.4. Bauwow

Bauwow [4] je web aplikacija za vlasnike kućnih ljubimaca koju je moguće preuzeti i u mobilnom obliku preko Google Playa ili Apple Storea. Neprijavljeni korisnik ima ograničene mogućnosti poput čitanja bloga, pregleda veterinarskih klinika, skloništa za životinje, parkova i brojnih drugih mjesta na karti, te pregleda stranice posvećene preminulim ljubimcima. Nakon uspješno obavljene registracije u aplikaciju, kreira se račun u ime kućnog ljubimca, nakon čega se otvara zaslon kao na slici 2.4.

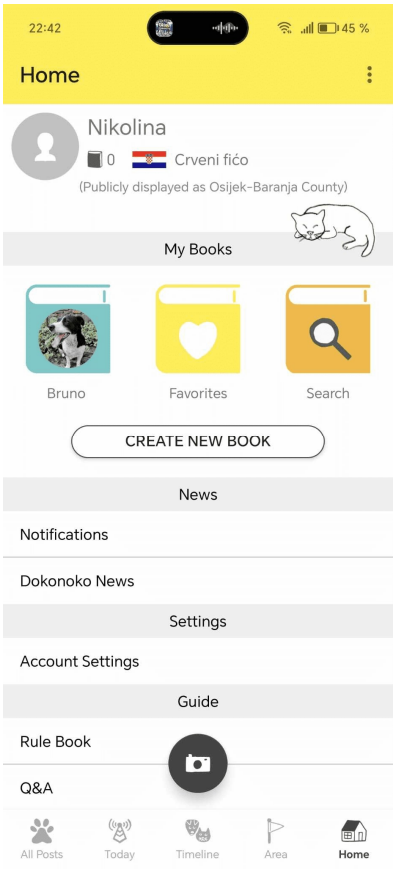


Sl. 2.4. Početni zaslon aplikacije Bauwow

Na profilu korisnik može kreirati objavu, pri čemu može odabrati želi li da objava bude javna ili vidljiva prijateljima, pregledavati svoje fotografije, značke, koje može dobiti interakcijom s drugim korisnicima i drugo. Također, moguće je predložiti mjesta i aktivnosti za koje korisnik želi prikazati na karti kako bi ih i drugi korisnici mogli pregledavati. Objave drugih korisnika moguće je komentirati i označiti sa sviđanjem. Komunikacija se odvija putem foruma, te ne postoji izravna komunikacija kao unutar izrađene aplikacije. Također, pri dodavanju novog ljubimca kreira se novi račun za svakog ljubimca koji se može mijenjati na profilu, dok unutar izrađene aplikacije svi ljubimci se dodaju pod jedan račun, što omogućuje jednostavno upravljanje i pregled ljubimaca s jednog mjesta.

2.5. Dokonoko

Dokonoko [5] mobilna je aplikacija namijenjena vlasnicima pasa i mačaka koja omogućuje korisnicima objavljivanje fotografija grupiranih po knjigama. Ukoliko korisnik želi objaviti fotografiju svog ljubimca, potrebno je prije toga kreirati knjigu za njega. Knjiga se kreira na početnom zaslonu aplikacije vidljivom na slici 2.5., klikom na gumb *Create new book* nakon čega je potrebno odabrati fotografiju, unijeti ime ljubimca te odabrati je li riječ o psu ili mački. Nakon objavljivanja fotografije, fotografija se dodaje u knjigu te na zaslon označen sa *All posts* u navigaciji. Fotografije drugih korisnika mogu se komentirati i označiti sa sviđanjem, a korisnike je moguće pratiti te pregledavati njihove knjige. Također je moguće dodati lokaciju, te na karti pronaći druge ljubimce koji se nalaze u blizini. Ne postoji nikakav oblik komunikacije, kao ni informacije da korisnik ima novu obavijest, što znači da ih korisnik stalno treba provjeravati. Izrađena aplikacija pruža mogućnost razmjene poruka, a pri primanju nove obavijesti korisnika se odmah informira o novim aktivnostima unutar aplikacije. Osim toga, dodavanje novih ljubimaca je praktičnije unutar izrađene aplikacije, jer se osigurava da su svi ljubimci na jednom mjestu, dok se unutar Dokonoko aplikacije svaki dodaje u zasebnu knjigu.



Sl. 2.5. Početni zaslon aplikacije Dokonoko

3. ZAHTJEVI ZA APLIKACIJU

Funkcionalni i nefunkcionalni zahtjevi dvije su temeljne kategorije zahtjeva u razvoju softvera. U ovom poglavlju navedeni su svi ključni funkcionalni i nefunkcionalni zahtjevi za razvoj web aplikacije. Osim toga, prikazan je i opisan arhitekturalni dizajn aplikacije.

3.1. Funkcionalni zahtjevi

Funkcionalni zahtjevi odnose se na značajke i funkcionalnosti koje sustav treba pružiti kako bi zadovoljio potrebe korisnika i omogućio im obavljanje njihovih zadataka [6]. Općenito, oni opisuju ponašanje sustava u različitim situacijama i uvjetima, uključujući što sustav treba raditi, kako treba reagirati na korisničke ulaze, obraditi podatke i prikazati rezultate. Funkcionalni zahtjevi koje izrađena aplikacija obuhvaća su:

- Registracija i prijava korisnika
- Promjena zaporke u slučaju zaborava
- Pregled objava drugih korisnika
- Označavanje objava sa sviđanjem
- Pregled oznaka sviđanja na objavama
- Komentiranje objava i brisanje komentara
- Pregled komentara
- Dodavanje, izmjena i brisanje slike profila
- Pregled slike profila unutar modala
- Dodavanje, uređivanje i brisanje ljubimca
- Dodavanje i brisanje objava
- Pretraživanje korisnika
- Praćenje korisnika
- Pregled pratitelja i praćenih osoba
- Komunikacija između korisnika (privatne poruke)
- Sustav obavijesti
- Ocjenjivanje korisničkog iskustva
- Statistička obrada rezultata koju obavlja administrator

3.2. Nefunkcionalni zahtjevi

Nefunkcionalni zahtjevi nisu povezani s funkcionalnošću sustava, već definiraju kako sustav treba raditi. Ključni su za osiguranje upotrebljivosti, pouzdanosti i učinkovitosti sustava, te često imaju značajan utjecaj na ukupno korisničko iskustvo [6]. Nefunkcionalni zahtjevi koje izrađena aplikacija obuhvaća su:

- Performanse
- Sigurnost
- Dostupnost
- Upotrebljivost
- Održavanje i nadogradnja
- Kompatibilnost

U smislu performansi, vrijeme učitavanja stranice treba biti manje od tri sekunde. Angular aplikacija treba optimizirati korištenje resursa, uključujući odgođeno učitavanje[1] (engl. *lazy loading*) modula za brže učitavanje, dok Firebase baza podataka treba omogućiti ažuriranje podataka u stvarnom vremenu s minimalnim kašnjenjem. U smislu sigurnosti, autentikacija korisnika putem Firebase Authentication treba biti osigurana korištenjem Oauth 2.0, a Angular Auth Guard treba preusmjeriti neautorizirane korisnike na stranicu za prijavu. S aspekta dostupnosti, aplikacija treba biti dostupna 99,9% vremena. Treba koristiti infrastrukturu temeljenu na oblaku (engl. *cloud-based*), što je omogućeno korištenjem Firebase Hosting. Upotrebljivost znači da aplikacija treba imati responzivan dizajn, prilagođavajući se različitim veličinama ekrana i uređajima, dok korisničko sučelje treba biti intuitivno i jednostavno za korištenje, bez zahtijevanja dodatnih uputa za korištenje. Za održavanje i nadogradnju aplikacije, potrebno je osigurati modularnu arhitekturu. Aplikacija treba biti dizajnirana korištenjem Angular modula i servisa, čime se omogućuje laka promjena i nadogradnja pojedinih dijelova bez utjecaja na cijeli sustav. Kompatibilnost označava da aplikacija treba biti kompatibilna sa svim popularnim web preglednicima te pružati podršku za različite operacijske sustave [8]. Navedeni zahtjevi osiguravaju sigurnost i stabilnost aplikacije, kao i njezinu dugoročnu održivost i prilagodbu potrebama korisnika.

[1] Odgođeno učitavanje učitava NgModule prema potrebi, čime se smanjuje vrijeme učitavanja. U slučaju ne korištenja odgođenog učitavanja, učitaju se svi NgModuli, bez obzira jesu li oni odmah potrebni ili ne [7]

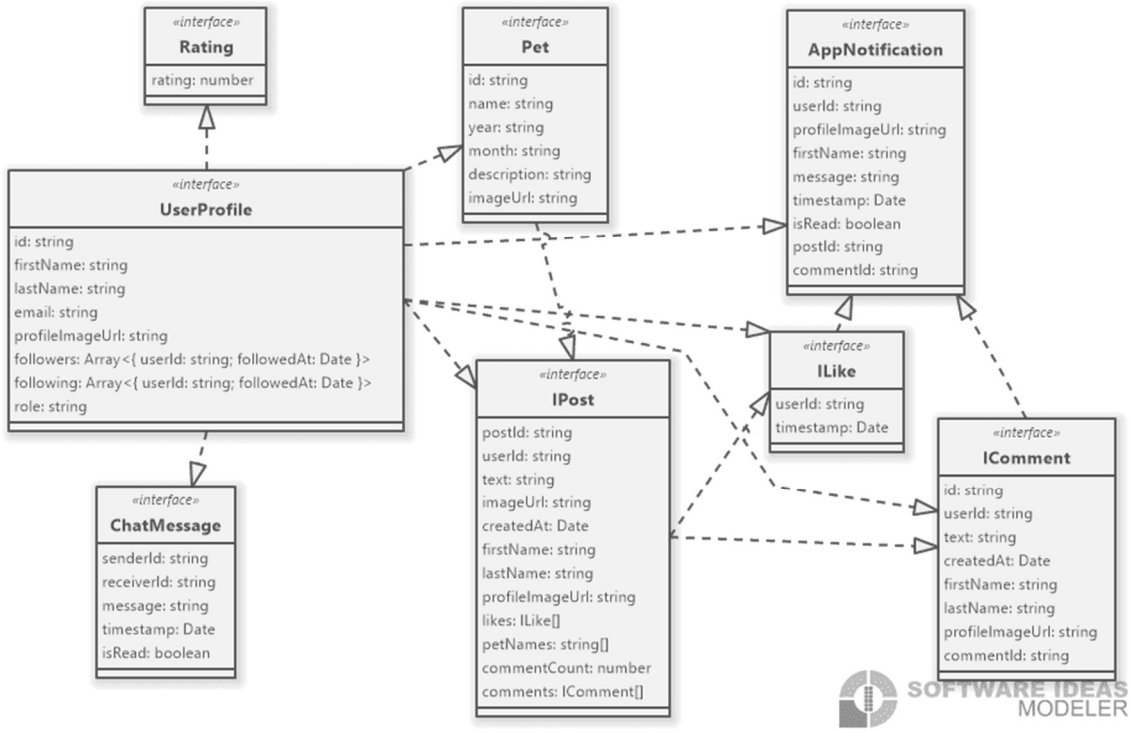
3.3. Arhitekturalni dizajn aplikacije

Izrađena aplikacija klijentski je orijentirana i razvijena korištenjem Angulara i Firebasea. Angular je JavaScript okvir otvorenog koda napisan u TypeScriptu. Održava ga Google, a njegova primarna svrha je razvoj jednostraničkih aplikacija (engl. *Single Page Application, SPA*) [9]. Takve aplikacije sastoje se od jednog HTML (engl. *Hypertext Markup Language*) dokumenta koji se pokreće u pregledniku i ne treba ga ponovno učitavati svaki put kada korisnik stupi u interakciju sa stranicom. Prilikom odgovaranja na događaj, preuzima se odgovarajući sadržaj s poslužitelja i dinamički ažurira određeni dio korisničkog sučelja, umjesto da se sve generira od nule. Neki od primjera jednostranačkih aplikacija su Facebook, X, Gmail i Github [10]. Uz Angular, korišten je i Angular Material, biblioteka komponenti posebno dizajnirana za AngularJS programere. Ova biblioteka pomaže u strukturiranju aplikacija pružajući unaprijed definirane komponente koje omogućuju izradu privlačnih, dosljednih i funkcionalnih web stranica i aplikacija [11]. Angular slijedi MVC (engl. *Model-View-Controller*) arhitekturu, čime se održava jasna razlika između podataka (engl. *Model*), korisničkog sučelja (engl. *View*) i logike (engl. *Controller*) [12]. S druge strane, Firebase je usluga pozadinskog sustava[2] (engl. *Backend-as-a-Service*), a predstavlja Googleovu platformu za razvoj mobilnih i web aplikacija. Kategoriziran je kao NoSQL (engl. *not only SQL*) baza podataka, koja pohranjuje podatke u dokumente slične JSON-u (engl. *JavaScript Object Notation*) [14], pri čemu se dokumenti organiziraju u kolekcije. U nastavku je opisana struktura baze podataka izrađene web aplikacije, gdje svaka kolekcija odgovara specifičnom entitetu unutar aplikacije i predstavlja logičku grupu podataka.

3.3.1. Struktura baze podataka

Na slici 3.1. prikazana je shema Firestore baze podataka koju koristi izrađena aplikacija. Shema baze podataka izrađena je korištenjem alata Software Ideas Modeler Ultimate, koji se koristi za modeliranje, dizajn i vizualizaciju baza podataka.

[2] Model usluge u oblaku koji pruža gotove backend komponente i infrastrukturu potrebne za razvoj i upravljanje aplikacijama, što omogućuje programerima da se usredotoče na razvoj korisničkog sučelja i poslovne logike bez potrebe za održavanjem backend infrastrukture [13]



Sl. 3.1. Shema baze podataka izrađene web aplikacije

Kolekcija *UserProfile*, prikazana na shemi, predstavlja glavni entitet baze, zadužen za pohranu osnovnih podataka o korisnicima. Ona pohranjuje atribute kao što su korisnički identifikator, ime, prezime, adresu e-pošte, sliku profila, a osim toga, ova kolekcija pohranjuje i polja za praćenje korisnika. Polja *followers* i *following* pohranjuju niz objekata koji uključuju identifikatore korisnika te vrijeme kada je akcija praćenja izvršena. Kolekcija *Pet* omogućava pohranu podataka o kućnim ljubimcima korisnika, uključujući njihovo ime, godinu i mjesec rođenja te opis. Svaki ljubimac može biti povezan s korisnikom putem korisničkog profila ili objava. Kolekcija *IPost* služi za pohranu objava korisnika. Svaka objava povezana je s korisnikom putem polja *userId*, a dodatno sadrži informacije poput teksta objave, URL-a (engl. *Uniform Resource Locator*) slike objave te vremena kreiranja objave. Kolekcija je povezana s ostalim entitetima kroz polja *likes* i *comments*, koja pohranjuju reference na kolekcije *ILike* i *IComment*. *IComment* kolekcija pohranjuje podatke o komentarima. Komentar je povezan s objavom putem polja *postId*, a s korisnikom koji ga je napisao pomoću polja *userId*. Svaki komentar sadrži dodatne informacije kao što su tekst komentara, ime i prezime korisnika, URL slike profila te vrijeme kada je komentar objavljen. Kolekcija *ILike* prati oznake sviđanja na objavama. Svaki zapis sadrži *userId* korisnika koji je označio objavu sa sviđanjem te vrijeme kada je označena. Kolekcija *AppNotification* služi za pohranu obavijesti koje korisnici dobivaju unutar aplikacije. Svaka obavijest povezana je s korisnikom putem polja *userId* te može sadržavati poruku, vrijeme obavijesti i status je li obavijest

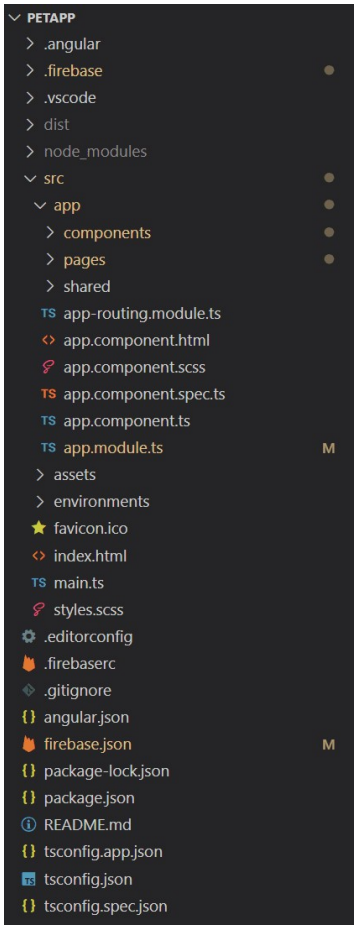
pročitana. Također, obavijest može biti povezana s objavom ili komentarom putem polja *postId* i *commentId*, čime se omogućava informiranje korisnika svaki put kada netko komentira njegovu objavu ili je označi sa sviđanjem. Kolekcija *ChatMessage* pohranjuje podatke o porukama razmijenjenim između korisnika. Svaka poruka uključuje korisnički identifikator pošiljatelja i primatelja, sadržaj poruke, vrijeme slanja te status je li poruka pročitana. Kolekcija *Rating* služi za praćenje ocjena korisnika, a s kolekcijom *UserProfile* je povezana preko korisničkog identifikatora koji odgovara nazivu dokumenta u koji se spremaju ocjene.

4. IZRADA WEB APLIKACIJE

U ovom poglavlju prikazana je i opisana struktura izrađene web aplikacije. Opisana je izrada osnovnih funkcionalnosti aplikacije te su prikazani primjeri koda. Također je opisan način objavljivanja aplikacije na Firebase Hosting.

4.1. Struktura web aplikacije

Na slici 4.1. prikazana je struktura izrađene web aplikacije. Mapa *.angular* sadrži cache i druge konfiguracijske datoteke koje Angular CLI (engl. *Angular Command-Line Interface*) koristi prilikom pokretanja aplikacije, testiranja ili pokretanja *npm run build* naredbe. Unutar mape *.firebase* nalaze se interne datoteke koje omogućuju praćenje i upravljanje integracijom s Firebase servisima. Mapa *dist* generira se prilikom pokretanja *npm run build* naredbe, a koristi se za implementaciju aplikacije na web poslužitelj, odnosno Firebase Hosting. Mapa *src* glavna je mapa projekta u kojoj se nalazi izvorni kod cijele aplikacije. Angular komponente, koje su osnovni gradivni blokovi aplikacije nalaze se unutar mape *components*. Svaka od njih nakon kreiranja sastoji se od četiri osnovne datoteke: TypeScript datoteke za logiku, HTML datoteke za prikaz, SCSS datoteke za stilove i datoteke za testiranje. Radi bolje organizacije i grupiranja pojedinih dijelova aplikacije svakoj komponenti dodan je Angular modul, a pojedine komponente koje komuniciraju s bazom sadrže servise. Neki primjeri komponenti su navigacijska traka, objave, dijalozi za dodavanje ljubimca i komentara. Mapa *pages* sadrži komponente koje predstavljaju zaslone aplikacije, poput korisničkog profila, poruka ili obavijesti. Unutar *shared* nalaze se Angular guards, koji služe za ograničavanje pristupa korisnicima koji nemaju određena prava. Primjerice, ukoliko korisnik koji nije administrator pokuša pristupiti stranici za prikaz statistike, preusmjeriti će se na početnu stranicu aplikacije. Unutar *app-routing.module.ts* datoteke definiraju se putanje do različitih dijelova aplikacije. Datoteka *app.component.ts* predstavlja glavnu komponentu aplikacije, dok *app.module.ts* predstavlja glavni modul aplikacije. Mapa *environments* omogućuje definiranje različitih konfiguracija ovisno o okruženju u kojem se aplikacija pokreće. Primjerice, aplikacija može imati različitu Firebase konfiguraciju za razvoj i za produkciju. Datoteka *angular.json* glavna je konfiguracijska datoteka koja definira ključne postavke projekta, poput naziva projekta, direktorija izvornog koda, načina na koji se aplikacija kompajlira i drugih. Datoteka *firebase.json* glavna je konfiguracijska datoteka za Firebase servise, koja definira postavke i ponašanje prilikom objavljivanja aplikacije. Treba još istaknuti *package.json* datoteku u kojoj se nalaze sve informacije o instaliranim npm paketima te osnovni podaci o projektu poput imena, verzije, autora i drugih.



Sl. 4.1. Struktura web aplikacije

4.2. Registracija i prijava korisnika

Za registraciju i prijavu korisnika u aplikaciju koristi se Firebase Authentication. Prije samog korištenja potrebno je instalirati Firebase SDK (engl. *Firebase Software Development Kit*) unutar projekta, dodati Firebase konfiguraciju za različita okruženja te inicijalizirati Firebase i Firebase Authentication korištenjem AngularFireModule i AngularFireAuthModule. Unutar Firebase Authentication potrebno je definirati metodu prijave, a za ovaj projekt odabrana je prijava pomoću adrese e-pošte i zaporke te Google računa.

4.2.1. Registracija korisnika

Na slici 4.2. prikazana je metoda za registraciju korisnika definirana unutar servisa. Prilikom registracije, korisnik unosi ime, prezime, adresu e-pošte i zaporku u odgovarajuća polja unutar forme. Kada korisnik popuni potrebne podatke i pošalje zahtjev za registracijom, metoda *createUserWithEmailAndPassword* kreira novi korisnički račun na temelju unesenih podataka. Nakon uspješne registracije, Firebase vraća objekt korisnika koji sadrži osnovne informacije o

korisniku, uključujući jedinstveni identifikator. Ovaj identifikator koristi se za pohranu korisničkih podataka unutar Firestore baze podataka kako bi se omogućila njihova kasnija upotreba unutar aplikacije. Također, identifikator se pohranjuje unutar *localStorage*[3], a korisnika se preusmjerava na početnu stranicu aplikacije.

```
async signUp(
  email: string,
  password: string,
  firstName: string,
  lastName: string
): Promise<any> {
  try {
    this.isRegistering = true;
    const result = await this.afAuth.createUserWithEmailAndPassword(
      email,
      password
    );

    const user = result.user;

    if (user) {
      await this.firestore.collection('users').doc(user.uid).set({
        firstName,
        lastName,
        email,
        id: user.uid,
      });

      localStorage.setItem('accessToken', user.uid);
      this.isRegistering = false;
      this.router.navigate(['/']);
    }

    return result;
  } catch (error) {
    console.error('Error registering:', error);
    this.isRegistering = false;
    this.snackBar.open(
      'The email address is already in use by another account.',
      'OK',
      {
        duration: 5000,
      }
    );
    throw error;
  }
}
```

Sl. 4.2. Metoda za registraciju korisnika unutar servisa

Unutar .ts dokumenta obavlja se provjera unesenih podataka, a sve dok korisnik ne unese ispravne podatke, oni se neće slati Firebase servisu. Na slici 4.3. prikazana su pravila validacije definirana unutar *ngOnInit()* metode gdje je vidljivo da su sva polja obavezna, te da ime i prezime trebaju imati najmanje dva znaka, adresa e-pošte treba biti u ispravnom formatu, a zaporka treba imati najmanje 8 znakova te treba sadržavati veliko slovo, malo slovo i brojku.

[3] Lokalna pohrana, omogućuje spremanje parova ključ/vrijednost u pregledniku. Pohranjuje podatke bez datuma isteka. Podaci se ne brišu kada se preglednik zatvori i dostupni su za buduće sesije [15].

```
ngOnInit(): void {
  this.form = this.formBuilder.group({
    firstname: [
      '',
      [Validators.required, Validators.pattern('^[a-zA-ZČčĆćĐđŽžŠš]{2,}$')],
    ],
    lastname: [
      '',
      [Validators.required, Validators.pattern('^[a-zA-ZČčĆćĐđŽžŠš]{2,}$')],
    ],
    email: [
      '',
      [
        Validators.required,
        Validators.pattern(/^[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,4}$/),
      ],
    ],
    password: [
      '',
      [
        Validators.required,
        Validators.minLength(8),
        Validators.pattern(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d).*$/),
      ],
    ],
  });
}
```

Sl. 4.2. Validacija unesenih podataka unutar .ts dokumenta

4.2.2. Prijava korisnika

Slika 4.3. prikazuje metodu za prijavu korisnika definiranu unutar servisa. Prilikom prijave, korisnik unosi adresu e-pošte i zaporku unutar forme. Nakon što popuni potrebne podatke i pošalje zahtjev za prijavom, pomoću signInWithEmailAndPassword metode, Firebase Authentication verificira unesene podatke. Ako su podaci ispravni, korisnikov jedinstveni identifikator pohranjuje se u *localStorage*, a korisnik se automatski preusmjerava na početnu stranicu aplikacije. Kao i pri registraciji, podaci koje korisnik unosi prvo trebaju proći validaciju prije nego se pošalju servisu.

```
async signIn(email: string, password: string): Promise<any> {
  this.isLoggingIn = true;
  try {
    const result = await this.afAuth.signInWithEmailAndPassword(
      email,
      password
    );

    localStorage.setItem('accessToken', result.user?.uid);
    this.isLoggingIn = false;
    this.router.navigate(['/']);

    return result;
  } catch (error) {
    console.error('Error signing in:', error);
    this.isLoggingIn = false;
    this.snackBar.open(
      'Error when entering email or password. Please try again.',
      'OK',
      {
        duration: 5000,
      }
    );
    throw error;
  }
}
```

Sl. 4.3. Metoda za prijavu korisnika definirana unutar servisa

Prilikom prijave pomoću Google računa, korisnik aktivira direktivu prikazanu na slici 4.4. koja pokreće metodu *signInWithGoogle()* definiranu unutar servisa. Ova metoda koristi Google Oauth 2.0 sustav za autentikaciju putem skočnog prozora (*signInWithPopup*), čime se korisniku omogućuje siguran pristup aplikaciji putem vlastitog Google računa. Nakon uspješne autentikacije, aplikacija dohvaća korisničke podatke i provjerava postoji li već u Firestore bazi podataka zapis za korisnika. Ukoliko ne postoji, kreira se novi dokument s podacima.

```
import { Directive, HostListener } from '@angular/core';
import { AuthenticationService } from '../services/authentication.service';

@Directive({
  selector: '[googleSso]',
})
export class GoogleSsoDirective {
  constructor(private authService: AuthenticationService) {}

  @HostListener('click')
  async onClick() {
    try {
      await this.authService.signInWithGoogle();
    } catch (error) {
      console.error('Google sign-in error:', error);
    }
  }
}
```

Sl. 4.4. Direktiva za prijavu korisnika putem Google računa

4.3. Korisnički profil

Informacije o korisnicima pohranjene unutar Firestore baze podataka korištene su na profilu kako bi se prikazalo ime i prezime korisnika. Osim imena, radi bolje prepoznatljivosti među korisnicima, omogućeno je i dodavanje slike profila čime se dodaje novo polje u kolekciju korisnika. Na profilu također, korisnici mogu dodavati informacije vezane za svoje ljubimce te objave, pri čemu se svaka od njih sprema u zasebnu kolekciju.

4.3.1. Slika profila

Nakon što korisnik odabere sliku profila, ona se obrađuje u metodi *onImageSelected()*. Unutar metode provjerava se ispravnost odabrane datoteke. Ako je datoteka ispravna, selectedImage se postavlja na odabranu datoteku te se poziva *uploadProfileImage()* metoda prikazana na slici 4.5. U metodi *uploadProfileImage()*, dohvaća se korisnički identifikator iz *localStorage*. Ako identifikator postoji te ako je slika odabrana, poziva se metoda *uploadImage()* iz servisa koja prenosi sliku na Firebase Storage. Ova metoda vraća Observable koji se preplaćuje za praćenje statusa prijenosa. Kada prijenos završi, *uploadImage()* vraća URL slike koji se prosljeđuje metodi *saveImageUrl()*. Metoda *saveImageUrl()* ažurira korisnički profil u Firestore bazi podataka, pohranjujući novi URL slike kao dio korisnikovih podataka.

```
uploadProfileImage() {
  this.isUploading = true;
  const userId = localStorage.getItem('accessToken');
  if (this.selectedImage && userId) {
    this.userService.uploadImage(this.selectedImage, userId).subscribe(
      (imageUrl: string) => {
        this.userService.saveImageUrl(userId, imageUrl).then(() => {
          console.log('Profile image uploaded successfully');
          this.snackBar.open('Profile image uploaded successfully.', 'OK', {
            duration: 5000,
          });
          this.loadUserProfile(userId);
          this.isUploading = false;
        });
      },
      (error) => {
        console.error('Error uploading profile image:', error);
        this.snackBar.open(
          'Error while trying to upload profile image. Please try again.',
          'OK',
          {
            duration: 5000,
          }
        );
      }
    );
  } else {
    console.error('No image selected or user ID missing');
  }
}
```

Sl. 4.5. Metoda uploadProfileImage() za dodavanje slike profila

Odabranu sliku moguće je prikazati u uvećanom prikazu korištenjem modala. Klikom na sliku, poziva se metoda *openModal()* definirana na slici 4.6. Metoda *openModal()* preuzima referencu na modalni prozor i sliku unutar tog modala. Postavlja *modal.style.display* na *block*, što prikazuje modalni prozor na zaslonu te postavlja URL slike na src atribut slike, što omogućava uvećani prikaz slike unutar modalnog prozora. Modal je moguće zatvoriti klikom na bilo koje mjesto unutar modala (izvan slike) ili klikom na ikonu X, čime se poziva metoda *closeModal()* također prikazana na slici 4.6.

```
openModal(imageUrl: string): void {
  const modal = document.getElementById('myModal') as HTMLElement;
  const modalImg = document.getElementById('img01') as HTMLImageElement;
  modal.style.display = 'block';
  modalImg.src = imageUrl;

  modal.onclick = () => {
    modal.style.display = 'none';
  };
}

closeModal(): void {
  const modal = document.getElementById('myModal') as HTMLElement;
  modal.style.display = 'none';
}
```

Sl. 4.6. Metode za otvaranje i zatvaranje modala

Korisnik može obrisati odabranu sliku profila pozivom metode *deleteImage()* prikazanom na slici 4.7. Prvo se otvara dijaloški okvir u kojem se korisnika pita želi li zaista obrisati sliku profila. Nakon zatvaranja dijaloškog okvira, ukoliko je korisnik potvrdio brisanje, dohvaća se korisnički identifikator iz *localStorage*. Nakon provjere o postojanju korisničkog identifikatora, podataka o korisniku i profilne slike, poziva se metoda *deleteProfileImage()* definirana unutar servisa koja briše profilnu sliku iz Firebase Storagea i ažurira Firestore bazu podataka postavljanjem polja *profileImageUrl* na *null*.

```
deleteImage() {
  const dialogRef = this.dialog.open(ConfirmDialogComponent, {
    width: '320px',
    height: 'auto',
    data: {
      message: `Are you sure you want to delete your profile image?`,
    },
  });

  dialogRef.afterClosed().subscribe((result) => {
    if (result) {
      const userId = localStorage.getItem('accessToken');
      if (userId && this.userProfile && this.userProfile.profileImageUrl) {
        this.userService
          .deleteProfileImage(userId, this.userProfile.profileImageUrl)
          .then(() => {
            console.log('Profile image deleted successfully');
            this.snackBar.open('Profile image deleted successfully.', 'OK', {
              duration: 5000,
            });
            this.loadUserProfile(userId);
          })
          .catch((error) => {
            console.error('Error deleting profile image:', error);
            this.snackBar.open(
              'Error while trying to delete profile image. Please try again.',
              'OK',
              {
                duration: 5000,
              }
            );
          });
      } else {
        console.error('User ID missing or profile image URL not available');
      }
    }
  });
}
```

Sl. 4.6. Metoda deleteImage() za brisanje slike profila

4.3.2. Dodavanje, uređivanje i brisanje ljubimca

Dodavanje ljubimca vrši se kroz dijaloški okvir u koji korisnik unosi informacije o ljubimcu, kao što su ime, godina i mjesec rođenja, opis i slika ljubimca. Po potvrdi unosa, aktivira se metoda *savePet()* prikazana na slici 4.7, koja prvo provjerava je li korisnik vlasnik ljubimca. Zatim dohvaća korisnički identifikator iz *localStorage* i priprema objekt *petData* s podacima o ljubimcu. Ako korisnik uređuje ljubimca, metoda poziva *updatePet()*, koja ažurira podatke o ljubimcu u Firestore bazi. Ako je nova slika priložena, učitava je u Firebase Storage i ažurira URL slike. Nakon toga, ažurira podatke o ljubimcu u Firestore bazi podataka s novim informacijama i URL-om slike. Ako korisnik dodaje novog ljubimca, metoda *savePet()* poziva metodu *addNewPet()*. Ova metoda koristi *addPetWithImage()* metodu iz servisa koja dodaje novog ljubimca u Firestore, generira jedinstveni identifikator za ljubimca, učitava sliku na Firebase Storage i sprema podatke o ljubimcu zajedno s URL-om slike u Firestore pod dokument specifičan za korisnika.

```
savePet() {
  if (!this.isOwner) {
    return;
  }

  this.isSaving = true;
  const accessToken = localStorage.getItem('accessToken');

  const petData = {
    id: this.petId,
    name: this.petName,
    year: this.petYear,
    month: this.petMonth,
    description: this.petDescription,
    image: this.selectedImage,
  };

  if (this.isEditing && this.petId) {
    this.updatePet(petData, accessToken);
  } else {
    this.addNewPet(petData, accessToken);
  }
}
```

Sl. 4.7. Metoda savePet() za uređivanje i dodavanje ljubimca

Brisanje ljubimca vrši metoda *deletePet()*, prikazana na slici 4.8. Nakon što korisnik potvrdi brisanje unutar dijaloškog okvira za potvrdu, dohvaća se korisnički identifikator iz *localStorage*. Zatim se poziva metoda *deletePet()* iz servisa koja prvo uklanja sliku ljubimca iz Firebase Storage, ako postoji, a potom briše podatke o ljubimcu iz Firestore baze podataka. Nakon uspješnog brisanja, korisniku se prikazuje poruka o uspjehu, a podaci o preostalim ljubimcima se ažuriraju.

```
deletePet(pet: Pet) {
  const dialogRef = this.dialog.open(ConfirmDialogComponent, {
    width: '320px',
    height: 'auto',
    data: {
      message: `Are you sure you want to delete ${pet.name}?`,
    },
  });

  dialogRef.afterClosed().subscribe((result) => {
    if (result) {
      const userId = localStorage.getItem('accessToken');
      if (userId) {
        this.petService
          .deletePet(userId, pet.id)
          .then(() => {
            this.snackBar.open('Pet deleted successfully.', 'OK', {
              duration: 5000,
            });
            this.getPets(userId);
          })
          .catch((error) => {
            console.error('Error deleting a pet:', error);
          });
      }
    }
  });
}
```

Sl. 4.8. Metoda deletePet() za brisanje ljubimca

4.3.3. Dodavanje i brisanje objava

Kod dodavanja objave, korisnik može unijeti tekst i/ili odabrati sliku unutar dijaloškog okvira. Osim toga, ako je prethodno dodao ljubimca, može odabrati na kojeg se ljubimca objava odnosi. Nakon što korisnik odabere sliku, poziva se metoda *onImageSelected()* koja provjerava valjanost slike i koristi *FileReader* za učitavanje i prikaz slike unutar korisničkog sučelja. Nakon klika na gumb za dodavanje objave unutar dijaloga, poziva se metoda *addPost()*. Prvo provjerava jesu li uneseni potrebni podaci. Ako jesu, pripremaju se podaci za korisnika te se poziva istoimena metoda iz servisa prikazana na slici 4.9. Unutar servisa preuzimaju se podaci iz objave te se generira jedinstveni identifikator za objavu. Ukoliko je odabrana slika, ona se pohranjuje u Firebase Storage te se generira URL slike. Nakon toga, metoda sprema sve podatke o objavi u Firestore bazu podataka, a nakon uspješnog dodavanja objave, komponenta prikazuje obavijest o uspjehu, zatvara dijalog i resetira dijalog za unos.

```
async addPost(
  text: string,
  imageFile: File | null,
  userId: string,
  firstName: string,
  lastName: string,
  profileImageUrl: string,
  petNames: string[]
) {
  const storage = getStorage();
  const postId = this.firestore.createId();

  try {
    let imageUrl: string | null = null;

    if (imageFile) {
      const filePath = `posts/${userId}/${imageFile.name}`;
      const fileRef = ref(storage, filePath);
      const snapshot = await uploadBytes(fileRef, imageFile);
      imageUrl = await getDownloadURL(snapshot.ref);
    }

    await this.firestore
      .collection('posts')
      .doc(userId)
      .collection('posts')
      .doc(postId)
      .set({
        userId: userId,
        text: text,
        imageUrl: imageUrl,
        postId: postId,
        createdAt: new Date(),
        firstName: firstName,
        lastName: lastName,
        profileImageUrl: profileImageUrl,
        likes: [],
        petNames: petNames,
      });

    return 'Post added successfully!';
  } catch (error) {
    console.error('Error adding post with image:', error);
    throw error;
  }
}
```

Sl. 4.9. Metoda addPost() definirana unutar servisa

Dodana objava sadrži padajući izbornik u kojem se nalazi opcija za brisanje objava. Klikom na opciju, poziva se metoda *deletePost()* koja otvara dijalog za potvrdu brisanja. Nakon potvrde, poziva se metoda *deletePost()* iz servisa prikazana na slici 4.10. U servisu se prvo pokušava obrisati slika povezana s objavom iz Firebase Storagea, ukoliko ona postoji. Zatim se poziva metoda *deleteAllPostNotifications()*, koja briše sve obavijesti vezane za određenu objavu iz kolekcije obavijesti u Firestoreu. Nakon što su svi povezani podaci uklonjeni, pristupa se brisanju same objave iz kolekcije objava u Firestoreu, korištenjem odgovarajućeg *postId*. Ako su svi koraci uspješno izvedeni, korisniku se vraća potvrda o uspješnom brisanju, a u slučaju greške, sustav hvata iznimke i ispisuje odgovarajuću poruku.

```
async deletePost(userId: string, postId: string): Promise<string> {
  const storage = getStorage();
  const filePath = `posts/${userId}/${postId}`;
  const fileRef = ref(storage, filePath);

  try {
    await deleteObject(fileRef);
  } catch (error) {
    console.error('Error deleting image file:', error);
  }

  try {
    await this.deleteAllPostNotifications(userId, postId);

    await this.firestore
      .collection('posts')
      .doc(userId)
      .collection('posts')
      .doc(postId)
      .delete();

    return 'Post deleted successfully!';
  } catch (error) {
    console.error('Error deleting post:', error);
    throw error;
  }
}
```

Sl. 4.9. Metoda deletePost() definirana unutar servisa

4.4. Interakcija s drugim korisnicima

Interakcija među korisnicima predstavlja ključni aspekt svake društvene mreže, omogućujući korisnicima međusobnu komunikaciju i povezivanje kroz različite funkcionalnosti. U razvijenoj aplikaciji, korisnici mogu pretraživati i pratiti druge korisnike, pregledavati njihove objave, reagirati na sadržaj putem sviđanja, komentirati objave te slati privatne poruke, a u nastavku će biti opisana implementacija svake od navedenih.

4.4.1. Pretraživanje i praćenje korisnika

Pretraživanje korisnika implementirano je unutar navigacijske trake, čineći ovu funkcionalnost dostupnom s bilo kojeg zaslona. Omogućeno je pretraživanje prema imenu i prezimenu, a rezultati pretraživanja prikazuju se u realnom vremenu putem *mat-autocomplete* komponente. Prikaz rezultata uključuje profilnu sliku korisnika, ako je dostupna, te ime i prezime. U slučaju da profilna slika nije postavljena, prikazuje se zadana slika, čime se osigurava konzistentan izgled sučelja. Polje za pretraživanje povezano je s *FormControl*, koji prati promjene tijekom unosa korisnika. Svaka promjena u polju za pretraživanje aktivira *Observable valueChanges*, koji uz pomoć

switchMap operatora poziva metodu *searchUsers()* prikazanu na slici 4.10. Unutar metode *searchUsers()*, pretraga se izvodi u Firestore kolekciji korisnika te se rezultati filtriraju prema imenu i prezimenu koji odgovaraju unesenom upitu. Dohvaćeni korisnici potom se vraćaju kao Observable, čime se osigurava da se rezultati pretraživanja ažuriraju u stvarnom vremenu.

```
searchUsers(query: string): Observable<UserProfile[]> {
  return this.firestore
    .collection('users')
    .snapshotChanges()
    .pipe(
      map((actions) =>
        actions.map((a) => {
          const data = a.payload.doc.data() as UserProfile;
          const id = a.payload.doc.id;
          return { id, ...data };
        })
      ),
      map((users) =>
        users.filter((user) => {
          const userName = `${user.firstName || ''} ${user.lastName || ''}`;
          return userName.toLowerCase().includes(query.toLowerCase());
        })
      )
    );
}
```

Sl. 4.10. Metoda searchUsers() definirana unutar servisa

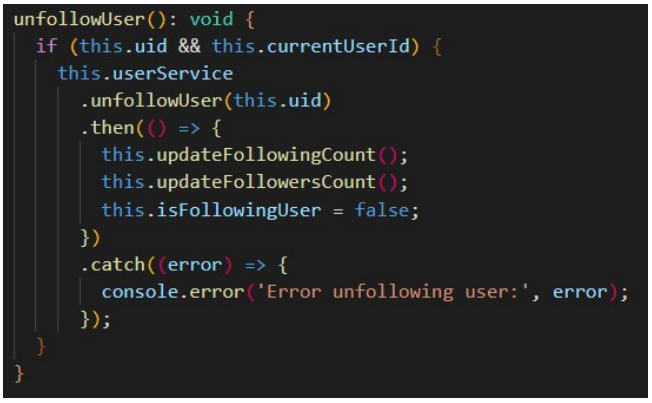
Klikom na nekog od ponuđenih korisnika unutar tražilice, Angular router preusmjerava korisnika na profil odabranog korisnika korištenjem njegovog jedinstvenog identifikatora unutar URL-a. U komponenti profila, metoda *ngOnInit(),* prikazana na slici 4.11., reagira na promjene u URL-u putem ActivatedRoute i dohvaća korisnički identifikator iz URL parametara. Nakon toga, pokreće se niz funkcija za učitavanje korisničkih podataka. Ovo uključuje dohvaćanje imena i prezimena, slike profila, te dodanih ljubimaca. Također se učitavaju broj pratitelja i broj osoba koje korisnik prati. Ovaj pristup omogućuje dinamičko prikazivanje ažuriranih informacija o korisniku odmah nakon navigacije na njegov profil.

Sl. 4.10. Metoda ngOnInit() definirana unutar komponente profila

Prikazanog korisnika može se zapratiti. Kada korisnik klikne na gumb za praćenje, poziva se metoda *followUser()* prikazana na slici 4.11. Metoda najprije provjerava prisutnost identifikatora trenutno prijavljenog korisnika i korisnika kojeg se prati, a ako su oba prisutna, poziva se istoimena metoda iz servisa. U servisu, metoda *followUser()* upravlja procesom praćenja tako da najprije provjerava postoji li identifikator trenutno prijavljenog korisnika u *localStorage*. Nakon toga, kreira se batch operacija za izvođenje promjena u bazi podataka u jednoj transakciji. Ove promjene uključuju ažuriranje podataka o korisnicima koji prate i koji su zapraćeni. Servis zatim dohvaća reference na dokumente korisnika u Firestore bazi podataka i provjerava jesu li podaci o korisnicima dostupni. Kreira se nova obavijest za korisnika kojeg se prati, korištenjem jedinstvenog identifikatora i relevantnih informacija kao što su profilna slika i ime. Ova obavijest obavještava korisnika da ga je netko zapratio. U batch operaciji se ažurira polje *following* trenutno prijavljenog korisnika, gdje se dodaje novi zapis s korisnikovim identifikatorom i vremenskom oznakom praćenja. Također, ažurira se polje *followers* korisnika kojeg se prati, gdje se dodaje zapis s identifikatorom trenutnog korisnika i vremenskom oznakom. Nakon uspješnog praćenja, ažurira se broj korisnika koje trenutni korisnik prati i broj pratitelja, te se isFollowingUser varijabla postavlja na true. Ova varijabla koristi se za praćenje stanja praćenja korisnika, čime se omogućava promjena gumba iz *Follow* u *Unfollow*.

Sl. 4.11. Metoda followUser() definirana unutar komponente

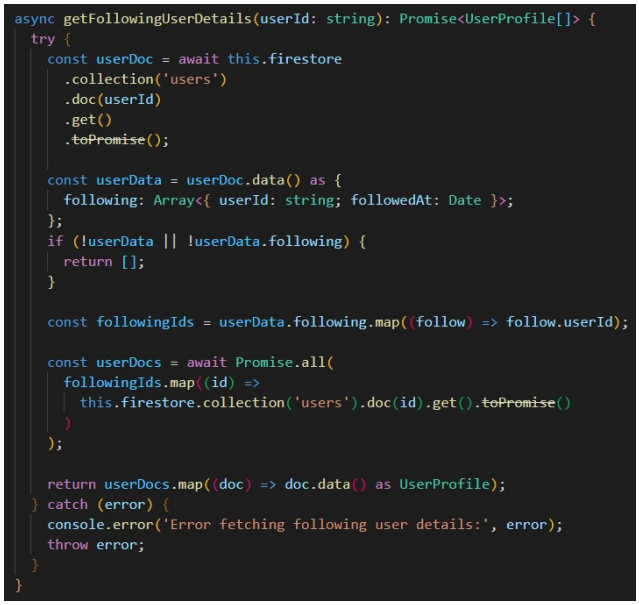
Kada se tekst gumba promijeni u *Unfollow*, ponovni klik na gumb poziva metodu *unfollowUser()* prikazanu na slici 4.12. Metoda najprije provjerava prisutnost identifikatora trenutno prijavljenog korisnika i korisnika kojeg se otpraćuje. Ako su oba identifikatora prisutna, poziva se istoimena metoda iz servisa. U servisu, metoda *unfollowUser()* upravlja procesom otpraćivanja tako da najprije provjerava postoji li identifikator trenutno prijavljenog korisnika u *localStorage*. Nakon toga, kreira se batch operacija za izvođenje svih potrebnih promjena u bazi podataka u jednoj transakciji. Prvi korak batch operacije uključuje dohvaćanje referenci na dokumente korisnika u Firestore bazi podataka. Metoda zatim provjerava jesu li podaci o trenutnom korisniku i korisniku kojeg se otpraćuje dostupni. U nastavku batch operacije, ažuriraju se dokumenti korisnika u bazi podataka tako da se iz polja *following* trenutno prijavljenog korisnika uklanja identifikator korisnika kojeg se otpraćuje, dok se iz polja *followers* korisnika kojeg se otpraćuje uklanja identifikator trenutno prijavljenog korisnika. Nakon ažuriranja polja *following* i *followers*, sljedeći korak batch operacije uključuje brisanje svih povezanih obavijesti u vezi s praćenjem iz kolekcije *notifications*. Metoda pretražuje obavijesti korisnika kojeg se otpraćuje, a zatim, ako pronađe obavijesti koje sadrže identifikator trenutno prijavljenog korisnika, te obavijesti se brišu. Nakon brisanja obavijesti, batch operacija se potvrđuje, što osigurava da su sve promjene u bazi podataka izvršene u jednoj transakciji. Nakon uspješno izvršene operacije otpraćivanja, ažuriraju se brojevi pratitelja i praćenja, varijabla *isFollowingUser* se postavlja na false, a gumb se ponovno mijenja u *Follow*.



Sl. 4.12. Metoda unfollowUser() definirana unutar komponente

4.4.2. Pregled pratitelja i praćenih osoba

Korisnik na profilu ima mogućnost pregledavanja pratitelja i praćenih osoba, a za to su zaslužne komponente *FollowingComponent* i *FollowersComponent*. Kada korisnik odabere opciju za prikaz pratitelja ili praćenih korisnika, komponente komuniciraju sa servisom koji dohvaća potrebne podatke iz Firestore baze. Metode *getFollowingUserDetails()* i *getFollowersDetails()* unutar servisa implementiraju logiku za dohvaćanje podataka o korisnicima koje trenutni korisnik prati, odnosno koji prate trenutnog korisnika. Metoda *getFollowingUserDetails()*, prikazana na slici 4.13., prvo dohvaća dokument korisnika iz kolekcije *users* unutar Firestore baze podataka. Unutar tog dokumenta, nalazi se polje *following*, koje sadrži listu objekata s identifikatorima korisnika koje trenutni korisnik prati, kao i datume kada je korisnik počeo pratiti te osobe. Nakon dohvaćanja identifikatora, metoda koristi *Promise.all()* kako bi paralelno dohvatila profile svih praćenih korisnika. Na kraju, dohvaćeni podaci se mapiraju u objekte tipa *UserProfile*, koji predstavljaju podatke o korisničkim profilima. Metoda *getFollowersDetails()*, prikazana na slici 4.14., radi na sličan način, ali dohvaća listu pratitelja korisnika. Ova metoda također koristi Firestore kako bi dohvatila dokument korisnika, unutar kojeg se nalazi polje *followers*. To polje sadrži listu identifikatora korisnika koji prate trenutnog korisnika, a zatim se koriste isti mehanizmi za paralelno dohvaćanje korisničkih podataka.



Sl. 4.13. Metoda getFollowingUserDetails() za dohvaćanje osoba koje korisnik prati

Sl. 4.14. Metoda getFollowerUserDetails() za dohvaćanje osoba koje prate korisnika

4.4.3. Pregled objava

Metoda *loadPosts()*, definirana unutar *PostComponent*, zadužena je za dohvaćanje i prikaz objava korisnika. Ova metoda, prikazana na slici 4.15., provjerava prisutnost korisničkog identifikatora, te u skladu s tim dohvaća ili objave specifičnog korisnika ili objave korisnika koje trenutni korisnik prati. Korisnički identifikator uid dohvaća se iz URL-a, te ako je on prisutan, poziva se metoda *getPosts()* iz servisa koja dohvaća objave korisnika s tim identifikatorom. U suprotnom, poziva se metoda *getPostsByFollowing()* koja dohvaća objave korisnika koje trenutni korisnik

prati. Dohvaćene objave sortiraju se prema datumu kreiranja, a prikazuju se unutar *ProfileComponent* i *HomeComponent*.

```
loadPosts(): void {
  if (this.uid) {
    this.postService.getPosts(this.uid).subscribe(
      (posts) => {
        this.posts = posts;
        this.postCount.emit(posts.length);
      },
      (error) => {
        console.error('Error fetching posts:', error);
      }
    );
  } else if (this.currentUserId) {
    this.postService.getPostsByFollowing(this.currentUserId).subscribe(
      (posts) => {
        this.posts = posts;
        this.postCount.emit(posts.length);
      },
      (error) => {
        console.error('Error fetching posts by following:', error);
      }
    );
  }
}
```

Sl. 4.15. Metoda loadPosts() za pozivanje metoda za dohvaćanje objava

4.4.4. Označavanje objava sa sviđanjem

Ukoliko korisnik želi označiti objavu drugog korisnika sa sviđanjem, to može učiniti klikom na ikonu srca koja se nalazi ispod svake objave. Ova funkcionalnost realizirana je kroz metodu *toggleLike()*, prikazanoj na slici 4.16. Metoda *toggleLike()* koristi korisnički identifikator kako bi utvrdila je li trenutni korisnik već označio objavu sa sviđanjem. Ako je, metoda uklanja sviđanje pozivom funkcije *dislikePost()* definiranom unutar servisa. U suprotnom, dodaje se novo sviđanje korištenjem funkcije *likePost()*. Metoda *likePost()* implementirana je tako da koristi Firebase batch operaciju kako bi omogućila simultano ažuriranje više dokumenata. Prilikom dodavanja sviđanja, u kolekciji *posts* ažurira se polje *likes*, koje pohranjuje identifikator korisnika koji je označio objavu sa sviđanjem i vremensku oznaku kada je ta akcija izvršena. Istovremeno, kreira se nova obavijest u kolekciji *notifications*, kako bi se obavijestilo vlasnika objave da je njegova objava označena sa sviđanjem. Ta obavijest sadrži osnovne podatke o korisniku koji je označio objavu sa sviđanjem, uključujući njegovo ime, prezime i profilnu sliku.

```
toggleLike(post: IPost): void {
  if (this.currentUserId) {
    const isLiked = post.likes.some(
      (like) => like.userId === this.currentUserId
    );
    const delta = isLiked ? -1 : 1;

    this.updatePostLikes(post.postId, delta);

    if (isLiked) {
      this.postService
        .dislikePost(post.userId, post.postId)
        .then(() => {
          console.log('Post disliked successfully.');
        })
        .catch((error) => {
          console.error('Error disliking post:', error);
          this.updatePostLikes(post.postId, -delta);
        });
    } else {
      this.postService
        .likePost(post.userId, post.postId)
        .then(() => {
          console.log('Post liked successfully.');
        })
        .catch((error) => {
          console.error('Error liking post:', error);
          this.updatePostLikes(post.postId, -delta);
        });
    }
  }
}
```

Sl. 4.16. Metoda toggle() za postavljanje i uklanjanje oznake sviđanja

Metoda *dislikePost()* koristi Firebase transakciju za uklanjanje sviđanja iz polja *likes* unutar dokumenta objave. Unutar transakcije prvo se dohvaćaju trenutni podaci objave kako bi se provjerilo postoji li korisnički identifikator unutar niza sviđanja. Ako sviđanje postoji, metoda ga uklanja korištenjem *arrayRemove*, a zatim ažurira dokument objave. Paralelno s tim, metoda dohvaća obavijesti iz kolekcije *notifications* koje odgovaraju korisničkom identifikatoru i objavi, te ih briše korištenjem batch operacije.

4.4.5. Komentiranje objava

Klikom na ikonu komentara, korisnici otvaraju dijalog za komentare, koji omogućuje pregled, dodavanje i brisanje komentara. Dijalog za komentare implementiran je pomoću *CommentsDialogComponent*, koji se otvara pozivom metode *openCommentsDialog()*. Unutar dijaloga, komentari su prikazani u obliku kartica. Svaka kartica uključuje profilnu sliku, ime korisnika, tekst komentara i vremensku oznaku. Prikaz komentara omogućava metoda *loadComments()* prikazana na slici 4.17.

```
loadComments(): void {
  this.postService.getComments(this.data.postId, this.data.userId).subscribe(
    (comments) => {
      this.comments = comments;
    },
    (error) => {
      console.error('Error fetching comments:', error);
    }
  );
}
```

Sl. 4.17. Metoda loadComments() za prikaz komentara

Metoda *submitComment(),* prikazana na slici 4.18., omogućuje korisnicima dodavanje novog komentara. Kada korisnik unese komentar i pritisne Enter ili klikne na gumb za slanje, metoda postavlja *isSaving* na true i provjerava je li tekst komentara prazan. Ako nije, poziva *addComment()* metodu iz servisa, koja se nakon uspješnog dodavanja komentara resetira i ažurira popis komentara. Metoda *addComment()* dohvaća podatke o korisniku iz Firestorea, stvara novi komentar s jedinstvenim identifikatorom i dodaje ga u kolekciju komentara za određenu objavu. Ako objava nije vlasništvo trenutnog korisnika, kreira se obavijest u kolekciji *notifications*, obavještavajući vlasnika objave o novom komentaru. Ova metoda također koristi Firebase batch operacije i transakcije kako bi osigurala da se svi podaci ispravno ažuriraju u bazi podataka.

```
submitComment(): void {
  this.isSaving = true;
  if (this.newComment.trim()) {
    this.postService
      .addComment(this.data.postId, this.data.userId, this.newComment)
      .then(() => {
        this.isSaving = false;
        this.newComment = '';
        this.loadComments();
      })
      .catch((error) => {
        this.isSaving = false;
        console.error('Error adding comment:', error);
      });
  }
}
```

Sl. 4.18. Metoda addComment() za dodavanje komentara

Kada korisnik želi obrisati komentar, metoda *deleteComment()*, prikazana na slici 4.19., poziva *deleteComment()* metodu iz servisa. Metoda iz servisa koristi Firestore transakcije za sigurno uklanjanje komentara iz dokumenta objave. Unutar transakcije, metoda dohvaća dokument objave, filtrira komentare kako bi uklonila komentar s navedenim commentId, i ažurira broj komentara.

Ako je korisnik prijavljen, metoda također uklanja sve obavijesti povezane s obrisanim komentarom iz kolekcije *notifications* korištenjem batch operacija.

```
deleteComment(commentId: string): void {
  this.postService
    .deleteComment(this.data.postId, this.data.userId, commentId)
    .then(() => {
      this.loadComments();
    })
    .catch((error) => {
      console.error('Error deleting comment:', error);
    });
}
```

Sl. 4.18. Metoda deleteComment() za brisanje komentara

4.4.6. Slanje poruka

Zaslon za slanje poruka podijeljen je na dva dijela. Lijevi panel sadrži tražilicu koja filtrira korisnike na temelju unesenog imena i prezimena, korištenjem *searchControl* za prikaz rezultata u stvarnom vremenu. Svaki pronađeni korisnik prikazuje se s osnovnim informacijama, uključujući profilnu sliku i ime. Moguće je pretraživati samo zajedničke pratitelje, a za to je zaslužna metoda *searchMutualUsers()* definirana unutar servisa. Klikom na nekog od ponuđenih korisnika otvara se desni panel. Desni panel prikazuje razgovor s odabranim korisnikom, a svaka poruka se prikazuje s datumom i vremenom slanja. U donjem dijelu panela nalazi se polje za unos novih poruka. Nakon unosa, poruka se može poslati pritiskom na tipku Enter ili klikom na ikonu za slanje. Kada se poruka pošalje, poziva se metoda *sendMessage()*. Ova metoda koristi istoimenu metodu, prikazanu na slici 4.19., kako bi dodala poruku u Firestore bazu podataka na dvije lokacije: jednu za pošiljatelja i jednu za primatelja. Na taj način oba korisnika imaju pristup povijesti poruka. Nakon slanja poruke, polje za unos se automatski prazni, a razgovor se ažurira prikazom nove poruke.

Sl. 4.19. Metoda sendMessage() za slanje poruka

Sve poruke unutar razgovora dohvaćaju se korištenjem metode *getMessages()* iz servisa, prikazanom na slici 4.20. Ova metoda osigurava dohvaćanje i prikaz poruka između trenutnog korisnika i odabranog korisnika u stvarnom vremenu. Ako poruka sadrži atribut *isRead* postavljen na false, poruka se označava kao nepročitana. Metoda *markMessagesAsRead()* ažurira status poruka u Firestore bazi, postavljajući atribut *isRead* na true kada primatelj otvori razgovor.

Sl. 4.20. Metoda getMessages() za prikaz poruka

Ispod tražilice na lijevom panelu dohvaćaju se zadnje poruke korisnika korištenjem metode *getLatestMessages()*. Ova metoda radi na način da kombinira podatke iz dvije lokacije: jednu koja pohranjuje poruke koje je korisnik poslao, i drugu koja pohranjuje poruke koje je primio. Metoda koristi *combineLatest()* kako bi simultano pratila promjene u više kolekcija poruka, omogućavajući dohvaćanje najnovije poruke.

4.5. Dodatne funkcionalnosti

Dodatne funkcionalnosti implementirane unutar aplikacije uključuju ocjenjivanje korisničkog iskustva i statističku obradu rezultata koju obavlja administrator. Korisnik ocjenjuje korisničko iskustvo označavanjem broja zvjezdica, nakon čega se ti rezultati statički obrađuju u obliku grafa korištenjem Chart.js.

4.5.1. Ocjenjivanje korisničkog iskustva

Proces ocjenjivanja korisničkog iskustva započinje odabirom opcije za ocjenjivanje unutar izbornika na navigacijskoj traci. Klikom na opciju, poziva se funkcija *openRateAppDialog()* koja otvara dijalog za ocjenjivanje. Ocjenjivanje je omogućeno označavanjem zvjezdica koje predstavljaju ocjenu od jedan do pet. Kada se dijalog otvori, komponenta *RateAppDialogComponent* provjerava je li korisnik već ocijenio aplikaciju putem metode *checkIfUserHasRated()*. Ova metoda koristi jedinstveni korisnički identifikator kako bi iz Firestore baze podataka dohvatila postojeću ocjenu. Ako je ocjena pronađena, korisnik vidi svoju prethodnu ocjenu i poruku zahvale, a daljnje ocjenjivanje je onemogućeno. Ako korisnik nije prethodno ocijenio aplikaciju, na sučelju se prikazuje sustav zvjezdica, omogućujući odabir željene ocjene. Odabirom zvjezdica poziva se metoda *rate()* koja ažurira vizualni prikaz zvjezdica i sprema trenutnu ocjenu u varijablu rating. Nakon što je korisnik zadovoljan odabranom ocjenom, može pritisnuti gumb *Submit* koji poziva metodu *submitRating()*. Metoda *submitRating()* poziva metodu *saveRating()*, prikazanu na slici 4.21., koja uzima korisnički identiikator i ocjenu, te ih pohranjuje u Firestore bazu podataka. Nakon uspješnog spremanja, metoda *submitRating()* zatvara dijalog za ocjenjivanje i obavještava korisnika o uspjehu.

```
async saveRating(userId: string, rating: number) {
  try {
    await this.firestore.collection('ratings').doc(userId).set({
      rating: rating,
    });

    return 'App rated successfully!';
  } catch (error) {
    console.error('Error rating app:', error);
    throw error;
  }
}
```

Sl. 4.21. Metoda saveRating() za spremanje ocjena korisnika

4.5.2. Statistička obrada rezultata koju obavlja administrator

Ukoliko je korisnik postavljen kao administrator, u izborniku mu se, umjesto opcije za ocjenjivanje korisničkog iskustva, pojavljuje opcija za pregled statistike rezultata. Odabirom te opcije, korisnik se preusmjerava na zaslon gdje je prikazan detaljan pregled svih prikupljenih ocjena. Komponenta *StatisticsComponent* inicijalno dohvaća sve spremljene ocjene putem servisa RatingService, korištenjem metode *getRatingCounts()*, prikazane na slici 4.22., koja vraća broj dodijeljenih ocjena, od jedne do pet zvjezdica. Dohvaćene ocjene pohranjuju se u niz ratingCounts, a zatim se izračunavaju dodatni statistički podaci, uključujući prosječnu ocjenu, modalnu ocjenu i postotak svake ocjene u odnosu na ukupni broj ocjena. Nakon što su svi potrebni podaci dohvaćeni i izračunati, metoda *renderChart()* prikazuje grafikon ocjena korištenjem Chart.js biblioteke. Ocjene su prikazane u obliku stupčastog grafikona, gdje se na osi X nalaze različite ocjene (od jedne do pet zvjezdica), dok os Y prikazuje broj korisnika koji su dodijelili određenu ocjenu. Za isticanje stupaca unutar grafikona koriste se različite nijanse ružičaste.

```
async getRatingCounts(): Promise<number[]> {
  const counts = [0, 0, 0, 0, 0];
  try {
    const snapshot = await this.firestore
      .collection('ratings')
      .get()
      .toPromise();
    snapshot.forEach((doc) => {
      const data = doc.data() as Rating;
      if (data.rating >= 1 && data.rating <= 5) {
        counts[data.rating - 1]++;
      }
    });
    return counts;
  } catch (error) {
    console.error('Error fetching rating counts:', error);
    throw error;
  }
}
```

Sl. 4.22. Metoda getRatingCounts() za dohvaćanje ocjena korisnika

4.6. Objavljivanje aplikacije

Za objavljivanje aplikacije korišten je Firebase Hosting. Prvi korak je instaliranje Firebase CLI upisivanjem narede *npm install -g firebase-tools* unutar komandne linije. Nakon toga, potrebno se prijaviti u Firebase upisivanjem *firebase login* naredbe. Sljedeći korak je inicijalizacija Firebase projekta unutar direktorija aplikacije korištenjem naredbe *firebase init*. U ovom procesu korisnik bira Firebase Hosting kao opciju, odabire korištenje trenutnog projekta te direktorija projekta,

lokaciju direktorija (obično je to mapa *dist*, koja se generira naredbom *npm run build*) te postavlja opcije vezane sa mogućnost korištenja jednostranačke aplikacije (engl. *Single Page Application, SPA*). Kada je inicijalizacija dovršena, Firebase generira konfiguracijske datoteke kao što su *firebase.json* i *.firebaserc*. U te datoteke spremaju se osnovni parametri poput putanja za objavljivanje i postavki preusmjeravanja. Sadržaj datoteke *firebase.json* prikazan je na slici 4.23. Parametar "public": "dist/browser" definira direktorij iz kojeg će Firebase dohvatiti statičke datoteke aplikacije, dok opcija "rewrites" osigurava da se sve zahtjeve koji ne odgovaraju specifičnoj datoteci preusmjeri na *index.html*. Također, datoteka isključuje nepotrebne resurse kao što su *.firebase.json* i *node_modules*.

Sl. 4.23. Sadržaj *firebase.json* datoteke

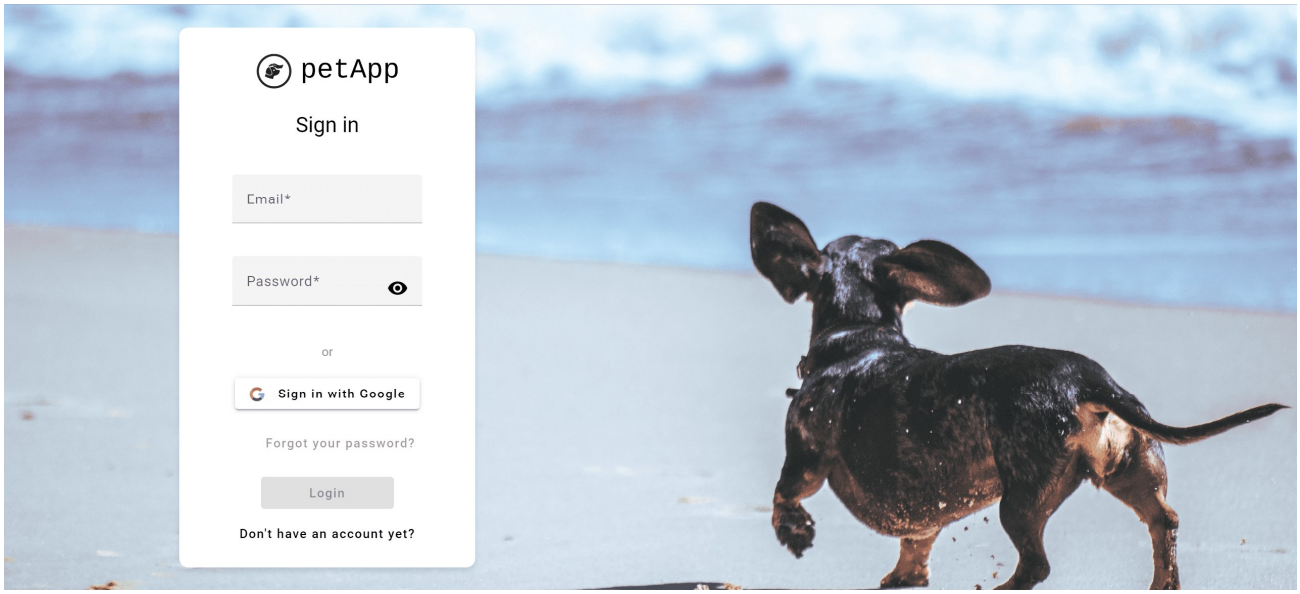
Zadnji korak je objavljivanje aplikacije korištenjem *firebase deploy* naredbu. Nakon što je objavljivanje završeno, u konzoli se dobiva link na čiji klik se može pristupiti aplikaciji. Nakon dodavanja novih značajki, svako novo objavljivanje aplikacije zahtjeva ponovno upisivanje *npm run build* i *firebase deploy* naredbi.

5. PRIKAZ RADA WEB APLIKACIJE

U ovom poglavlju dan je prikaz rada izrađene web aplikacije. Razvijena aplikacija omogućava korisnicima razne mogućnosti. Prije samog korištenja, korisnik se treba prijaviti u aplikaciju ili registrirati, ukoliko već nema izrađen račun.

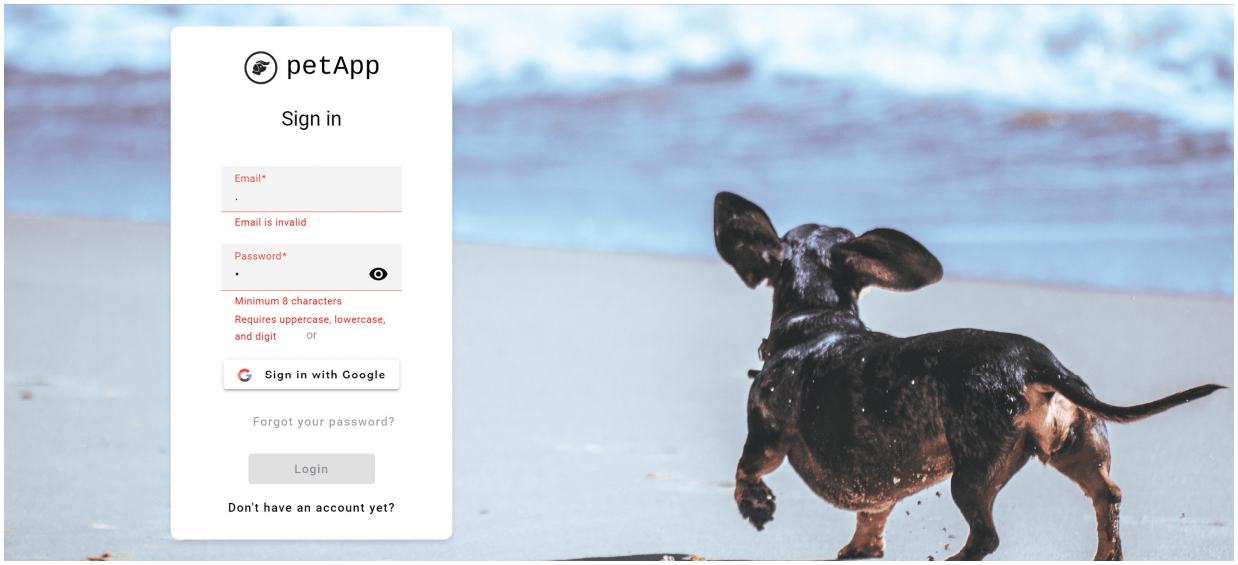
5.1. Prijava korisnika

Nakon pokretanja aplikacije, korisniku se otvara zaslon za prijavu prikazan na slici 5.1. Korisnik se može prijaviti unosom adrese e-pošte i zaporke ili pomoću Google računa.



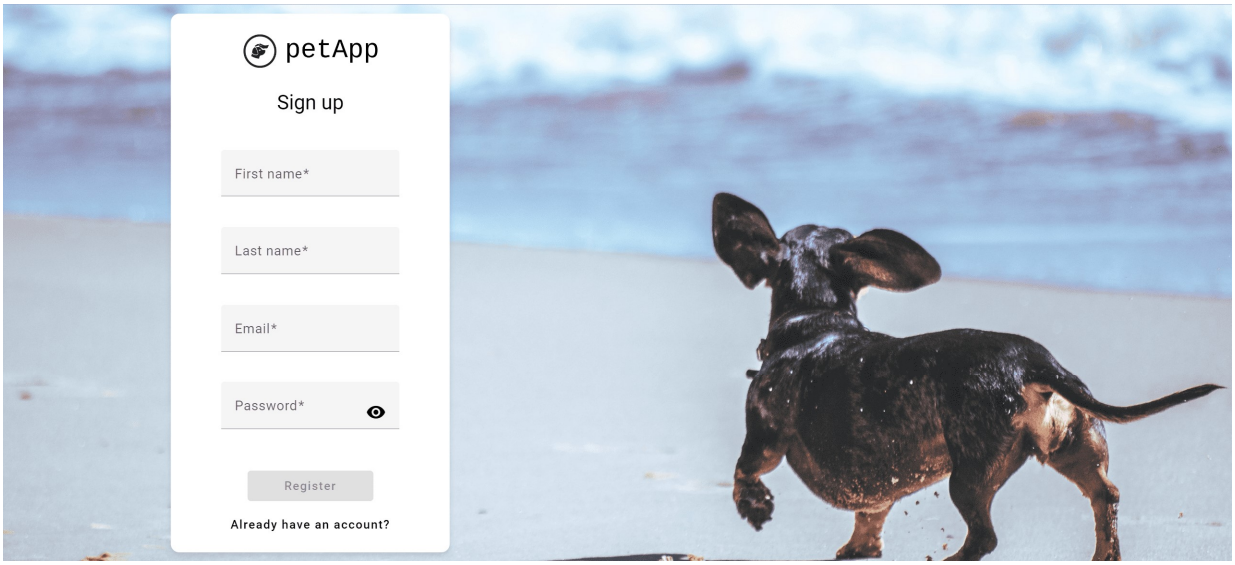
Sl. 5.1. Prijava korisnika u aplikaciju

Prilikom unosa adrese e-pošte, potrebno je paziti na format podatka. Ukoliko korisnik unese neispravan oblik podatka, pojaviti će se poruka o grešci, što je vidljivo na slici 5.2. Osim toga, polje za unos adrese e-pošte je obavezno te će se, ukoliko ga korisnik ostavi praznim, pojaviti poruka o grešci. Polje za unos zaporke također je obavezno, te treba sadržavati minimalno osam znakova, te barem jedno veliko slovo, malo slovo i brojku, što je također vidljivo na slici 5.2. Gumb za prijavu biti će onemogućen sve dok korisnik ispravno ne popuni sva polja za unos.



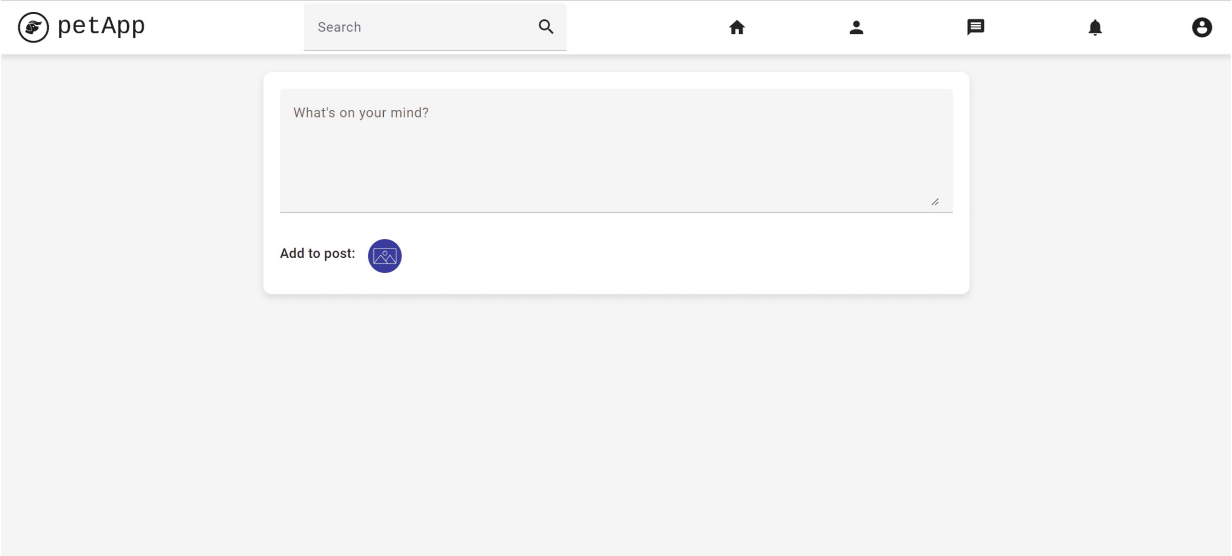
Sl. 5.2. Validacija adrese e-pošte i zaporke

Ako je korisnik zaboravio zaporku, postoji mogućnost resetiranja zaporke. Klikom na gumb *Forgot your password?* korisniku se na adresu e-pošte šalje link koji vodi na mogućnost unosa nove zaporke. Ukoliko korisnik nema kreiran račun u aplikaciji, može ga kreirati klikom na gumb *Don't have an account yet?*, nakon čega se otvara zaslon prikazan na slici 5.3.



Sl. 5.3. Registracija korisnika u aplikaciju

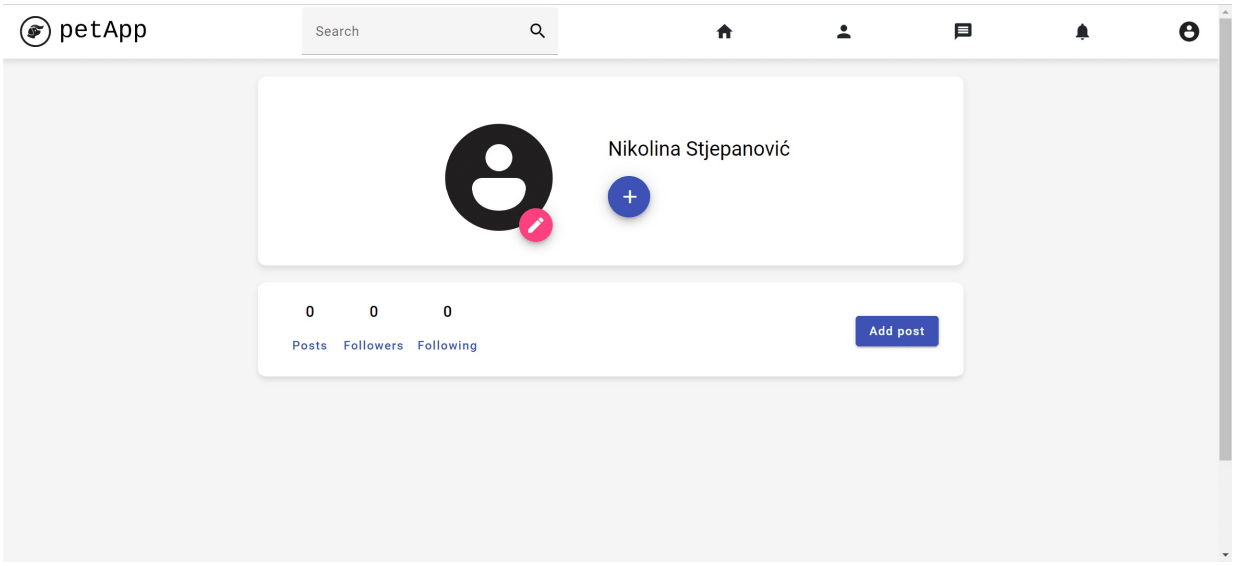
Polja za unos imena i prezimena također ne smiju biti prazna, te ne smiju biti kraća od dva znaka. Nakon uspješne prijave ili registracije, korisnika se vodi na zaslon prikazan na slici 5.4.



Sl. 5.4. Početni zaslon aplikacije

5.2. Korisnički profil prijavljenog korisnika

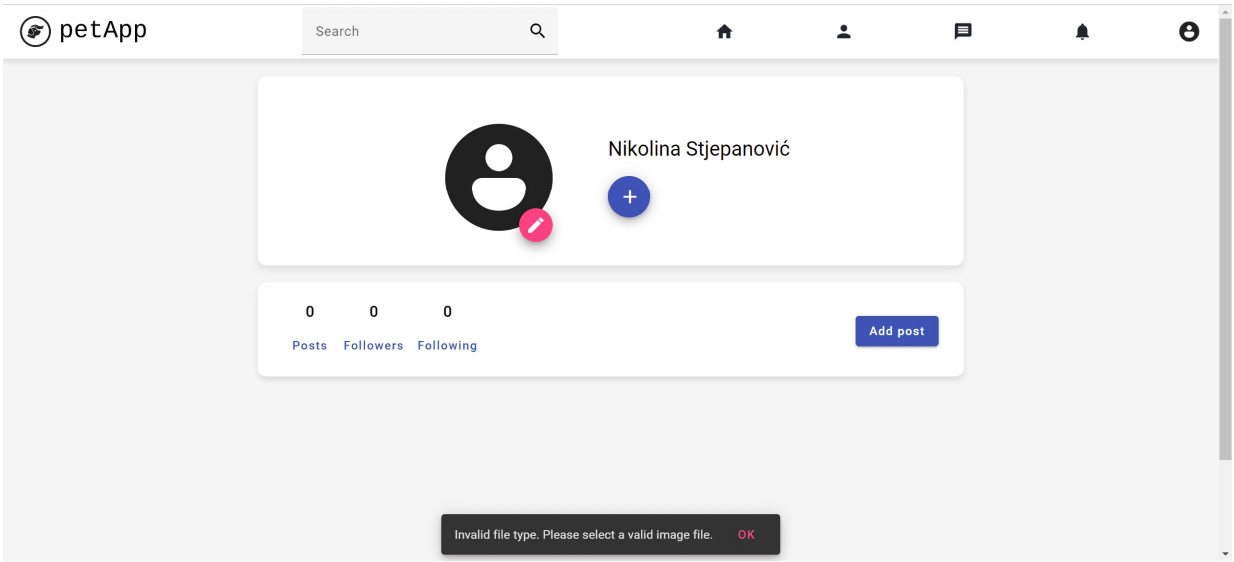
Klikom na ikonu osobe u navigacijskoj traci, korisnika se vodi na vlastiti profil prikazan na slici 5.5. Na profilu se korisniku pružaju brojne mogućnosti poput dodavanja slike profila, dodavanja svojeg kućnog ljubimca, uređivanja i brisanja ljubimca, dodavanja i brisanja objave, prikaza pratitelja i drugih.



Sl. 5.5. Korisnički profil prijavljenog korisnika

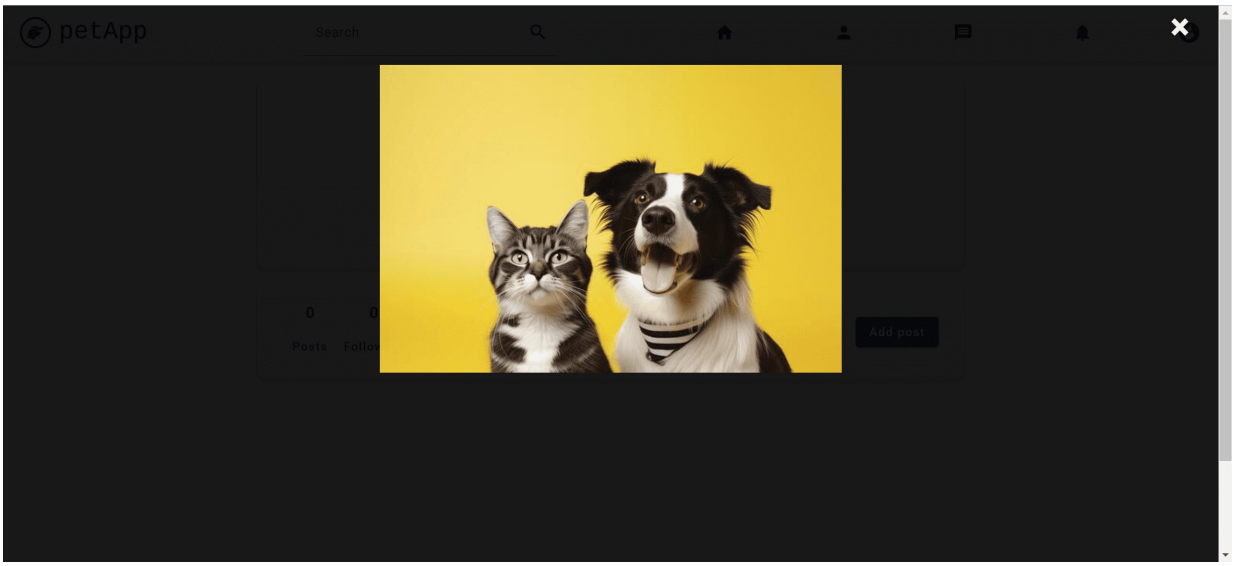
5.2.1. Dodavanje slike profila

Klikom na ikonu olovke, korisniku se otvara izbornik sa opcijom *Choose Image* na čiji klik može odabrati fotografiju. Ukoliko odabere neispravnu datoteku, na dnu zaslona pojavit će se prikladna poruka, što je prikazano na slici 5.6.



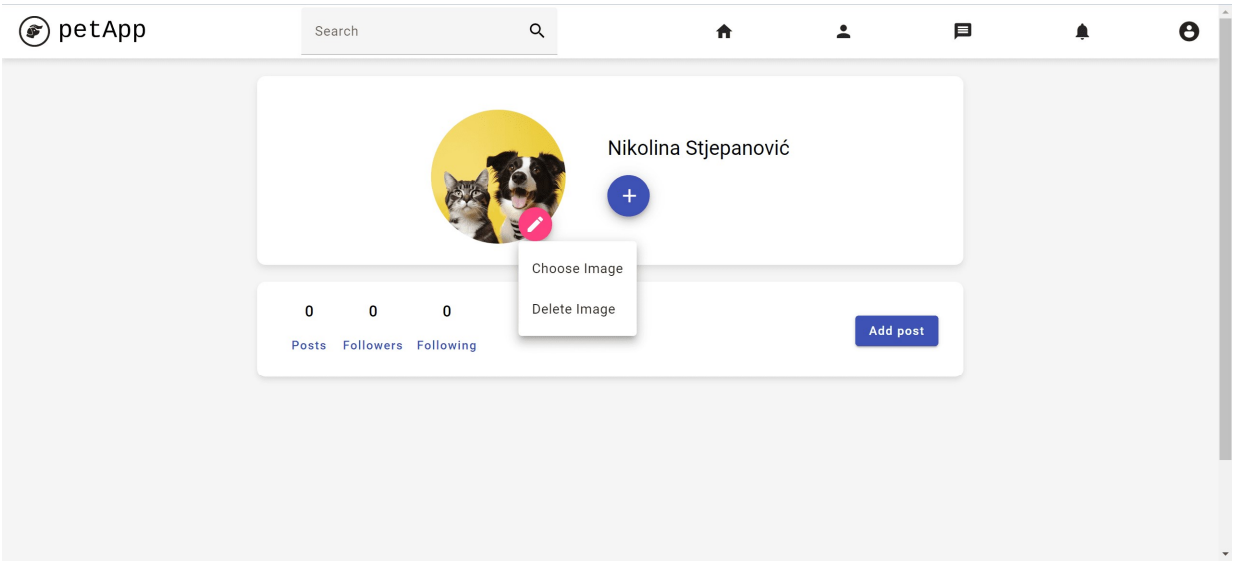
Sl. 5.6. Poruka o neispravnoj datoteci

Nakon učitavanja ispravne datoteke, na dnu zaslona pojavljuje se poruka da je slika uspješno učitana. Ukoliko korisnik želi bolje vidjeti odabranu sliku profila, može kliknuti na učitanu fotografiju, nakon čega se otvara modal prikazan na slici 5.7.

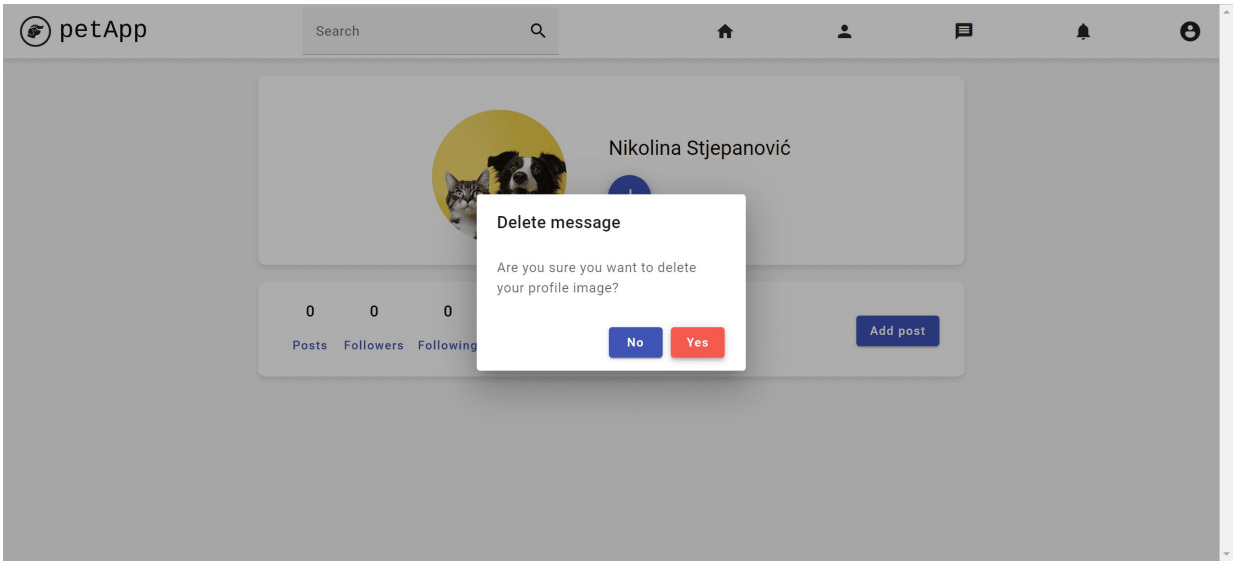


Sl. 5.7. Slika profila unutar modala

Korisnik može obrisati odabranu sliku profila ponovnim klikom na ikonu olovke, u čijem izborniku je sada vidljiva dodatna opcija *Delete Image*. Prikaz izbornika vidljiv je na slici 5.8., a klikom na opciju *Delete Image* otvara se dijalog za potvrdu brisanja prikazan na slici 5.9.



Sl. 5.8. Izbornik za upravljanje slikom profila

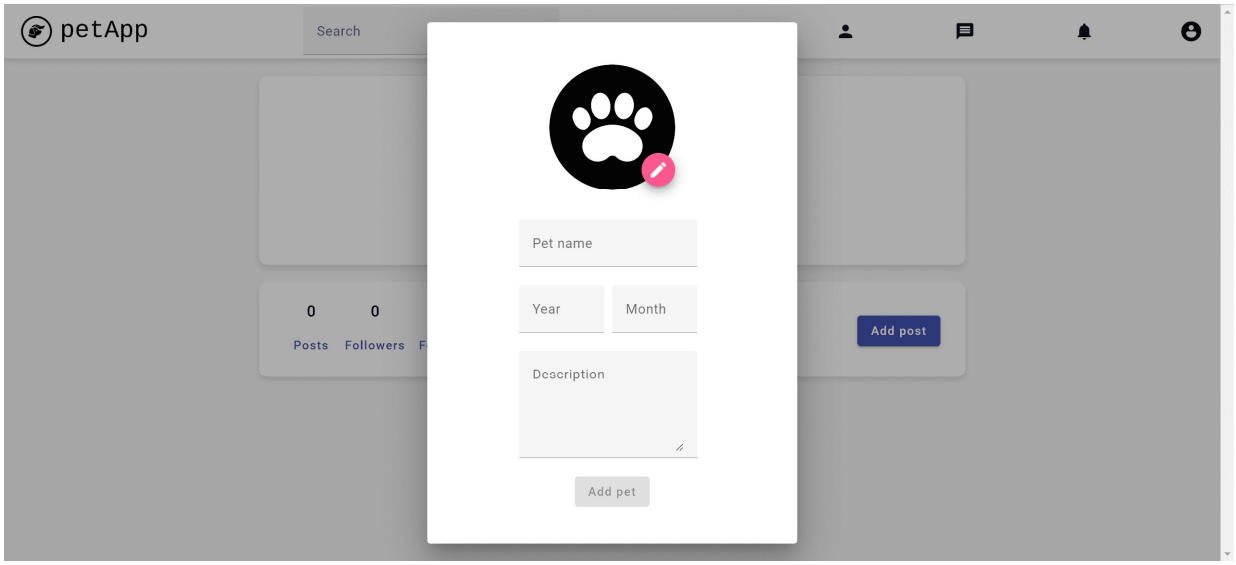


Sl. 5.9. Dijalog za potvrdu brisanja slike profila

5.2.2. Dodavanje, uređivanje i brisanje ljubimca

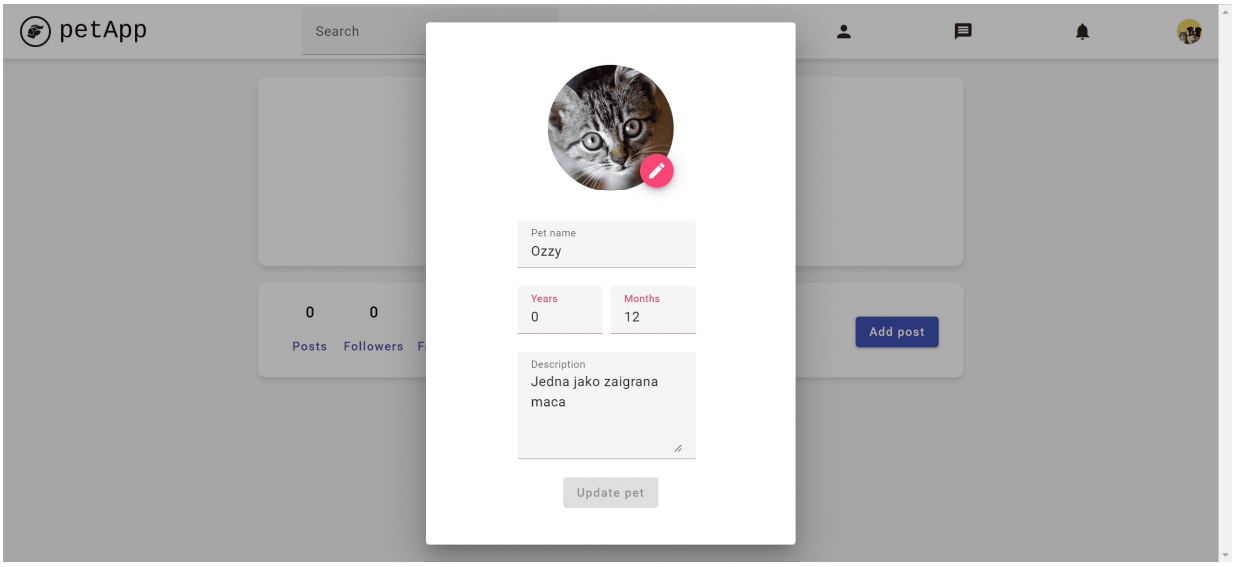
Pored slike profila vidljiv je gumb za dodavanje, na čiji klik se otvara dijalog za dodavanje ljubimca prikazan na slici 5.10. Unutar dijaloga korisnik može dodati fotografiju svojeg ljubimca,

te unijeti informacije poput imena ljubimca, broja godina i kratkog opisa. Svi navedeni elementi su obavezni, te ukoliko korisnik koji izostavi, ljubimac neće biti dodan.



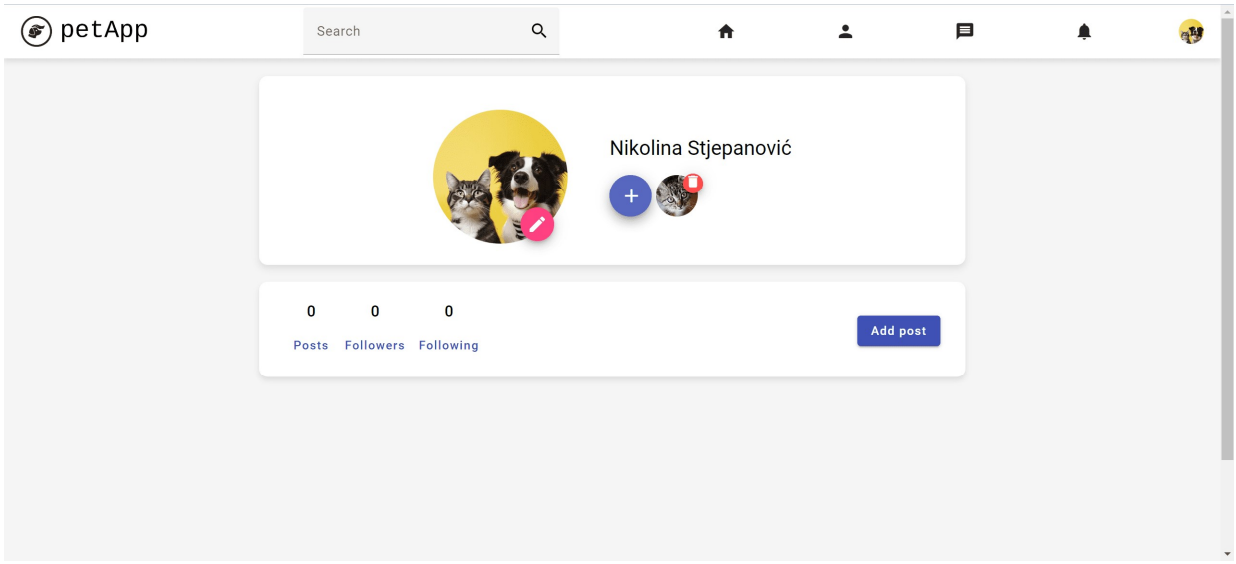
Sl. 5.10. Dijalog za dodavanje ljubimca

Prilikom dodavanja fotografije, odabir neispravne datoteke prikazat će prikladnu poruku na dnu zaslona. Jedno od polja kod unosa godina ljubimca smije ostati prazno, ali ne smije biti negativno ili nula. Također broj mjeseci ne smije biti veći od jedanaest, što je prikazano na slici 5.11.



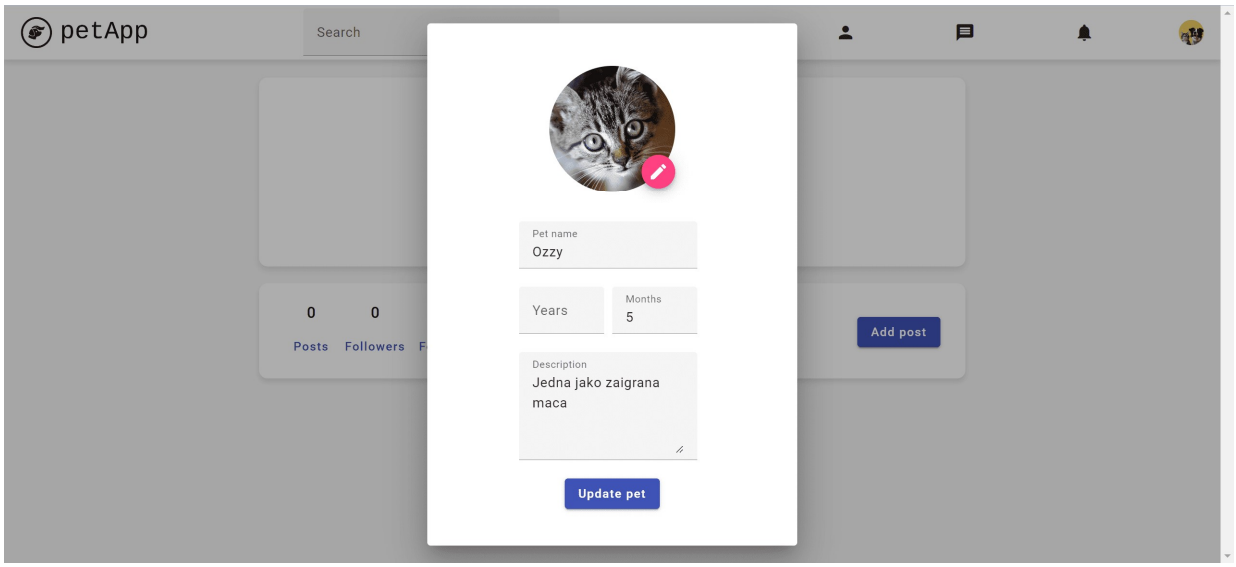
Sl. 5.11. Validacija za unos godina ljubimca

Nakon uspješnog prolaska validacije te klika na gumb *Add pet*, na dnu zaslona pojavljuje se prikladna poruka da je ljubimac uspješno dodan. Dodani ljubimac prikazuje se na profilu pored gumba za dodavanje u obliku male ikone, što je prikazano na slici 5.12.



Sl. 5.12. Prikaz dodanog ljubimca na korisničkom profilu

Klikom na dodanog ljubimca, moguće je izmijeniti informacije o ljubimcu, što je prikazano na slici 5.13. Nakon izmjena, klik na gumb *Update pet* prikazat će poruku da je ljubimac uspješno izmijenjen.

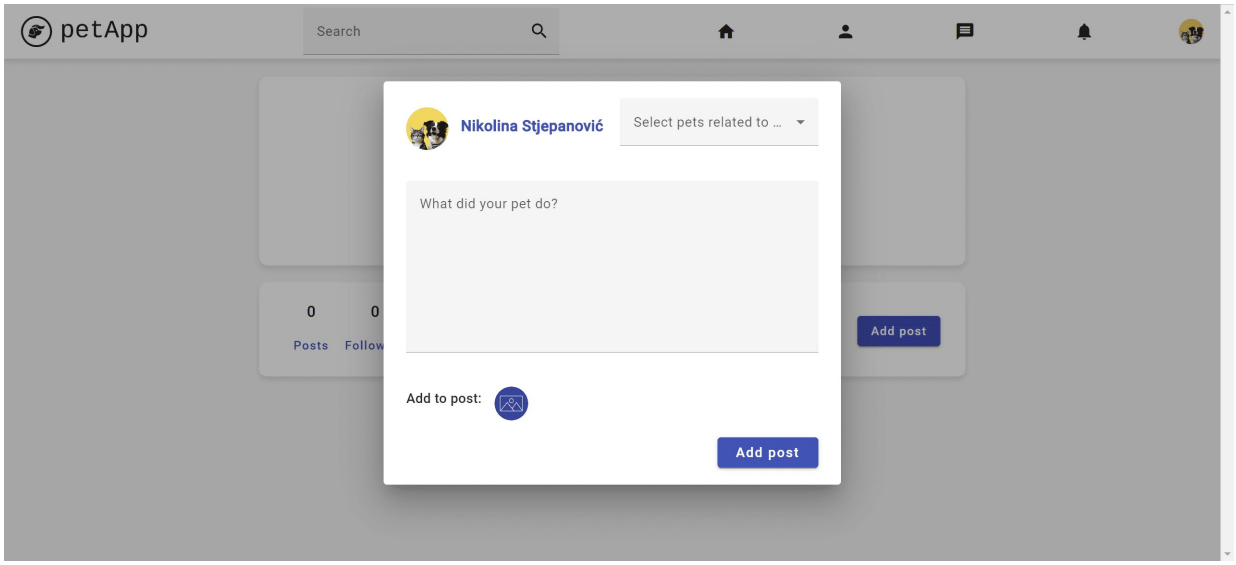


Sl. 5.13. Dijalog za izmjenu ljubimca

Na desnoj strani dodanog ljubimca prikazana je ikona za brisanje, na čiji klik korisnik može obrisati ljubimca. Prije brisanja, pojavljuje se dijalog za potvrdu brisanja slično kao na slici 5.9.

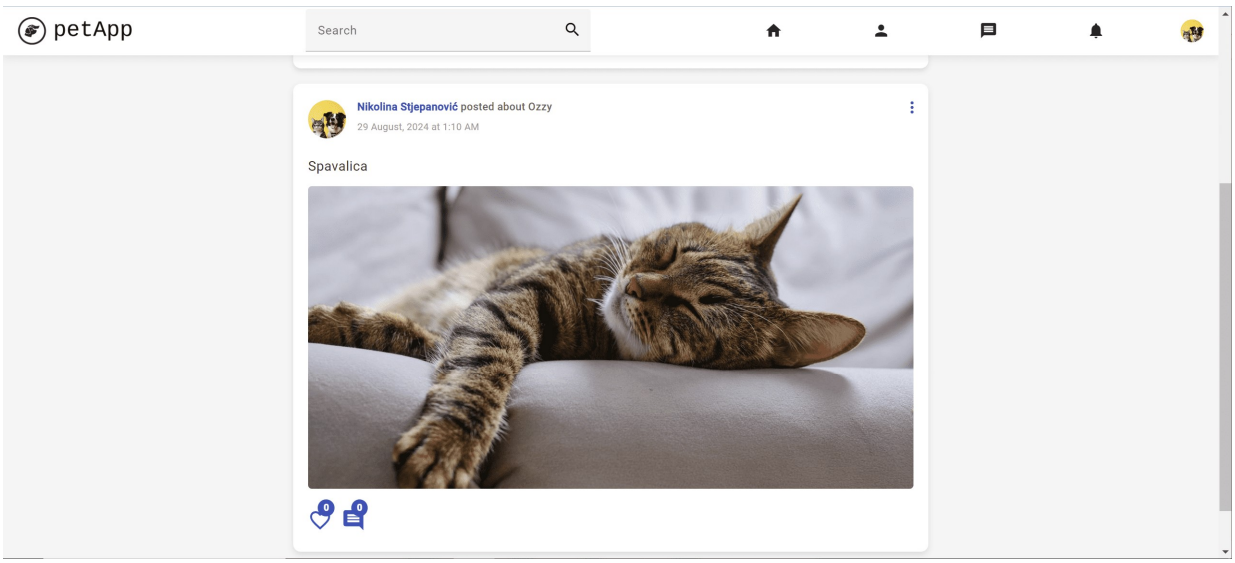
5.2.3. Dodavanje i brisanje objave

Objava se može dodati klikom na gumb *Add post*, nakon čega se otvara dijalog prikazan na slici 5.14.



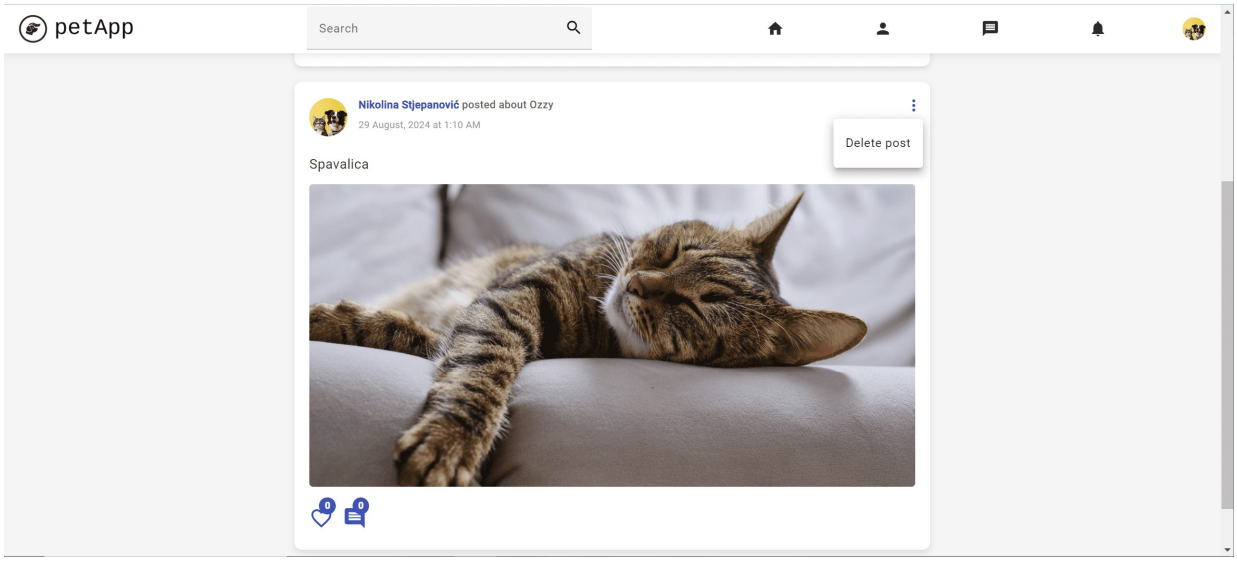
Sl. 5.14. Dodavanje objave

Objavu je moguće podijeliti u obliku teksta, fotografije ili oboje. Ukoliko je korisnik dodao ljubimca, na gornjoj desnoj strani dijaloga pojavljuje se izbornik za odabir unutar kojeg su prikazani svi ljubimci koje se korisnik dodao. Odabirom ljubimca korisnik određuje na kojeg se ljubimca objava odnosi. Ukoliko korisnik pokuša objaviti prazan post, na dnu zaslona pojavljuje se prikladna poruka. Nakon unosa željene objave, klik na gumb *Add post* objava se pojavljuje na profilu korisnika, što je vidljivo na slici 5.15. Osim na profilu, objava se prikazuje i na početnoj stranici aplikacije, s koje je također moguće dodati objavu.



Sl. 5.15. Prikaz objave na korisničkom profilu

U gornjem desnom kutu objave vidljiva je ikona izbornika, prikazana u obliku tri točkice poredane okomito, na čiji klik se prikazuje opcija za brisanje objave, što je vidljivo na slici 5.16.

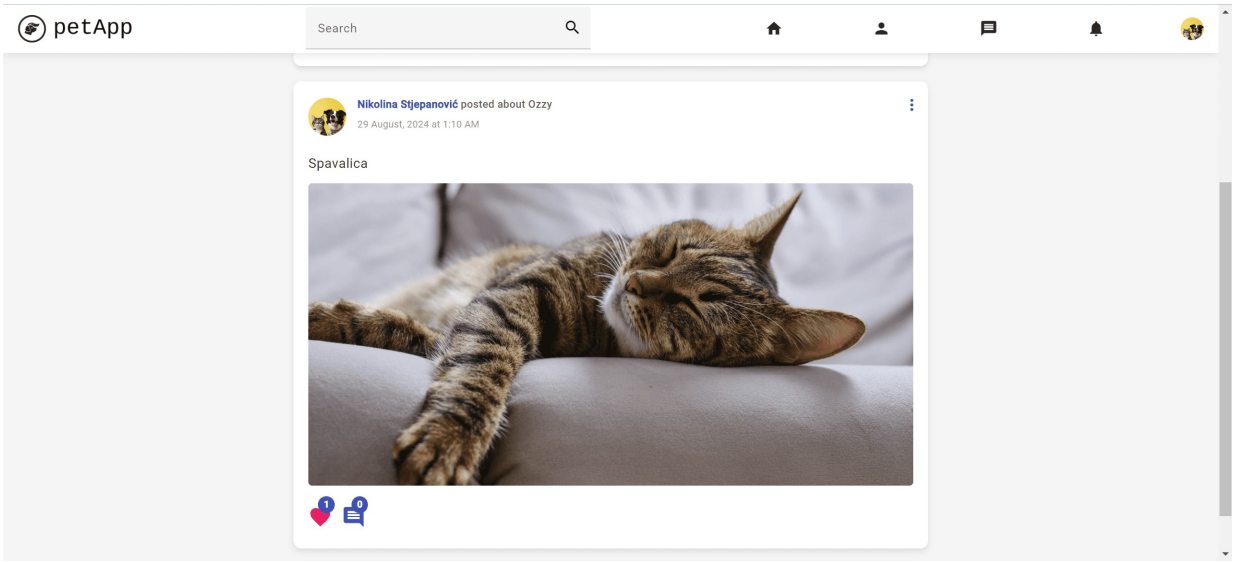


Sl. 5.16. Opcija za brisanje objave

Odabirom opcije otvara se dijalog za potvrdu brisanja objave slično kao na slici 5.9.

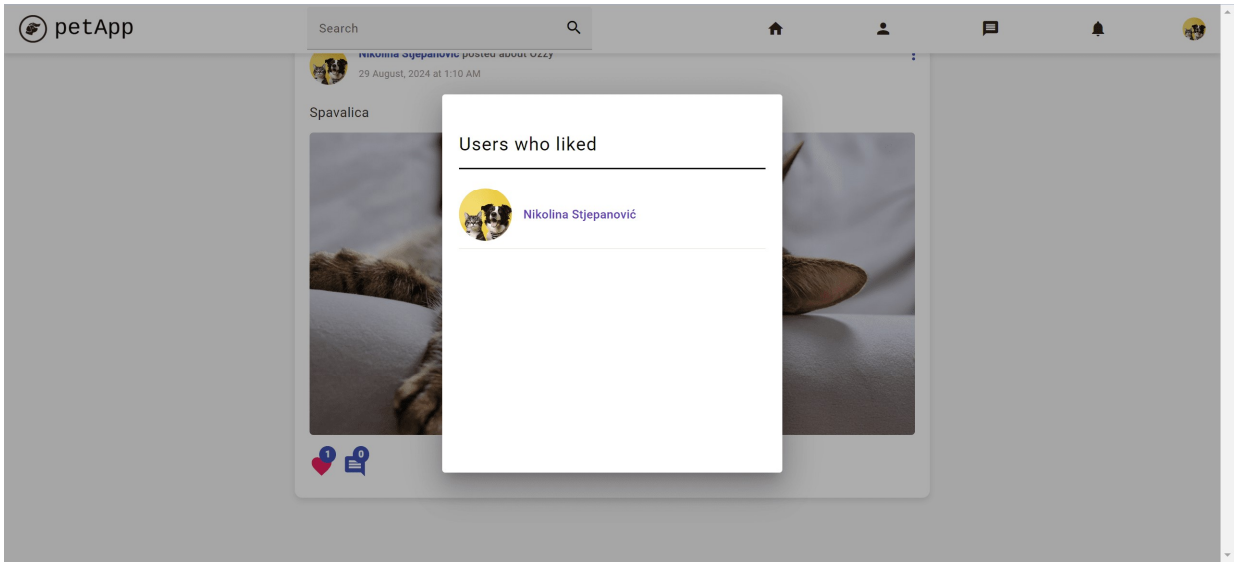
5.2.4. Označavanje objava sa sviđanjem

U donjem lijevom kutu objave prikazanoj na slici 5.16. nalazi se ikona za označavanje objave sa sviđanjem. Ikona je inicijalno bijela s ljubičastim okvirom, a nakon označavanja prelazi u ružičastu, što je prikazano na slici 5.17.



Sl. 5.17. Prikaz objave nakon označavanja objave sa sviđanjem

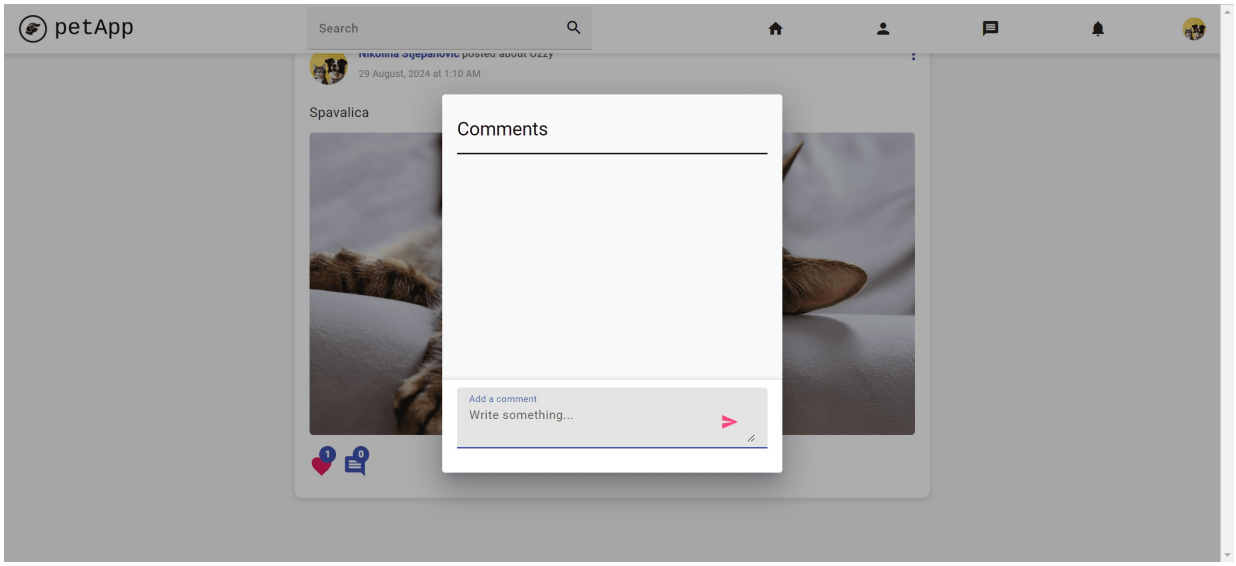
Iznad ikone nalazi se oznaka koja prikazuje broj osoba koje su označile objavu sa sviđanjem, a klikom na oznaku otvara se dijalog prikazan na slici 5.18. Unutar dijaloga prikazane su sve osobe kojima se objava sviđa, a klikom na neku od osoba korisnika se vodi na profil odabrane osobe.



Sl. 5.18. Prikaz osoba koje su označile objavu sa sviđanjem

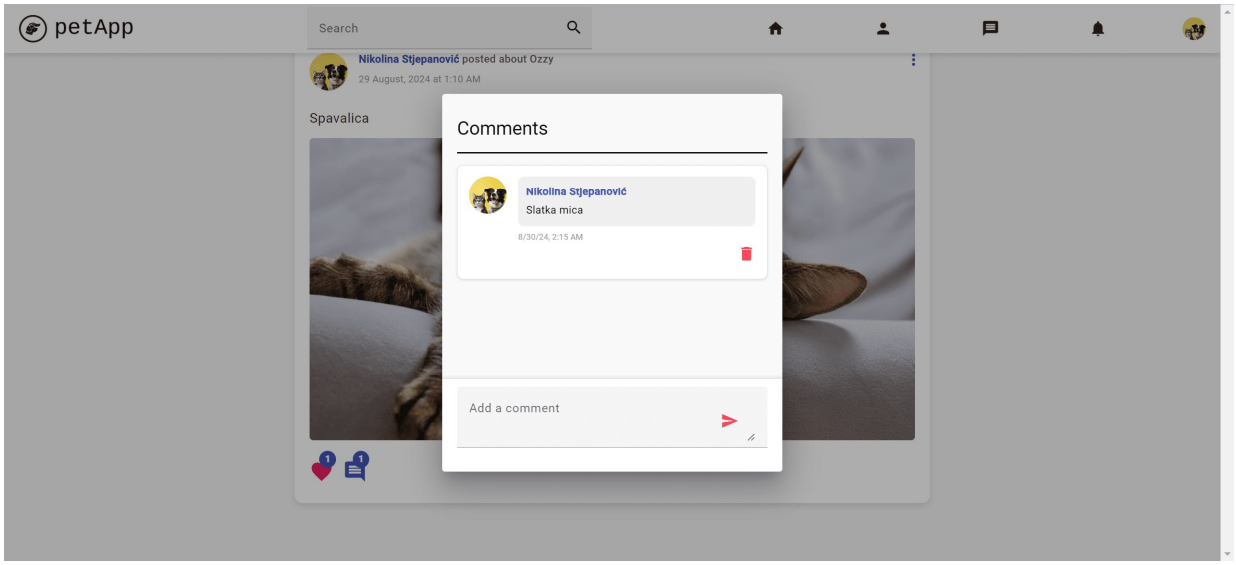
5.2.5. Komentiranje objava

Pored ikone za označavanje objave sa sviđanjem nalazi se ikona za komentiranje objave na čiji se klik otvara dijalog prikazan na slici 5.19.



Sl. 5.19. Dijalog za komentiranje objave

Nakon unosa željenog sadržaja te klika na ikonu za slanje, komentar se objavljuje te se prikazuje kao na slici 5.20. Osim klika na ikonu, komentar se može objaviti pritiskom tipke Enter.

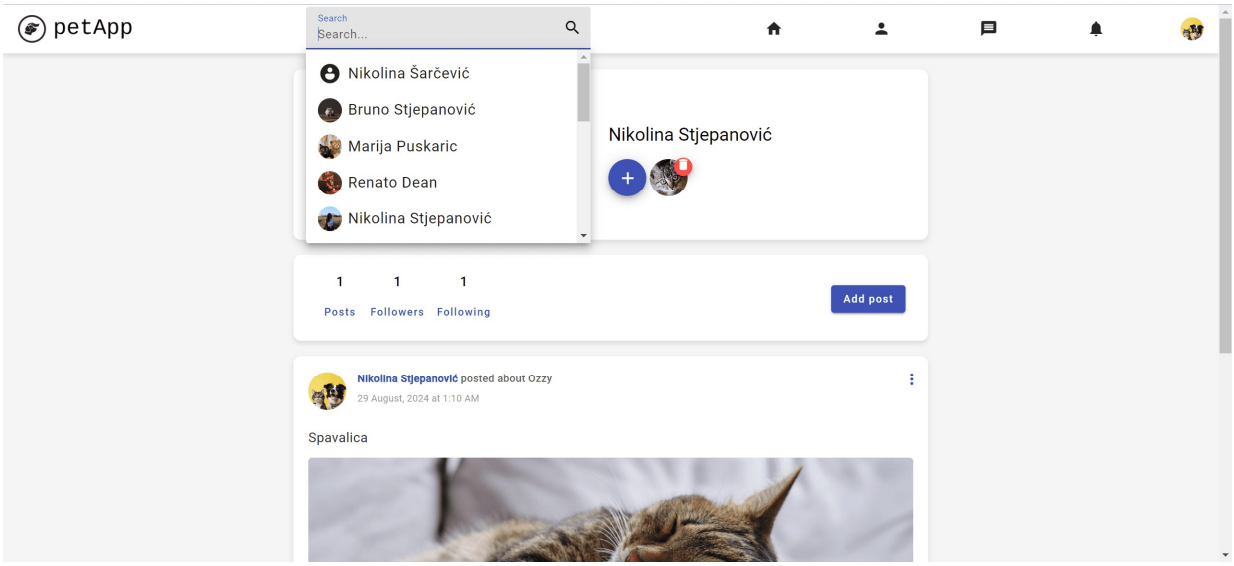


Sl. 5.20. Prikaz komentara unutar dijaloga

Vlasnik komentara ima mogućnost brisanja komentara klikom na ikonu za brisanje prikazanoj u donjem desnom kutu komentara, nakon čega se komentar odmah briše, bez potvrde za brisanje. Ikona za komentiranje također kao i ikona za označavanje objave sa sviđanjem sadrži oznaku, koja u ovom slučaju, prikazuje broj komentara na objavi.

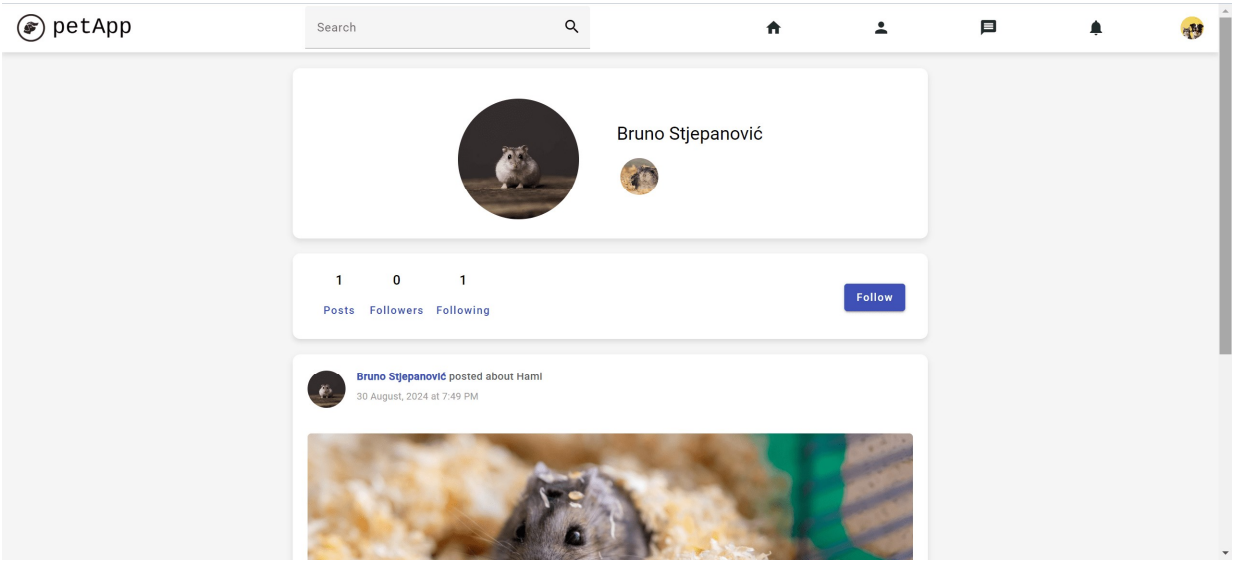
5.3. Korisnički profil ostalih korisnika

Na navigacijskoj traci nalazi se tražilica na čiji klik se otvara popis svih korisnika aplikacije, što je prikazano na slici 5.21. Uz ime i prezime korisnika vidljiva je i slika profila, kako bi se korisnici s istim imenom lakše razlikovali. Upisivanjem imena, korisnici se filtriraju te se prikazuju samo korisnici koji odgovaraju traženom upitu, a pretraživanje je omogućeno po imenu i prezimenu.



Sl. 5.21. Popis svih korisnika aplikacije

Klikom na nekog od ponuđenih osoba, korisnika se vodi na profil odabrane osobe, što je prikazano na slici 5.22.

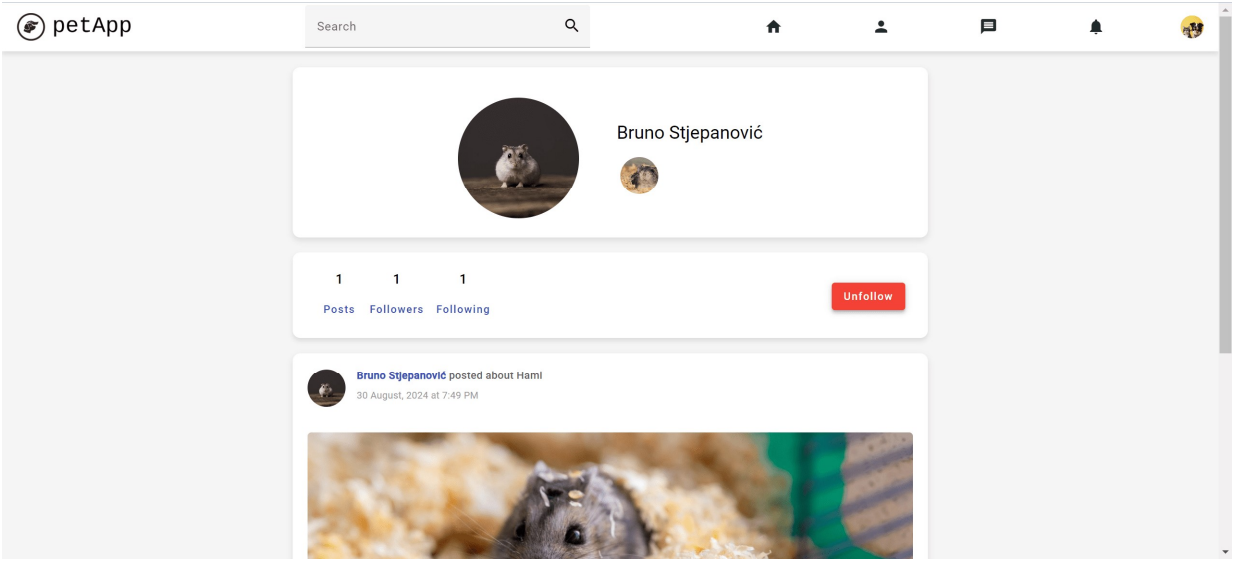


Sl. 5.22. Prikaz korisničkog profila odabranog korisnika iz tražilice

Na profilu drugih korisnika, prijavljeni korisnik može vidjeti njihove objave, označiti ih sa sviđanjem, komentirati, vidjeti informacije o njihovim ljubimcima te ih zapratiti.

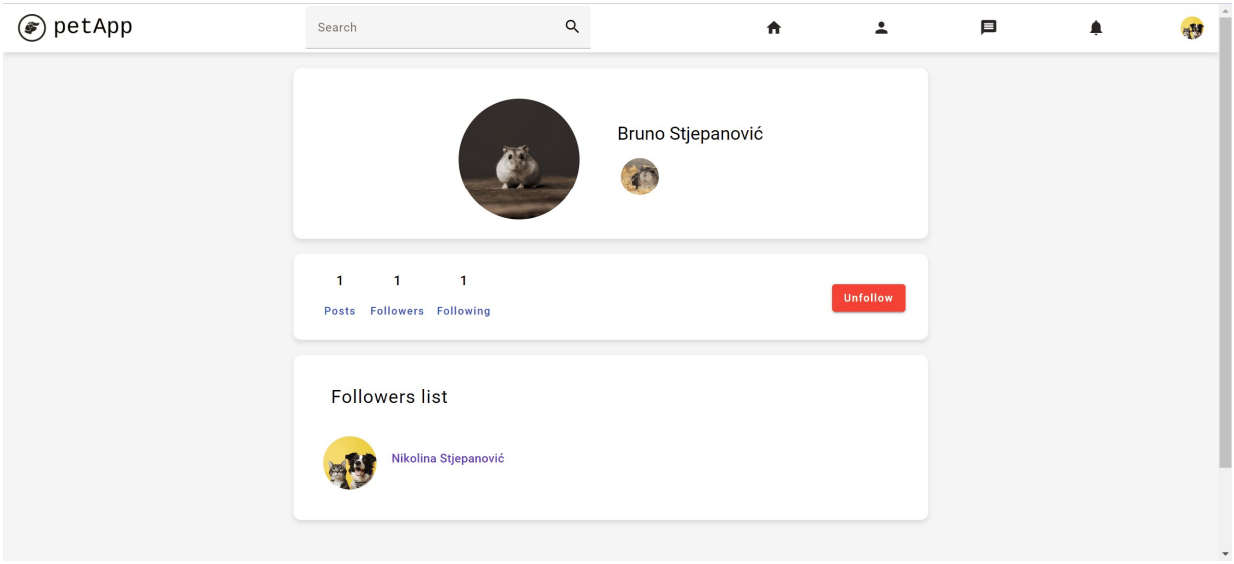
5.3.1. Praćenje korisnika

Klikom na gumb *Follow* vidljivog na slici 5.22. moguće je zapratiti prikazanog korisnika. Nakon klika ažurira se broj pratitelja, a gumb mijenja tekst i boju u *Unfollow* i crvenu, što je prikazano na slici 5.23. Ponovnim klikom na gumb korisnik će se otpratiti.

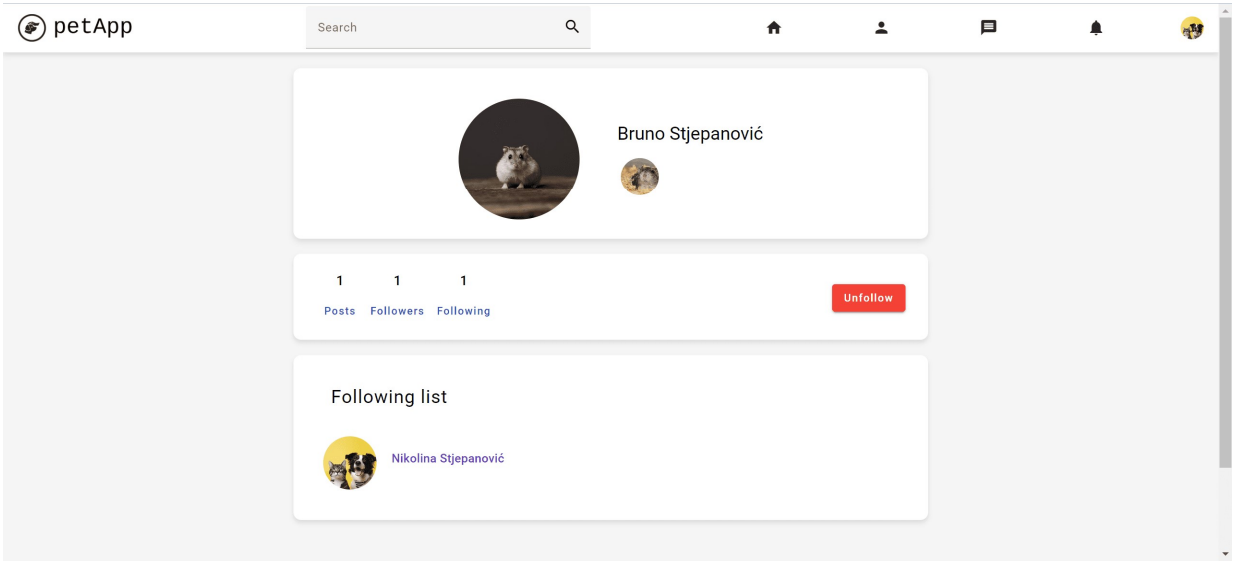


Sl. 5.23. Promjena gumba nakon praćenja korisnika

Moguće je vidjeti popis praćenih osoba, kao i osoba koje prate prikazanog korisnika klikom na gumbe *Following* i *Followers*, nakon čega se otvara prikaz kao na slikama 5.24. i 5.25. Klikom na neku od osoba s popisa, korisnika se vodi na profil odabrane osobe.



Sl. 5.24. Prikaz pratitelja prikazanog korisnika



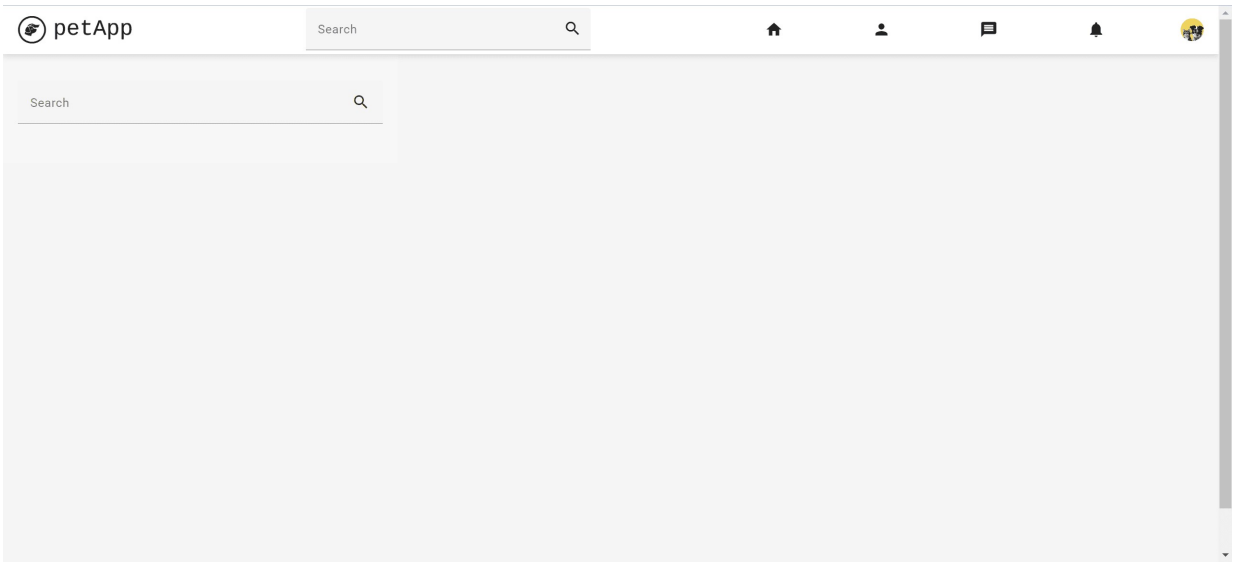
Sl. 5.25. Prikaz praćenih osoba prikazanog korisnika

Nakon praćenja korisnika, sve korisnikove objave prikazat će se na početnoj stranici korisnika koji ga prati.

5.4. Slanje poruka

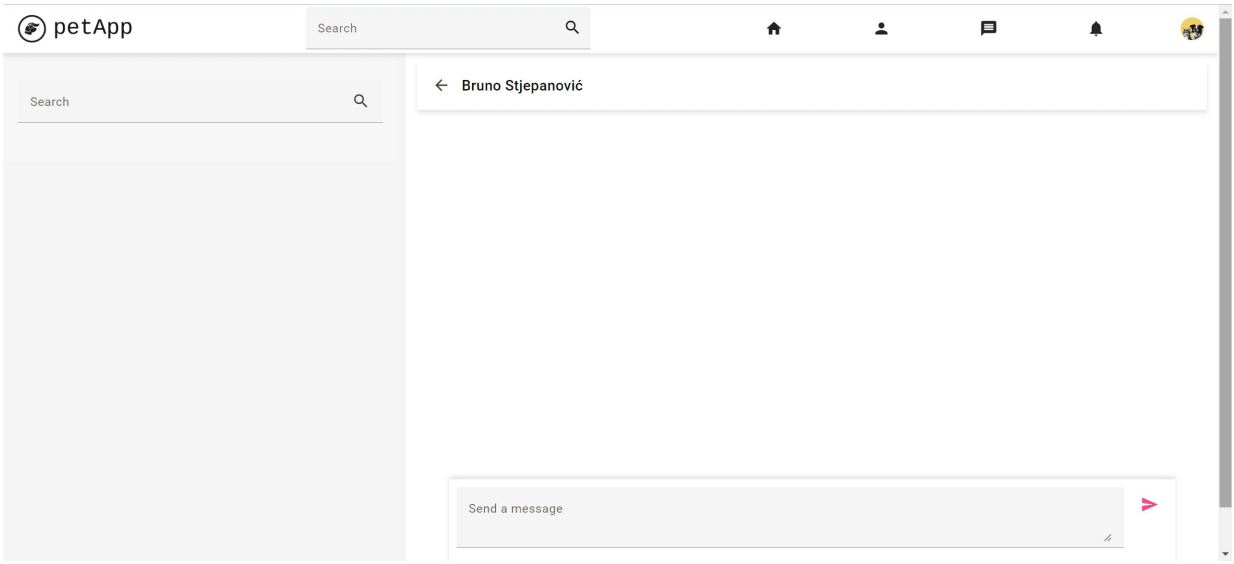
Klikom na ikonu poruke na navigacijskoj traci otvara se zaslon kao na slici 5.26. S lijeve strane zaslona nalazi se tražilica unutar koje se može vidjeti popis svih osoba, slično kao na slici 5.21.,

kojima se može poslati poruka. Slanje poruka omogućeno je samo između osoba koje se međusobno prate, kako bi se izbjegle neželjene poruke.



Sl. 5.26. Zaslon za slanje poruka

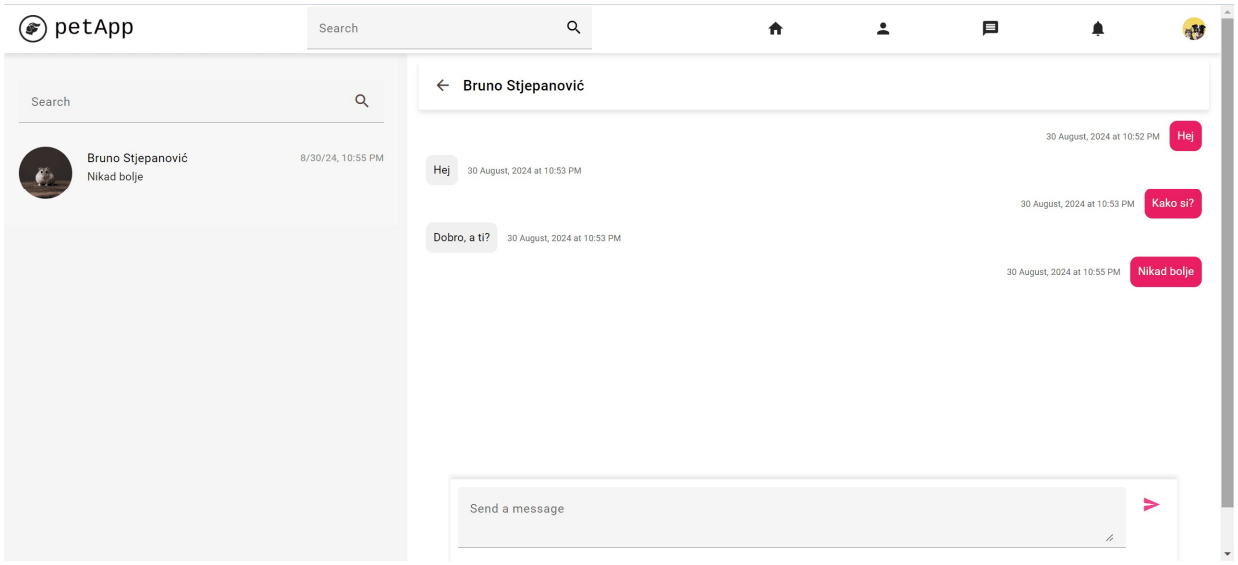
Nakon odabira osobe unutar tražilice, na desnoj strani zaslona otvara se prozor za komunikaciju kao na slici 5.27. Na vrhu prozora vidljivo je ime odabranog korisnika, a klikom na strelicu izlazi se iz prozora.



Sl. 5.27. Prozor za slanje poruka odabranom korisniku

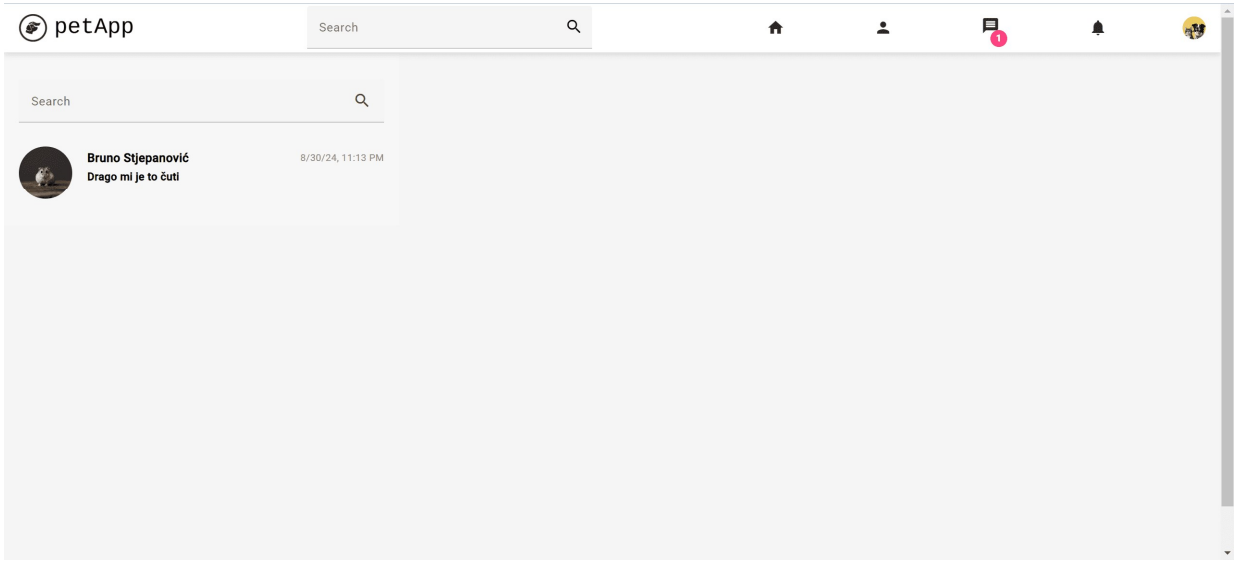
Na dnu prozora unosi se željena poruka, a nakon unosa poruka se može poslati klikom na ikonu za slanje ili pritiskom na tipku Enter. Poruke koje korisnik pošalje označene su ružičastom bojom, dok su primljene poruke označene sivom bojom, što je prikazano na slici 5.28. Pored svake poruke

nalazi se vremenska oznaka kako bi korisnik znao kada je poruka poslana ili primljena. Nakon izmjene poruka, s lijeve strane zaslona pojavljuje se odabrani korisnik zajedno sa zadnjom porukom unutar razgovora, a klikom na korisnika ponovno se otvara prozor za komunikaciju.



Sl. 5.28. Primjer slanja poruka između korisnika

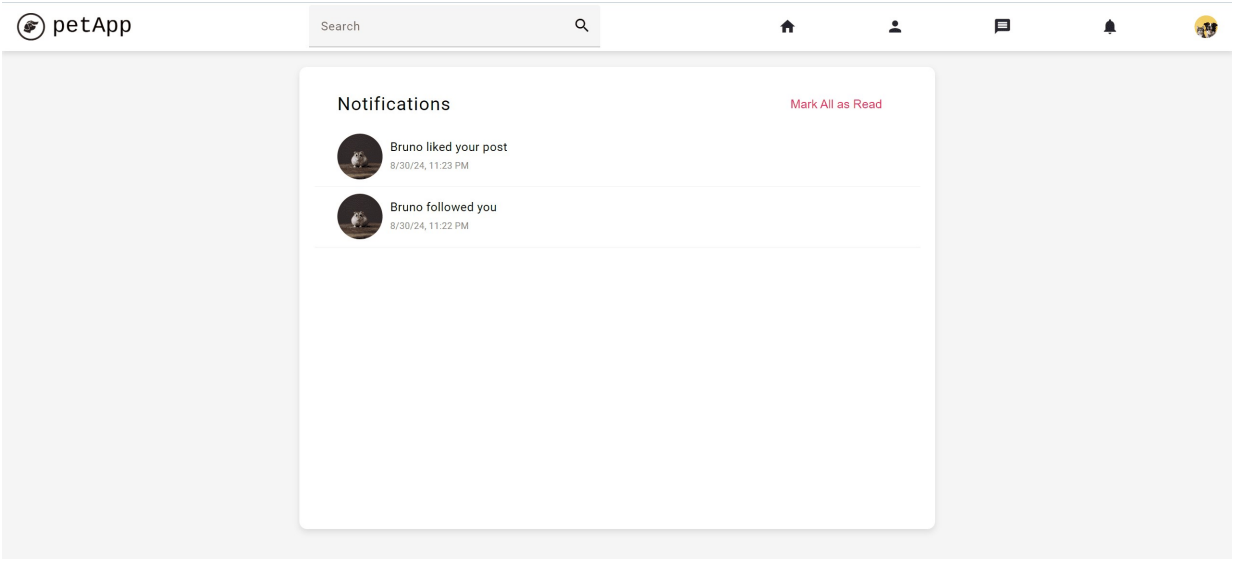
Kada korisnik primi poruku, pored ikone za slanje na navigacijskoj traci pojavljuje se oznaka koja naznačuje korisniku da ima novu poruku koju nije pročitao, a nepročitana poruka je naglašena kao na slici 5.29. Nakon klika na poruku te otvaranja prozora za komunikaciju, poruka se označava pročitanom.



Sl. 5.29. Primjer nepročitane poruke

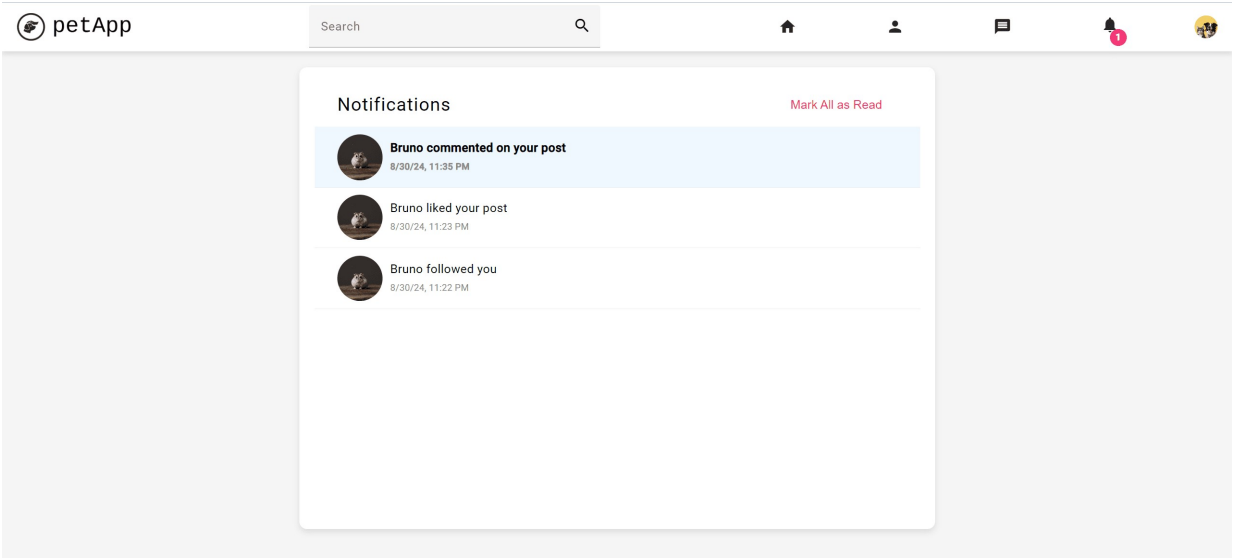
5.5. Obavijesti

Na navigacijskoj traci također je dostupna ikona za obavijesti na čiji klik se otvara zaslon kao na slici 5.30. Kada drugi korisnik komentira, označi objavu sa sviđanjem ili zaprati prijavljenog korisnika, prijavljeni korisnik će dobiti obavijest.



Sl. 5.30. Zaslon za prikaz obavijesti

Kada korisnik primi obavijest, označena je kao nepročitana te se prikazuje oznaka slična kao kod poruka na slici 5.29. Osim toga, nova obavijest će biti naglašena kako bi se razlikovala od već pročitanih obavijesti, a primjer je prikazan na slici 5.31.

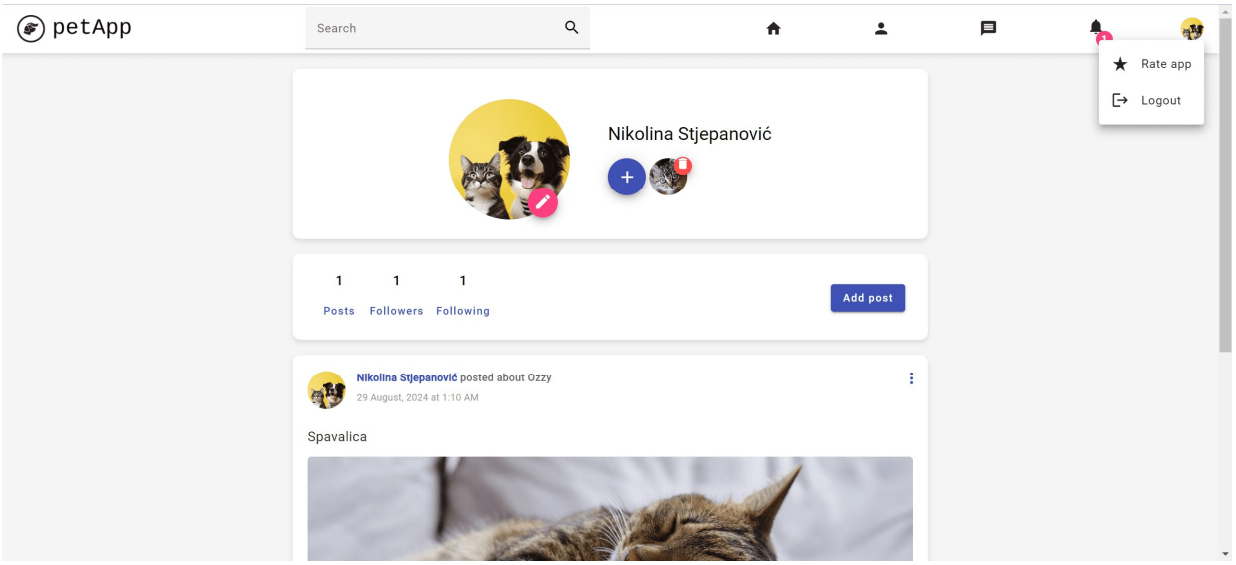


Sl. 5.31. Primjer nepročitane obavijesti

Obavijest je moguće pojedinačno označiti kao pročitanu klikom na obavijest, pri čemu će obavijest o novom komentaru ili oznaci sviđanja voditi korisnika na vlastiti profil, a obavijest o praćenju na korisnika koji je zapratio prijavljenog korisnika. Ukoliko postoji više nepročitanih obavijesti, moguće je sve obavijesti odjednom pročitati klikom na gumb *Mark All as Read*.

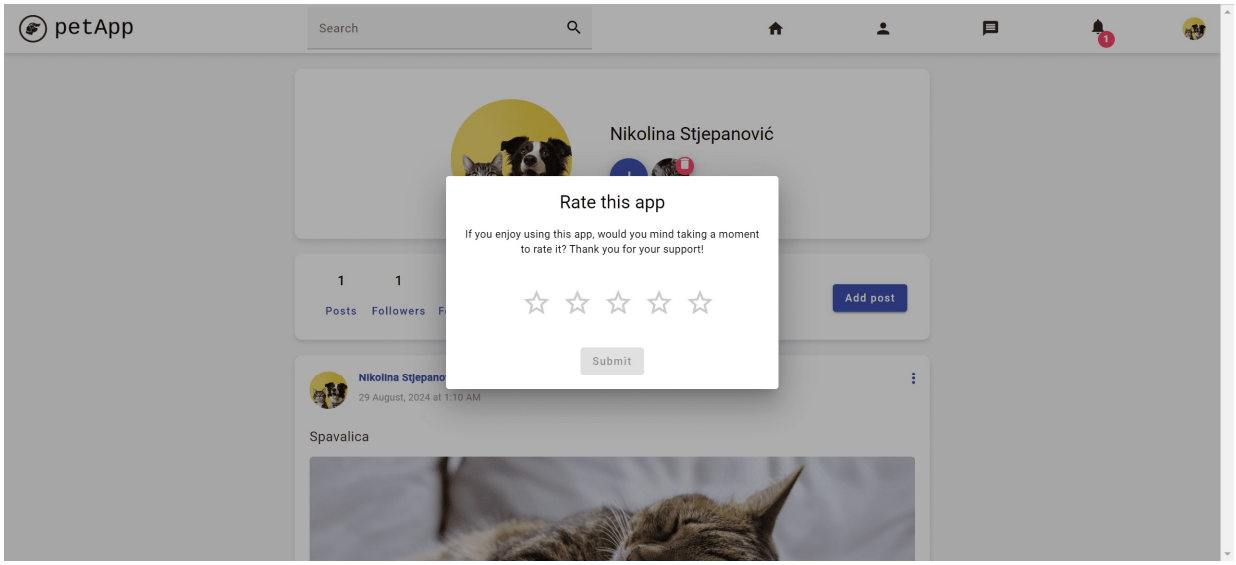
5.6. Ocjenjivanje korisničkog iskustva i obrada rezultata

Unutar aplikacije postoji mogućnost ocjenjivanja korisničkog iskustva. Na navigacijskoj traci, nakon klika na ikonu s profilnom slikom prijavljenog korisnika otvara se izbornik prikazan na slici 5.32.



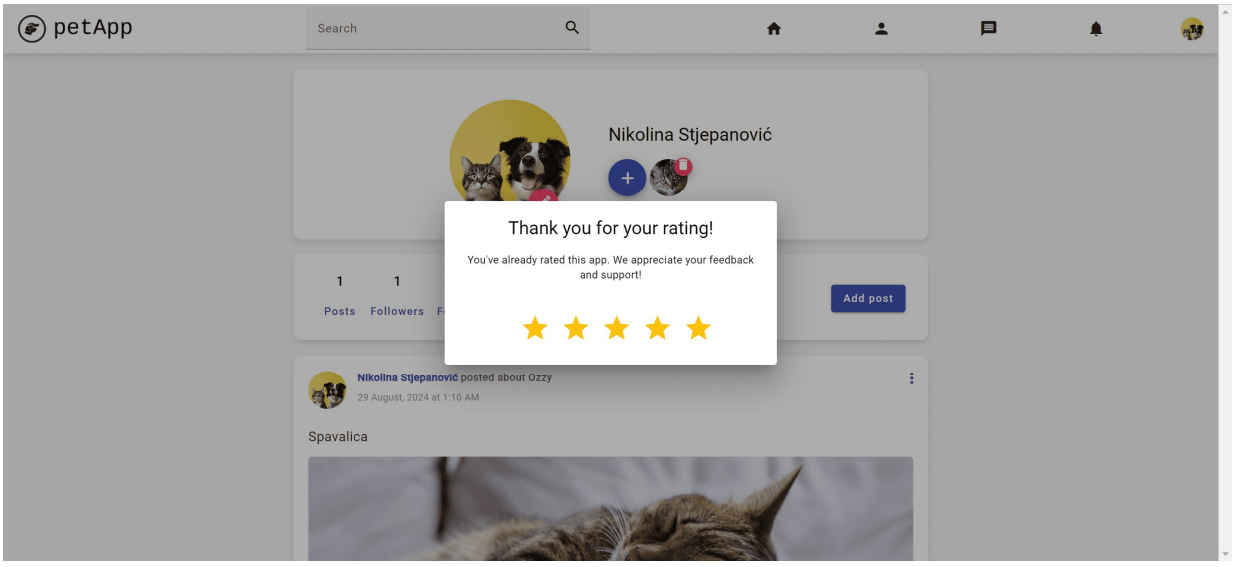
Sl. 5.32. Izbornik za ocjenjivanje korisničkog iskustva i odjavu

Odabirom opcije *Rate app* otvara se dijalog prikazan na slici 5.33. unutar kojeg korisnik može ocijeniti aplikaciju označavanjem zvjezdica. Nakon označavanja te pritiska na gumb *Submit* na dnu zaslona prikazuje se poruka zahvale.



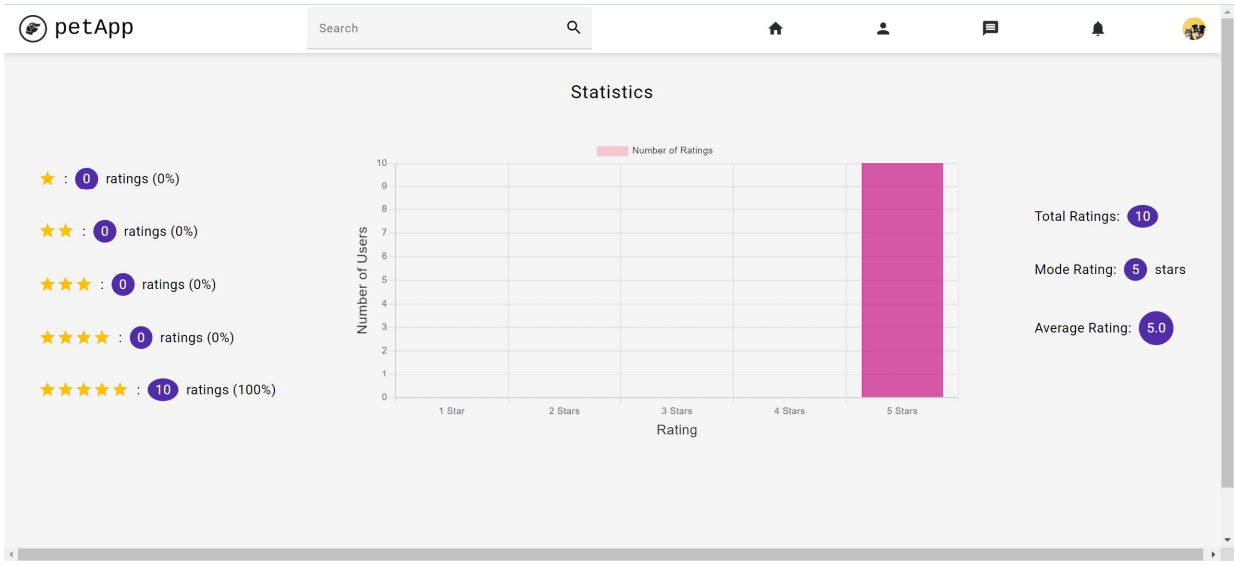
Sl. 5.33. Prikaz dijaloga za ocjenjivanje korisničkog iskustva aplikacije

Ponovno ocjenjivanje aplikacije nije moguće, te ako korisnik ponovno pokuša ocijeniti aplikaciju, prikazuje mu se dijalog kao na slici 5.34. Unutar dijaloga vidljiva je poruka zahvale te ocjena koju je korisnik dao.



Sl. 5.34. Prikaz statistike rezultata ocjena korisnika

Ukoliko korisnik ima ulogu administratora, njegov izbornik će izgledati drugačije te će umjesto *Rate app* imati opciju *View rating statistics*. Unutar *View rating statistics* korisnik može vidjeti koliko je korisnika dalo koju ocjenu, koliki je postotak pojedine ocjene u odnosu na ukupan broj ocjena, ukupan broj ocjena, najčešću ocjenu te prosjek ocjena, a osim tekstualnih informacija vidljiv je i grafički prikaz svih ocjena što je vidljivo na slici 5.35.



Sl. 5.35. Prikaz statistike rezultata ocjena korisnika

6. ZAKLJUČAK

Ljubav prema kućnim ljubimcima, te dijeljenje njihovih svakodnevnih trenutaka i avantura na društvenim mrežama potaknula je razvoj specijalizirane platforme koja bi vlasnicima kućnih ljubimaca omogućila povezivanje, dijeljenje sadržaja te razmjenu korisnih savjeta i iskustava. Iz tog razloga, svrha rada bila je razviti platformu koja bi vlasnicima kućnih ljubimaca omogućila da sav sadržaj vezan za ljubimce pronađu i dijele na jednom mjestu.

Zadatak rada bio je izraditi web aplikaciju koja se može koristiti kao oblik društvene mreže za vlasnike kućnih ljubimaca. Omogućena je prijava i registracija korisnika, te je implementirana mogućnost kreiranja vlastitog profila, dodavanja ljubimaca, dijeljenja objava, reagiranja na objave, pretraživanja i praćenja korisnika, kao i sustav za komunikaciju među korisnicima. Također je ugrađena funkcionalnost za ocjenjivanje korisničkog iskustva. Osim toga, izrađena je baza podataka za pohranu svih relevantnih podataka. Za realizaciju navedenih funkcionalnosti korišten je Angular za frontend te Firebase za backend. Od mogućnosti koje Firebase nudi korišten je Firebase Authentication za prijavu i registraciju korisnika, Firestore baza podataka za pohranu svih podataka unutar aplikacije, Storage za pohranu fotografija te Hosting za objavljivanje aplikacije.

U radu su navedeni svi funkcionalni i nefunkcionalni zahtjevi aplikacije, prikazana je i opisana korištena baza podataka, detaljno je opisana implementacija svih funkcionalnosti aplikacije korištenjem navedenih tehnologija te je prikazan rad izrađene aplikacije. Prikupljene su ocjene korisnika koje su statistički obrađene korištenjem Chart.js biblioteke. Osim toga navedeno je i opisano nekoliko sličnih rješenja, te su navedene njihove glavne prednosti i nedostaci. Neki od glavnih nedostataka su gomila nefunkcionalnog sadržaja, kao i ne postojanje mogućnosti razmjene privatnih poruka. Također, većina rješenja nema obavijesti kojem bi se korisnika obavijestilo o novim aktivnostima nad njegovim profilom i sadržajem u aplikaciji. Aplikacija izrađena u ovom radu pokriva sve navedene nedostatke no, iako je većina korisnika dala izuzetno visoku ocjenu, mogla bi se unaprijediti poboljšanjem sigurnosnih značajki, ali i implementacijom mogućnosti filtriranja sadržaja te kreiranja grupa, što bi korisnicima pružilo još veću mogućnost povezivanja.

LITERATURA

[1] Yummypets, »Yummypets, the social petwork, community for pet lovers!,« [Mrežno]. Available: https://www.yummypets.com/. [Pokušaj pristupa 30, 6. 2024.].

[2] PetLoversChat, »PetLovers Chat,« [Mrežno]. Available: https://petloverschat.com/. [Pokušaj pristupa 30. 6. 2024.].

[3] U. Pets, »United Pets,« [Mrežno]. Available: https://www.unitedpets.net/#0. [Pokušaj pristupa 30. 6. 2024.].

[4] Bauwow, »Bauwow - Your Social Petwork,« [Mrežno]. Available: https://www.bauwowworld.com/. [Pokušaj pristupa 30. 6. 2024.].

[5] Dokonoko, »Dokonoko App,« [Mrežno]. Available: https://www.dokonoko.jp/en/. [Pokušaj pristupa 15. 9. 2024.].

[6] Altexsoft, »Functional and Nonfunctional Requirements: Specification and Types,« [Mrežno]. Available: https://www.altexsoft.com/blog/functional-and-non-functional-requirements-specification-and-types/. [Pokušaj pristupa 15. 9. 2024.].

[7] Altexsoft, »Nonfunctional Requirements in Software Engineering: Examples, Types, Best Practices,« [Mrežno]. Available: https://www.altexsoft.com/blog/non-functional-requirements/. [Pokušaj pristupa 19. 9. 2024.].

[8] Angular, »Lazy-loading feature modules,« [Mrežno]. Available: https://angular.dev/guide/ngmodules/lazy-loading. [Pokušaj pristupa 19. 9. 2024.].

[9] Simplilearn, »What Is Angular? A Guide to the Angular Framework,« [Mrežno]. Available: https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular. [Pokušaj pristupa 19. 9. 2024.].

[10] Altexsoft, »The Good and the Bad of Angular Development,« [Mrežno]. Available: https://www.altexsoft.com/blog/the-good-and-the-bad-of-angular-development/. [Pokušaj pristupa 19. 9. 2024.].

[11] Javatpoint, »Angular Material Tutorial,« [Mrežno]. Available: https://www.javatpoint.com/angular-material. [Pokušaj pristupa 19. 9. 2024.].

[12] Medium, »The MVC Architecture in Angular,« [Mrežno]. Available: https://eugeniucozac.medium.com/the-mvc-architecture-in-angular-9eecd586bb94. [Pokušaj pristupa 19. 9. 2024.].

[13] Cloudflare, »What is BaaS? | Backend-as-a-Service vs. serverless,« [Mrežno]. Available: https://www.cloudflare.com/learning/serverless/glossary/backend-as-a-service-baas/. [Pokušaj pristupa 19. 9. 2024.].

[14] Educative, »https://www.educative.io/answers/what-is-firebase,« [Mrežno]. Available: https://www.educative.io/answers/what-is-firebase. [Pokušaj pristupa 19. 9. 2024.].

[15] W3schools, »Window localStorage,« [Mrežno]. Available: https://www.w3schools.com/jsref/prop_win_localstorage.asp. [Pokušaj pristupa 19. 9. 2024.].

SAŽETAK

Vlasnici kućnih ljubimaca često dijele zanimljive trenutke i avanture svojih kućnih ljubimaca na društvenim mrežama. Kako bi im se omogućilo dijeljenje sadržaja, ali i razmjena savjeta i iskustava na jednom mjestu, u ovom radu izrađena je web aplikacija koja služi kao društvena mreža namijenjena samo vlasnicima kućnih ljubimaca. Aplikacija je razvijena korištenjem Angulara za frontend i Firebasea za backend, uključujući Firebase Authentication za prijavu i registraciju korisnika, Firestore za pohranu podataka, Storage za pohranu fotografija te Hosting za objavljivanje aplikacije. Implementirane su ključne funkcionalnosti poput kreiranja profila, dodavanja ljubimaca, dijeljenja objava, komentiranja i označavanja objava sa sviđanjem, mogućnosti pretraživanja i praćenja korisnika, razmjena poruka te mogućnost ocjenjivanja korisničkog iskustva. Rezultati korisnika su statistički obrađeni, a za grafički prikaz ocjena korisnika korištena je Chart.js biblioteka.

Ključne riječi: Angular, Chart.js, društvena mreža, Firebase, kućni ljubimci, web aplikacija

ABSTRACT

Web application for pet owners

Pet owners often share interesting moments and adventures of their pets on social media. To enable them to share content, as well as exchange advice and experiences in one place, this thesis presents a web application that serves as a social network specifically for pet owners. The application was created using Angular for the frontend and Firebase for the backend, including Firebase Authentication for user login and registration, Firestore for data storage, Storage for photo management, and Hosting for deploying the application. Key functionalities include profile creation, adding pets, posting content, commenting and liking posts, user search and following, messaging, and the ability to rate user experience. User ratings were statistically analyzed, and Chart.js was used for graphical representation of ratings.

Keywords: Angular, Chart.js, Firebase, pets, social network, web application

PRILOZI

Link na Github: https://github.com/NikolinaS1/PetApp