Vue aplikacija za prodaju i udomljavanje životinja

Duvnjak, Lora

Undergraduate thesis / Završni rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:842298

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-21**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Stručni studij

VUE APLIKACIJA ZA PRODAJU I UDOMLJAVANJE ŽIVOTINJA

Završni rad

Lora Duvnjak

Osijek, 2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1S: Obrazac za ocjenu završnog rada na stručnom prijediplomskom studiju

Ocjena završnog rada na stručnom prijediplomskom studiju

Ime i prezime pristupnika:	Lora Duvnjak
Studij, smjer:	Stručni prijediplomski studij Računarstvo
Mat. br. pristupnika, god.	AR 4721, 19.07.2019.
JMBAG:	0122229865
Mentor:	izv. prof. dr. sc. Ivica Lukić
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Mirko Köhler
Član Povjerenstva 1:	izv. prof. dr. sc. Ivica Lukić
Član Povjerenstva 2:	Miljenko Švarcmajer, univ. mag. ing. comp.
Naslov završnog rada:	Vue aplikacija za prodaju i udomljavanje životinja
Znanstvena grana završnog rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak završnog rada:	Izraditi Vue aplikaciju za prodaju i udomljavanje životinja gdje će korisnici prodavati ili udomljavati životinje, nuditi hranu za životinje i druge potrepštine. Prijavljeni korisnici mogu postaviti podatke o životinji i svoje podatke za kontakt. Provesti različite vrste testiranja aplikacije kako bi se osigurao ispravan rad. Rezervirano za: Lora Duvnjak

Datum ocjene pismenog dijela završnog rada od strane mentora:	31.08.2024.
Ocjena pismenog dijela završnog rada od strane mentora:	Vrlo dobar (4)
Datum obrane završnog rada:	18.9.2024.
Ocjena usmenog dijela završnog rada (obrane):	Vrlo dobar (4)
Ukupna ocjena završnog rada:	Vrlo dobar (4)
Datum potvrde mentora o predaji konačne verzije završnog rada čime je pristupnik završio stručni prijediplomski studij:	23.09.2024.

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

IZJAVA O IZVORNOSTI RADA

Osijek, 23.09.2024.

Ime i prezime Pristupnika:	Lora Duvnjak
Studij:	Stručni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	AR 4721, 19.07.2019.
Turnitin podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Vue aplikacija za prodaju i udomljavanje životinja**

izrađen pod vodstvom mentora izv. prof. dr. sc. Ivica Lukić

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 završnog rada 1
2. POSTOJEĆE WEB APLIKACIJE 2
2.1. Njuškalo 2
2.2. Index Oglasi 3
2.3. Gumtree 4
2.4. Petfinder 5
3. TEHNOLOGIJE KORIŠTENE ZA IZRADU APLIKACIJE 6
3.1. Visual Studio Code 6
3.2. HTML 6
3.3 CSS 7
3.4. Bootstrap 7
3.5. Vue.js 8
3.6. Firebase 9
4. FUNKCIONALNOST WEB APLIKACIJE 10
4.1. Izrada Vue.js projekta 10
4.2. Struktura projekta 10
4.2.1. Main.js 11
4.2.2. App.vue 11
4.2.3. Router.js 12
4.2.4. Store.js 13
4.3. Stranice aplikacije 14
4.3.1. Naslovna stranica 14
4.3.2. Stranice „Cats“ i „Dogs“ 17
4.3.3. Stranica za detalje o oglasu 22
4.3.4. Stranice za autentifikaciju 25
4.3.5. Stranica za predaju oglasa 30
4.3.6. Korisnikov profil 36
5. ZAKLJUČAK 46
Literatura 47
SAŽETAK 48
ABSTRACT 49

1. UVOD

U današnje vrijeme sve više ljudi posjeduje kućne ljubimce, te se izradom ove web aplikacije želi olakšati pregledavanje oglasa korisnicima koji žele udomiti ili kupiti životinju. Pošto se u današnje vrijeme više ljudi koristi internetom i većina oglasa se nalazi na internetu umjesto kao prije u novinama ili preko radija, oglašavanje putem interneta je lakše nego oglašavanje u novinama jer će ljudi prije potražiti oglase preko interneta nego u novinama, što je puno lakše, i tako će više ljudi vidjeti te oglase. Zbog lakoće koju Internet pruža osobe koje se bave uzgojem čistokrvnih životinja mogu besplatno objaviti svoj oglas za prodaju na ovoj web aplikaciji uz izradu korisničkog računa kako bi mogli objaviti oglas, te osobe koje žele staviti životinju na udomljavanje isto tako mogu to napraviti. Iako već postoje neke stranice za oglašavanje, na njima se mogu stavljati bilo kakvi oglasi, a ne samo za životinje stoga je izrađena ova web aplikacija koja olakšava korisnicima koje samo zanimaju oglasi o životinjama pregledavanje oglasa o njima.

Prvo se opisuju postojeće slične web aplikacije koje imaju isti cilj kao i ova aplikacija, zatim se predstavljaju tehnologije koje su bile korištene za izradu ove web aplikacije, te se a kraju opisuju funkcionalnosti i izgled aplikacije.

1.1. Zadatak završnog rada

Zadatak završnog rada je izraditi web aplikaciju preko koje se korisnici koji žele objaviti oglas moraju registrirati i prije objave ulogirati. Pri objavi oglasa moraju postaviti sliku životinje, podatke o njoj, te podatke za kontakt. Korisnici koji nisu logirani imaju mogućnost slobodnog kretanja po aplikaciji, te pregledavanja oglasa.

2. POSTOJEĆE WEB APLIKACIJE

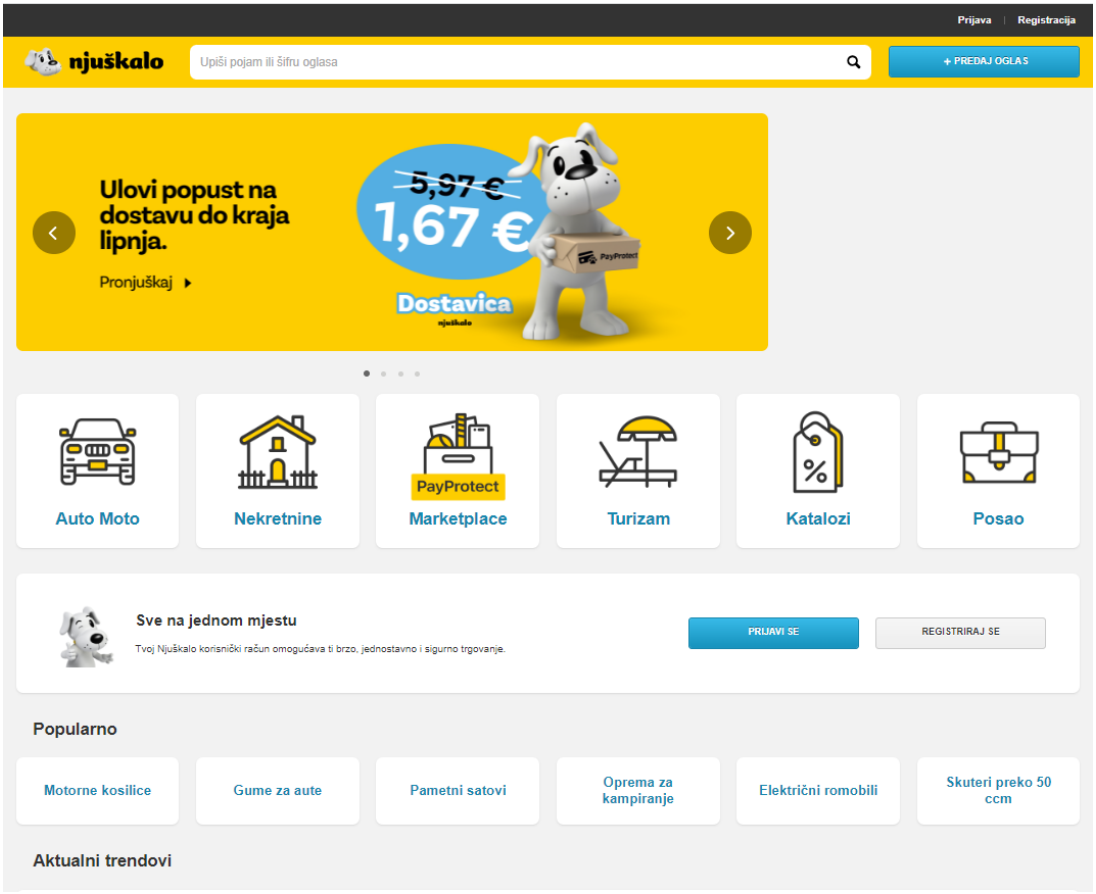
Web aplikacija je interaktivni računalni program kojem se pristupa preko web preglednika putem interneta. Za razliku od web stranica, koje su većinom informativnog sadržaja i pružaju posjetitelju čitanje sadržaja te poneku interakciju putem ispunjavanja forme ili komentara, web aplikacije su interaktivne i omogućuju izvođenje zadataka, obradu i manipulaciju podacima te su usmjerene na interakciju s korisnikom. Radi na principu klijent-poslužitelj. Klijent se odnosi na korisnička sučelja preko kojih korisnici rukuju aplikacijom, a poslužitelj se odnosi na server na kojemu se nalazi aplikacija i na kojemu se obrađuju i pohranjuju podaci.

Primjeri predstavljenih web aplikacija s istom ili sličnom svrhom su:

- Njuškalo
- Index Oglasi
- Gumtree
- Petfinder

2.1. Njuškalo

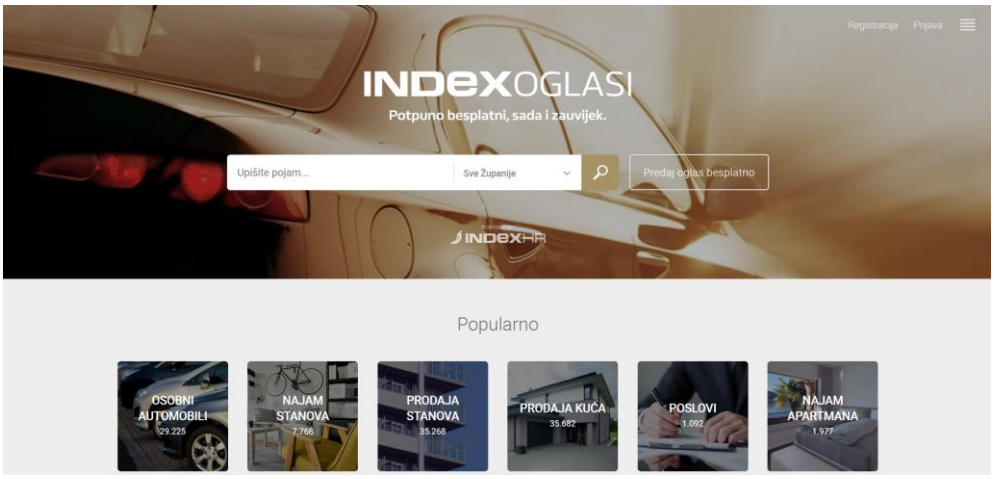
Njuškalo [1] je jedna od najpoznatijih hrvatskih stranica za oglašavanje. Nastalo je 2007. godine i od tada ima 1,5 milijuna aktivnih korisnika, te dnevno primi 10 tisuća novih oglasa. Naslovna stranica Njuškala se sastoji od prijave i registracije koje se nalaze u desnom vrhu, ispod toga se u lijevom kutu nalazi logo te pored njega tražilica za pretraživanje oglasa. Ispod tražilice se nalazi *carousel* s pomičnim slikama koji nije centriran nego je smješten ulijevo. Sredina stranice je ispunjena određenim kategorijama za lakšu navigaciju, pojmovima koji su trenutno popularni, aktualnim trendovima, te "Super Vau" oglasima. Na samom dnu stranice se nalazi *footer* u kojem se nalaze određene informacije o Njuškalu.



Slika 2.1. Naslovna stranica Njuškala

2.2. Index Oglasi

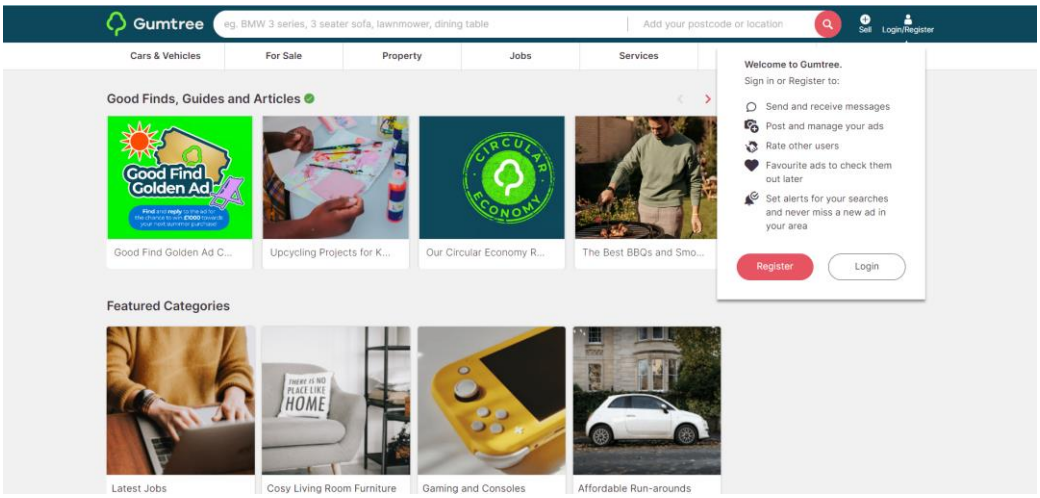
Index Oglasi [2] su također jedna od poznatijih hrvatskih stranica za besplatno oglašavanje. Na njihovoj stranici se prijava i registracija također nalaze u desnom vrhu pored kojih se nalazi *hamburger* koji se pritiskom otvara prema lijevo prikazujući kategorije oglasa za lakše snalaženje na stranici. Pri vrhu u sredini se nalazi logo i traka za pretraživanje s pomoću koje se mogu odabrati i županije ako se oglas traži na određenoj lokaciji. Ispod toga se popularni pojmovi. Listanjem niže se tražilica, logo, prijava i registracija pretvaraju u navigacijsku traku (eng. Navbar). Pri sredini stranice se nalaze iste kategorije koje se nalaze pritiskom na *hamburger*. Na dnu stranice se nalazi najnoviji oglasi i podnožje (eng. Footer).



Slika 2.2. Naslovna stranica Index Oglasa

2.3. Gumtree

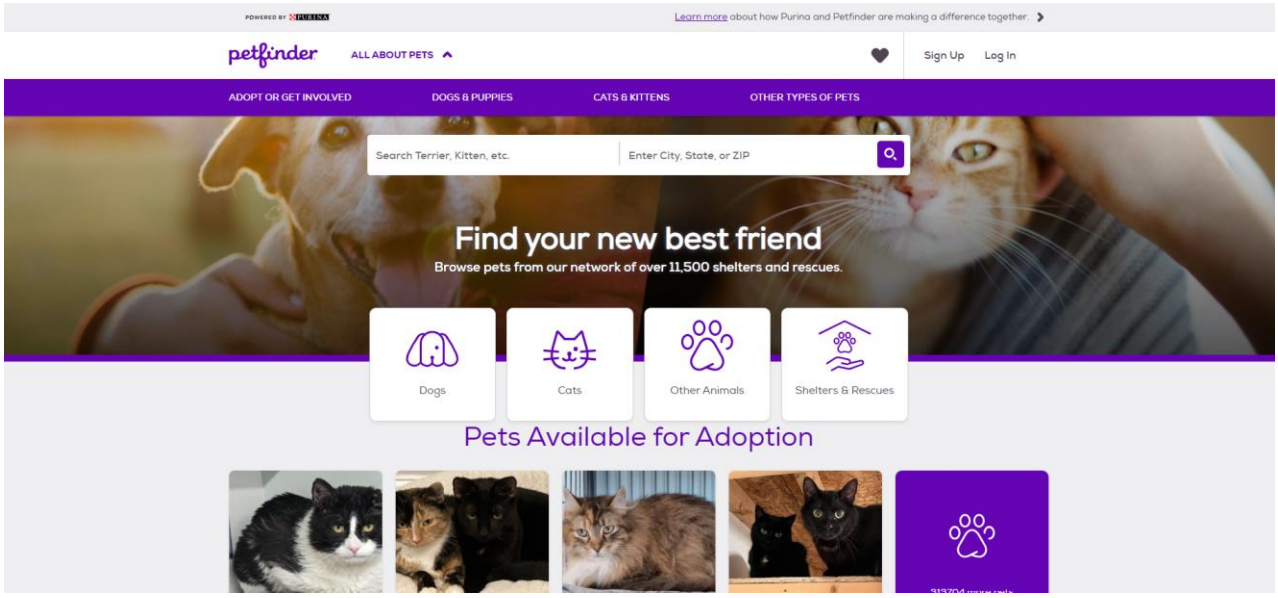
Gumtree [3] je britanski oglasnik preko kojega se mogu objavljivati besplatni ili plaćeni oglasi ovisno o kategoriji proizvoda. Njihova stranica također prati općeniti izgled s logom na vrhu u lijevom kutu, tražilice pored loga te prijave i registracije u desnom vrhu pored tražilice uz opciju za postavljanje oglasa. Ispod toga se nalazi *navbar* na kojem se prelaskom miša preko kategorija otvara *dropdown* s više opcija. Uz sve to pri dolasku na stranicu iskače prozorčić koji nudi prijavu ili registraciju uz predstavljanje nekih od funkcionalnosti aplikacije kao što je slanje i primanje poruka. Sredina stranice je ispunjane raznim oglasima, a dno sadrži *footer* i izbor gradova Velike Britanije kako bi korisnici mogli filtrirati oglase blizu svoje lokacije.



Slika 2.5. Početna stranica Gumtreeja

2.4. Petfinder

Petfinder [4] je najveća web aplikacija za udomljavanje kućnih ljubimaca koja obuhvaća Sjevernu Ameriku. Na stranici se nalazi više od 315 000 oglasa od kojih je 14 000 iz aktivnih skloništa za životinje. Kao i na ostalim primjerima na Petfinder stranici prijava i registracija se mogu naći na desnom vrhu, logo je u lijevom kutu te se ispod nalazi *navbar* s kategorijama koje prilikom prelaska mišem otvaraju *dropdown* s još više opcija. Uz tražilicu se nalazi i opcija da se unese grad, poštanski broj ili savezna država kako bi se suzilo područje gdje se nalaze. Sredina stranice je također kao i u prijašnjim primjerima popunjena oglasima te se na dnu nalazi *footer* s određenim informacijama o stranici.



Slika 2.3. Naslovna stranica Petfindera

3. TEHNOLOGIJE KORIŠTENE ZA IZRADU APLIKACIJE

Za izradu bilo kakvih aplikacija potrebni su *frontend* i *backend*. Oni se razlikuju po tome što se *frontend* bazira na korisničkom iskustvu i elementima koje korisnik vidi na sučelju kao što su boje, slike, obrasci, gumbi te drugih vizualnih elemenata uz korištenje tehnologija kao što su HTML (engl. *Hypertext Markup Language*) , CSS (engl. *Cascading Style Sheets*) i JavaScript, dok se *backend* sastoji od svega što korisnik ne može vidjeti kao što su baze podataka, servera te pozadinske logike. *Backend* upravlja cjelokupnom funkcionalnošću aplikacije, obradom, dohvaćanjem i pohranjivanjem podataka koji se pošalju s *frontenda* korištenjem tehnologija kao što su Java, Python, Ruby, PHP, C#, Go itd.

3.1. Visual Studio Code

Visual Studio Code [5] je Microsoftov besplatni uređivač koda koji ima podršku za programske jezike kao što su Java, JavaScritp, Python, C++, C# itd. Uključuje značajke kao što su inteligentno dovršavanje koda, isticanje sintakse, podršku za otklanjanje pogrešaka (debugging) i ugrađeni Git. Također omogućuje korisnicima mijenjanje teme (npr. svijetla ili tamna), dodavanje prečaca na tipkovnici, dodavanje ekstenzija kao što su poravnanje koda itd.

3.2. HTML

HTML [6] tj. HyperText Markup Language je standardni jezik korišten za izradu web stranica. On nije programski jezik jer ne može izvršavati nikakve matematičke operacije niti zadatke već definira sadržaj i strukturu web sadržaja u obliku HTML dokumenta. Ti dokumenti se sastoje od HTML elemenata kao što su <!DOCTYPE html> koji označava da se radi o HTML dokumentu, <html> koji označava početak dokumenta, <head> u kojemu se obično postavlja naslov stranice koristeći elemente <h1>,<h2>,<h3>... također se može dodati CSS za ljepši izgled i skripte za funkcionalnost elemenata izrađene u JavaScriptu. Ispod <head> unutar <html> se dodaje <body> gdje se nalazi sav sadržaj izrađen od HTML elemenata kao što su , , <a> za linkove, <img> za slike, <p> za paragrafe, <ul> za liste, <li> za elementa te liste, <button> za gumbove te mnogih drugih. Svaki taj element završava kosom crtom ispred riječi elementa npr. .

3.3 CSS

CSS [7] ili Cascading Style Sheet je stilski jezik koji se koristi za uređivanje izgleda HTML dokumenta. Može se pisati unutar HTML tagova , ispod HTML dokumenta unutar <style></style> elemenata, te se može pisati u posebnom dokumentu s ekstenzijom .css koji se onda poziva unutar <link> elementa kojeg postavljamo u <head> element.

```
<p style="color:green;">test</p>
```

Slika 3.1. Primjer CSS-a unutar HTML tagova

CSS pišemo s pomoću nekoliko pravila kao što su selektor kojim određujemo koji element želimo urediti. Selektori mogu biti imena HTML elemenata kao što su *p, a, body, nav, div* itd. Također mogu biti imena klasa koju smo odredili unutar određenih elemenata, te mogu biti ID selektori koji se koriste za jedan element. Kako bi pristupili klasi ispred imena klase dodajemo točku, a kod ID-ja dodajemo ljestve #. Svojstva selektora se pišu unutar vitičastih zagrada {}, nakon svakog svojstva pišemo dvotočku : zatim vrijednost koju želimo pridodati tom svojstvu te na kraju točku sa zarezom ; kako odvojili svojstvo od drugih unutar selektora.

```
ul {
  list-style: none;
  padding: 0;
  display: flex;
  align-items: center;
}

.logo {
  margin-right: auto;
}
```

Slika 3.2. Primjer CSS selektora, svojstva i vrijednosti

3.4. Bootstrap

Bootstrap [8] je besplatni popularni front-end framework dizajniran kako bi omogućio lakšu izradu responzivnih web stranica i web aplikacija. Sastoji se od predefiniranih komponenti koje omogućuju web programerima lakše i brže stiliziranje stranice. Neki od tih komponenti su navigacijske trake, gumbovi, padajući izbornici itd.

3.5. Vue.js

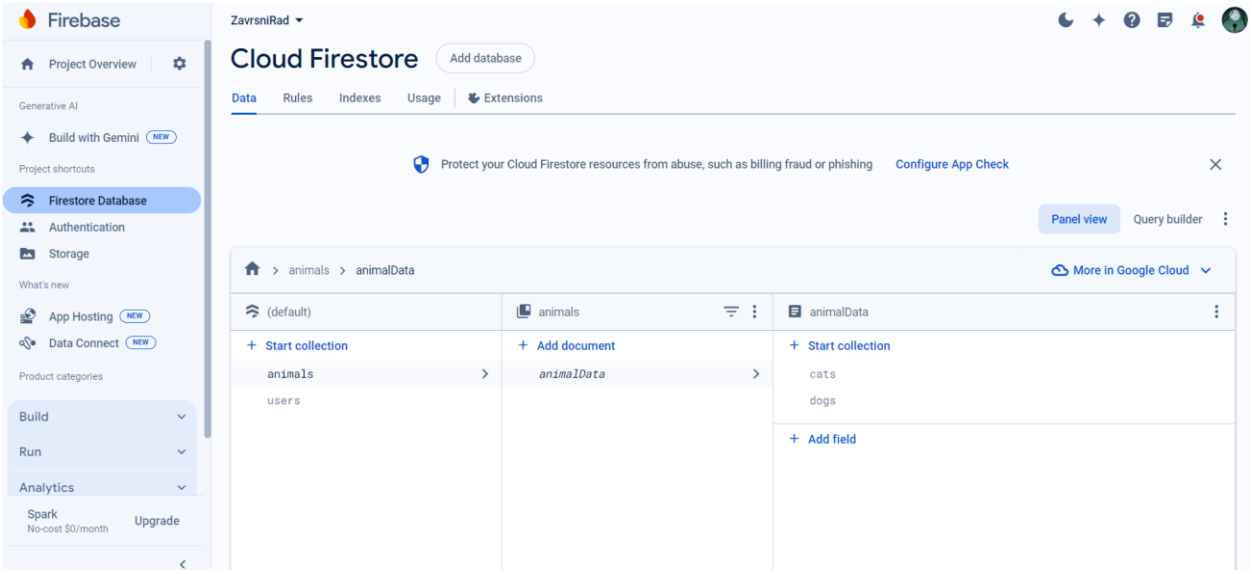
Vue.js [9] je JavaScript framework koji se koristi za izradu korisničkog sučelja i jednostraničnih aplikacija (Single Page Applications). Jednostranične aplikacije omogućuju korisniku dinamičko kretanje po aplikaciji s pomoću tzv. routera gdje se registriraju stranice koje se žele prikazati. Sav sadržaj se učitava iz jedne „index.html“ datoteke umjesto iz više HTML datoteka te se zbog toga aplikacija učitava brzo. Vue je izrađen kako bi olakšao i ubrzao razvijanje aplikacija gdje se koristi JavaScript, usmjeren je na sloj prikaza i omogućava developerima da prave aplikacije uz pomoć komponenata. Komponente se sastoje od dijela za template gdje se nalazi HTML, dijela za JavaScript i style dijela za CSS. One su predviđene kao dijelovi aplikacije koji se mogu ponovno koristiti, npr. komponenta za izgled gumba kojeg ćemo koristiti u više dijelova aplikacije ili za nekakav padajuću izbornik koji nam treba na više mjesta. Glavna roditeljska komponenta se zove App.vue u koju dodajemo komponente za koje želimo da se pojave na svakoj stranice. Sve ostale komponente su komponente djeca.

```
<template>

    <div class="user-profile-card" v-if="show">
        <div class="user-profile-card-overlay" @click="close"></div>
        <div class="user-profile-content">
            <header>
                <h3>{{ title }}</h3>
                <button class="close-button" @click="close">X</button>
            </header>
            <div class="user-profile-info">
                0 references
                <slot></slot>

            </div>
        </div>

    </div>

</template>
```

Slika 3.3. Primjer Vue template-a u komponenti za prikaz korisničkog profila

3.6. Firebase

Firebase [10] je platforma napravljena od strane Googlea koja pruža razne backend usluge koje pomažu programerima u razvoju aplikacija. Neke od mogućnosti na Firebaseu su Real-time Database gdje se podaci spremaju u JSON formatu i sinkroniziraju u stvarnom vremenu, zbog toga je ovakva baza podataka pogodna za aplikacije koje trebaju izvršavanje radnja u stvarnom vremenu. Osim Real-time baze podataka postoji i Firestore [11] baza podataka koja također nudi podršku u stvarnom vremenu, ali i izvan mreže. Osim te dvije baze podataka postoji i Firebase Authentication [12] preko kojeg se može rukovati korisnikovim podacima za prijavu, i omogućuje prijavu putem e-maila, lozinki, telefonskog broja Googlea itd. Još jedna od mogućnosti koje Firebase pruža je Firebase Storage [13] gdje se spremaju fotografije ili videozapisi koje su korisnici učitali.



Slika 3.4. Prikaz Firebase sučelja

4. FUNKCIONALNOST WEB APLIKACIJE

U ovom poglavlju će biti opisani koraci izrade web aplikacije za udomljavanje i prodaju mačaka i pasa koja je osmišljena kako bi olakšala postavljanje oglasa na stranicu koja je fokusirana samo na oglase za kućne ljubimce te omogućila lakše pronalaženje kućnog ljubimca. Registrirani korisnici mogu postavljati oglase i uređivati iste, a gosti stranice, tj. korisnici koji nemaju profil mogu samo gledati oglase. Kod svake stranice bit će opisan dizajn, te zatim funkcionalnosti koje sadrži.

4.1. Izrada Vue.js projekta

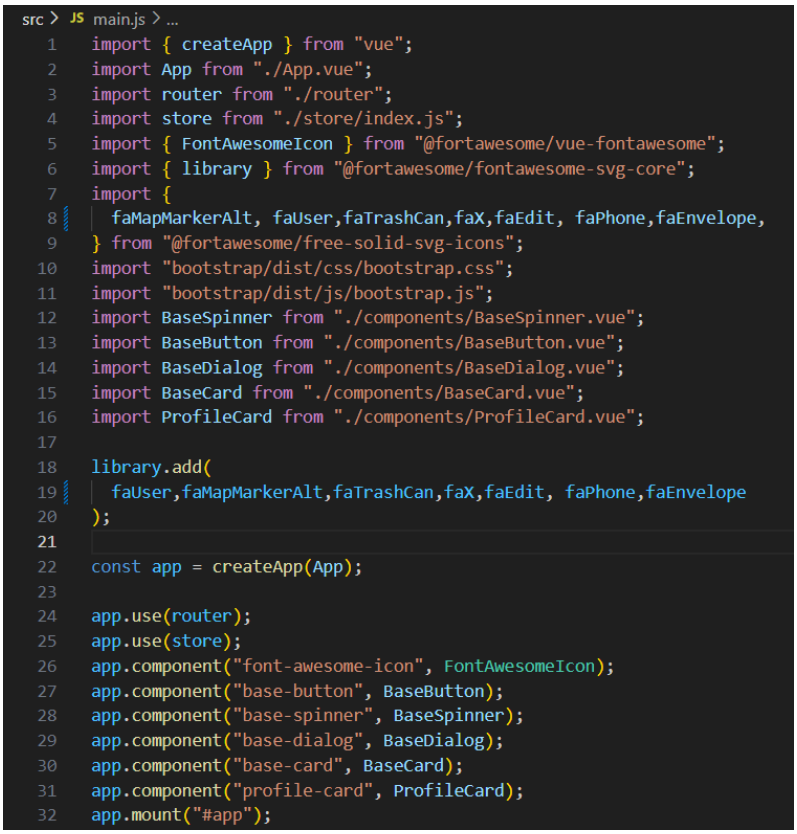
Prije početka pisanja koda moramo instalirati Node.js tj. Vue CLI kako bi mogli koristiti *npm* naredbu. Sa službene stranice Node.js-a skinemo verziju za, u ovom slučaju, Windows, u terminalu ili command prompt-u pokrenemo naredbu *npm install @vue/cli –g* i globalno ga instaliramo. Nakon instalacije pokrenemo naredbu *vue create ime-projekta* te mu proizvoljno damo ime. Kako bi mogli početi raditi na projektu pozicioniramo se u direktorij u kojem se nalazi projekt naredbom „*cd ime-projekta*“. Pri otvaranja projekta na webu pokrenemo dvije naredbe: *npm run build* kako bi izgradili (eng. Build) projekt, te *npm run serve* koji pokreće razvojni server kojemu možemo pristupiti preko *http://localhost:8080* URL-a gdje možemo vidjeti projekt.

4.2. Struktura projekta

Glavni dio koda se nalazi u *src* folderu. Unutar njega nalaze se folderi *components* koji sadrži komponente koje se mogu ponovno koristiti bilo gdje u projektu, zatim folder *pages* u kojem se nalaze sve stranice projekta, i na kraju folder *store* u kojem se nalazi Vuex koji služi kao spremište za sve komponente u aplikaciji. Osim foldera unutar *src*-a se također nalazi ulazna točka aplikacije *main.js*, korijenska komponenta *App.vue*, *firebase.js* preko kojega se spajamo na Firebase i *router.js* gdje su postavljene sve rute koje želimo koristiti.

4.2.1. Main.js

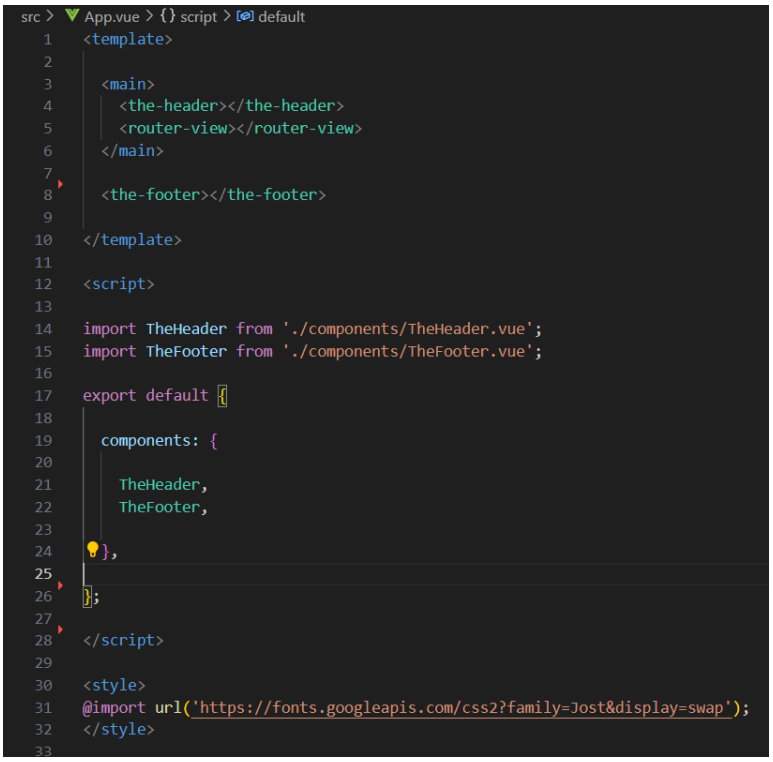
Datoteka main.js inicijalizira Vue instancu, postavlja router, dodaje Vuex i postavlja aplikaciju na DOM (Document Object Model). To je mjesto gdje registriramo komponente, dodajemo globalne dodatke, biblioteke ili ikonice.



Slika 4.1. Prikaz main.js datoteke

4.2.2. App.vue

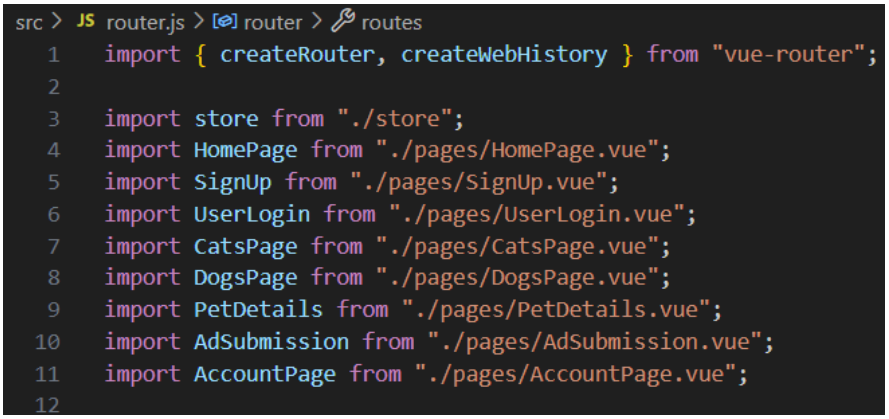
App.vue je korijenska komponenta aplikacije. U njoj se nalaze komponente koje se pojavljuju na svakoj stranici u aplikaciji kao što su navigacijska traka <the-header>, footer <the-footer>, te komponenta za rute <the-router> preko koje usmjeravamo korisnika kada klikne link na neku drugu stranicu u aplikaciji. Uobičajena praksa pisanja imena komponenata je da se ime sastoji od dvije riječi i crtice između. Također ovdje možemo dodati izgled fonta kojeg želimo da se koristi u cijeloj aplikaciji.



Slika 4.2. Prikaz App.vue datoteke

4.2.3. Router.js

U router.js datoteci registriramo sve rute koje želimo da su dostupne za kretanje po stranici. Osim *path* preko kojeg definiramo URL koji će se koristiti za rutu možemo dodati i opcionalno ime za rutu, što nam može pomoći u kodu, zatim možemo specificirati koja će se komponenta prikazati kada se ruta aktivira i možemo, ako su potrebni, proslijediti tzv. Svojstva (eng. Props) određenoj komponenti.



Slika 4.3. Prikaz dodavanja ruta u router.js

Slika 4.4. Prikaz definiranih ruta

4.2.4. Store.js

Store.js je središnji dio aplikacije kod korištenja Vuexa koji je biblioteka za upravljanje stanjem Vuea. Sastoji se od četiri ključne komponente; State, Getters, Mutations i Actions. Unutar obrnutog stanja (eng. State) komponente definiramo podatke kojima želimo globalno upravljati u aplikaciji. To može uključivati bilo koje podatke koje trebamo dijeliti među više komponenata. Dohvatitelji (eng. Getters) su funkcije preko kojih možemo pristupiti nekim svojstvima stanja na primjer getter za dohvaćanje životinje po ID-ju. Oni su samo za čitanje i ne mogu mijenjati podatke. Mutacije (eng. Mutations) su sinkrone funkcije koje izravno mijenjaju stanje, te su one jedini način da se promjeni stanje u Vuexu. Obično se pokreću preko radnje (eng. Actions) koji su funkcije koje mogu sadržavati asinkrone operacije. Njih se koristi za složene operacije i rukovanje logikom. Osim te četiri komponente nalazi se još jedna pod imenom index.js gdje se upravlja globalnim stanjem aplikacije kao što su provjera autentičnosti i podaci o životinjama.

Slika 4.5. Prikaz index.js gdje su postavljena dva modula; animals i users

Na slici 4.5. prikazan je index.js u koji se iz Vuex biblioteke uvozi *createStore*, zatim se uvoze dva modula koji rukuju stanjem, mutacijama, radnjama i dohvatiteljima za životinje i za korisnike. Oni su odvojeni u posebne module radi lakšeg rukovanja Vuex pohranom (eng. Vuex Store) jer svaki modul upravlja vlastitim stanjem, mutacijama, radnjama i dohvatiteljima koji se zatim spajaju u korijensku pohranu (eng. Store). Nakon toga *createStore* se poziva stvaranjem nove instance Vuex pohrane koja je konfigurirana modulima koji su uvezeni. Na kraju se izvozi instanca pohrane kako bi se mogla uvesti i koristiti u glavnoj Vue.js aplikaciji.

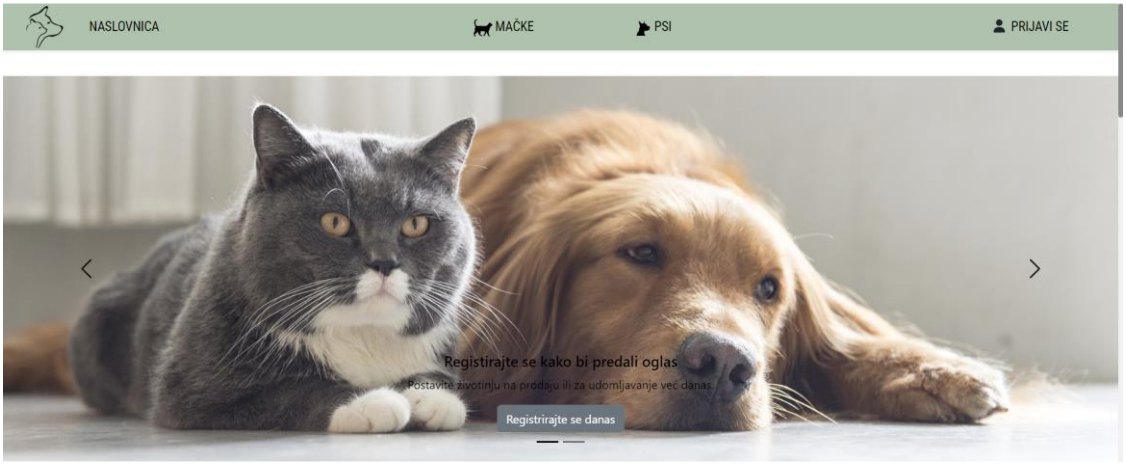
4.3. Stranice aplikacije

Aplikacija se sastoji od osam stranica gdje za svaku postoji ruta preko koje dođemo do određene stranice.

4.3.1. Naslovna stranica

Kada otvorimo aplikaciju preko localhost-a prva stranica koju vidimo je naslovna stranica to jest HomePage.vue. Stranica je napravljena da bude informativna i da se s nje može navigirati na druge stranice. Na njoj se nalazi navigacijska traka preko koje se možemo navigirati na stranicu za mačke *Mačke*, stranicu za pse *Psi* te se možemo logirati klikom na *Prijavi se* koji nas preusmjerava na stranicu gdje upisujemo podatke za prijavu. Pošto se ova navigacijska traka nalazi

na svakoj stranici aplikacije, klikom na *Naslovnica* možemo se vratiti na naslovnu stranicu. Ispod naslovne stranice se nalazi tzv. Bootstrapov klizač (eng. Carousel) koji se sastoji od dvije slike. Na prvoj postoji gumb koji nas preusmjerava na stranicu za izradu korisničkog profila, a na idućoj se opet nalaze dva linka za stranice *Mačke* i *Psi*. Nakon toga se pojavljuje par odjeljaka kao što su odjeljak koji bi korisnicima mogao pomoći odlučiti paše li više njihovom načinu života mačka ili pas, zatim odjeljak za savjete za brigu o kućnim ljubimcima koji bi mogao biti koristan za nove vlasnike, te odjeljak za zdravlje kućnih ljubimaca. Na dnu stranice se nalaze izjave zadovoljnih korisnika stranice koji su pronašli svog novog kućnog ljubimca i podnožje (eng. Footer).



Slika 4.6. Prikaz sučelja navigacijske trake i klizača

Slika 4.7. Prikaz sučelja odjeljka koji korisniku može pomoći u odabiru mačke ili psa

Slika 4.8. Prikaz sučelja odjeljka za savjete za brigu o kućnim ljubimcima

Slika 4.9. Prikaz sučelja odjeljka za brigu o zdravlju kućnih ljubimaca

Slika 4.10. Prikaz sučelja korisničkih izjava i podnožje

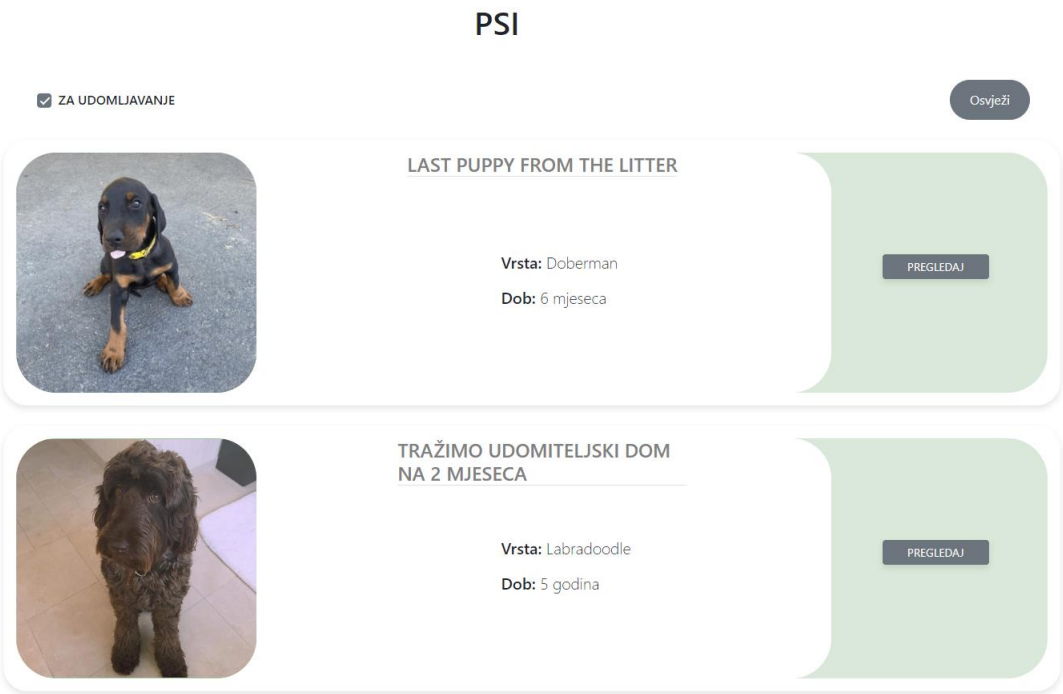
Stranica HomePage.vue se sastoji od <template> oznaka (eng. Tag) u kojim se nalaze djeca komponente poredane po redu kako želimo da se prikazuju na stranici. Unutar <script> oznaka uvozimo te komponente iz njihovih lokacija u projektu i registriramo ih pod *components*. Ovakav način pisanja koda omogućava da on bude čist, čitljiv i lakše održiv.

Slika 4.11. Prikaz koda datoteke HomePage.vue

4.3.2. Stranice „Mačke“ i „Psi“

Na stranicama za mačke i pse nalaze se filter za filtriranje oglasa mačaka tj. pasa koji su za udomljavanje tj. na prodaju. Na desnoj strani nalazi se i gumb za osvježavanje oglasa. Oglasi su poredani jedan ispod drugog i napravljeni su od komponente na kojoj se nalazi slika, naslov, vrsta mačke ili psa, starosti i cijene ako je životinja na prodaju, te gumba koji nas vodi na detaljnije podatke o oglasu. Na stranici o određenom oglasu, do koje dolazimo preko id parametra iz rute, se nalaze dodatni podaci o životinji kao što su još slika, opis, lokacija, broj za kontakt i ime korisnika koji je objavio oglas, također možemo kliknuti na ime tog korisnika kako bi otvorili prozorčić u kojem se nalaze dodatni podaci o tom korisniku kao što je e-mail, slika, lokacija i kontakt broj.

Slika 4.12. Prikaz sučelja oglasa za prodaju na stranici *Mačke*



Slika 4.13. Prikaz sučelja oglasa za udomljavanje na stranici *Psi*

Slika 4.14. Prikaz sučelja detalja za oglas

Kako bi odredili hoće li se na stranici pokazati oglasi za pse ili mačke treba nam *prop*, kratica za *property* tj. svojstvo kojeg dobivamo iz usmjerivača (eng. router) ovisno idemo li preko rute /cats ili /dogs, osim što prikazujemo mačke ili pse na stranici se ispiše naslov ovisno o ruti.

```
export default {
    props: {
        animalType: {
            type: String,
            required: true,
        },
    },
```

Slika 4.15. Prop *animalType* preko kojeg prikazujemo mačke ili pse

```
<h1 class="page-title">{{ animalType === 'cat' ? 'MAČKE' : 'PSI' }}</h1>
```

Slika 4.16. Linija koda u predlošku (eng. Template) gdje se provjerava treba li se ispisati MAČKE ili PSI

```
{
  path: "/dogs",
  name: "Dogs",
  component: DogsPage,
  props: { animalType: "dog" },
},
{
  path: "/cats",
  name: "Cats",
  component: CatsPage,
  props: { animalType: "cat" },
},
```

Slika 4.17. *Props* u router-u za pripadajuću rutu

Klikom na gumb za osvježenje stranice pojavljuje se pokazivač učitavanja (eng. Spinner) koji se pokreće metodom *refreshAnimals* i koji se vrti sve dok se ne dohvate oglasi životinja. Ako nešto pođe krivo ispiše se poruka greške. U metodi *refreshAnimals* učitavanje tj. *isLoading* se postavi na *true* dok se ne učitaju oglasi, zatim se pokušavaju učitati oglasi radnjom *loadAnimals*, ako sve prođe u redu *isLoading* se postavi na *false* kako bi se spinner maknuo i oglasi prikazali. Metoda *refreshAnimals* se također poziva pri prvom ulasku na stranicu preko kuke *created* (eng. Hook).

```
async refreshAnimals() {
    this.isLoading = true;
    try {
        await this.loadAnimals();
    } catch (error) {
        this.error = error.message || 'Something went wrong.';
    }
    this.isLoading = false;
},
```

Slika 4.18. Metoda *refreshAnimals*

```
<div class="refresh-btn">
    <base-button mode="outline" @click="refreshAnimals">Osvježi</base-button>
</div>
```

Slika 4.19. Gumb s metodom *refreshAnimals*

```
async loadAnimals({ commit }) {
  try {
    const animalDataDocRef = doc(db, "animals", "animalData");

    const catsSnapshot = await getDocs(collection(animalDataDocRef, "cats"));
    const dogsSnapshot = await getDocs(collection(animalDataDocRef, "dogs"));

    const cats = catsSnapshot.docs.map((doc) => ({
      id: doc.id,
      ...doc.data(),
    }));
    const dogs = dogsSnapshot.docs.map((doc) => ({
      id: doc.id,
      ...doc.data(),
    }));

    commit("setCats", cats);
    commit("setDogs", dogs);
  } catch (error) {
    console.error("Error loading animals:", error.message);
  }
},
```

Slika 4.20. Asinkrona radnja *loadAnimals* iz *animals/actions.js*

Slika 4.20. prikazuje radnju preko koje učitavamo životinje iz Firebasea. U try-catch bloku, koji se koristi za asinkrone operacije, referenciramo *animalDataDocRef* na dokument *animalData* u kolekciji *animals* u Firestore bazi podataka. Zatim u *catsSnapshot* i *dogsSnapshot* preko *getDocs* dohvaćamo sve dokumente u određenoj kolekciji imena *cats* ili *dogs*. (collection(animalDataDocRef, „cats“)) stvara referencu na kolekciju *cats* ugniježđenu unutar dokumenta animalData. Isto to radi i za kolekciju *dogs*. Polje dokumenata dohvaćeno preko *catsSnapshot.docs* se sprema u konstantu *cats*, i svaki dokument se transformira u objekt sa svojim id-om i ostalima poljima podataka. doc.id je jedinstveni identifikator dokumenta, a doc.data dohvaća sve podatke pripadajućeg dokumenta. Isto tako slijedi i za kolekciju pasa. Preko commit se polje *cats* predajue Vuex pohrani koristeći mutaciju *setCats*. Na kraju ako se dogodi pogreška, ona se ispisuje u konzolu. Podaci koje smo dohvatili iz Firebstorea se mogu ispisati na stranici, u ovom slučaju na stranici gdje su svi oglasi samo se ispsuje naslov, vrsta životinje i dob, te cijena ako je životinja na prodaju, dok se slika učitava iz Firebase Storagea.

4.3.3. Stranica za detalje o oglasu

Kao što je već rečeno, do detalja određenog oglasa se dolazi preko parametra id iz trenutne rute, dohvaća ga *routeId* computed property (računato svojstvo). Kako bi dohvatili životinju iz Vuex pohrane koristimo metodu *fetchAnimals* kojoj kod poziva predajemo id koji se može napisati i kao *this.routeId*. Ta se metoda poziva u lifecycle hook-u *created* koristeći this.routeId. Unutar *fetchAnimals* metode pokušava se dohvatiti životinja iz pohrane koristeći *getAnimalById* getter iz modula *animals*, ako životinja nije pronađena poziva se radnja *loadAnimals* koja učitava sve životinje iz Firestore-a i pokušava ponovno dohvatiti traženu životinju. Ako je pronađena postavlja *selectedItem* za tu životinju i ažurira računata svojstva, kao što su naslov, cijena, opis itd., prema vrijednosti iz *selectedItem.*

```
created() {
    if (this.routeId) {
        this.fetchAnimal(this.routeId);
    } else {
        console.error('Route ID is undefined on created');
    }
},
```

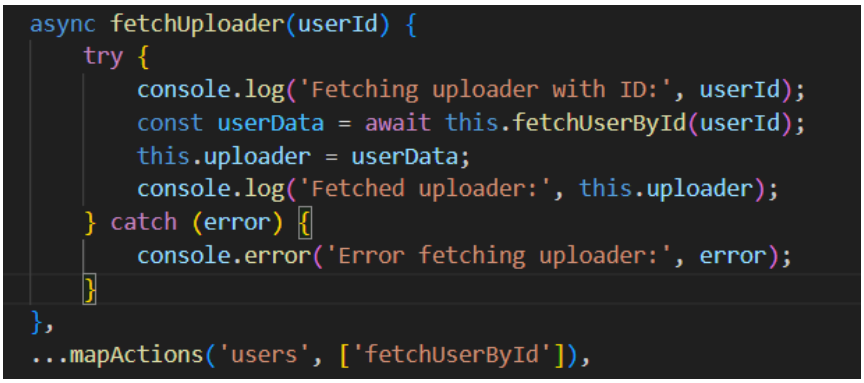
Slika 4.21. Created lifecycle kuka koja se pokreće kada se komponenta napravi

```
methods: {
    async fetchAnimal(id) {
        let animal = this.getAnimalById(id);
        if (!animal) {
            console.log('Animal not found in store, loading animals from Firestore...');
            await this.$store.dispatch('animals/loadAnimals');
            animal = this.getAnimalById(id);
        }
        if (animal) {
            this.selectedItem = animal;
            console.log('Fetched animal:', this.selectedItem);
            if (animal.userId) {
                console.log('Animal has userId:', animal.userId);
                await this.fetchUploader(animal.userId);
            } else {
                console.error('Animal does not have a userId');
            }
        } else {
            console.error('Animal not found after loading from Firestore.');
        }
    },
```

Slika 4.22. Metoda *fetchAnimals* uz predani id

Slika 4.23. Computed properties tj. računata svojstva koja se ažuriraju kada se životinja dohvati

Osim što dohvaćamo životinju, ako u njenim poljima podataka postoji *userId*, dohvaćamo i korisnika koji je postavio oglas životinje. Slično kao i za životinju imamo metodu za dohvaćanje korisnika *fetchUploader* kojoj predajemo *userId* te se u njoj dohvaćaju podaci o korisniku preko radnje fetch*UserById*. Kako bi radnje iz bilo kojeg modula bile dostupne moramo ih mapirati preko ...*mapActions* gdje zatim pišemo ime modula i ime radnje koju želimo koristiti kao metodu.

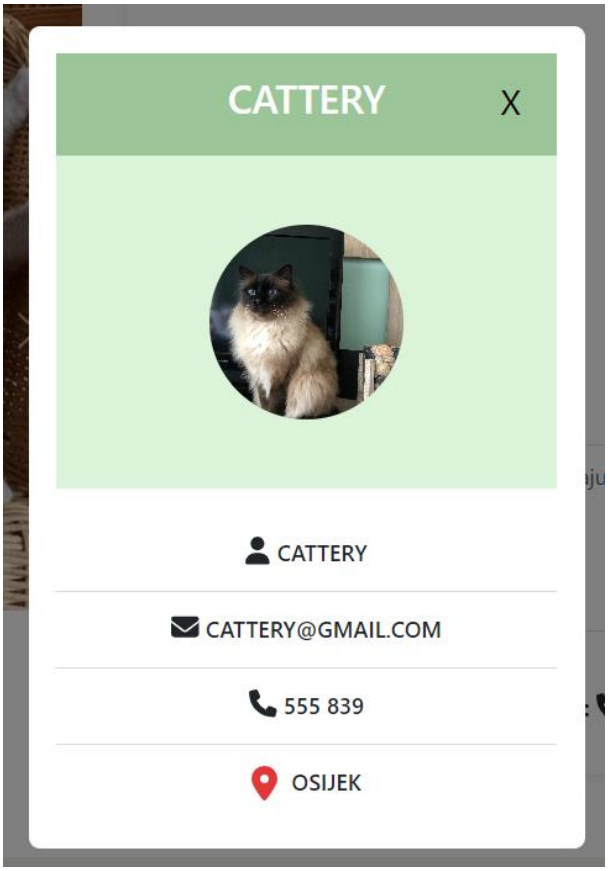


Slika 4.24. Metoda *fetchUploader*

Dohvaćeni podaci o korisniku se mogu prikazati klikom na korisnikov profil gdje se otvara komponenta <profile-card> s podacima o korisniku kao što su njegovo ime profila, slika, e-mail, broj za kontakt i lokacija.

```
uploaderUsername() {
    return this.uploader ? this.uploader.username : "";
},
email() {
    return this.uploader ? this.uploader.email : "";
},
profilePicture() {
    return this.uploader && this.uploader.profilePicture ? this.uploader.profilePicture : '';
},
profileTitle() {
    return this.uploader ? this.uploader.username + "'s profile" : "Profile";
}
```

Slika 4.25. *Computed properties* za korisnikove podatke



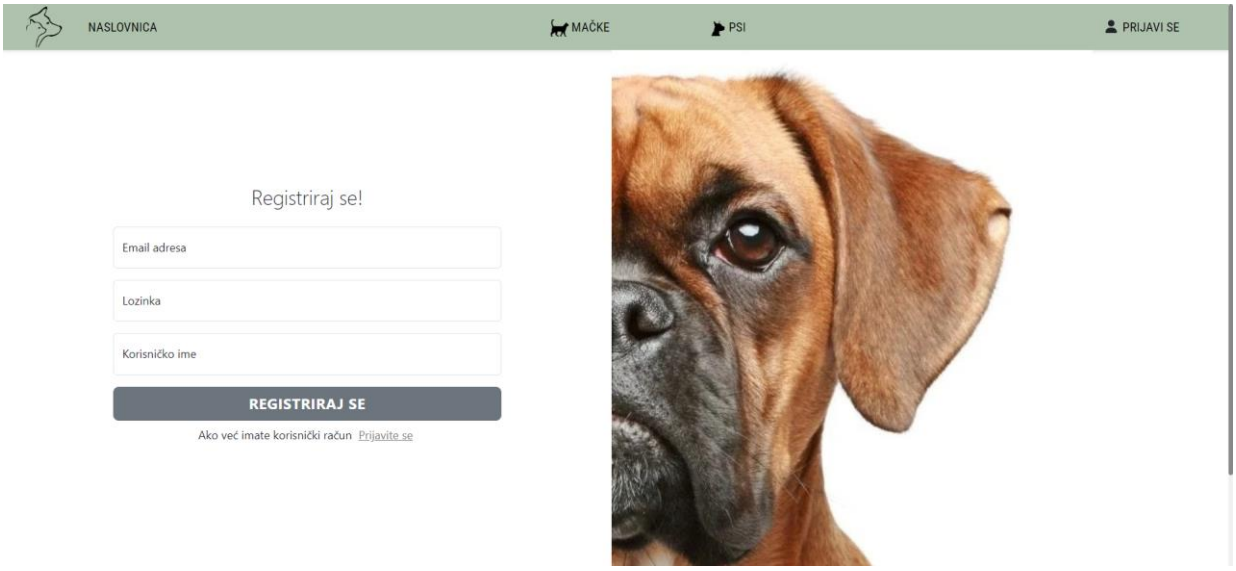
Slika 4.26. Iskočni prozor korisnikovog profila

```
<div class="uploader">UPLOADER:</div>
<button class="profile-button" @click="showProfileCard = true">
    {{ uploaderUsername }}</button>
<profile-card :show="showProfileCard" :title="profileTitle" class="profile-title"
    @close="showProfileCard = false">
    <div class="profile-content">
        <div class="profile-picture-container">
            <div class="profile-picture">
                <img v-if="profilePicture" :src="profilePicture" alt="Profile Picture" />
            </div>
        </div>
        <div class="profile-data">
            <div class="data"> <font-awesome-icon icon="user"
                    class="icon" />{{ uploaderUsername }}</div>
            <span class="horizontal-line"></span>
            <div class="data"> <font-awesome-icon icon="fa-envelope" class="icon" />{{ email }}
            </div>
            <span class="horizontal-line"></span>
            <div class="data"> <font-awesome-icon icon="fa-phone" class="icon" />{{ contact }}
            </div>
            <span class="horizontal-line"></span>
            <div class="data"> <font-awesome-icon icon="fa-map-marker-alt"
                    class="location-marker" /> {{ location }}</div>
        </div>
    </div>
</profile-card>
```

Slika 4.27. Primjer koda s podacima u komponenti <profile-card>

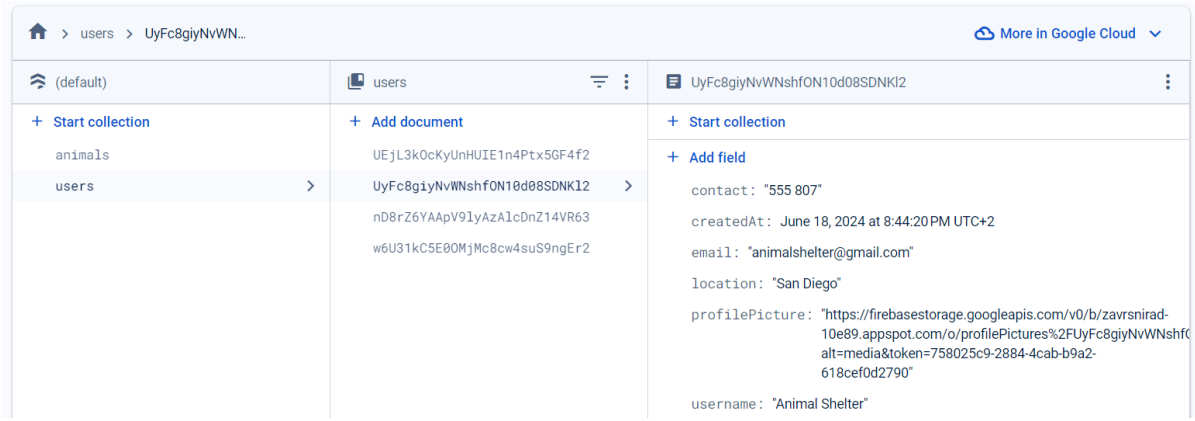
4.3.4. Stranice za autentifikaciju

Kako bi oglasi mogli biti postavljeni na stranicu korisnik se prvo mora registrirati. Za registraciju se koristi Firebaseov Authentication koji olakšava implementaciju sigurne autentifikacije u aplikaciji uz mnoge metode prijave kao što su prijava preko e-maila i lozinke, Googleovog računa, telefonskog broja itd. Za početak se na stranici za prijavu *SignUp.vue* nalaze tri obrasca za upis podataka, a to su obrazac za upis e-maila, lozinke i korisničkog imena. Ispod njih se nalazi gumb za prijavu s kojim s kreira novi korisnički račun. Ako korisnik već ima račun ispod gumba se nalazi tekst i link na stranicu za prijavu s postojećim računom.



Slika 4.28. Sučelje za kreiranje korisničkog računa

Kada korisnik unese željene podatke klikom na „Registriraj se“ gumb pokreće se metoda *submitData* u kojoj se prvo poziva metoda za provjeru unesenih podataka u polja obrasca, tj. provjerava jesu li polja prazna i provjerava postoji li znak @ u polju za e-mail. Ako je jedno od polja prazno ili u e-mail-u nema znak @ izbacuje se prozorčić gdje se ispisuje greška. Ako provjera obrasca prođe onda se kroz metodu *signUp* iz radnji šalju uneseni e-mail, lozinka i korisničko ime i pravi se novi korisnik s tim podacima koji se onda spremaju u Firestore bazu podataka u kolekciju *users* u polja podataka kod odgovarajućeg dokumenta nazvanog po jedinstvenom korisnikovom ID-om. Ta polja podataka u dokumentu sadrže datum izrade korisničkog računa, e-mail i korisničko ime također, i polja koja korisnik kasnije može dodati kao sliku profila, kontakt broj i lokaciju. Nakon uspješnog pravljenja novog korisničkog računa, korisnika se ulogira i preusmjeri na naslovnu stranicu te se na navigacijskoj traci pojavljuju nove rute tj. stranice za dodavanje oglasa, pregleda korisničkog profila na kojem se nalaze korisnički podaci i pripadajući oglasi, te gumba za odjavu iz računa.



Slika 4.29. Primjer kolekcije *users* s podacima dokumenta

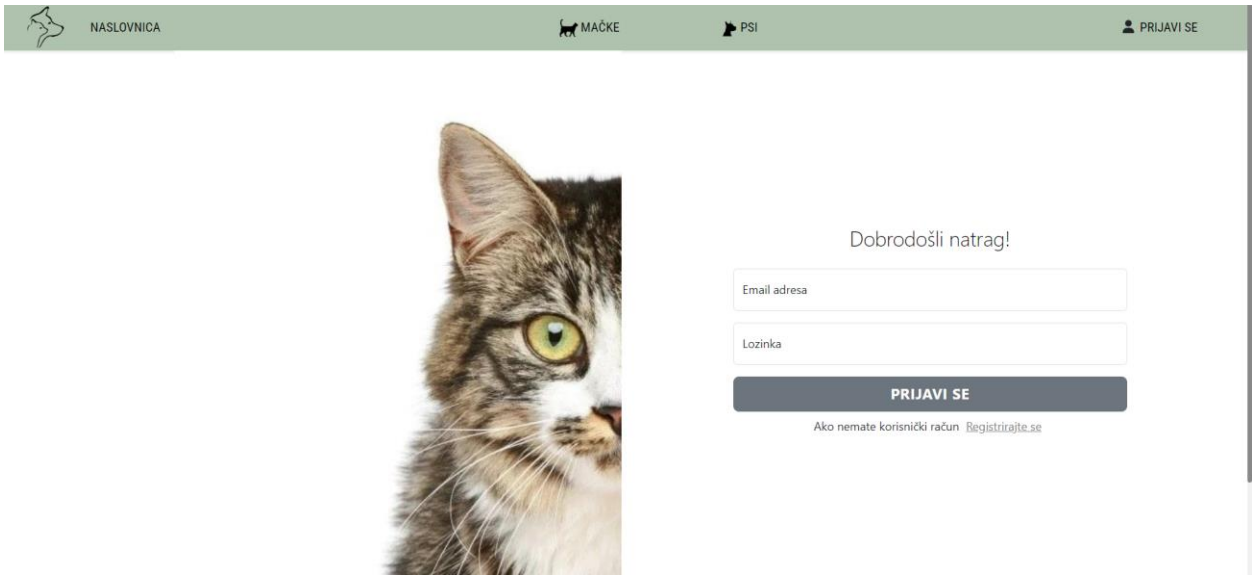
Unutar asinkrone radnje *signUp* prvo se stanje učitavanja mutacijom postavlja na „true“ kako bi se dalo do znanja da je u tijeku proces pravljenja korisničkog računa i dohvaćaju se instance za autentifikaciju i bazu podataka. U bloku *try-catch* se provjerava postoji li već korisnik s unesenim korisničkim imenom tako da se napravi upit prema Firestoreu za kolekciju *users*. Ako korisničko ime postoji javlja se greška, u suprotnom pokreće se Firebase-ova metoda *createUserWithEmailAndPassword* preko koje se pravi novi korisnički račun. Zatim se preko Firebase-ove metode *setDoc* postavlja dokument s korisnikovim jedinstvenim ID-om kao ime u kolekciju *users*. U tom se dokumentu nalaze uneseni podaci. Započinje se mutacija *setUser* za

slanje korisničkih podataka u Vuex pohranu. Catch blok hvata bilo koje greške koje se dogode dok se pokušava izvršiti kod u try bloku. U pohranu se šalje mutacija za pohranu greške, ako objekt za grešku *error.message* nema poruku ispisuje se zadana poruka. Na kraju unutar bloka *finally*, koji se pokreće dogodila se greška ili ne, učitavanje se postavlja na *false* kako bi se naznačilo da je proces kreiranje korisničkog računa završio.

```
async signUp({ commit }, { email, password, username }) {
  commit("setLoading", true);
  const auth = getAuth();
  const db = getFirestore();

  try {
    const usernameQuery = query(
      collection(db, "users"),
      where("username", "==", username)
    );
    const usernameSnapshot = await getDocs(usernameQuery);
    if (!usernameSnapshot.empty) {
      throw new Error("Username already exists.");
    }

    const userCredential = await createUserWithEmailAndPassword(
      auth,
      email,
      password
    );
    const user = userCredential.user;

    await setDoc(doc(db, "users", user.uid), {
      username,
      email,
      createdAt: new Date(),
    });

    commit("setUser", { uid: user.uid, email, username });
  } catch (error) {
    commit("setError", error.message || "Failed to sign up, try later.");
  } finally {
    commit("setLoading", false);
  }
},
```

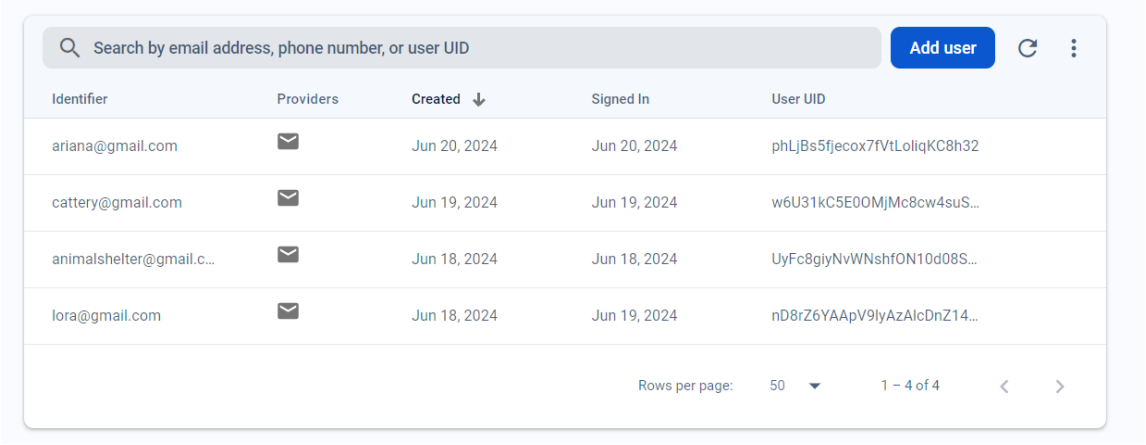
Slika 4.30. Radnja za pravljenje korisničkog računa

Stranica za prijavu je slična stranici za pravljenje korisničkog računa. Razlika je u tome što nema tri obrasca za unos podataka nego dva i to su obrazac za unos e-mail adrese i lozinke. Ispod gumba za prijavu se nalazi ruta koja korisnika šalje na stranicu za pravljenje korisničkog računa u slučaju da ga korisnik nema.



Slika 4.31. Prikaz sučelja za prijavu

Kada korisnik unese podatke za prijavu i klikne gumb za prijavu, pokreće se metoda *loginUser* u kojoj se dohvaća instanca za autentifikaciju. U try-catch bloku se autentificira korisnik preko unesenog e-maila i lozinke. Firebase-ova funkcija *signInWithEmailAndPassword* prihvaća tri argumenta, instancu autentifikacije, e-mail i lozinku, pokušava prijaviti korisnika u račun preko predanog e-maila i lozinke. Ako je prijava uspješna vraća objekt *userCredential* u koji se sprema rezultat funkcije, tj. informacije o prijavljenom korisniku i stanju autentifikacije. Nakon toga šalje se radnja *login* koja se brine o dodatnoj logici kao što je spremanje korisničkih podataka u Vuex pohranu. Klikom na gumb „Prijavi se“ pojavljuje se spinner dok se izvršavaju radnje iz metode *loginUser* i ako je prijava uspješna korisnika se ulogira i preusmjeri na naslovnu stranicu, u suprotnom se izbaci prozorčić u kojem piše kakva se greška dogodila.



Slika 4.32. Firebase Authentication

```vue
<script>
import { getAuth, signInWithEmailAndPassword } from 'firebase/auth';
import { mapActions, mapGetters } from 'vuex';

export default {
    data() {
        return {
            email: '',
            password: '',
        };
    },
    computed: {
        ...mapGetters('users', ['error', 'isLoading']),
    },
    methods: {
        ...mapActions('users', ['login']),

        async loginUser() {
            const auth = getAuth();
            try {
                const userCredential = await signInWithEmailAndPassword(auth, this.email, this.password);
                userCredential.user;
                await this.login({ email: this.email, password: this.password });
                this.$router.push('/home');
            } catch (error) {
                console.error('Error logging in:', error);
                this.handleError(error.message);
            }
        },

        handleError(message) {
            this.$store.commit('users/setError', message);
        }
    },
};
</script>
```

Slika 4.33. Script kod stranice za logiranje

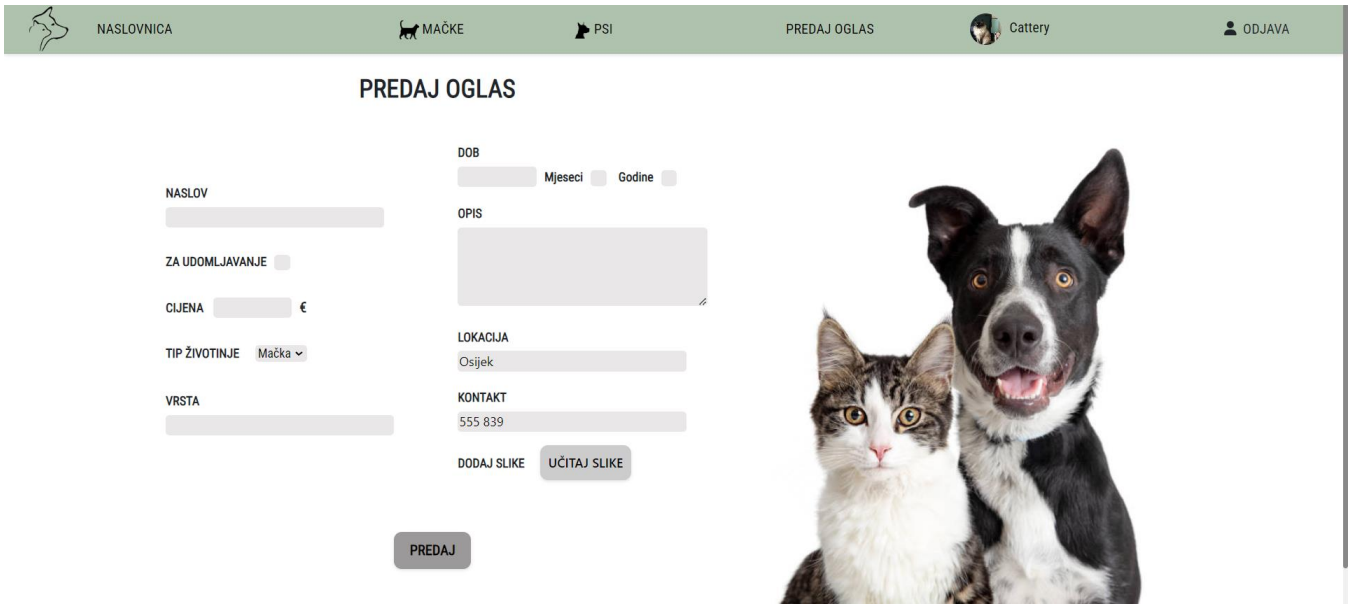
Asinkrona funkcija *login* u radnjama iz modula za korisnike prima dva argumenta, metodu *commit* koju nam pruža Vuex kako bi mogli pohraniti mutacije u pohranu (eng. Store) i objekt s korisnikovim podacima za prijavu tj. e-mailom i lozinkom. Kao i u funkciji za pravljenje korisničkog računa, Vuexu se šalje mutacija koja postavlja stanje učitavanja na *true* kojim se onda pokazuje spinner koji označuje da je u procesu nekakva radnja. Ta radnja se odvija u try-catch bloku gdje se korisnika pokušava prijaviti preko funkcije *signInWithEmailAndPassword*. Pošto je funkcija asinkrona koristimo *await* koji osigurava da funkcija čeka da se operacija završi. Iz Firestore baze podataka se dohvaća dokument iz kolekcije *users* koji odgovara UID-ju provjerenog korisnika. Zatim se odvija provjera postoji li dokument u bazi podataka, ako postoji podaci se dohvaćaju iz dokumenta i postavljaju u konstantu *userData*. Unutar *commit* Vuexu se šalje mutacija s dohvaćenim podacima kako bi pohranili korisnikovo stanje za daljnju uporabu. U slučaju da korisnik ne postoji izbacuje se greška. Na kraju se učitavanje završi postavljanjem *isLoading* na *false* bez obzira je li pronađen korisnik koji je zatim ulogiran ili se dogodila greška.

```
async login({ commit }, { email, password }) {
  commit("setLoading", true);
  const auth = getAuth();
  const db = getFirestore();

  try {
    const userCredential = await signInWithEmailAndPassword(
      auth,
      email,
      password
    );
    const user = userCredential.user;

    const userDoc = await getDoc(doc(db, "users", user.uid));
    if (userDoc.exists()) {
      const userData = userDoc.data();
      commit("setUser", {
        uid: user.uid,
        email: userData.email,
        username: userData.username,
        profilePicture: userData.profilePicture || "",
        location: userData.location,
        contact: userData.contact,
      });
    } else {
      throw new Error("User data not found in Firestore");
    }
  } catch (error) {
    commit("setError", error.message || "Failed to log in, try later.");
  } finally {
    commit("setLoading", false);
  }
},
```

Slika 4.34. Radnja *login* u modulu *users*

4.3.5. Stranica za predaju oglasa

Stranica za predaju oglasa se sastoji od obrasca u kojem se treba unijeti naslov oglasa, cijena ili označiti kvačicu ako je za udomljavanje čime se polje za cijenu miče, tip životinje (mačka ili pas), vrsta, dob uz dodatno specificiranje misli li se na mjesece ili godine, opis, lokacija, kontakt broj i slike koje su opcionalne. Ispod forme se nalazi gumb kojim se pokreće provjera obrasca te ako je obrazac valjan preusmjerava korisnika, ovisno o odabranom tipu životinje, na stranicu gdje se nalaze oglasi mačaka tj. pasa.



Slika 4.35. Obrazac za predaju oglasa

Klikom na okvir za udomljavanje sakriva se cijena, svojstvo stanja *adoptable* koje je u ovom slučaju objekt, kao i sva ostala svojstva unutar *data()* funkcije koja su potrebna kao objekti. To svojstvo sadrži dva ključa *val* koji drži *false* vrijednost i *isValid* s vrijednošću *true* koji se koristi za provjeru obrasca. Uvjetnim renderiranjem koristeći Vue.js direktivu *v-if* kada je vrijednost *adoptable false* cijena je prikazana, a kada se klikne okvir vrijednost postaje *true* i polje za cijenu se sakriva.

```
<div class="form-control adoption-checkbox">
    <label for="adoptable">FOR ADOPTION</label>
    <input type="checkbox" id="adoptable" v-model="adoptable.val">
</div>
<div v-if="!adoptable.val" class="form-control price-input">
    <label class="price-label" for="price">PRICE</label>
    <input type="number" id="price" v-model.number="price.val">
    <label class="euro" for="price">€</label>
</div>
```

Slika 4.36. Kod koji sakriva ili prikazuje polje za cijenu

```
data() {
    return {
        isLoading: false,
        adoptable: { val: false, isValid: true },
        type: { val: 'cat', isValid: true },
        title: { val: '', isValid: true },
        price: { val: null, isValid: true },
        age: { val: null, isValid: true },
        months: { val: false, isValid: true },
        years: { val: false, isValid: true },
        description: { val: '', isValid: true },
        location: { val: '', isValid: true },
        breed: { val: '', isValid: true },
        contact: { val: '', isValid: true },
        images: { val: [], isValid: true },
        formIsValid: true,
        adData: null,
    };
},
```

Slika 4.37. data() funkcija s pohranjenim vrijednostima

```
<div class="form-control type-dropdown">
    <label for="type">TIP ŽIVOTINJE</label>
    <select id="type" v-model="type.val">
        <option value="cat">Mačka</option>
        <option value="dog">Pas</option>
    </select>
</div>
```

Slika 4.38. Padajući izbornik s opcijama *Mačka* ili *Pas*

Kada se odabere opcija *Mačka* ili *Pas* prikazana na slici 4.39. ta vrijednost se kasnije koristi kao vodič na koju stranicu trebamo usmjeriti korisnika, stranicu s oglasima mačaka ili pasa. Vrijednosti svih polja se povezuju koristeći Vue.js direktivu *v-model* za dvosmjerno povezivanje. *V-model* povezuje *input*, *textarea* ili *select* elemente s varijablom u Vue instanci. Postoje i modifikatori koje možemo koristiti na *v-modelu* kao što je npr *.number* kod cijene na slici 4.36. koji osigurava da unesena vrijednost bude broj, te *.trim* kojim odrežemo razmake nakon unesenih vrijednosti.

```
<div class="form-control age-input" :class="{ invalid: !age.isValid }">
    <label class="top-label" for="age">DOB</label>
    <input type="number" id="age" v-model.number="age.val" @blur="clearValidity('age')">
    <label class="months-checkbox" for="age" value="months">Mjeseci</label>
    <input type="checkbox" id="months" @change="handleCheckboxClick('months')"
        v-model="months.val">
    <label class="years-checkbox" for="age" value="years">Godine</label>
    <input type="checkbox" id="years" @change="handleCheckboxClick('years')"
        v-model="years.val">
    <p v-if="!age.isValid">Dob ne može biti prazna.</p>
```

Slika 4.39. Polje za unos dobi i kvadratići za odabir mjeseca ili godina

Kod unosa godina mora biti barem jedan kvadratić označen, u slučaju da nije izbacit će se greška preko *v-if*-a gdje se provjerava je li ključ *isValid* objekta *age true* ili *false*. Ista greška se pojavi i ako je polje za godine prazno. Ako je prvotno označen kvadratić za mjesece te se zatim označi kvadratić za godine, kvadratić za mjesece se odznačuje, a kvadratić za godine postaje označen, isto to se događa i ako je prvotno bio označen kvadratić za godine.

```
handleCheckboxClick(checkbox) {
    if (checkbox === 'months' && this.months.val) {
        this.years.val = false;
    } else if (checkbox === 'years' && this.years.val) {
        this.months.val = false;
    }
},
```

Slika 4.40. Metoda kojom provjeravamo koji kvadratić je označen i drugom mičemo oznaku

Kako bi obrazac bio predan bez grešaka mora proći kroz provjeru gdje se provjerava jesu li određena polja prazna, je li cijena veća od nule i je li joj polje prazno, jesu li godine manje od nule i je li polje prazno. Klikom na gumb „Predaj“ pokreće se *submitForm* metoda u kojoj se odvija ta provjera te ako je sve u redu na stranici iskače prozorčić sa spinnerom koji se vrti dok se forma s podacima ne preda, te ovisno pošalje korisnika na odgovarajuću stranicu predanog oglasa ili iskoči novi prozorčić u kojem piše da se dogodila greška kod pokušavanja predaje oglasa.

```
submitForm() {
    this.validateForm();
    if (this.formIsValid) {
        this.isLoading = true;
        const auth = getAuth();
        const user = auth.currentUser;

        const ad = {
            userId: user.uid,
            title: this.title.val,
            type: this.type.val,
            breed: this.breed.val,
            price: this.price.val,
            age: this.age.val,
            months: this.months.val,
            years: this.years.val,
            description: this.description.val,
            location: this.location.val,
            contact: this.contact.val,
            images: this.images.val.map(image => image.url ? { url: image.url } : image),
            adoptable: this.adoptable.val,
        };
```

Slika 4.41. submitForm metoda

Na slici 4.41. svi podaci predani u obrascu se spremaju u objekt *ad* koji se zatim predaje kao argument metodi *addAnimal* iz radnji u modulu za životinje, te preusmjerava korisnika ovisno je li tip životinje bio mačka, na stranicu oglasa mačaka ili pasa.

```
this.addAnimal(ad)
    .then(() => {
        const redirectUrl = ad.type === 'cat' ? '/cats' : '/dogs';
        this.$router.push(redirectUrl);
    })
    .catch(err => {
        this.error = err.message || 'Failed to submit the ad.';
        this.isLoading = false;
    });
```

Slika 4.42. Predaja podataka u radnju *addAnimal* i preusmjeravanje korisnika na određenu stranicu

Slike predane u obrascu se ne spremaju u bazu podataka u predanom obliku već se pretvaraju u URL koji možemo spremiti u bazu podataka, dok se slike spremaju u Firebase Storage. Korištenjem URL-a za slike nam olakšava rukovanje njima kada ih trebamo ispisati na ekranu.

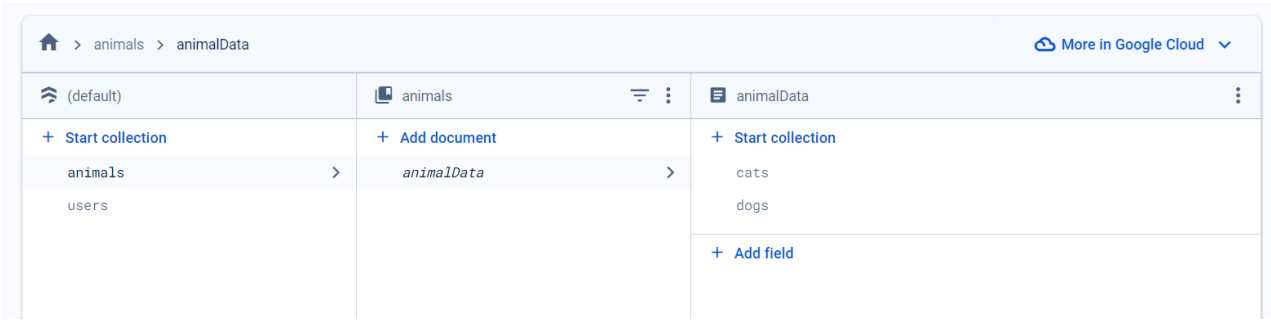
```
...mapActions('animals', ['addAnimal', 'updateAnimal']),
handleImageChange(event) {
    const files = Array.from(event.target.files);
    const newImages = files.map(file => ({
        name: file.name,
        size: file.size,
        type: file.type,
        file: file,
    }));
    this.images.val = [...this.images.val.filter(img => img.url), ...newImages];
    this.images.isValid = true;
},
```

Slika 4.43. Dodavanje metoda iz Vuex radnji i metoda za stvaranje URL-a slike

Na slici 4.43. u metodi *handleImageChange* se lista datoteka tj. slika pretvara u standardno JavaScript polje kako bi bilo lakše za rukovati njime. Zatim se u objekt *newImages* spremaju vrijednosti iterirane kroz svaku datoteku u polju *files.* Nakon toga se u podacima komponente ažurira *images.val* svojstvo. Prvo filtrira postojeće slike koje imaju URL i zatim dodaje novostvorene objekte *newImages* u polje. Ovom operacijom se spajaju postojeće slike s novododanim. Na kraju se označuje uspješno učitavanje slike.

U funkciji *addAnimal* iz radnja u modulu *animals* se odvija provjera je li korisnik koji je predao oglas autentificiran, u slučaju da nije pokazuje se greška, a u suprotnom vrati se objekt korisnika. Nakon toga se definira ime foldera u koji će biti spremljene slike predane oglasom ovisno je li predana životinja mačka ili pas. Zatim se stvara referenca na mjesto gdje će slika biti pohranjena

koristeći *ref(storage, ${folderName}/${image.name})*, slika se postavlja u Firebase Storage pomoću *uploadBytes* i pomoću *getDownloadURL* dohvaćamo URL slike. Na kraju se čeka da se sve slike postave koristeći *Promise.all* što rezultira poljem URL-ova slika. Unutar objekta *animalData* nalaze se svi podaci o životinji. Oglas životinje dodajemo u Firestore potkolekciju (eng. Subcollection), prvo referenciramo roditeljski dokument u bazi, zatim potkolekciju unutar tog dokumenta. S *addDoc* dodajemo novi dokument unutar *animalData* dokumenta i vraćamo referencu na dokument. Baza podataka je struktuirana tako da postoji kolekcija *animals* s dokumentom *animalData* koji sadrži dvije potkolekcije *cats* i *dogs*, te svaki od njih sadrži svoje dokumente s jedinstvenim ID-om koji drže svoja polja podataka. Strukturirano je tako zato što Firestore kolekcije mogu sadržavati grupu dokumenata, ali ne mogu sadržavati potkolekciju dok dokumenti mogu sadržavati potkolekcije koje u sebi sadržavaju dokumente.



Slika 4.44. Prikaz strukture Firestore baze podataka

Unutar *newAnimal* objekta generira se novi ID dokumenta, i postavlja se mutacija koja ažurira stanje u Vuexu pozivajući odgovarajuće mutacije (addCat ili addDog) i na kraju vraća taj objekt. U slučaju da se dogodila greška ona se ispisuje u konzoli.

```
async addAnimal({ commit }, animal) {
  try {
    const auth = getAuth();
    const user = auth.currentUser;
    if (!user) {
      throw new Error("User is not authenticated");
    }

    const folderName = animal.type === "dog" ? "dogs" : "cats";
    const imageUploads = animal.images.map(async (image) => {
      const storageRef = ref(storage, `${folderName}/${image.name}`);
      await uploadBytes(storageRef, image.file);
      const url = await getDownloadURL(storageRef);
      return url;
    });

    const imageUrls = await Promise.all(imageUploads);

    const animalData = {
      title: animal.title,
      adoptable: animal.adoptable,
      type: animal.type,
      breed: animal.breed,
      age: animal.age,
      months: animal.months,
      years: animal.years,
      price: animal.price,
      description: animal.description,
      location: animal.location,
      contact: animal.contact,
      userId: user.uid,
      images: imageUrls,
    };
```

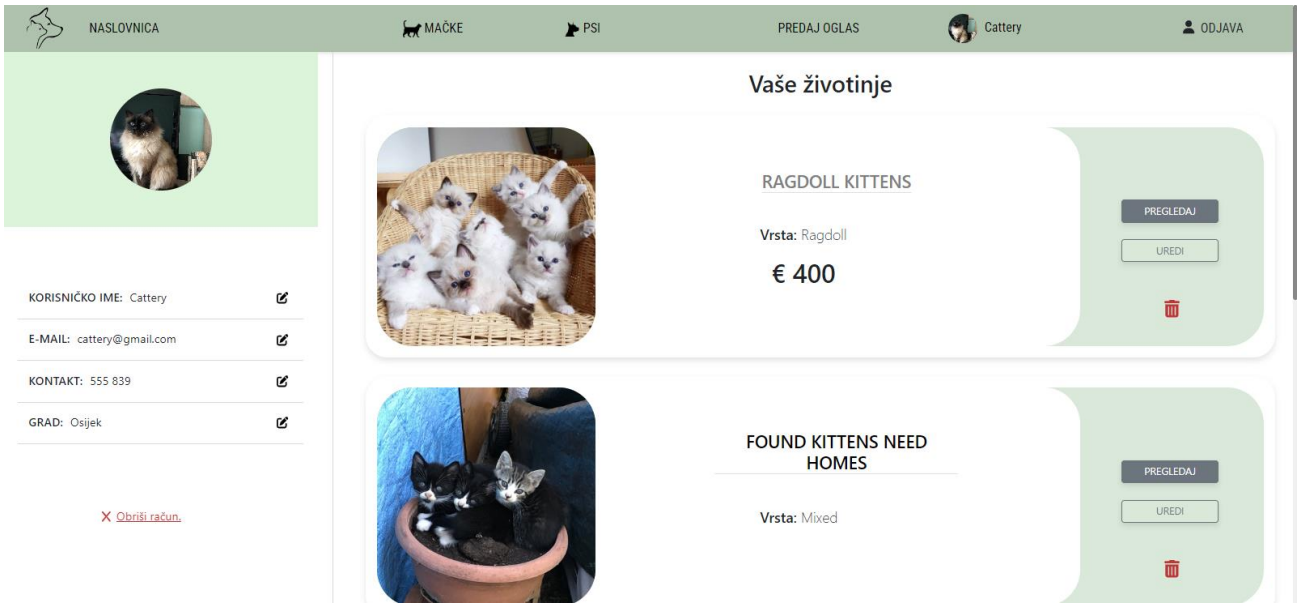
Slika 4.45. Prikaz prvog dijela *addAnimal* funkcije

```
    const parentDocRef = doc(db, "animals", "animalData");
    const subcollectionPath = animal.type === "dog" ? "dogs" : "cats";
    const subcollectionRef = collection(parentDocRef, subcollectionPath);
    const docRef = await addDoc(subcollectionRef, animalData);

    const newAnimal = { id: docRef.id, ...animalData };
    commit(animal.type === "dog" ? "addDog" : "addCat", newAnimal);

    return newAnimal;
  } catch (error) {
    console.error("Error adding animal:", error.message);
    throw error;
  }
},
```

Slika 4.46. Prikaz drugog dijela *addAnimal* funkcije

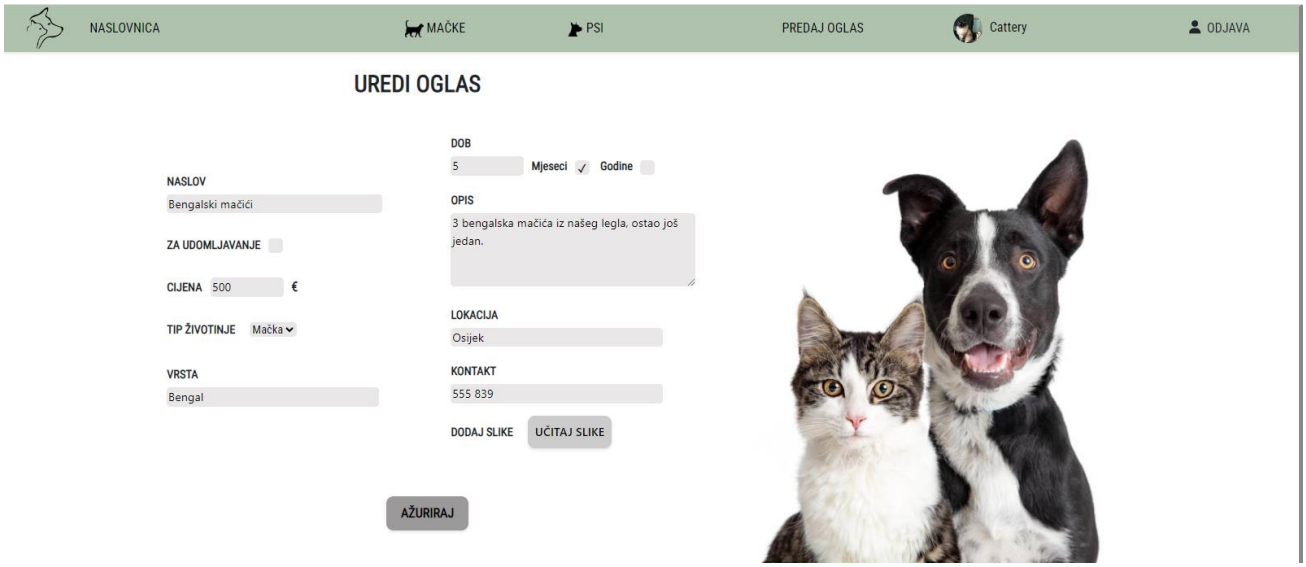
4.3.6. Korisnikov profil

Logirani korisnik ima mogućnost pregledavanja svojih podataka i oglasa na stranici svog profila prikazanoj na slici 4.47. Također može dodati profilnu sliku, mijenjati svoje podatke, uređivati oglase, brisati oglase i obrisati svoj račun.



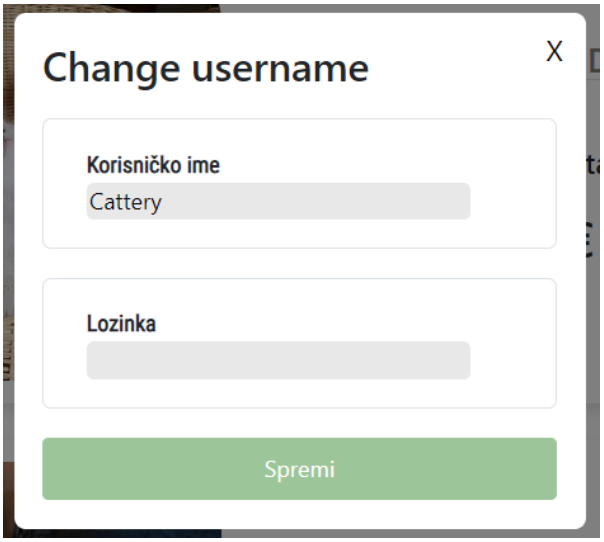
Slika 4.47. Prikaz sučelja korisnikovog profila i njegovih oglasa

Kada korisnik klikne na „Uredi“ gumb, otvara se isti obrazac kojim je prvotno predao oglas, samo s drugim naslovom koji naznačuje da je u pitanje uređivanje obrasca. Obrazac se također popuni s vrijednostima od prije kako se ništa ne bi izgubilo. Te vrijednosti dobijemo preko parametra iz rute.



Slika 4.48. Prikaz sučelja obrasca za uređivanje oglasa

Ako korisnik želi promijeniti jedan od svojih podataka, klikom na ikonicu za uređivanje otvara se prozorčić u kojem je napisana prijašnja vrijednost i polje za unos lozinke kako bi se potvrdila promjena.



Slika 4.49. Prikaz prozorčića za mijenjanje korisničkog imena

Za mijenjanje podataka oglasa ponovno se koristi metoda *submitForm* kao za predaju oglasa samo se na kraju prije preusmjeravanja provjerava je li riječ o uređivanju obrasca i zatim se koristi radnja *updateAnimal* kojoj se predaju izmijenjeni i stari podaci.

```
if (this.isEditMode) {
    ad.id = this.adData.id;
    this.updateAnimal(ad)
        .then(() => {
            const redirectUrl = ad.type === 'cat' ? '/cats' : '/dogs';
            this.$router.push(redirectUrl);
        })
        .catch(err => {
            this.error = err.message || 'Failed to update the ad.';
            this.isLoading = false;
        });
```

Slika 4.50. Provjera radi li se o uređivanju obrasca

Kako bi obrazac bi popunjen postojećim podacima tog oglasa metoda *initializeFormFields* popunjava ta polja podacima iz parametra URL-a. *adData* izvlači podatke iz trenutne rute, zatim se provjerava postoji li, ako postoji parsira string iz JSON formata u JavaScript objekt, pridružuje objekt *adData* svojstvu komponente *parseAdData*, zatim se popunjavaju polja obrasca odgovarajućim vrijednostima.

```
    initializeFormFields() {
        const { adData } = this.$route.query;
        if (adData) {
            const parsedAdData = JSON.parse(adData);
            this.adData = parsedAdData;
            this.title.val = parsedAdData.title;
            this.type.val = parsedAdData.type;
            this.breed.val = parsedAdData.breed;
            this.age.val = parsedAdData.age;
            this.price.val = parsedAdData.price;
            this.description.val = parsedAdData.description;
            this.location.val = parsedAdData.location;
            this.contact.val = parsedAdData.contact;
            this.images.val = parsedAdData.images;
        }
    }
},
```

Slika 4.51. Provjera radi li se o uređivanju obrasca

```
<div class="profile-container">
    <div class="profile-picture-container">
        <div class="profile-picture" @click="triggerFileUpload">
            <template v-if="isUploading">
                <base-spinner></base-spinner>
            </template>
            <template v-else>
                <img v-if="profilePicture" :src="profilePicture" alt="Profile Picture" />
                <button v-else class="upload-button">SET IMAGE</button>
            </template>
            <input type="file" ref="fileInput" @change="onFileChange" class="file-input" />
        </div>
    </div>
```

Slika 4.52. Prikaz koda kojim se dodaje profilna slika

Ako je profilna slika dostupna ona se prikaže u lijevom gornjem kutu na stranici, a ako nije korisnik ju može dodati. Tijekom dodavanja vrti se spinner dok se slika ne učita. Slika se može dodati klikom na mjesto predviđeno za sliku.

```
editField(fieldType) {
    this.editFieldType = fieldType;
    if (fieldType === 'username') {
        this.editUsername.val = this.username;
    } else if (fieldType === 'email') {
        this.editEmail.val = this.email;
    } else if (fieldType === 'contact') {
        this.editContact = this.contact;
    } else if (fieldType === 'location') {
        this.editLocation = this.location;
    }
    this.showEditCard = true;
},
```

Slika 4.53. Metoda za uređivanje korisničkih podataka

Kada korisnik klikne na određeno polje za uređivanje svojih podataka poziva se *editField* metoda. U njoj se postavi *editFieldType* prema odgovarajućem polju koje se uređuje i ispuni ga s trenutnom vrijednošću. Nakon što korisnik potvrdi promjenu poziva se metoda *saveEdit* u kojoj se provjerava je li obrazac valjan, provjerava je li unesena točna lozinka i provjerava postoji li već uneseno korisničko ime. Radnja *updateUserData* ažurira podatke u bazi podataka i polja na profilu se ažuriraju novim vrijednostima.

```
async saveEdit() {
    console.log('saveEdit called');
    this.isSaving = true;
    this.passwordIsEmpty = false;
    this.validateForm();

    if (!this.formIsValid) {
        console.log('Form is invalid');
        this.isSaving = false;
        return;
    }

    try {
        if (this.requiresPassword && !await this.checkPassword()) {
            this.password.isValid = false;
            console.error("Invalid password");
            throw new Error("Invalid password");
        }

        if (this.editFieldType === 'username' && !await this.isUsernameUnique()) {
            this.editUsername.isValid = false;
            console.log('Username is not unique');
            this.isSaving = false;
            return;
        }
```

Slika 4.54. Prvi dio metode *saveEdit*

```
        const updateData = {};
        if (this.editFieldType === 'username') {
            updateData.username = this.editUsername.val;
        } else if (this.editFieldType === 'email') {
            updateData.email = this.editEmail.val;
        } else if (this.editFieldType === 'contact') {
            updateData.contact = this.editContact;
        } else if (this.editFieldType === 'location') {
            updateData.location = this.editLocation;
        }

        await this.updateUserData(updateData);

        const db = getFirestore();
        const userDoc = doc(db, 'users', this.user.uid);
        await updateDoc(userDoc, updateData);

    } catch (error) {
        console.error("Error updating user info:", error);
    } finally {
        this.isSaving = false;
        this.showEditCard = false;
        this.password.val = '';
    }
},
```

Slika 4.55. Drugi dio metode *saveEdit*

Korisnikovi oglasi se dohvaćaju prilikom stvaranja komponente *created* koja dohvaća korisnika i sve njegove oglase. Ako je korisnik ulogiran dohvati sve oglase za mačke i pse pozivom *fetchAnimals(„cats“)* tj. *fetchAnimals(„dogs“). fetchAnimals* šalje upit Firestoreovom dokumentu u hijerarhiji oblika *animals/animalData/animalType* gdje userId odgovara ID-ju prijavljenog korisnika.

```
async created() {
    if (this.user) {
        this.isLoading = true;
        try {
            const cats = await this.fetchAnimals("cats");
            const dogs = await this.fetchAnimals("dogs");

            this.animals = [...cats, ...dogs];
        } catch (error) {
            console.error("Error fetching animals:", error);
        } finally {
            this.isLoading = false;
        }
    }
}
```

Slika 4.56. *created* metoda

```
async fetchAnimals(animalType) {
    const db = getFirestore();
    const animalsQuery = query(
        collection(db, "animals", "animalData", animalType),
        where("userId", "==", this.user.uid)
    );
    const animalsSnapshot = await getDocs(animalsQuery);
    return animalsSnapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));
},
```

Slika 4.57. Metoda *fetchAnimals*

U slučaju da korisnik želi obrisati jedan od oglasa poziva se metoda *removeAdFromList* za uklanjanje oglasa kojoj se predaje ID oglasa te metoda filtrira oglas s danim ID-om i briše ga iz liste oglasa. U template dijelu se nalazi prilagođeni event *adDelete* preko kojeg se poziva metoda *removeAdFromList. adDelete* se emitira iz komponente *animal-ad*.

```
showDeleteDialog() {
    this.isDialogVisible = true;
},
async confirmDelete() {
    try {
        await this.deleteAnimal({ id: this.id, type: this.type });
        this.$emit('adDeleted', this.id);
    } catch (error) {
        console.error('Failed to delete the ad:', error);
    } finally {
        this.isDialogVisible = false;
    }
},
```

Slika 4.58. Metode za pokazivanje prozorčića i potvrde brisanja oglasa u komponenti *animal-ad*

```
<ul v-else-if="animals.length > 0">
    <animal-ad v-for="animal in animals" :key="animal.id" :id="animal.id" :title="animal.title"
        :type="animal.type" :gender="animal.gender" :breed="animal.breed" :age="animal.age"
        :price="animal.price" :description="animal.description" :location="animal.location"
        :contact="animal.contact" :images="animal.images" @adDeleted="removeAdFromList"></animal-ad>
</ul>
```

Slika 4.59. Prikaz korištenja Vue.js direktiva za prikaz oglasa

Prije brisanja oglasa iskače prozorčić koji pita korisnika želi li stvarno obrisati oglas. Korisnik ima dvije mogućnosti, kliknuti na *Da* za potvrdu i *Ne* za odgodu brisanja.

```
<base-dialog :show="isDialogVisible" title="Confirm Delete" @close="isDialogVisible = false">
    <template #default>
        <p>Jeste li sigurni da želite obrisati životinju?</p>
    </template>
    <template #actions>
        <base-button @click="confirmDelete">Da</base-button>
        <base-button @click="isDialogVisible = false">Ne</base-button>
    </template>
</base-dialog>
```

Slika 4.60. Komponenta za prozorčić unutar komponente *animal-ad*

```
async deleteAnimal({ commit }, { id, type }) {
  try {
    const parentDocRef = doc(db, "animals", "animalData");
    let subcollectionPath = type === "dog" ? "dogs" : "cats";

    const animalDocRef = doc(parentDocRef, subcollectionPath, id);
    await deleteDoc(animalDocRef);

    commit(type === "dog" ? "removeDog" : "removeCat", id);
  } catch (error) {
    console.error("Error deleting animal:", error);
    throw error;
  }
},
```

Slika 4.61. Radnja *deleteAnimal* u modulu *animals*

Unutar *deleteAnimal* radnje se briše određeni oglas preko njegovog ID-ja i tipa. Integrira Firestore operacije za brisanje dokumenta i Vuex mutacije za ažuriranje stanja aplikacije.

```
<base-dialog :show="showDeleteDialog" title="Confirm Delete Account" @close="showDeleteDialog = false">
    <template #default>
        <p>Jeste li sigurni da želite obrisati račun? Ova radnja se ne može poništiti i obrisati će sve vaše
            oglase.</p>
    </template>
    <template #actions>
        <base-button @click="confirmDeleteAccount">Da</base-button>
        <base-button @click="showDeleteDialog = false">Ne</base-button>
    </template>
</base-dialog>
```

Slika 4.62. Prikaz komponente prozorčića prije brisanja korisničkog računa

```
async confirmDeleteAccount() {
    this.isDeleting = true;
    try {
        await this.$store.dispatch('users/deleteAccount');
        this.$router.push({ path: '/home' });
    } catch (error) {
        console.error("Error deleting account:", error);
    } finally {
        this.isDeleting = false;
    }
},
```

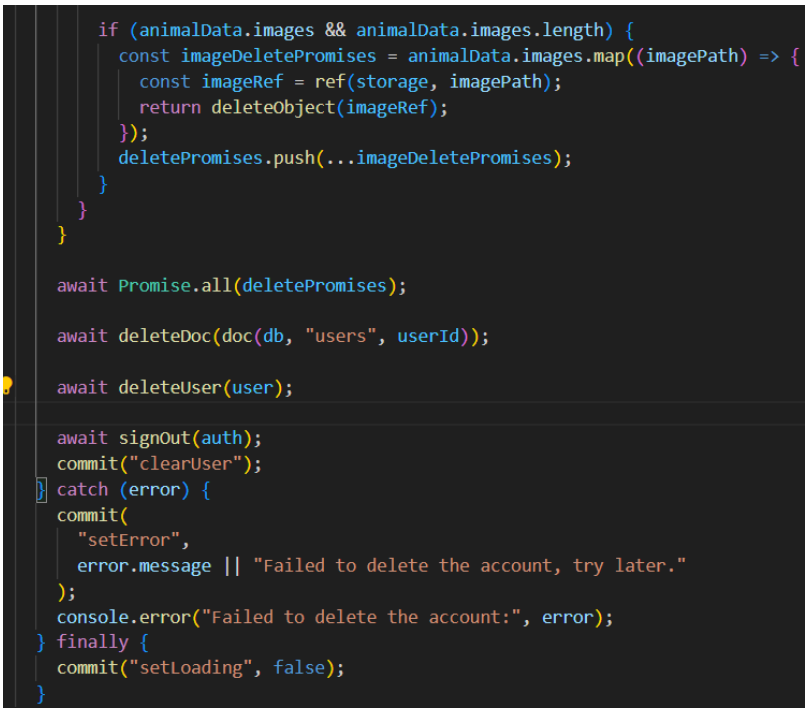
Slika 4.63. Metoda za potvrdu brisanja korisničkog računa

U slučaju da korisnik poželi obrisati svoj korisnički račun, to može obaviti klikom na „Obriši račun“ gumb u donjem lijevom kutu na svom profilu. Klikom na taj gumb poziva se metoda *confirmDeleteAccount* koja prikazuje spinner dok u try-catch bloku pokušava izvršiti brisanje računa slanjem radnje *deleteAccount* iz modula *users* Vuexu koji se treba pobrinuti za brisanje. Ako je brisanje uspjelo, korisnika se izlogira i preusmjeri na naslovnu stranicu. Također kao i kod brisanja oglasa klikom na gumb za brisanje računa korisnika se pita želi li sigurno obrisati račun. U tom prozorčiću na slici 4.63. kao *prop* se dobije *showDeleteDialog* koji određuje hoće li se on prikazati ili ne.

```
async deleteAccount({ commit, getters, dispatch }) {
  commit("setLoading", true);
  const db = getFirestore();
  const auth = getAuth();
  const storage = getStorage();
  const user = auth.currentUser;
  const userId = getters.user.uid;

  try {
    const subcollections = ["cats", "dogs"];
    const deletePromises = [];

    for (const subcollection of subcollections) {
      const animalsQuery = query(
        collection(db, "animals", "animalData", subcollection),
        where("userId", "==", userId)
      );
      const animalsSnapshot = await getDocs(animalsQuery);

      for (const animalDoc of animalsSnapshot.docs) {
        const animalData = animalDoc.data();
        const animalId = animalDoc.id;
        const animalType = animalData.type;

        deletePromises.push(
          dispatch(
            "deleteAnimal",
            { id: animalId, type: animalType },
            { root: true }
          )
        );
```

Slika 4.64. Prvi dio koda radnje *deleteAccount*

Na slici 4.64. u radnji za brisanje korisničkog računa inicijalizira se Firestore, Authentication i Storage, zatim se dohvaća trenutni korisnik i njegov ID. U try-catch bloku definiraju se potkolekcije u kojima se nalaze svi oglasi koji pripadaju tom korisniku i pokušavaju se obrisati, u slučaju da ih korisnik nije obrisao prije brisanja računa.



```
        if (animalData.images && animalData.images.length) {
          const imageDeletePromises = animalData.images.map((imagePath) => {
            const imageRef = ref(storage, imagePath);
            return deleteObject(imageRef);
          });
          deletePromises.push(...imageDeletePromises);
        }
      }
    }

    await Promise.all(deletePromises);

    await deleteDoc(doc(db, "users", userId));

    await deleteUser(user);

    await signOut(auth);
    commit("clearUser");
  } catch (error) {
    commit(
      "setError",
      error.message || "Failed to delete the account, try later."
    );
    console.error("Failed to delete the account:", error);
  } finally {
    commit("setLoading", false);
  }
```

Slika 4.65. Drugi dio koda radnje *deleteAccount*

Na slici 4.65. u slučaju da pripadajući oglasi koji se brišu imaju slike, pristupa se tim slikama i briše se i njih. Zatim se briše dokument iz kolekcije *users*, korisnik se briše iz Firebase Authentication, izlogira ga se i Vuexu se pošalje mutacija kako bi se korisnik maknuo iz stanja. U slučaju bilo kakve greške, ona se ispiše.

5. ZAKLJUČAK

Cilj ovog završnog rada je bio napraviti web aplikaciju za udomljavanje životinja, u ovom slučaju mačaka i pasa gdje posjetitelji aplikacije mogu slobodno listati oglase na stranicama za mačke i pse, dok prijavljeni korisnik može predavati oglase, uređivati ih i brisati. Prijavljeni korisnici također mogu dodati profilnu sliku, mijenjati svoje podatke i obrisati svoj račun. Za spremanje korisničkih računa i spremanje oglasa, te bilo koje radnje s njima kao što su brisanje ili uređivanje, korišten je Firebase koji nudi mogućnosti za lakše baratanje podacima koristeći Firestore za spremanje korisničkih profila i oglasa, Firebase Authentication preko kojeg je omogućeno pravljenje profila i Storage u koji se spremaju slike. Rad je izrađen u JavaScript-ovom framework-u Vue.js uz pomoć CSS-a, Bootstrapa i HTML-a u programu Visual Studio Code. Prednost aplikacije je što je fokusirana samo na oglase mačaka i pasa pa korisnici ne moraju koristiti druge oglasnike koji su obogaćeni mnogim drugim sadržajem, već se mogu lakše snaći u potrazi za kućnim ljubimcem.

Literatura

[1] Njuškalo: https://www.njuskalo.hr/. [7. 4. 2024.].

[2] Index Oglasi: https://www.index.hr/oglasi/. [7. 4. 2024.].

[3] Gumtree: https://www.gumtree.com/. [7. 4. 2024.].

[4] Petfinder: https://www.petfinder.com/. [7. 4. 2024.].

[5] Visual Studio Code: https://code.visualstudio.com/. [7. 4. 2024.].

[6] HTML: https://www.w3schools.com/html/. [14. 4. 2024.].

[7] CSS: https://www.w3schools.com/css/. [17. 5. 2024.].

[8] Bootstrap: https://getbootstrap.com/docs/5.0/getting-started/introduction/. [17. 5. 2024.].

[9] Vue.js: https://vuejs.org/guide/introduction.html. [22. 5. 2024.].

[10] Firebase: https://firebase.google.com/docs. [18. 6. 2024.].

[11] Firestore: https://firebase.google.com/docs/firestore. [18. 6. 2024.].

[12] Firebase Authentication: https://firebase.google.com/docs/auth. [18. 6. 2024.].

[13] Firebase Storage: https://firebase.google.com/docs/storage. [18. 6. 2024.].

SAŽETAK

Vue aplikacija za prodaju i udomljavanje životinja napravljena uz pomoć Vue.js framework-a i tehnologija kao što su HTML, CSS, Bootstrap i Firebase u programu Visual Studio Code, omogućuje posjetiteljima strance slobodno kretanje po naslovnici, stranicama gdje se nalaze oglasi mačaka i pasa te pregledavanje tih oglasa. Korisnici koji žele predati oglas moraju prvo kreirati korisnički račun kako bi to mogli učiniti. Prijavljeni korisnici dodatno, uz mogućnost predaje oglasa, imaju mogućnost uređivanja vlastitih oglasa, brisanja tih oglasa, kao i dodavanja profilne slike na svom profilu, mijenjanja vlastitih podataka i brisanja korisničkog računa.

Ključne riječi: CSS, Firebase, oglasnik, Vue.js, web aplikacija

ABSTRACT

Title: Vue applicatoin for selling and adopting animals

This Vue application, intended for selling and adopting animals, was created using the Vue.js framework and technologies such as HTML, CSS, Bootstrap and Firebase in the Visual Studio Code program, allows visitors to freely access the homepage and the pages containing the cat and dog ads as well as viewing those ads. Users wishing to upload an ad must first register and create a user account. Logged-in users, in addition to uploading an ad, can edit their ads' information and delete them. Additionaly, users can add a profile picture to their profile, update their profile information and delete their user account.

Keywords: Classifieds website, CSS, Firebase, Vue.js, web application

PRILOZI

GitHub repozitorij: https://github.com/Lora9299/Zavrsni-Rad