Izrada web aplikacije za oglašavanje i iznajmljivanje turističkih objekata

Pavlaković, Mihael

Undergraduate thesis / Završni rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka / Sveučilište u Rijeci**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:195:443893

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-09-13**

Sveučilište u Rijeci
Fakultet informatike i digitalnih tehnologija

Repository / Repozitorij:

Repository of the University of Rijeka, Faculty of Informatics and Digital Technologies - INFORI Repository

zir.nsk.hr

DIGITALNA KNJIŽNICA

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

Sveučilište u Rijeci – Odjel za informatiku

Jednopredmetna informatika

Mihael Pavlaković

Izrada web aplikacije za oglašavanje i iznajmljivanje turističkih objekata

Završni rad

Mentor: Doc. dr. sc. Lucia Načinović Prskalo

Rijeka, 03.09.2021

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 25.4.2021.

Zadatak za završni rad

Pristupnik: Mihael Pavlaković

Naziv završnog rada: Izrada web aplikacije za oglašavanje i iznajmljivanje turističkih objekata

Naziv završnog rada na eng. jeziku: Development of a web application for advertising and renting tourist facilities

Sadržaj zadatka: Glavni zadatak završnog rada je izraditi web aplikaciju za oglašavanje i iznajmljivanje turističkih objekata. Pritom će se koristiti odabrani alati za razvoj frontend i backend dijela aplikacije koji će se u radu i detaljno opisati. Također će se opisati i demonstrirati svi važni elementi i funkcionalnosti izrađene web aplikacije.

Mentor

Doc. dr. sc. Lucia Načinović Prskalo

Načinović Prskalo

Voditelj za završne radove

doc. dr. sc. Miran Pobar

Zadatak preuzet: 16.4.2021.

Pavlaković

(potpis pristupnika)

Sadržaj

1. Sažetak

U završnom radu opisan je cijeli postupak i koncept izrade web aplikacije *Svijet Apartmana*. *Svijet Apartmana* je e-commerce aplikacija koja služi za iznajmljivanje apartmana, soba ili kuća na internetu. Na početku se pojašnjavaju osnovni pojmovi, korištene aplikacije i alati kako bi se čitatelj lakše uputio u sam proces izrade aplikacije. Nakon toga se detaljno opisuju elementi aplikacije gdje su kroz slikovni prikaz gotove web aplikacije opisane funkcionalnosti pojedinih elemenata aplikacije. Potom su izneseni detalji vezani uz postavljanje razvojnog okruženja, opisivanje samog koda i funkcija. Na kraju je dan zaključak.

Ključne riječi: Web development, Svijet Apartmana, framework, funkcija, uvjet, razvoj web aplikacija, softverski razvojni okvir

2. Uvod

Kako bi se pobliže pojasnila odabrana tema završnog rada, potrebno je prvo objasniti osnovne pojmove kao što su web aplikacije, te što su to točno e-commerce aplikacije.

„Web aplikacija (web app) je kompjuterski program koji se izvodi u internet pregledniku. Web aplikacija radi na principu povezanosti klijent-server, i klijentu pruža grafičko sučelje kakvo je definirano na serveru.“ (Geek.hr, 2020).

U današnje se vrijeme web aplikacije koriste u velikoj mjeri. Neke od najpoznatijih su primjerice *Gmail* ili *Outlook* (vezano uz web mail), od oglasnika imamo primjerice aplikacije *Njuškalo*, *eBay*, *Amazon*, zatim društvene mreže tipa *Facebook* i *Twitter* te se taj popis nastavlja.

Kao što je navedeno unutar definicije, svim web aplikacijama je zajednička karakteristika da su izrađene i pohranjene na server. Kada mi kao korisnici utipkamo ime određenih aplikacija unutar web preglednika (*Google Chrome*, *Firefox*, *Opera* i sl.), prikažu nam se njihove početne stranice.

Za razliku od native mobilnih aplikacija, odnosno aplikacija koje su napravljene za određeni mobilni uređaj ili platformu, web aplikacije ne zahtijevaju upotrebu *Storea*. *Store* (hrv. dućan) je aplikacija koja unaprijed dolazi instalirana na našim mobilnim uređajima kako bismo preko nje mogli preuzeti daljnje mobilne aplikacije (primjerice *Instagram*, *Revolut*, *Njuškalo* i sl.). Također, web aplikacije je lakše održavati u smislu nadogradnji u budućim verzijama i fleksibilnost. Kod fleksibilnosti se misli na to da nije potreban razvoj za druge verzije sustava kao što je *iOS* ili *android* već je verzija na kojoj se radi kompatibilna sa oba sustava.Ovo su glavni razlozi zašto sam se odlučio na izradu web aplikacije, a ne native mobilne aplikacije.

Aplikacija *Svijet Apartmana* koja je izrađena u sklopu završnog rada, pripada e-commerce aplikacijama. Može se reći da je e-commerce aplikacija svaka aplikacija koja uključuje online transakciju. Kao primjer mogu poslužiti neke od prethodno navedenih aplikacija tipa *Amazona*, *eBay-a, Paypal-a* i sl. Osnovna definicija e-commerca glasi:„Electronic commerce ili e-commerce poslovni je model koji omogućuje tvrtkama i pojedincima da kupuju i prodaju stvari putem interneta.” (Bloomenthal, 2020).Korištenje e-commerce aplikacija također bilježi stalni rast. „Prema statistici, 73% ljudi unutar Hrvatske koristi Internet, od toga imamo 61% ljudi koji su kupili barem nešto preko interneta i na kraju 36% njih kupi nešto na mjesečnoj bazi.“ (Ecommerce News, 2021). Gledajući ukupno, online prodaja u 2020. godini iznosi rekordnih 464 milijuna eura. Analizirajući i prethodne godine možemo primijetit konstantan rast na tom području.

Prilikom odabira vrste aplikacije odlučeno je da će se raditi na izradi turističke e-commerce aplikacije. Jedna od bitnih karakteristika aplikacija tog tipa je mogućnost rezervacije sobe, apartmana i sl. unutar same aplikacije. U vrijeme Covid-19 pandemije mogli smo primijetiti porast u korištenju ovakvog tipa aplikacije. „Razlog tome je vrlo jednostavan - ljudi su htjeli izbjeći prenatrpane hotele i zamijeniti ih prostranijim apartmanima ili kućama.“ (Fox, 2021).

Prema svemu navedenom, tema ovog završnog rada pripada trenutno vrlo aktualnom području razvoja softvera i web aplikacija, koje će se i u budućem razdoblju ubrzano razvijati.

Rad je organiziran tako da su u 3. poglavlju opisane tehnologije korištene za izradu web aplikacije. U 4. poglavlju slijedi opis elementa i funkcionalnost aplikacije i kraju se nalazi 5. poglavlje u koje se iznose zaključci završnog rada.

3. Opis tehnologija korištenih za izradu web aplikacije

3.1. Alati i programski jezici za razvoj sučelja

Za pisanje i uređivanje koda korišten je *Visual Studio Code* i dodatak *Vetur*. Aplikacija je razvijena na *Vue.js* razvojnom okviru, dok je za vizualni dizajn zaslužan *Bootstrap*. Što se tiče podataka, kao baza podataka korištena je *Firebase Firestore*, a *Firebase Storage* za pohranu fotografija apartmana. Po završetku razvoja aplikacije, za host je izabran *GitHub Pages*.

3.1.1. Visual Studio Code

Za pisanje koda potrebna je neka vrsta tekstualnog editora (uređivača) koja olakšava njegovo pisanje. Najjednostavnija vrsta editora koja bi se mogla koristiti za izradu web stranica je ona slična primjerice *Microsoft Notepad* editoru koji je već ugrađen unutar operacijskog sustava *Windows* i ne zahtjeva nikakva dodatna preuzimanja. Postupak je jednostavan - otvori se editor, piše se kod i zatim se tu datoteku pohrani s .html ili .css ekstenzijom i dobijemo osnovni tip web stranice.
Iako bi se kod mogao pisati unutar editora kao što je *Notepad*, postoje brojni nedostaci istog. Neki od njih su: nemogućnost automatskog uvlačenja dijelova koda, nemogućnost primjene boje na tagove zbog lakšeg raspoznavanja elemenata, nemogućnost automatskog dopunjavanja i slično.

U izradi aplikacije *Svijet Apartmana* korišten je *Visual Studio Code* od strane Microsofta, iz razloga što je široko usvojen i ima veliki broj funkcija koje nedostaju osnovnim tekstualnim editorima. Njegova velika prednost je mogućnost uređivanja postavki, dodataka i samog izgleda. Od dodataka se koristio *Vetur* koji je koristan za sintaktičko podcrtavanje elemenata te dopunjavanje. *Vetur* je poseban dodatak koji se koristi kod programiranja sa *Vue.js* radnim okvirom.

3.1.2. Node.js

Node.js je server side platforma razvijena na *Google Chrome-ovom JavaScript Engine-u*. Pojednostavljeno rečeno, *Node.js* nam omogućava da koristimo *JavaScript* programski jezik na backend-u.

Za izradu web aplikacije *Svijet Apartmana*, *Node.js* se koristi za postavljanje željenog razvojnog okruženja. Na početku se treba preuzeti posljednja verzija *Node.js-a* nakon čega slijedi instalacija na računalo. Nakon toga pokreće se terminal i upisuje se naredba *npm install -g @vue/cli*. Ovom se naredbom pomoću node package managera globalno (-g atribut) instalirava *Vue.js framework* i njegov *command line interface*.

3.1.3. Vue.js

Vue.js je progresivni razvojni okvir (engl. Framework) za izradu korisničkih sučelja. Neki od sličnih framework-ova su Angular i React. Za *Vue.js* sam se odlučio iz jednostavnog razloga, a to je da je prikladniji za početnike.

Sam početak je vrlo jednostavan, prvo se instalira korisničko sučelje, a nakon toga se kreira sama aplikacija pomoću naredbi *vue create* "naziv-projekta" koji je u našem slučaju *Svijet Apartmana*. Zatim je odabrana verzija Vue.js-a 3.0 te željene značajke unutar projekta (odabrane su *Router* i *Vuex*).

Router služi za navigaciju unutar većine *Single Page Aplikacija*. „Single Page Aplikacije", ili skraćeno SPA, web su aplikacije koje stvaraju iluziju mijenjanja stranica, a korisnik je zapravo cijelo vrijeme na jednoj stranici." (Žarković, 2018). Pomoću *Router* značajke, mogu se kreirati pojedine komponente te im dodijeliti nazive ruta na kojima će se one nalaziti. Uz *Router* dolazi i *History* dodatak koji nam služi za vraćanje unatrag po prethodnim rutama.
Vuex je biblioteka za upravljanje sa globalnim *State-om. State* su podatci koje aplikacija treba i koji utječu na ono što korisnik vidi na ekranu. Ti podatci su najčešće reaktivni što znači da se mijenjaju ovisno o slučaju upotrebe. *State* se dijeli na *lokalni state* i *globalni state*. Lokalni je onaj koji utječe samo na jednu komponentu (primjerice kada korisnik unese nešto u formu, za prikaz nekog podatka i sl.) dok globalni utječu na više komponenata ili na cijelu aplikaciju (primjerice prijava korisnika, stanje košarice za kupovinu i sl.).

3.1.4. Bootstrap

Bootstrap je css razvojni okvir (engl. Framework) koji nam pomaže prilikom brzog kreiranja responzivnog dizajna za mobilne uređaje. Sastoji se od raznih komponenti koje su gotove za korištenje, a pojedine predloške je potrebno prilagoditi pojedinom projektu. Bootstrap se koristi tako da se unutar html elementa aplikacije dodaju željene klase. Često korištena klasa u aplikaciji *Svijet Apartmana* je *flex klasa* koja služi za prikazivanje *flexboxa (fleksibilni raspored kutija unutar containera)* zbog lakše organizacije elementa.

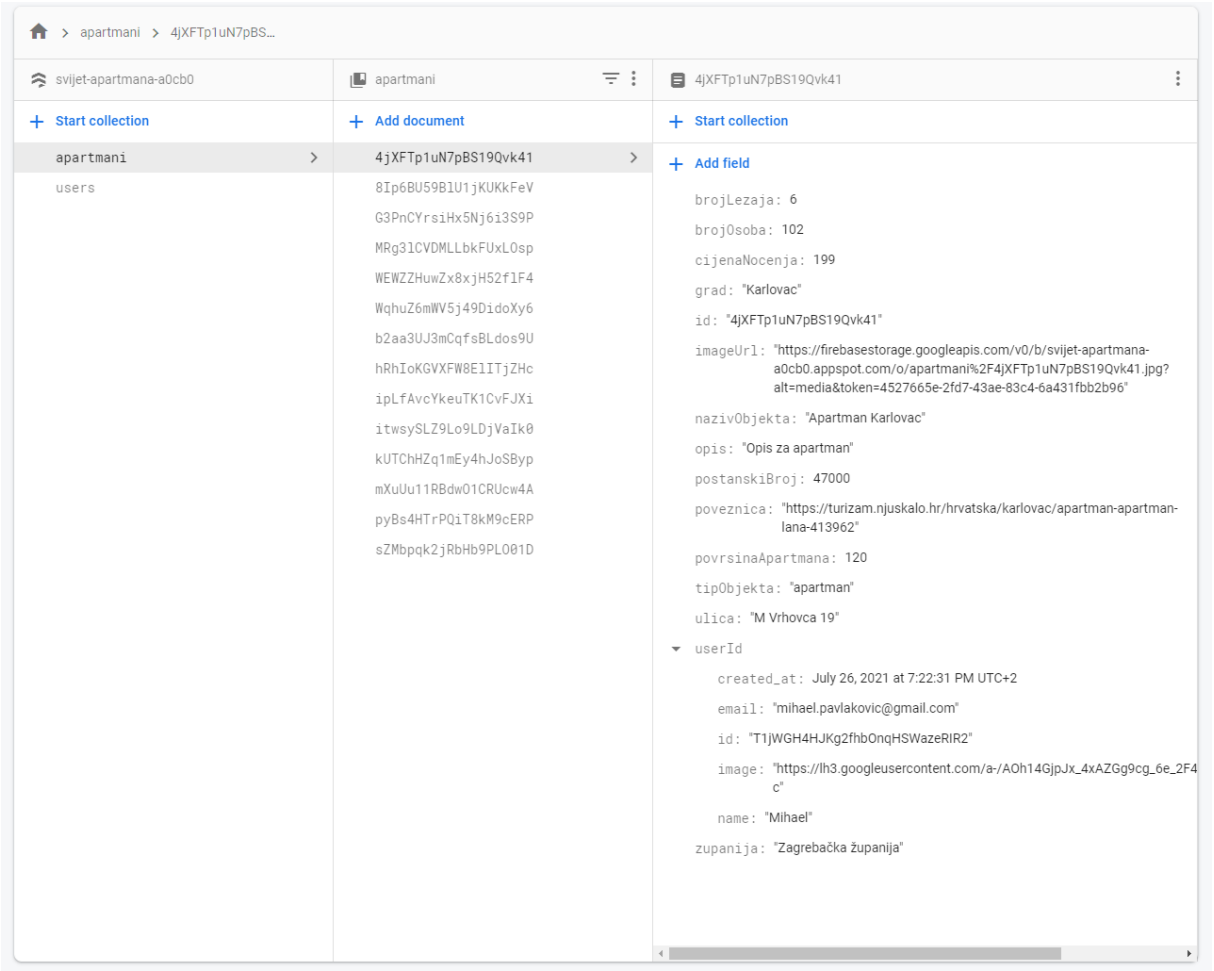
3.1.5. Firebase

Firebase je Googleova platforma za stvaranje mobilnih i web aplikacija. Sastoji se od raznih servisa koje možemo koristiti, a neki od njih su *Analitika*, *Autentifikacija*, *Realtime* baze podataka itd.
Unutar projekta koristilo se nekoliko Firebase servisa. Jedan od njih je *Autentifikacija* koji u ponudi ima razne načine prijave, ali među najraširenijima je onaj sa Google računom. Zatim servis koji je među najkorištenijima od strane Firebasea, a to je Firestore baza podataka te servis koji je dio Firebase Storagea kojeg smo koristili za pohranu fotografija apartmana. Služi nam kako ne bismo trebali pohranjivati velike fotografije unutar same baze, već samo njihove url linkove koji vode do Storagea.

3.1.6. Firebase Firestore

„Firebase Firestore je fleksibilna, skalabilna NoSQL cloud baza podataka koja pohranjuje, sprema i sinkronizira podatke za klijenta i server-side razvoj.“ (Google, Firebase, 2021).

Ova baza podataka prilično je jednostavna za korištenje. Osim toga, Firebase dokumentacija je izvrsno dokumentirana te se vrlo jednostavno nađu i implementiraju željeni elementi.



Slika 1 Prikaz baze podataka

Na slici 1 prikazana je baza podataka unutar Firestorea koja se sastoji od dvije kolekcije - *apartmani* i *users*. Kolekcija *apartmani* sadrži dokumente, a svaki dokument predstavlja jedan objavljeni apartman unutar aplikacije. Svaki dokument sadrži podatke o apartmanu te podatke o korisniku koji ga je objavio. U *users* kolekciji imamo zapise svih korisnika koji su se prijavili na stranicu te njihove podatke koji se prikazuju na korisničkim profilima. Isto kao i kod kolekcije *apartmani,* tako i se i unutar kolekcije *users* svi podatci pohranjuju unutar zasebnih dokumenta

3.1.7. Firebase Storage

„Firebase Storage nam omogućava upload-anje i dijeljenje korisnički generiranog sadržaja, kao što su slike i videozapisi.” (Google, Firebase, 2021).

U aplikaciji *Svijet Apartmana*, Firebase Storage se koristi za pohranu fotografija apartmana. Korisnik sam proizvoljno odabere svoju fotografiju apartmana i uploada ju na aplikaciju odnosno Storage. Svaka fotografija se pohranjuje unutar mape apartmani i dobiva jedinstveni naziv po id-u objave.
Pohranu vršimo na ovaj način kako bismo izbjegli opterećivanje baze podataka velikim datotekama. Kada se fotografija prenese na *Storage* kao odgovor dobijemo link i sljedeće što nam preostaje je pohranit taj link u *Firebase Firestoreu* kao string. Ovim načinom smo smanjili samu količinu podataka koja se čuva unutar same baze podataka.

3.1.8. Git

„*Git* je vrsta sustava kontrole verzija koja olakšava praćenje promjena na datotekama." (Cooper, 2017). Kako bismo ga mogli koristili na našem računalu, potrebno ga je preuzeti sa službene web stranice i instalirati.

Putem *Gita* se projekt može postaviti na neku od stranica za pohranjivanje koda. Među najpopularnijim je *GitHub*.

3.1.9. GitHub

GitHub, Inc. pružatelj je internetskog hostinga za razvoj softvera i kontrolu verzija pomoću *Gita*. Nudi distribuiranu kontrolu nad verzijama i funkcionalnost upravljanja izvornim kodom Gita, plus vlastite značajke. Neke od tih značajki kao *GitHub Pages* smo koristili na našem projektu za prikaz web aplikacije.

3.2. Server

„Poslužitelj je računalni program ili uređaj koji pruža uslugu drugom računalnom programu i njegovom korisniku, poznatom kao klijent." (Posey, 2021). Aplikacija *Svijet Apartmana* za server koristi *GitHub* dok je hosting izveden preko *GitHub Pages*.

3.2.1. GitHub Pages

GitHub sadrži vrlo korisnu značajku ukoliko želimo u kratkom vremenskom roku postavit stranicu na Internet, a ona se zove *GitHub Pages*. To je host (bilo koji uređaj povezan u računalnu mrežu, a koji može ostvarit komunikaciju s drugim sličnim uređajem) koji uzima html, css i javascript datoteke unutar našeg repozitorija i postavlja ih na određenu domenu koju korisnik odredi.

Prilikom postavljanja aplikacije koja koristi neke od softverskih razvojnih okvira (engl. Frameworka) potrebno je napraviti *build* te aplikacije kako bi se pravilno prikazala na webu. Build aplikacije se radi unosom naredbe *npm run build* u terminal. Nakon što se uspješno odradi build, novonastale datoteke će se nalaziti unutar *dist* mape u našem projektu. Ulaskom u *dist* mapu pokrećemo *git init* naredbu koja izvodi proces prenošenja stranice na *GitHub*. Po završetku izvođenja određujemo koje datoteke želimo prenijeti. U našem slučaju to su sve datoteke unutar kojih se dogodila neka promjena od prethodnog prijenosa ili sve datoteke ukoliko nam je to prvi prijenos. Naredba *git commit -am* "Poruka što smo promijenili" može se koristiti za pripremu svih promjena za postavljanje na GitHub

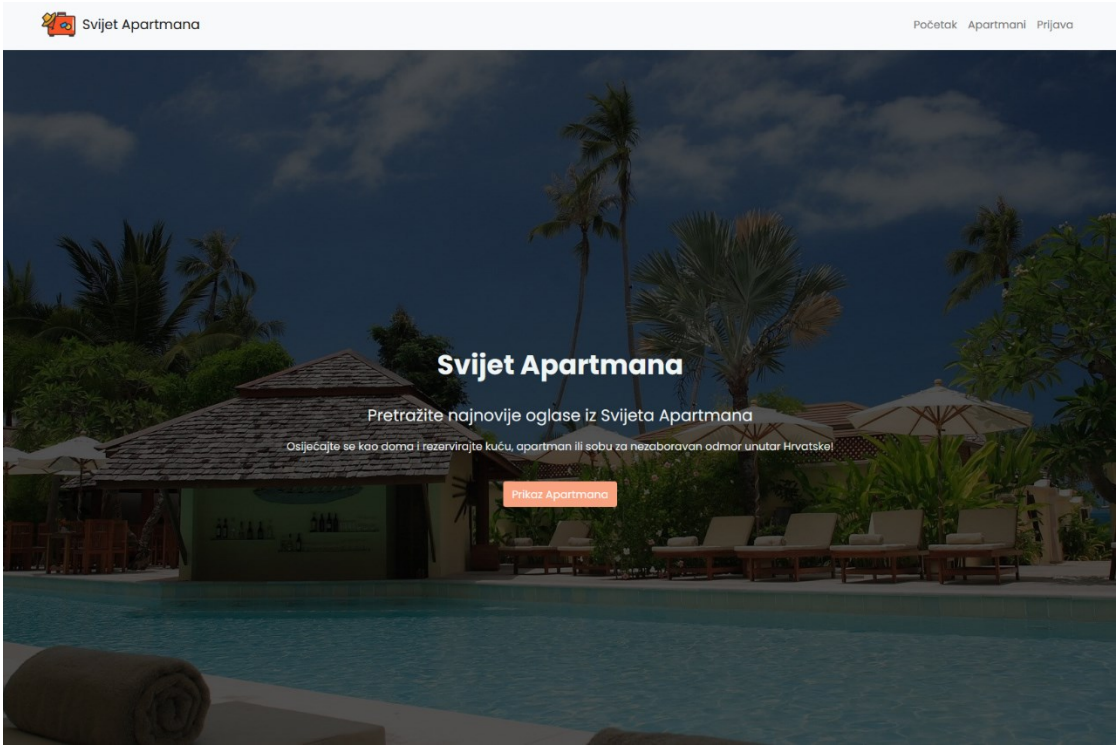
s odgovarajućom porukom. Posljednji korak je poslati sve te datoteke na njihov host naredbom *git push -f git@github.com:username/svijet-apartmana.git master:gh-pages*.

4. Opis elemenata i funkcionalnosti aplikacije

4.1. Opis elemenata aplikacije

U narednim su poglavljima detaljno opisani pojedini elementi i funkcionalnosti web aplikacije *Svijet Apartmana*.

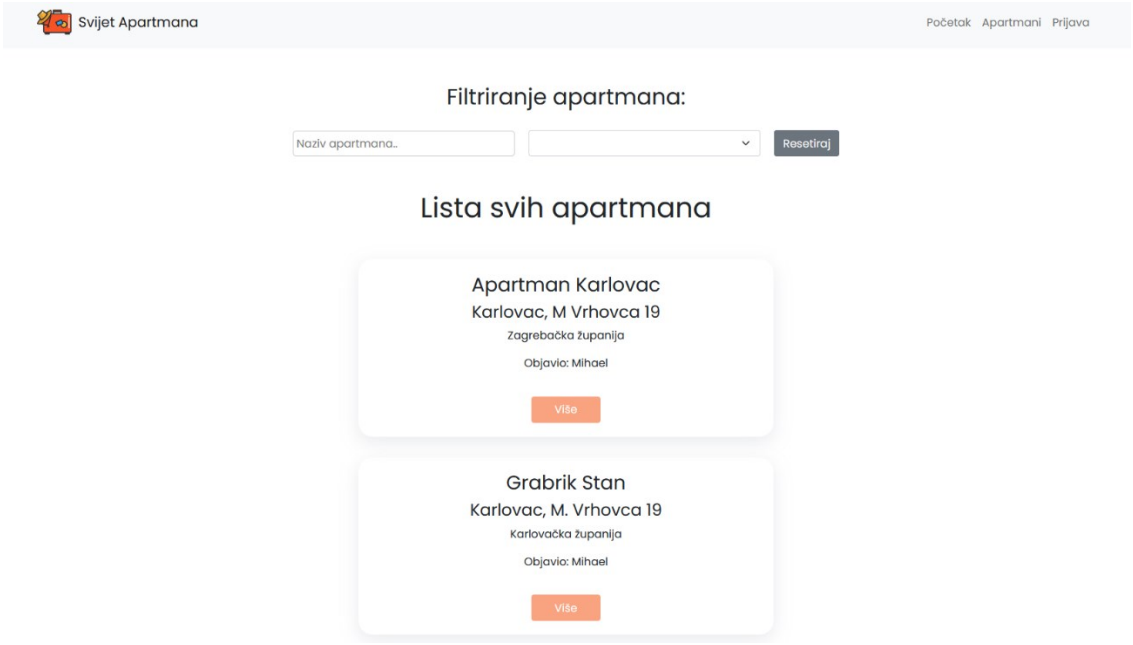
4.1.1. Početna stranica



Slika 2 Izgled početne stranice web aplikacije

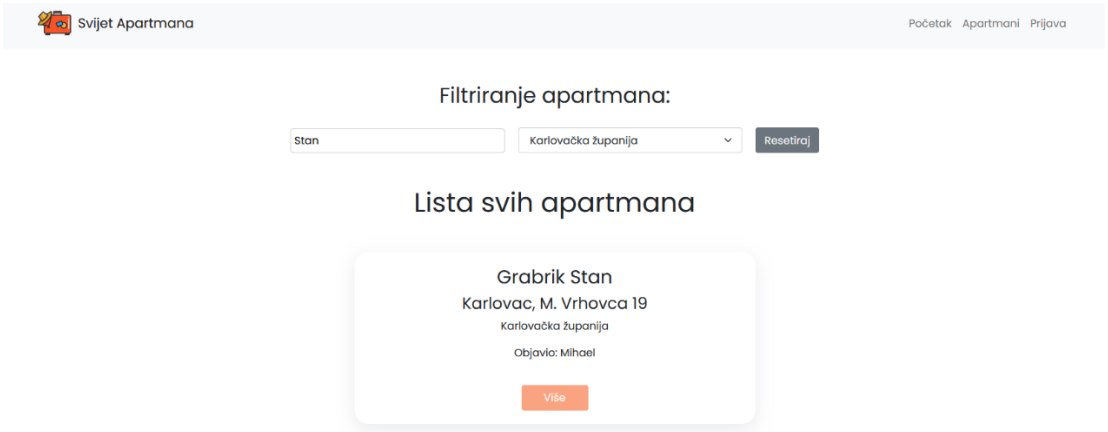
Nakon što se aplikacija učita, korisniku će biti predstavljena početna stranica aplikacije. Možemo odmah primijetit veliku pozadinsku sliku koja na sebi ima *overlay* kako bi bolje došao do izražaja glavni tekst stranice. Na slici 2 vidi se postavljen tekst početne stranice i gumb (engl. Button) koji nas vodi do prikaza svih apartmana unutar aplikacije. Isto tako na samom vrhu imamo prikazanu navigaciju dok korisnik nije ulogiran.

4.1.2. Apartmani



Slika 3 Izgled kartice apartmani

Stranica apartmani se sastoji od dvije velike komponente (slika 3) - jedna je filter, a druga je prikaz apartmana. Kod filtera imamo 2 načina filtriranja - jedan putem naziva apartmana, a drugi putem županije. Ukoliko imamo uključena oba filtera, pored polja se nalazi gumb za resetiranje filtera (prikazano na slici 4).



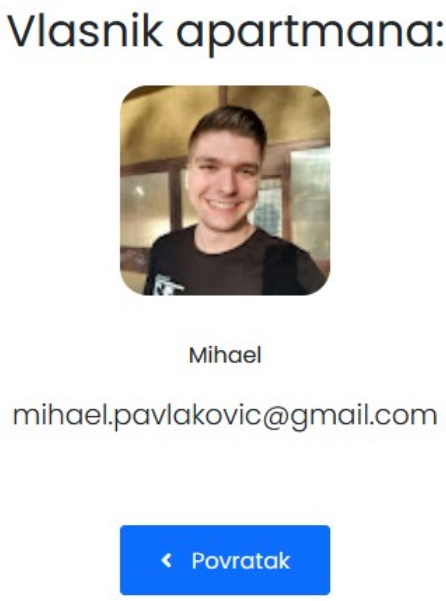
Slika 4 Izgled kartice Apartmani prilikom korištenja filtera

Ovisno o primijenjenim filterima, mijenja se prikaz apartmana koji odgovaraju zadanim vrijednostima (slika 4). Kod prikaza apartmana uočljiv je prikaz u obliku kartice. Unutar same kartice nalaze se neke najbitnije informacije o apartmanu kao što su naziv, adresa, grad, županija te osoba koja ga je objavila. Isto tako unutar kartice imamo i gumb koji nas vodi na detaljniji prikaz informacija o apartmanu (prikazano na slici 5).

4.1.3. Detaljan prikaz apartmana

Slika 5 Izgled detaljnijeg prikaza apartmana

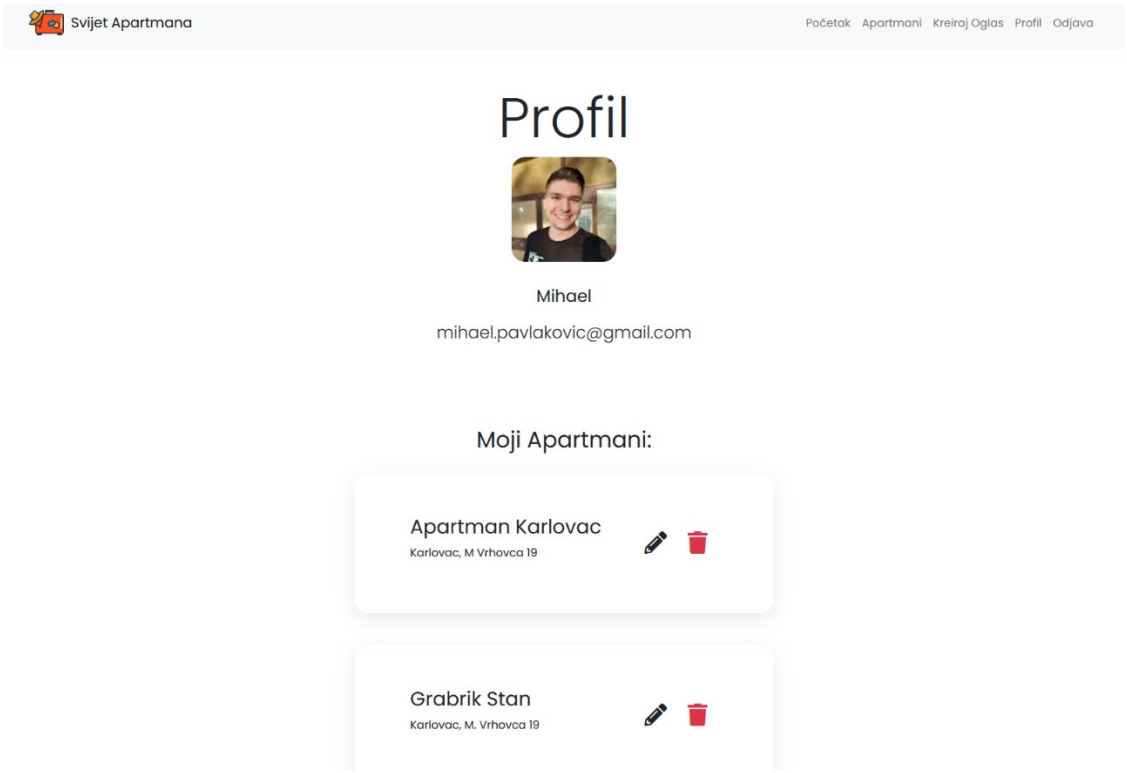
Na stranici *detaljan prikaz apartmana* će nas prvo dočekati fotografija apartmana (za potrebe slikanja početna fotografija je umanjena). Zatim slijede svi podatci o apartmanu od cijene najma, broja dozvoljenih osoba, broja ležaja i opisa ukoliko iznajmljivač želi upisati neke dodatne karakteristike. Sljedeći segment se odnosi na profil iznajmljivača ukoliko se zainteresiramo za apartman (prikazano na slici 6).



Slika 6 Izgled komponente vlasnika apartmana

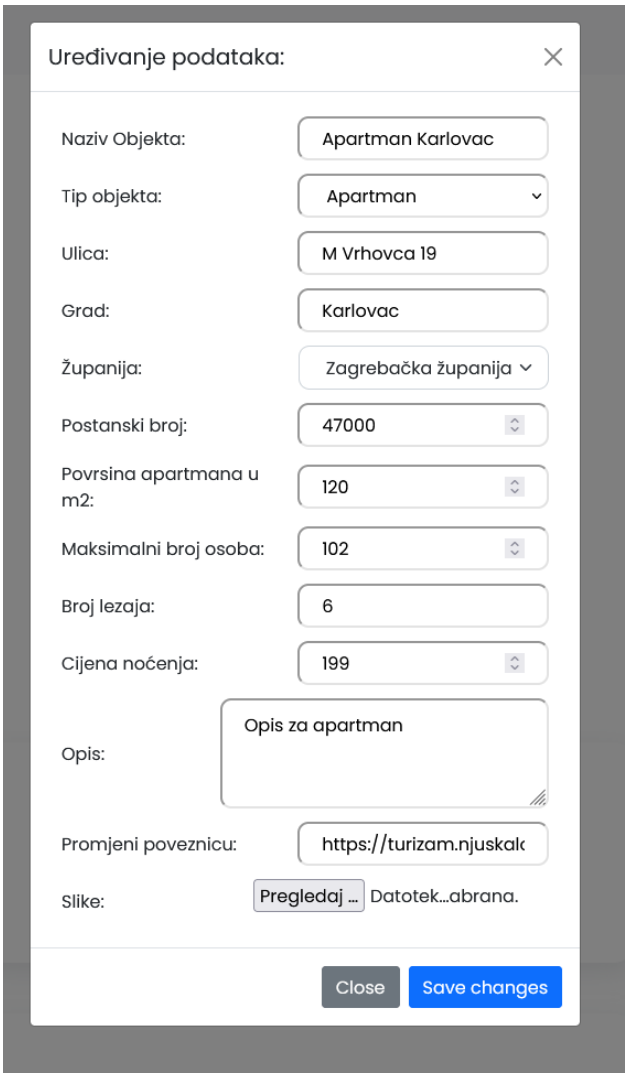
Ovdje imamo jasnu naznaku u obliku teksta da se radi o vlasniku apartmana. Ispod toga se nalazi slika profila korisnika, ime i e-mail adresa preko koje se kupac može javiti. Na kraju same stranice, ukoliko nam je navigacija predaleko, imamo gumb za povratak na prethodnu stranicu *apartmani*.

4.1.4. Profil



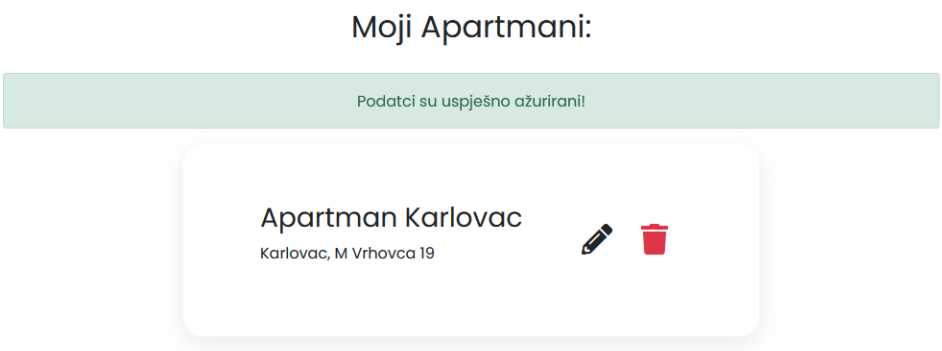
Slika 7 Izgled kartice Profil

Prikaz korisničkog profila je vrlo jednostavan (prikazano na slici 7). Na samom vrhu stranice imamo korisnika sa fotografijom, ispod toga ime i e-mail adresu. Nakon toga slijede svi apartmani koje korisnik ima u svom vlasništvu odnosno koje iznajmljuje. Prikaz je isto u obliku kartica kako bi se nastavila konzistentnost izgleda i dizajna aplikacije. Unutar kartice su navedeni ime apartmana, ulica, grad i 2 gumba koji pozivaju različite funkcije (prikazano na slici 8).



Slika 8 Izgled dijaloškog okvira za uređivanje podataka

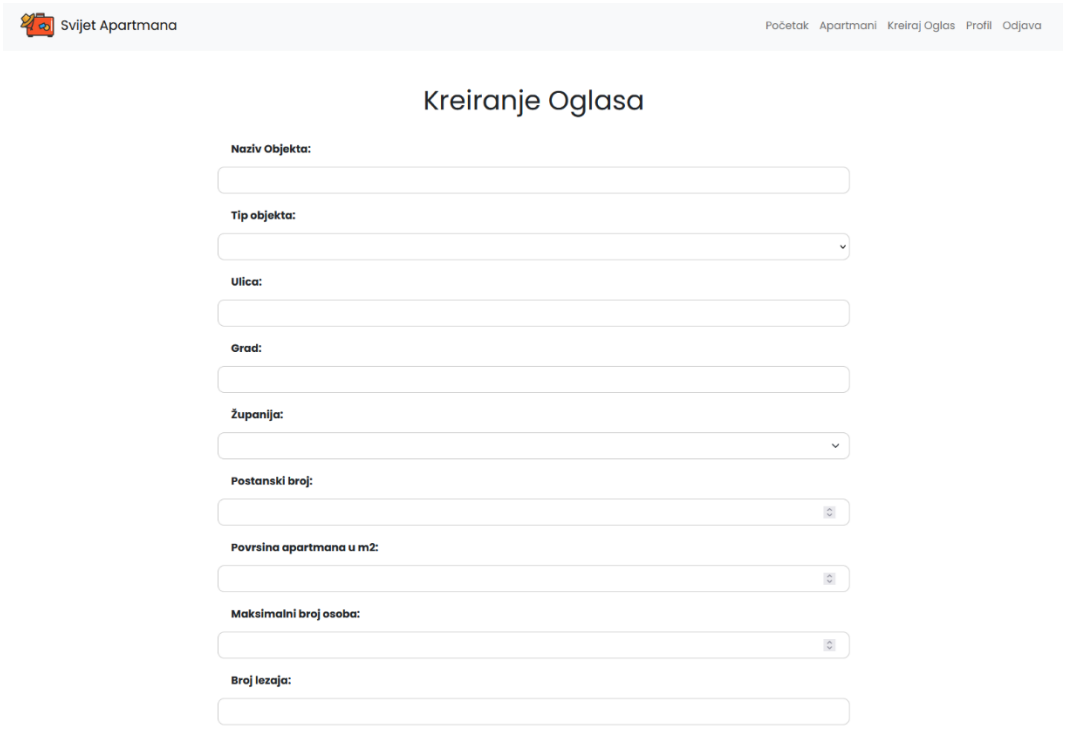
Prvi gumb služi za uređivanje podataka apartmana. Klikom na gumb olovke prikazuje se dijaloški okvir za uređivanje podataka. Nakon što smo izmijenili sve željene podatke klikom na gumb *Save changes*, zatvara se dijaloški okvir i ispisuje se poruka uspješnog ažuriranja podataka (prikazano na slici 9).



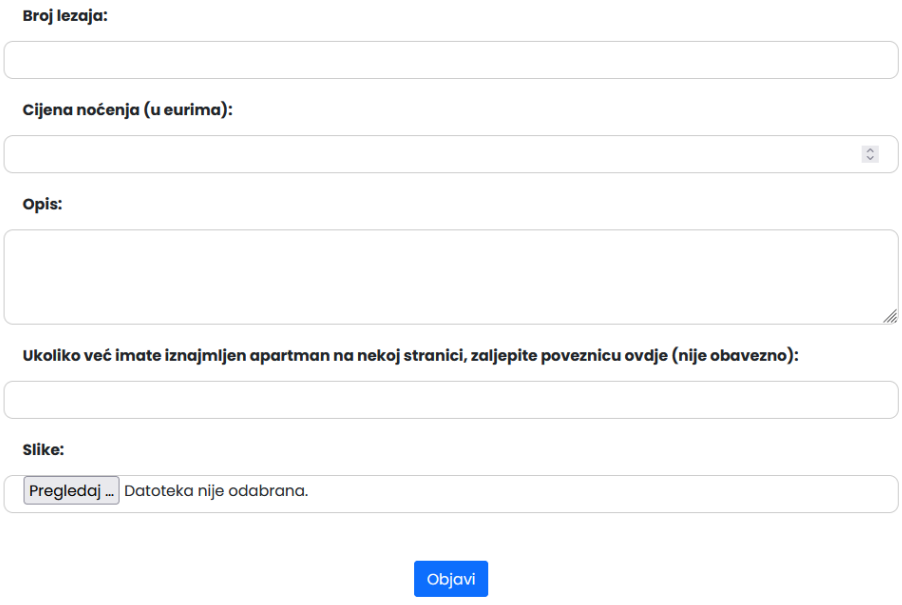
Slika 9 Izgled obavijesti o uspješnom ažuriranju podataka

Isto tako klikom na ikonicu smeća briše se objava apartmana i prikazuje se poruka o uspješnosti izvršavanja te radnje.

4.1.5. Kreiranje oglasa



Slika 10 Izgled kartice Kreiraj Oglas



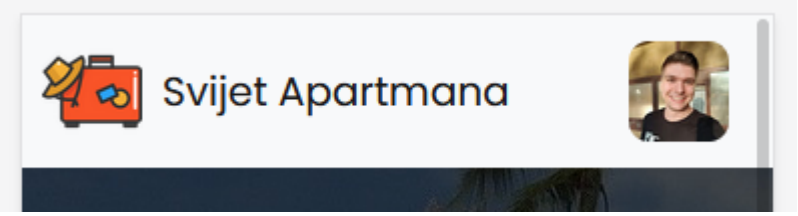
Slika 11 Izgled kartice Kreiraj Oglas 2

Kreiranje oglasa je napravljeno na principu velike forme (prikazano na slikama 10 i 11). Korisnika se traže sve bitne informacije za iznajmljivanje apartmana. Ukoliko korisnik želi dodati neke dodatne karakteristike apartmana, može to napisati u *Opis* dijelu forme. Po završetku ispunjavanja forme, na kraju stranice se nalazi

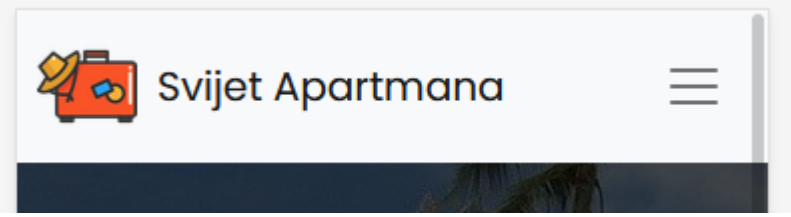
gumb *Objavi* čime se svi upisani podatci pohranjuju unutar baze podataka, te korisnika prebaci na stranicu *apartmani*.

4.1.6. Mobilni prikaz aplikacije

Što se tiče mobilnog prikaza, aplikacija je napravljena tako da se ispravno prikazuje na svim uređajima. Bitna razlika između desktop i mobilne aplikacije je unutar navigacije. Kod mobilnog prikaza navigacija u početnom izdanju (slika 13) sa desne strane sadrži *hamburger icon*. Prilikom prijave korisnika to se mijenja i prikazuje korisnikova profilna slika na tom mjestu (slika 12).



Slika 12 Izgled navigacije u mobilnom prikazu kada je korisnik prijavljen



Slika 13 Izgled navigacije u mobilnom prikazu kada je korisnik odjavljen

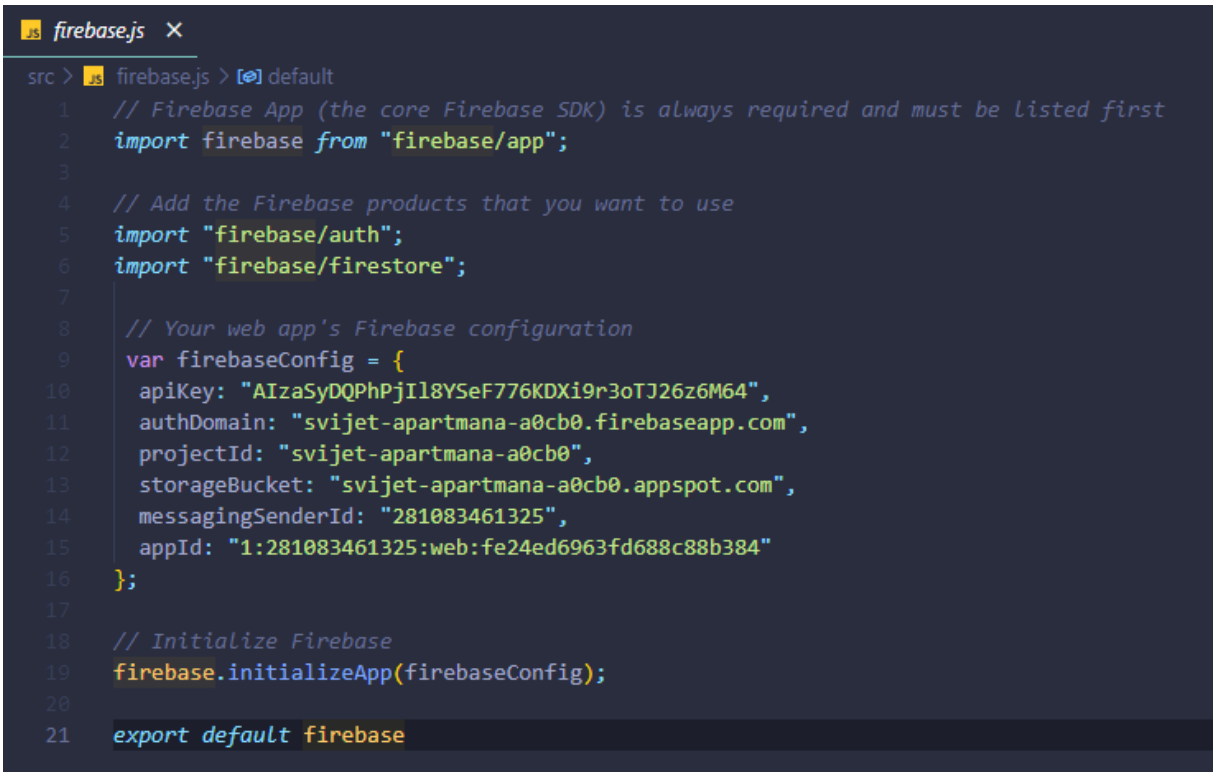
Isto tako na stranici apartmani kod filtera (slika 14) vidimo da su oni posloženi u retke umjesto u stupce kao na desktop prikazu.

Slika 14 Izgled filtera za apartmane u mobilnom prikazu

Ostali elementi su jednako prikazani i u desktop i mobilnoj verziji, samo su malo smanjeni u mobilnom prikazu kako bi se ispravno prikazali na tabletima i mobilnim uređajima.

4.2. Postavljanje Firebase-a

Prilikom izrade aplikacije *Svijet Apartmana*, Firebase kao alat bio je potreban za spremanje bitnih informacija o korisniku, apartmanima te za pohranu fotografija apartmana. Prvi korak je bio kreiranje samog projekta unutar *Firebase* konzole. Odabire se ime projekta, prihvaćaju se svi potrebni uvjeti i nastavlja se na konzolu projekta. Nakon što se uspješno kreira projekt, potrebno je kopirati kod *SDK setup and configuration* (slika 15) kako bi se projekt uspješno konfigurirao i kako bi se preuzeli svi važeći podatci o stvorenom projektu na Firebase platformi unutar razvojnog okruženja na lokalnom računalu. Nakon toga unutar konzole je potrebno pokrenuti naredbu *npm install ''save firebase* kako bi se kreirale sve potrebne datoteke unutar razvojnog okruženja. Po završetku kreiranja projekta unutar Firebase konzole i spajanja s razvojnim okruženjem na računalu, može se krenuti s daljnjim koracima razvoja aplikacije.

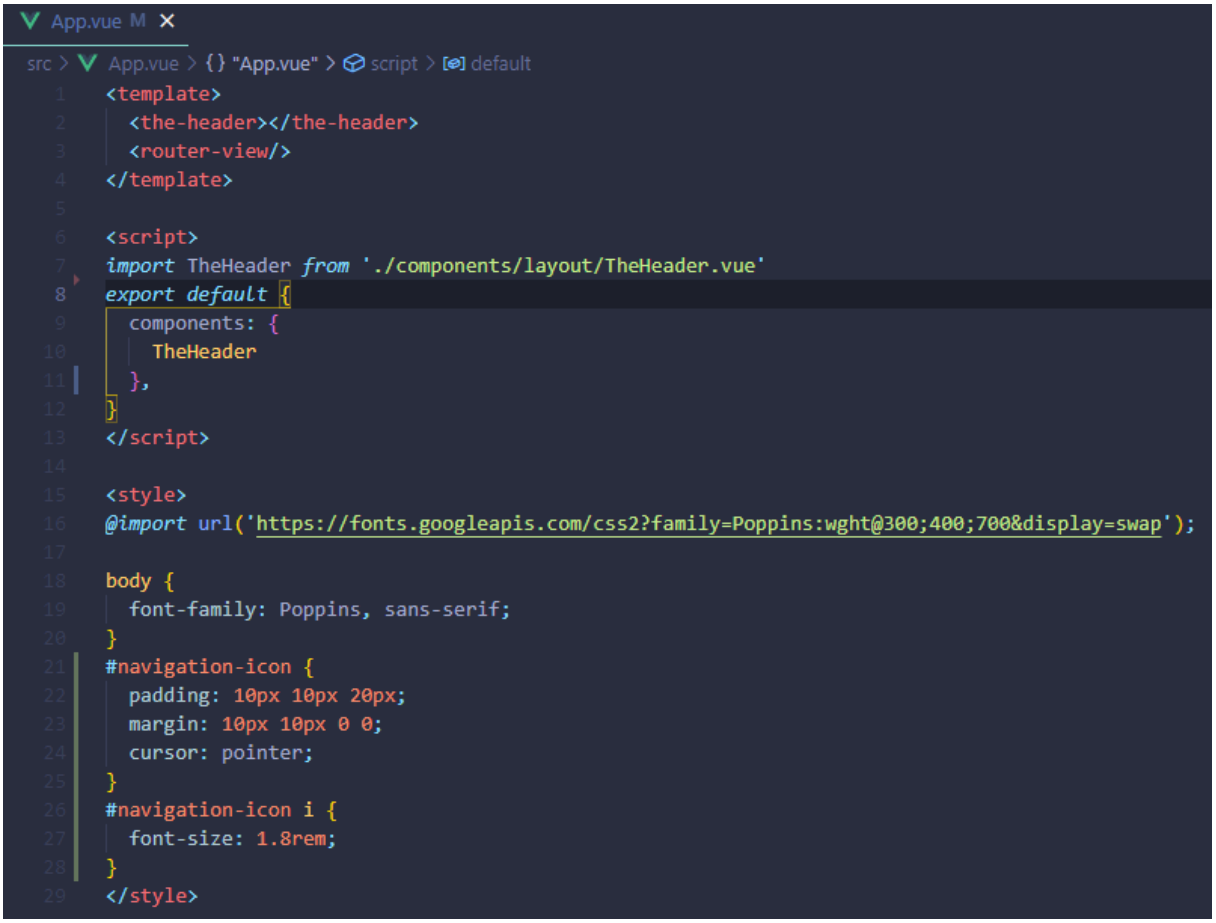


Slika 15 Kod firebase.js-a

4.3. Autentifikacija

Za prijavu u aplikaciju, korišten je *Firebase Authentication*. Uz Google prijavu moguće je odabrati još dosta drugih opcija, a za aplikaciju *Svijet Apartmana*, odabran je *Google SignIn* kao najbolja opcija. Na taj se način smanjuje nepotreban korak same registracije korisnika i kasnije prijavu, odnosno brža je prijava putem samog Google računa. Isto tako kada se prijavimo s Google računom, pohranjuju se podaci s kojima programer može raspolagati kao što su primjerice profilna fotografija korisnika, e-mail adresa, ime i prezime i slično. Ova opcija je bila korisna prilikom izrade aplikacije *Svijet Apartmana* jer se slika vlasnika apartmana može prikazati unutar same objave apartmana te prikaza profila.

4.4. App.vue



Slika 16 Kod App.vue komponente

App.vue je glavna komponenta aplikacije. Kôd App.vue komponente je prikazan na slici 16. U template dijelu datoteke se učitavaju i prikazuju komponente *the-header* i *router-view*. *The-header* komponentu je ovdje uključena jer je navigacija prisutna na svakoj stranici. Na ovaj se način izbjeglo uvođenje *the-header* komponente po svim ostalim komponentama - ovako je definirana na jednom mjestu i samim time ima manje pozivanja unutar aplikacije. *Router-view* komponenta je zadužena za prikazivanjerazličitih komponenti ovisno o tome na koji dio stranice kliknemo.

Script dio glavne komponente je poprilično jednostavan. Sadrži import za *the-header* komponentu. Definirana je u *components* dijelu kako bi bila dostupna trenutnoj komponenti unutar koje se trenutno nalazimo.

Style dio stranice je globalan, stoga se kroz cijelu aplikaciju protežu isti stilovi i importani font.

4.5. Navigacija

TheHeader.vue je univerzalna komponenta koja se koristi kod glavne komponente *App.vue* (sadržaj *The Header.vue* komponente je prikazan na slici 17). Napravljena je tako da

prikazom odgovara uz svaku komponentu te se prilagođava *stateu*, odnosno tome je li korisnik prijavljen ili ne.

TheHeader.vue M

src > components > layout > TheHeader.vue > {} "TheHeader.vue"

```
<template>
  <nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="container-fluid">
      <router-link class="navbar-brand" to="/">
        <div class="navbar-logo">
          <img src="../../assets/logo-case.png" alt="Logo od Svijeta Apartmana">
          <p>Svijet Apartmana</p>
        </div>
      </router-link>
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent"
      aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
        <div class="d-flex align-items-center user" v-if="isLoggedIn">
          <p>{{ user.name }}</p>
          <img id="profile-image" referrerpolicy="no-referrer" :src="user.image" alt="User image">
        </div>
        <span class="navbar-toggler-icon" v-else></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav ms-auto mb-2 mb-lg-0 text-center">
          <li class="nav-item">
            <router-link class="nav-link" to="/">Početak</router-link>
          </li>
          <li class="nav-item">
            <router-link class="nav-link" to="/apartmani">Apartmani</router-link>
          </li>
          <div v-if="!isLoggedIn">
            <li class="nav-item">
              <a class="nav-link" @click="googleSignIn">Prijava</a>
            </li>
          </div>
          <div v-else class="d-flex mobile-nav">
            <li class="nav-item">
              <router-link class="nav-link" to="/novi-oglas">Kreiraj Oglas</router-link>
            </li>
            <li class="nav-item">
              <router-link class="nav-link" to="/profil">Profil</router-link>
            </li>
            <li class="nav-item">
              <a class="nav-link" @click="googleSignOut">Odjava</a>
            </li>
          </div>
        </ul>
      </div>
    </div>
  </nav>
</template>
```

Slika 17 Kod template dijela TheHeader.vue komponente

Template dio koristi standardni *Bootstrap layout* za navigaciju uz par izmjena. Prva od njih je logo i naziv aplikacije koji je omotan s *router-link tagom* (linija broj 5 na slici 17). *Router-link* ima atribut „to" koji vodi do određene rute definirane unutar routes foldera. Nakon toga slijedi izmjena koja se implementira prilikom mobilnog prikaza aplikacije, kada je korisnik ulogiran. Onda se prikazuje njegovo ime i profilna sliku umjesto default hamburger ikonice. Ova logika je izvedena pomoću v-if uvjeta na 2 elementa (div i span) koja su na istoj razini. Izmjena je vidljiva od 11. linije do 14. linije na slici 17. Zadnja izmjena u template dijelu se odnosi na sam prikaz linkova unutar navigacije. Imamo 2 stanja, prvo je kada korisnik nije prijavljen (linija 25, slika 17), a drugo kada je korisnik prijavljen (linije 30 do 40, slika 17).

Slika 18 Kod script dijela TheHeader.vue komponente

Script dio navigacije se sastoji od dvije metode i dvije *computed* funkcije. Unutar metoda imamo *googleSignIn* koji kroz *store* poziva *login* funkciju te *googleSignOut* koja poziva *logout* funkciju unutar istog *storea*. *Computed* funkcija cijelo vrijeme „sluša“ jesu li nastupile promjene. Prva *computed* funkcija je *isLoggedIn* koja vraća *true* ili *false* vrijednost iz *storea* (linija 59 na slici 18), dok druga funkcija *user* vraća vrijednosti (linija 62 na slici 18).

4.6. Router

index.js M

src > router > index.js > routes > component

```
import { createRouter, createWebHistory } from 'vue-router'
import Home from '../views/Home.vue'
import Profil from '../views/Profil.vue'
import NoviOglas from '../views/NoviOglas.vue'
import Apartmani from '../views/Apartmani.vue'
import ApartmaniDetaljno from '../views/ApartmaniDetaljno.vue'
import NotFound from '../views/NotFound.vue'

const routes = [
  {
    path: '/',
    name: 'Home',
    component: Home
  },
  {
    path: '/profil',
    name: 'Profil',
    component: Profil
  },
  {
    path: '/apartmani',
    name: 'Apartmani',
    component: Apartmani
  },
  {
    path: '/apartmani/:id',
    name: 'ApartmaniDetaljno',
    component: ApartmaniDetaljno,
    props: true
  },
  {
    path: '/novi-oglas',
    name: 'NoviOglas',
    component: NoviOglas
  },
  {
    path: '/:notFound(.*)',
    component: NotFound
  }
]

const router = createRouter({
  history: createWebHistory(process.env.BASE_URL),
  routes
})

export default router
```

Slika 19 Kod index.js datoteke unutar router mape

Index.js (slika 19) je dokument koji se nalazi unutar mape router, a služi za određivanje ruta na stranici. Kako bismo kreirali router i povijest prvo je potrebno importati *createRouter* i *createWebHistory* komponente iz *vue-router*. Nakon toga slijedi import svih komponenata aplikacije te definiranje njihovih destinacija unutar projekta. Sljedeće,

potrebno je definirati rute, a to možemo u nekoliko koraka. Prvo se kreira konstanta *routes* kojoj se dodjeljuje polje s objektima. Svaki objekt ima zasebno definiran *path* kojim je označeno koji put prikazuje tu stranicu, zatim ime stranice, ime komponente koje se importaju i na određenim je komponentama *props* postavljen na *true* tako da se mogu prosljeđivati vrijednosti između *parent* i *child* komponenti. Preostaje još kreirati *router*, za što se može koristiti funkcija *createRouter* unutar koje se definira *history* pomoću funkcija *createWebHistory* i *routes*. Na kraju je potrebno izvršiti *export default router* kako bi on postao globalno dostupan za korištenje.

4.7. Početna stranica

Home.vue M

src > views > Home.vue > {} "Home.vue" > template

```
<template>
  <section id="pocetna" class="container-fluid">
    <h1>Svijet Apartmana</h1>
    <p class="subheading">Pretražite najnovije oglase iz Svijeta Apartmana</p>
    <p>Osijećajte se kao doma i rezervirajte kuću, apartman ili sobu za nezaboravan odmor unutar Hrvatske!</p>
    <button @click="$router.push('apartmani')" class="btn btn-color grow mt-4">Prikaz Apartmana</button>
  </section>
</template>
```

Slika 20 Kod template dijela Home.vue komponente

```
<script>
export default {
  name: 'Home'
}
</script>
```

Slika 21 Kod script dijela Home.vue komponente

U Home.vue komponenti (prikazana na slikama 20 i 21) je zapisan kod pomoću kojeg je određeno prvo što se vidi kada se aplikacija pokrene. Ona se sastoji od *sectiona* s *id-om* „pocetna“ koji služi za oblikovanje elemenata na stranici. Sadrži naslov koji je ujedno i naziv same aplikacije, podnaslov te element gumb koji nas vodi na stranicu s apartmanima. Odlazak do stranice apartmani je realiziran tako da je gumbu dodijeljen atribut *@click*. Unutar atributa se nalazi poziv do routera koji *„pusha“* ime te komponente.

4.8. Kreiranje objave

Slika 22 Kod template dijela NoviOglas.vue komponente

Unutar mape *views* kreirana je datoteka pod nazivom *NoviOglas.vue*. Mapa *views* služi za kreiranje velikih dijelova stranice - u ovom slučaju fokusiramo se na cijelu stranicu *Kreiranje Objave*. Kreiran je *section* (linija 2 na slici 22) koji ima *id* „oglas“ kako bi se mogli oblikovati elementi unutar te sekcije i klasu *container-fluid* zaslužnog za responzivni dizajn aplikacije. Sekcija sadrži naslov “Kreiranje Oglasa” s klasom *text-center* za centriranje na sredinu stranice. Ispod naslova kreirana je još jedna sekcija na stranici koja će prikazivati formu koju korisnik popunjava ukoliko želi kreirati objavu. Iz te se sekcije poziva komponenta *apartmani-forma* koja ima dodatan atribut *@save-data=”saveData”*. On je zaslužan za pohranjivanje ispunjene forme. *saveData* sadrži podatke koje forma pošalje, a to se sve prihvaća u metodi *save-data* koja je definirana u script dijelu stranice.

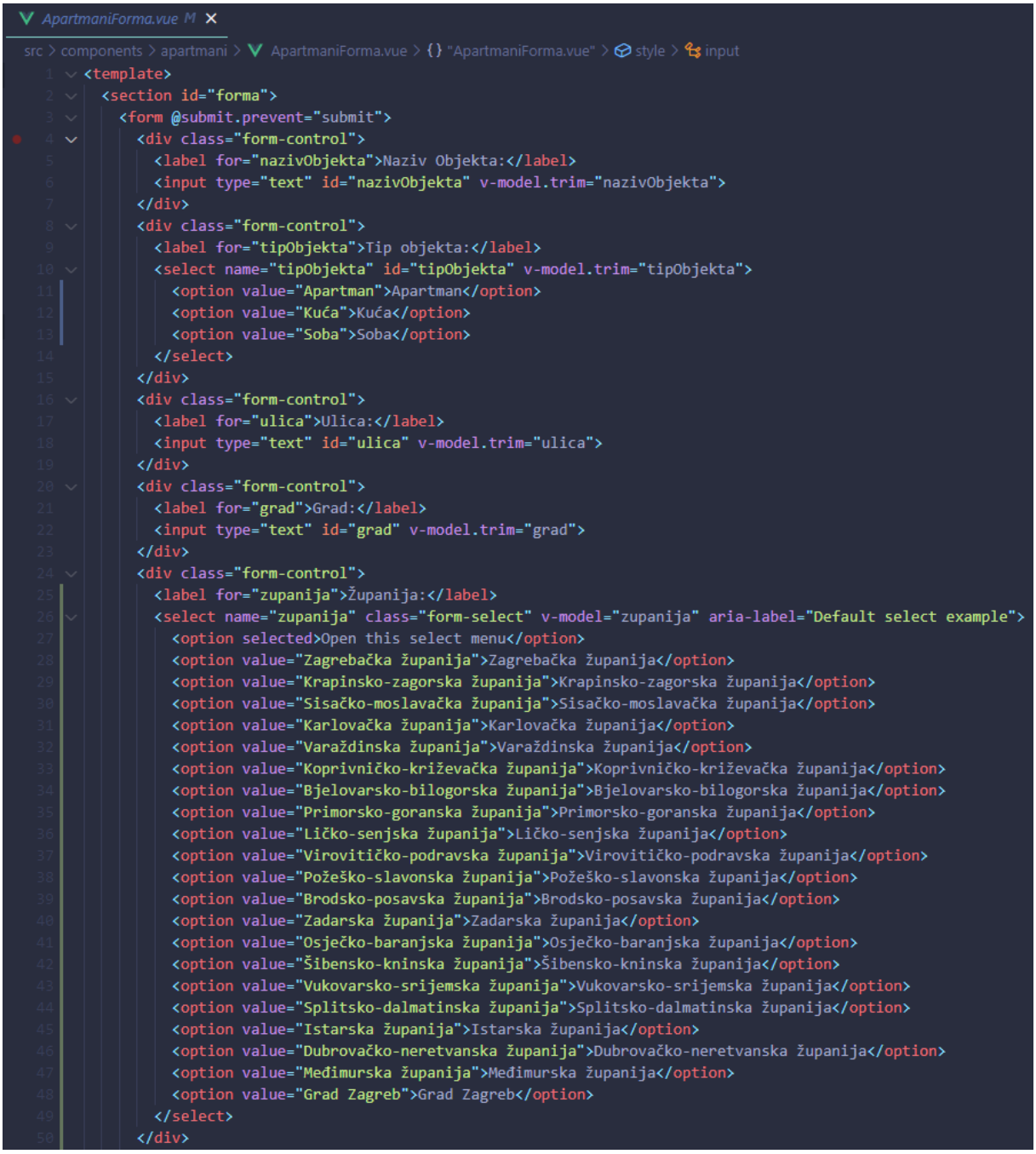
Slika 23 Kod script dijela NoviOglas.vue komponente

Script dio (slika 23) se sastoji od importanja komponente *ApartmaniForma* koje su pozvane u prethodno opisanoj sekciji, te metode *saveData* i *saveImages*.

SaveData prihvaća podatke koji su pohranjeni u objekt *data*. Funkcija sadrži poziv za slanje tih podataka u *store* pomoću naredbe *this.$store.dispatch()*. Kada se izvrši slanje podataka u *store*, zamjenjujemo trenutnu rutu *apartmani/noviApartman* sa */apartmani*. Korisnik ovu zamjenu vidi kao odlazak na stranicu gdje su prikazani *apartmani* (prikazano u liniji broj 20 na slici 23).
saveImages funkcija radi isto kao *saveData* funkcija samo umjesto tekstualnih podataka šalje fotografiju apartmana u *store*. Na taj se način forografija pohranjuje unutar *Storagea*, a zatim i u sami *Firestore*.

U Style dijelu kôda može se izdvojiti korištenje *scoped* načina oblikovanja stranica. Ovi stilovi će se odnositi samo na zadanu komponentu koja je prikazana gore prikazanim kôdom. Isto tako na dnu stranice koristi se *@media* kako bi se stil prilagodio stilu mobilnog prikaza.

4.9. Forma Apartmani



Slika 24 Kod template dijela ApartmaniForma.vue komponente 1

```
      </div>
      <div class="form-control">
        <label for="postanskiBroj">Postanski broj:</label>
        <input type="number" id="postanskiBroj" v-model.number="postanskiBroj">
      </div>
      <div class="form-control">
        <label for="povrsinaApartmana">Povrsina apartmana u m2:</label>
        <input type="number" id="povrsinaApartmana" v-model.number="povrsinaApartmana">
      </div>
      <div class="form-control">
        <label for="brojOsoba">Maksimalni broj osoba:</label>
        <input type="number" id="brojOsoba" v-model.number="brojOsoba">
      </div>
      <div class="form-control">
        <label for="brojLezaja">Broj lezaja:</label>
        <input type="text" id="brojLezaja" v-model.number="brojLezaja">
      </div>
      <div class="form-control">
        <label for="cijenaNocenja">Cijena noćenja (u eurima):</label>
        <input type="number" id="cijenaNocenja" v-model.number="cijenaNocenja">
      </div>
      <div class="form-control">
        <label for="opis">Opis:</label>
        <textarea row="4" id="opis" v-model.trim="opis"></textarea>
      </div>
      <div class="form-control">
        <label for="link">Ukoliko već imate iznajmljen apartman na nekoj stranici, zaljepite poveznicu ovdje (nije obavezno):</label>
        <input type="text" id="link" v-model.number="poveznica">
      </div>
      <div class="form-control">
        <label for="opis">Slike:</label>
        <input
          type="file"
          @change="onFilePicked"
          ref="fileInput"
          accept="image/*"
        >
          <img :src="imageURL" alt="">
      </div>
      <button class="btn btn-primary">Objavi</button>
    </form>
  </section>
</template>
```

Slika 25 Kod template dijela ApartmaniForma.vue komponente 2

Ulaskom u komponentu *ApartmaniForma.vue* (slika 24 i slika 25) koja je prethodno *impor*t-ana u *NoviOglas.vue,* vidimo kako izgleda forma koju korisnik popunjava. Kreirana je na način da sadrži sekciju s *id forma* koja služi za oblikovanje elemenata forme. Unutra se nalaze svi važni podatci za objavu apartmana. Svaki element forme objedinjen je klasom *form-control* unutar koje je *label* tag. *Label* služi za prikaz naslova polja, dok unutar *input* taga unosimo same podatke. *Input* tag sadrži atribute *type* koji određuje što korisnik unosi, *id* koji povezuje *label* sa *input* tagom i *v-model* atribut koji pohranjuje upisanu vrijednost u varijablu. Varijabla je navedena unutar dvostrukih navodnika. *Trim* funkcija je prisutna kako bi se uklonio višak razmaka ukoliko ga korisnik slučajno ostavi na početku ili kraju unosa.

```vue
<script>
export default {
  emits: ['save-data'],
  data() {
    return {
      nazivObjekta: '',
      tipObjekta: '',
      ulica: '',
      grad: '',
      zupanija: '',
      postanskiBroj: null,
      povrsinaApartmana: null,
      brojOsoba: null,
      brojLezaja: null,
      cijenaNocenja: null,
      opis: '',
      poveznica: '',
      userId: '',
      imageURL: '',
      image: null
    };
  },
  methods: {
    onPickFile() {
      this.$refs.fileInput.click()
    },
    onFilePicked(event) {
      const files = event.target.files
      let filename = files[0].name;
      if(filename.lastIndexOf('.') <= 0) {
        return alert('Please add a valid file!')
      }
      const fileReader = new FileReader()
      fileReader.addEventListener('load', () => {
        this.imageURL = fileReader.result
      })
      fileReader.readAsDataURL(files[0])
      this.image = files[0]
    },
    // Add form validation
    submit() {
      if (!this.image) {
        return;
      }
      const formData = {
        nazivObjekta: this.nazivObjekta,
        tipObjekta: this.tipObjekta,
        ulica: this.ulica,
        grad: this.grad,
        zupanija: this.zupanija,
        postanskiBroj: this.postanskiBroj,
        povrsinaApartmana: this.povrsinaApartmana,
        brojOsoba: this.brojOsoba,
        brojLezaja: this.brojLezaja,
        cijenaNocenja: this.cijenaNocenja,
        opis: this.opis,
        poveznica: this.poveznica,
        userId: this.$store.state.korisnik.user,
        image: this.image
      };

      this.$emit('save-data', formData);
    }
  }
}
</script>
```

Slika 26 Kod script dijela ApartmaniForma.vue komponente

Script dio (slika 26) se sastoji od *emits* opcije koja emitira zadanu funkciju definiranu unutar *NoviOglas.vue* datoteke. Na ovaj način prosljeđuju se funkcije koje su definirane u *parent* datotekama na *child* datoteke. Zatim se definira varijabla unutar koje se spremaju podatci koje korisnik unese. Metoda sadrži tri različite funkcije *onPickFile*, *onFilePicked* i *submit*. *onPickFile* se poziva kada korisnik klikne na input polje slike (izvedeno preko reference). *onFilePicked* koji prihvaća vrijednost *event* služi za dohvaćanje odabrane fotografije sa računala. Prvo moramo kreirat varijablu *files* koja dohvaća vrijednost datoteke, nakon toga dobivamo ime te datoteke tako da odaberemo prvu vrijednost u polju *files* atributom *.name*. Za provjeru jesmo li odabrali valjanu datoteku gledamo sadrži li neki nastavak. Ukoliko ne postoji nastavak izbacuje se *alert* sa komentarom. Nakon završetka provjere nastavljamo na učitavanje dokumenta na stranicu. Kreiramo novi *FileReader*, dodajemo *event listener* koji prilikom učitavanja datoteke postavi *imageUrl* na vrijednost *fileReader.result*. Nakon izvršavanja *eventa* i učitavanja fotografije, preostaje da postavimo fotografiju u varijablu *image* kako bi ju mogli prikazati na stranici.

Submit funkcija prvo provjerava postoji li već fotografija apartmana. Ako uvjet nije zadovoljen zaustavlja se njezino izvođenje, u suprotnom se nastavlja na kreiranje objekta *formData.* Unutar *formData* imamo *key* koji ima svoj naziv (npr. *nazivObjekta*) i *value*. U ovom slučaju se dohvaća vrijednost varijable tako da koristimo *this.nazivObjekta*. Jedino što preostaje je emitirati kreirani objekt u *parent* datoteku *NoviOglas* na način da odaberemo funkciju iz *parent* datoteke *save-data* i proslijedimo novo kreirani objekt.

4.10. Pregled svih apartmana

Apartmani.vue M

src > views > Apartmani.vue > {} "Apartmani.vue"

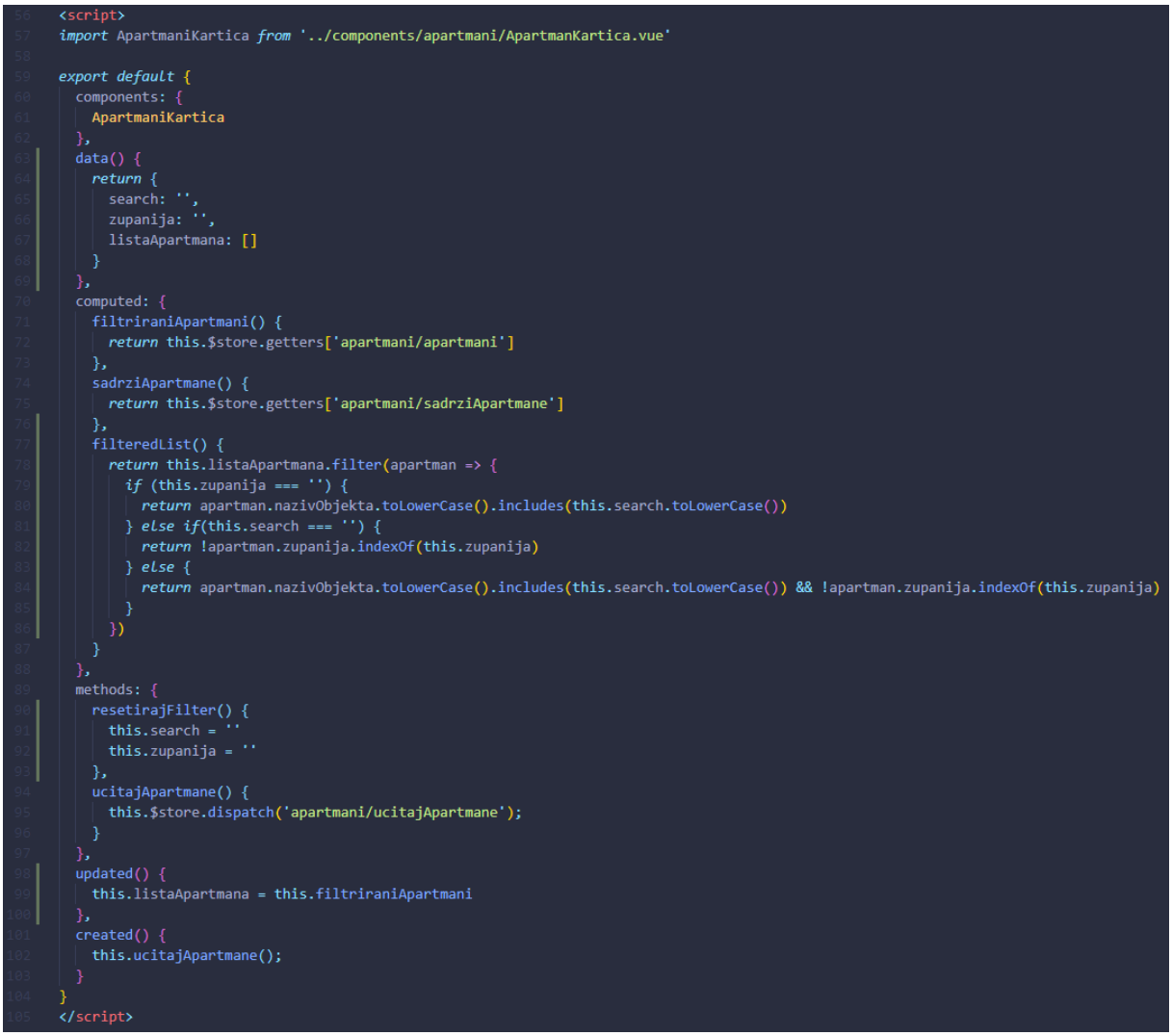
```
<template>
  <section id="pretrazivanje" class="container-fluid">
    <h2 class="text-center">Filtriranje apartmana:</h2>
    <div class="pretrazivanje">
      <div class="text-search">
        <input type="text" v-model="search" placeholder="Naziv apartmana.."/>
      </div>
      <div class="dropdown">
        <select class="form-select" v-model="zupanija" aria-label="Default select example">
          <option selected>Odaberite županiju</option>
          <option value="Zagrebačka županija">Zagrebačka županija</option>
          <option value="Krapinsko-zagorska županija">Krapinsko-zagorska županija</option>
          <option value="Sisačko-moslavačka županija">Sisačko-moslavačka županija</option>
          <option value="Karlovačka županija">Karlovačka županija</option>
          <option value="Varaždinska županija">Varaždinska županija</option>
          <option value="Koprivničko-križevačka županija">Koprivničko-križevačka županija</option>
          <option value="Bjelovarsko-bilogorska županija">Bjelovarsko-bilogorska županija</option>
          <option value="Primorsko-goranska županija">Primorsko-goranska županija</option>
          <option value="Ličko-senjska županija">Ličko-senjska županija</option>
          <option value="Virovitičko-podravska županija">Virovitičko-podravska županija</option>
          <option value="Požeško-slavonska županija">Požeško-slavonska županija</option>
          <option value="Brodsko-posavska županija">Brodsko-posavska županija</option>
          <option value="Zadarska županija">Zadarska županija</option>
          <option value="Osječko-baranjska županija">Osječko-baranjska županija</option>
          <option value="Šibensko-kninska županija">Šibensko-kninska županija</option>
          <option value="Vukovarsko-srijemska županija">Vukovarsko-srijemska županija</option>
          <option value="Splitsko-dalmatinska županija">Splitsko-dalmatinska županija</option>
          <option value="Istarska županija">Istarska županija</option>
          <option value="Dubrovačko-neretvanska županija">Dubrovačko-neretvanska županija</option>
          <option value="Međimurska županija">Međimurska županija</option>
          <option value="Grad Zagreb">Grad Zagreb</option>
        </select>
      </div>
      <button type="button" class="btn btn-secondary" @click="resetirajFilter">Resetiraj</button>
    </div>
  </section>
  <section id="prikaz-apartmana" class="container-fluid">
    <h1>Lista svih apartmana</h1>
    <ul v-if="sadrziApartmane">
      <apartmani-kartica
        v-for="apartman in filteredList"
        :key="apartman.id"
        :id="apartman.id"
        :nazivObjekta="apartman.nazivObjekta"
        :grad="apartman.grad"
        :ulica="apartman.ulica"
        :zupanija="apartman.zupanija"
        :user="apartman.userId.name"
      >
      </apartmani-kartica>
    </ul>
    <h3 v-else>Nema pronađenih apartmana.</h3>
  </section>
</template>
```

Slika 27 Kod template dijela Apartmani.vue komponente

Komponenta *Apartmani.vue* (slika 27) se sastoji od sekcije sa id-om *pretrazivanje* i još jedne pod sekcije unutar koje se prikazuju apartmani. Ulaskom u sekciju definiran je podnaslov "*Filtriranje apartmana*", koji ujedno naglašava početak dijela koji se odnosi na

filter i pretraživanje. *Input* polje je smješteno unutar dvije *div* komponente kako bi se bolje oblikovao element. Unutar *input*-a imamo atribute *type* pomoću kojeg se određje kojeg tipa će biti uneseni podatci, *v-model* koji pohranjuje unesenu vrijednost u varijablu i *placeholder* koji služi za objašnjenje što pojedino polje prihvaća. Pokraj pretraživanja imamo kreiranu *dropdown* komponentu iz *Bootstrapa* koja služi za filtriranje objava putem županije. *Select* tag ima definirane sve županije Republike Hrvatske unutar *option* taga. Kako bismo imali definiranu zadanu vrijednost ukoliko nismo odabrali nijednu opciju, stavljamo atribut *select* na željeni *option* tag. Na kraju imamo gumb koji služi za resetiranje unesenih vrijednosti pretraživanja i filtera. Gumb sadrži atribut *v-on* koji sluša za *click* (skraćeno *@click*) te poziva funkciju *resetirajFilter*.

Podsekcija ima definiran naslov "*Lista svih apartmana*" i *ul* tag s uvjetom ne prikazivanja apartmani ukoliko je *sadrziApartmane* prazan. Ulaskom u *ul* tag pozivamo drugu komponentu *apartmani-kartica* koja prikazuje kartice s odabranim podatcima. Izvodi se *for* petlja kojom se prolazi kroz sve apartmane u *filteredList* i proslijeđuje komponenti *apartmani-kartica* vrijednosti kao što su *id*, *nazivObjekta*, *grad*, *ulica*, *zupanija* i *user*. Ispod *ul* taga se nalazi *h3* s *v-else* atributom koji se pojavljuje ukoliko nema apartmana u polju *sadrziApartmane*. Nakon toga se prikazuje poruka „*Nema pronađenih apartmana*".



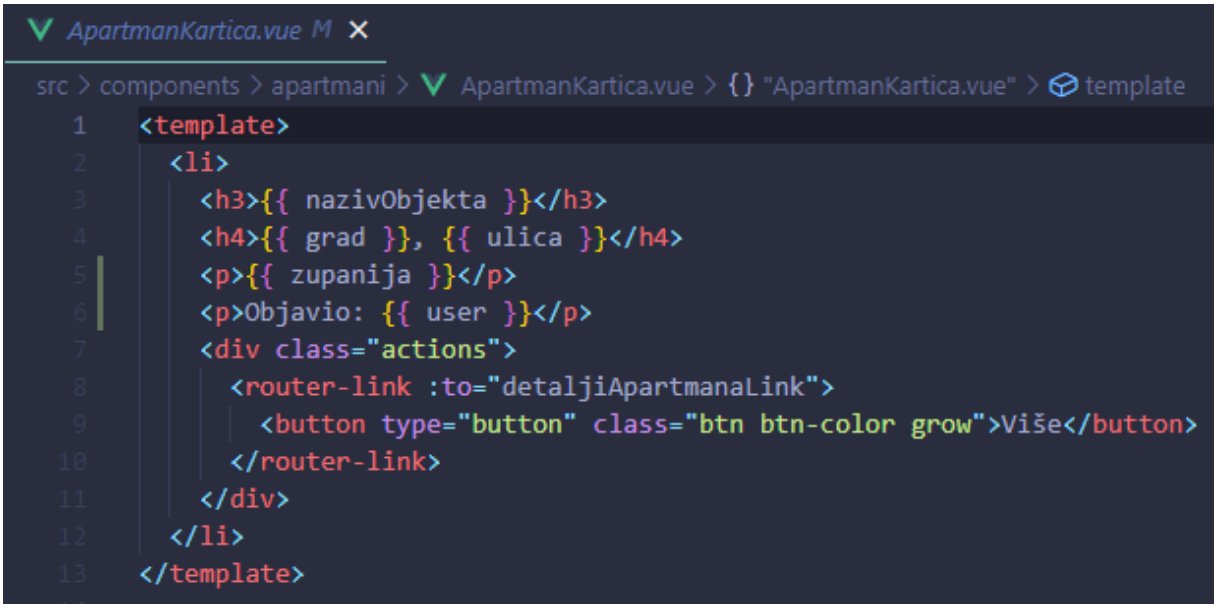
```vue
<script>
import ApartmaniKartica from '../components/apartmani/ApartmanKartica.vue'

export default {
  components: {
    ApartmaniKartica
  },
  data() {
    return {
      search: '',
      zupanija: '',
      listaApartmana: []
    }
  },
  computed: {
    filtriraniApartmani() {
      return this.$store.getters['apartmani/apartmani']
    },
    sadrziApartmane() {
      return this.$store.getters['apartmani/sadrziApartmane']
    },
    filteredList() {
      return this.listaApartmana.filter(apartman => {
        if (this.zupanija === '') {
          return apartman.nazivObjekta.toLowerCase().includes(this.search.toLowerCase())
        } else if(this.search === '') {
          return !apartman.zupanija.indexOf(this.zupanija)
        } else {
          return apartman.nazivObjekta.toLowerCase().includes(this.search.toLowerCase()) && !apartman.zupanija.indexOf(this.zupanija)
        }
      })
    }
  },
  methods: {
    resetirajFilter() {
      this.search = ''
      this.zupanija = ''
    },
    ucitajApartmane() {
      this.$store.dispatch('apartmani/ucitajApartmane');
    }
  },
  updated() {
    this.listaApartmana = this.filtriraniApartmani
  },
  created() {
    this.ucitajApartmane();
  }
}
</script>
```

Slika 28 Kod script dijela Apartmani.vue komponente

Script dio (slika 28) ove komponente se sastoji od *importa* komponente *ApartmaniKartica* koja prikazuje kartice na stranici. Unutar njega definirane su varijable koje se nalaze u *data()* funkciji, *computed* vrijednosti, *metode*, *updated* i *created* funkcije. *Computed* vrijednosti sadrže funkcije od kojih se očekuje da cijelo vrijeme „slušaju“ za pozivanje. *filtriraniApartmani* dohvaćaju iz *storea* putem *getters* funkcije vrijednosti definirane u *apartmani/apartmani* isto tako i *sadrziApartmane* dohvaća vrijednosti u *apartmani-sadrziApartmane*. Kod *filteredList* je stvar malo drugačija - ovdje vraćamo vrijednosti koje se podudaraju sa određenim slučajevima. Na primjer, iz *listaApartmana* filtriramo vrijednosti gdje *apartman* ukoliko je istina da varijabla *zupanija* nema vrijednost, vraća nazive apartmana koji se podudaraju s pretraživanjem. Ako je pretraživanje prazno, vraćamo sve apartmane čiji je *indexOf zupanije* isti kao i odabrani na *dropdown*u. Ili ukoliko koristimo pretraživanje po nazivu i filteru, onda vraćamo apartmane koji se podudaraju po jednoj i po drugoj vrijednosti.

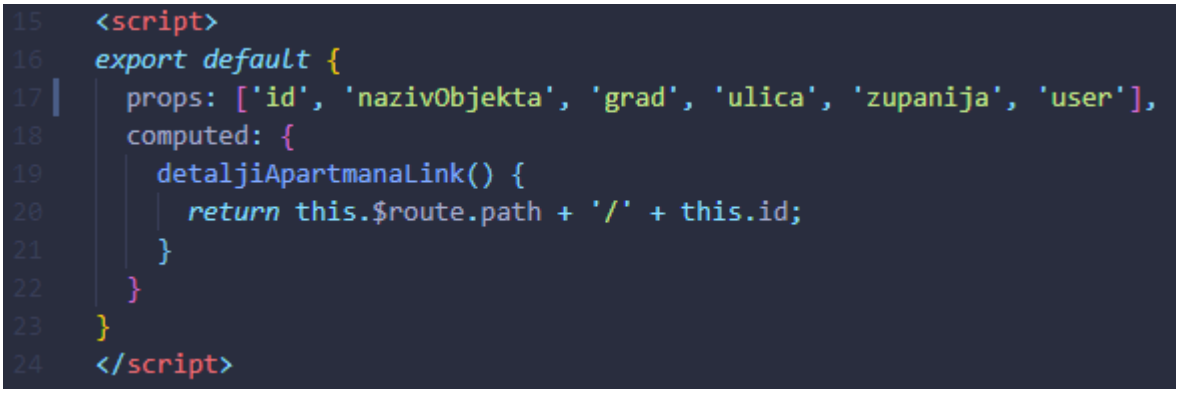
Unutar komponente *metoda* definirana funkcija *resetirajFilter* postavlja vrijednosti varijable *search* i *zupanija* na praznu vrijednost. *ucitajApartmane* poziva unutar *stora* učitavanje određene funkcije koja se nalazi unutar komponente *apartmani/ucitajApartman* gdje je to ujedno i naziv funkcije.

Updated je funkcija koja se izvodi ukoliko se bilo koja vrijednost unutar nje promijeni. Promjene se „slušaju“ tako da postavimo *listaApartmana* na novu vrijednost *filtriraniApartmani*. Ukoliko koristimo filter, *filtriraniApartmani* će se mijenjati. Navedene promjene želimo prikazati na stranici tako da uklonimo apartmane koji ne odgovaraju filteru ili pretraživanju.
Created funkcija se izvodi prilikom samog stvaranja komponente kada učitamo stranicu. Ovdje učitavamo sve apartmane, koji su korisnici objavili, pozivanjem funkcije *this.ucitajApartmane().*



Slika 29 Kod template dijela ApartmanKartica.vue komponente

ApartmaniKartica.vue (slika 29) je komponenta koja je zadužena za prikaz apartmana unutar komponente *Apartmani.vue*. Unutar ove komponente definiramo koji se podatci prikazuju u kartici i na koji način. Započinjemo sa *li* tagom iz razloga što smo u parent elementu smjestili ovu komponentu unutar *ul* taga. Unutar *li* taga imamo *h3* tag koji predstavlja *nazivObjekta.* Ispod njega prikazujemo *grad*, *ulicu*, *županiju* te *usera* koji je kreirao objavu. Nakon toga nas *router-link* odvodi do detaljnijeg prikaza apartmana. *Router-link* sadrži atribut *to* koji poziva funkciju *detaljiApartmanaLink*, a unutar njega imamo gumb za daljnje preusmjeravanje.



Slika 30 Kod script dijela ApartmanKartica.vue komponente

Script dio (slika 30) je poprilično jednostavan za ovu manju komponentu. Sastoji se od *props* opcije koja nasljeđuje varijable od roditeljske datoteke. Ovim načinom možemo prikazati podatke o apartmanu i *computed* vrijednosti. *Computed* sadrži funkciju *detaljiApartmanaLink* unutar koje vraćamo rutu na koju će korisnik bit odveden nakon klika na gumb. Njega generiramo dobivanjem trenutnog *patha* i spajanjem sa id-om od apartmana. Navedeno je vidljivo u liniji 20, slika 30.

4.11. Apartmani Detaljno

Slika 31 Kod template dijela ApartmaniDetaljno.vue komponente

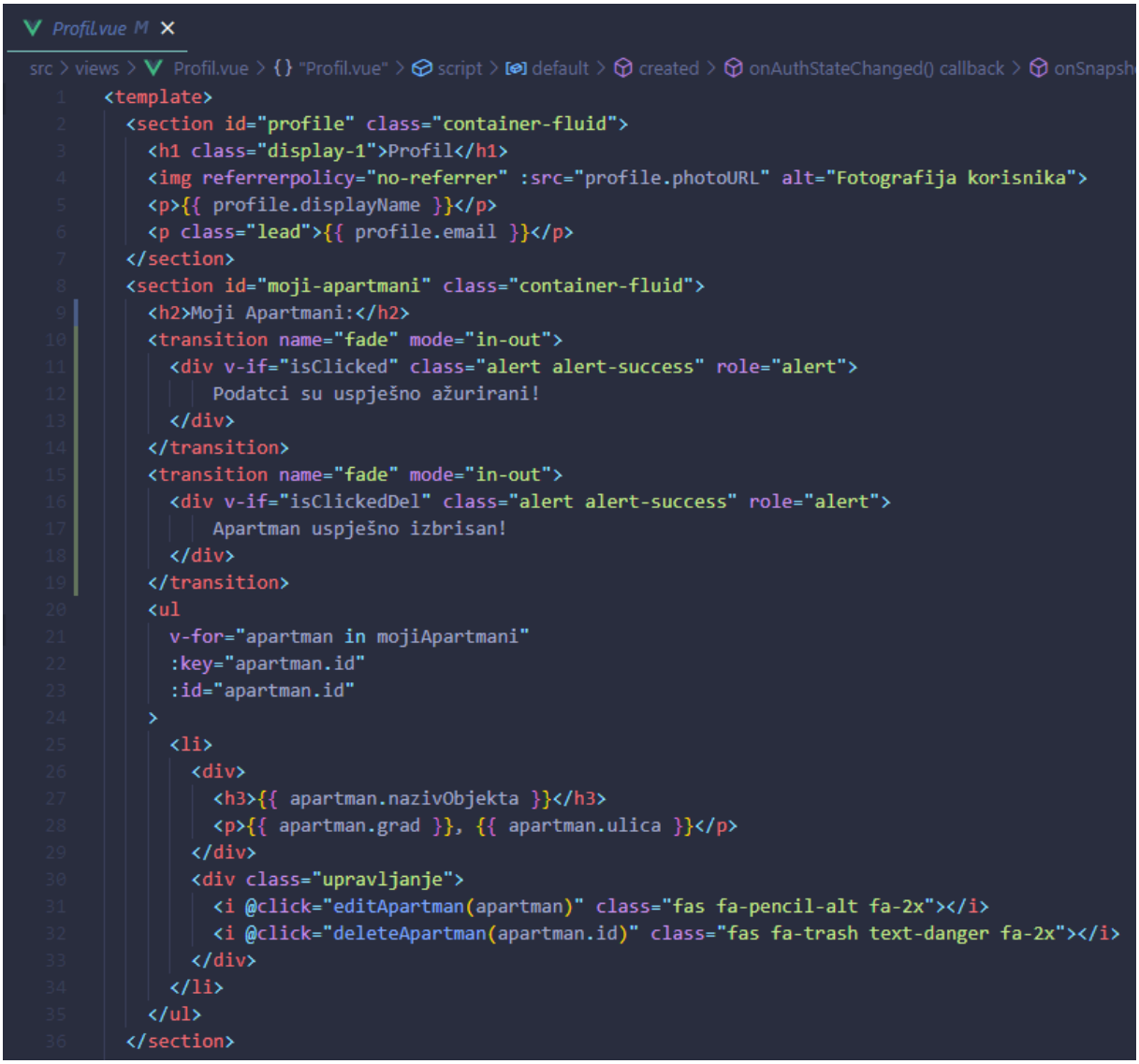
ApartmaniDetaljno.vue je komponenta koja se otvara prilikom klika na gumb „*Više*“ u kartici *apartmani*. Komponenta se sastoji od *section* dijela koji sadrži *id apartmani-detaljno* za oblikovanje sekcije. Unutar nje se prikazuju svi podatci koje je korisnik unio za apartman. Ovo je izvedeno tako da iz objekta *odabraniApartman* odaberemo *key* i *value* koji želimo prikazat. Druga sekcija je profil korisnika odnosno vlasnika apartmana. Imamo naslov „*Vlasnik apartmana*“ ispod kojeg se prikazuje slika vlasnika, ime, e-mail i link ukoliko korisnik ima objavljen apartman na nekom drugom oglasniku tipa *Njuškalo*, *Indeks oglasi* i sl. Na kraju stranice nalazi se gumb za povratak na prethodnu stranicu. Gumb sadrži *@click event* koji vraća na prethodnu rutu.

```vue
<script>
export default {
  props: ['id'],
  data() {
    return {
      odabraniApartman: {}
    }
  },
  methods: {
    ucitajApartman() {
      this.odabraniApartman = this.$store.getters['apartmani/apartmani'].find((apartman) => apartman.id === this.id);
    }
  },
  created() {
    this.ucitajApartman();
  },
}
</script>
```

Slika 32 Kod script dijela ApartmaniDetaljno.vue komponente

Script dio (slika 32) sadrži *props* unutar kojeg se nalazi *id* zbog lakšeg pronalaska zadanog apartmana. Metoda *ucitajApartman* sprema u objekt *odabraniApartman* vrijednost koju dohvaćamo iz *stora* putem *gettersa*. Detaljnije, unutar *apartmana* tražimo apartman koji ima isti *id* kao i ovaj koji dohvaćamo putem *propsa*. *Created* „učitava“ apartmane prilikom kreiranja stranice.

4.12. Korisnički profil



Slika 33 Kod template dijela Profil.vue komponente

Profil.vue (slika 33) je komponenta do koje se može doći ukoliko je korisnik ulogiran u aplikaciju. *Template* dio sadrži *section* s *id profile* unutar kojega prikazujemo korisnikove podatke. Prvo prikazujemo naslov sekcije „*Profil*“, zatim fotografiju korisnika, ispod toga njegovo ime i e-mail. Sljedeća sekcija se odnosi na prikaz apartmana koje je korisnik objavio. Krećemo s naslovom sekcije „*Moji apartmani*“, ispod se nalaze dva *transition* taga koja se prikazuju ovisno o *v-if* uvjetu. Uvjeti se ispunjavaju pozivom određene funkcije. Ispod tagova nalazi se *ul* tag koji sadrži *v-for* atribut. *V-for* prolazi kroz polje svih apartmana i prikazuje ih u listi kako smo definirali unutar samog *ul* taga. Korišten je identičan prikaz kao i u kartici *apartmani*. Jedina razlika između kartice *apartmani* i ovoga je dodatak dvije ikonice koje označavaju uređivanje podataka i brisanje objave (vidljivo u 31. i 32. liniji koda na slici 33). U narednim sekcijama će detaljnije biti objašnjene ove funkcije.

Created funkcija (slika 38) sadrži poziv u bazu podataka gdje provjeravamo je li korisnik prijavljen. Ukoliko je, pohranjujemo njegovu vrijednost u *profil* varijablu. Unutar varijable *mojiApartmani* nalazi se poziv u *store* za dohvaćanje svih apartmana koje trenutno

prijavljen korisnik posjeduje. *Else* dio uvjeta se izvodi ako korisnik nije prijavljen. Postavljamo sve vrijednosti na početak, odnosno na prazne vrijednosti i usmjeravamo korisnika na rutu */apartmani*.

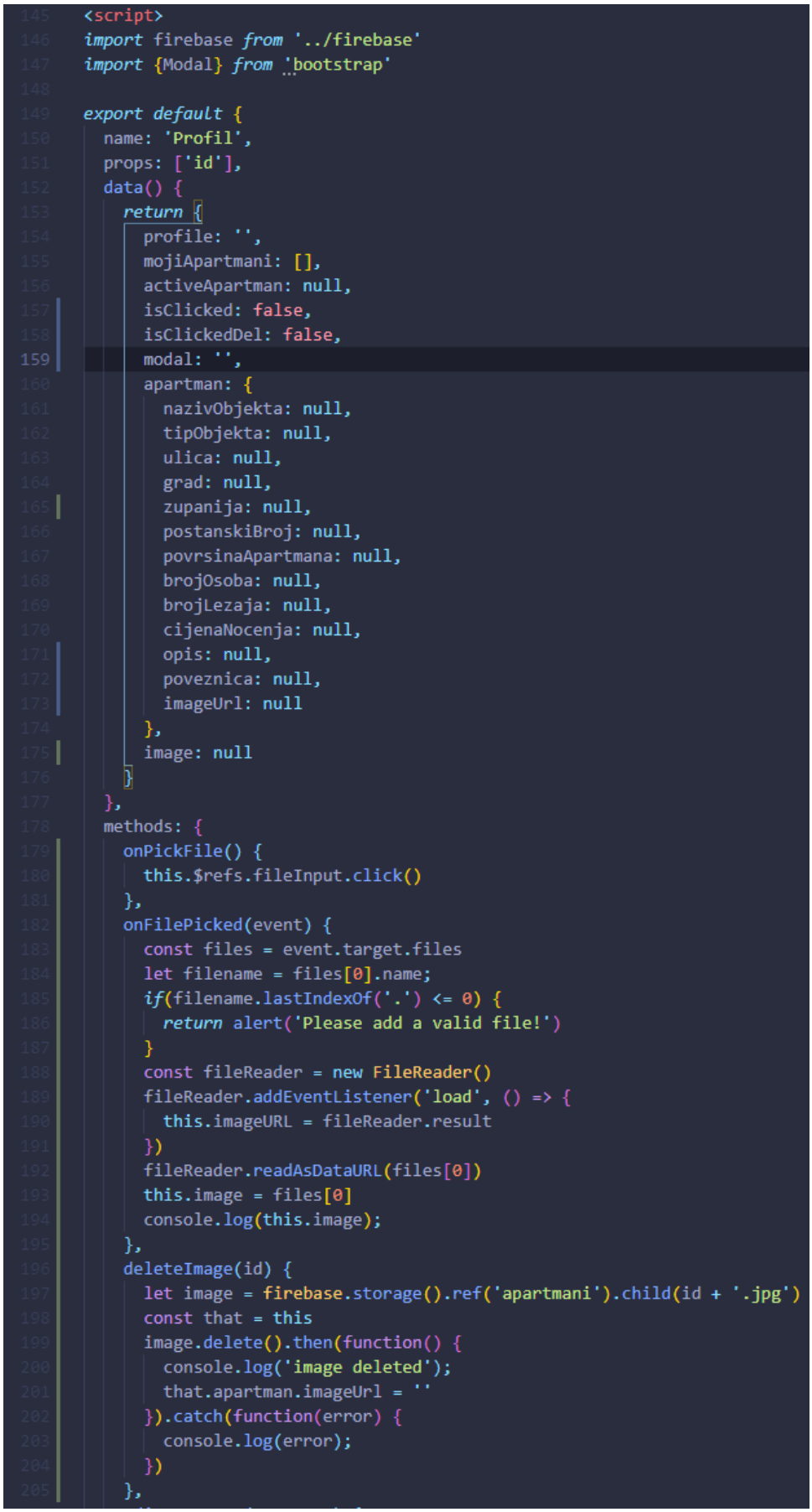
4.12.1. Uredi apartman funkcija

```
<div class="modal fade" id="edit" tabindex="-1" aria-labelledby="editLabel" aria-hidden="true">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="editLabel">Uređivanje podataka:</h5>
        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
      </div>
      <div class="modal-body">
        <div class="form-control">
          <label for="nazivObjekta">Naziv Objekta:</label>
          <input type="text" id="nazivObjekta" v-model.trim="apartman.nazivObjekta">
        </div>
        <div class="form-control">
          <label for="tipObjekta">Tip objekta:</label>
          <select name="tipObjekta" id="tipObjekta" v-model.trim="apartman.tipObjekta">
            <option value="apartman">Apartman</option>
            <option value="kuca">Kuća</option>
            <option value="soba">Soba</option>
          </select>
        </div>
        <div class="form-control">
          <label for="ulica">Ulica:</label>
          <input type="text" id="ulica" v-model.trim="apartman.ulica">
        </div>
        <div class="form-control">
          <label for="grad">Grad:</label>
          <input type="text" id="grad" v-model.trim="apartman.grad">
        </div>
        <div class="form-control">
          <label for="zupanija">Županija:</label>
          <select name="zupanija" id="zupanija" class="form-select" v-model="apartman.zupanija" aria-label="Default select example">
            <option selected>Open this select menu</option>
            <option value="Zagrebačka županija">Zagrebačka županija</option>
            <option value="Krapinsko-zagorska županija">Krapinsko-zagorska županija</option>
            <option value="Sisačko-moslavačka županija">Sisačko-moslavačka županija</option>
            <option value="Karlovačka županija">Karlovačka županija</option>
            <option value="Varaždinska županija">Varaždinska županija</option>
            <option value="Koprivničko-križevačka županija">Koprivničko-križevačka županija</option>
            <option value="Bjelovarsko-bilogorska županija">Bjelovarsko-bilogorska županija</option>
            <option value="Primorsko-goranska županija">Primorsko-goranska županija</option>
            <option value="Ličko-senjska županija">Ličko-senjska županija</option>
            <option value="Virovitičko-podravska županija">Virovitičko-podravska županija</option>
            <option value="Požeško-slavonska županija">Požeško-slavonska županija</option>
            <option value="Brodsko-posavska županija">Brodsko-posavska županija</option>
            <option value="Zadarska županija">Zadarska županija</option>
            <option value="Osječko-baranjska županija">Osječko-baranjska županija</option>
            <option value="Šibensko-kninska županija">Šibensko-kninska županija</option>
            <option value="Vukovarsko-srijemska županija">Vukovarsko-srijemska županija</option>
            <option value="Splitsko-dalmatinska županija">Splitsko-dalmatinska županija</option>
            <option value="Istarska županija">Istarska županija</option>
            <option value="Dubrovačko-neretvanska županija">Dubrovačko-neretvanska županija</option>
            <option value="Međimurska županija">Međimurska županija</option>
            <option value="Grad Zagreb">Grad Zagreb</option>
          </select>
        </div>
        <div class="form-control">
          <label for="postanskiBroj">Postanski broj:</label>
          <input type="number" id="postanskiBroj" v-model.number="apartman.postanskiBroj">
        </div>
        <div class="form-control">
          <label for="povrsinaApartmana">Povrsina apartmana u m2:</label>
          <input type="number" id="povrsinaApartmana" v-model.number="apartman.povrsinaApartmana">
        </div>
        <div class="form-control">
          <label for="brojOsoba">Maksimalni broj osoba:</label>
          <input type="number" id="brojOsoba" v-model.number="apartman.brojOsoba">
```

Slika 34 Kod template dijela vezanog za uređivanje apartmana Profil.vue komponente

Slika 35 Kod template dijela vezanog za uređivanje apartmana Profil.vue komponente 2

Unutar *template* dijela kôda (slike 34 i 35) korištena je identična forma kao kod kreiranja apartmana uz par malih izmjena. Ključna izmjena je prikaz vrijednosti apartmana unutar *input* polja, a ne kreiranje gdje na mjestu *v-model* imamo samo praznu varijablu. Ovdje koristimo *apartman* kao objekt, a zatim odabiremo vrijednost koju prikazujemo u pojedinom polju.



```
<script>
import firebase from '../firebase'
import {Modal} from 'bootstrap'

export default {
  name: 'Profil',
  props: ['id'],
  data() {
    return {
      profile: '',
      mojiApartmani: [],
      activeApartman: null,
      isClicked: false,
      isClickedDel: false,
      modal: '',
      apartman: {
        nazivObjekta: null,
        tipObjekta: null,
        ulica: null,
        grad: null,
        zupanija: null,
        postanskiBroj: null,
        povrsinaApartmana: null,
        brojOsoba: null,
        brojLezaja: null,
        cijenaNocenja: null,
        opis: null,
        poveznica: null,
        imageUrl: null
      },
      image: null
    }
  },
  methods: {
    onPickFile() {
      this.$refs.fileInput.click()
    },
    onFilePicked(event) {
      const files = event.target.files
      let filename = files[0].name;
      if(filename.lastIndexOf('.') <= 0) {
        return alert('Please add a valid file!')
      }
      const fileReader = new FileReader()
      fileReader.addEventListener('load', () => {
        this.imageURL = fileReader.result
      })
      fileReader.readAsDataURL(files[0])
      this.image = files[0]
      console.log(this.image);
    },
    deleteImage(id) {
      let image = firebase.storage().ref('apartmani').child(id + '.jpg')
      const that = this
      image.delete().then(function() {
        console.log('image deleted');
        that.apartman.imageUrl = ''
      }).catch(function(error) {
        console.log(error);
      })
    },
```

Slika 36 Kod script dijela Profil.vue komponente 1

```
    editApartman(apartman) {
      this.modal = new Modal(document.getElementById('edit'))
      this.modal.show()
      this.apartman = apartman
      console.log(apartman.id);
      console.log(this.apartman);
      this.activeApartman = apartman.id
    },
    updateApartman() {
      const state = this
      let imageUrl
      let apartmanRef = this.apartman.id
      if(this.image) {
        firebase.firestore().collection('apartmani').doc(apartmanRef).update({
        }).then(() => {
          const filename = state.image.name
          const ext = filename.slice(filename.lastIndexOf('.'))
          let uploadTask = firebase.storage().ref('apartmani/' + apartmanRef + ext).put(state.image)
          uploadTask.on('state_changed', () => {

          }, () => {
            // Handle unseccessful uploads
          }, () => {
            // Handle successful uploads on complete
            uploadTask.snapshot.ref.getDownloadURL().then((downloadURL) => {
              imageUrl = downloadURL
              firebase.firestore().collection('apartmani').doc(apartmanRef).update({imageUrl: imageUrl})
            })
          })
        })
      }

      firebase.firestore().collection('apartmani').doc(this.activeApartman).update(this.apartman)
        .then(function() {
          state.isClicked = true;
          setTimeout(() => {
            state.isClicked = false
          }, 2000)
          state.modal.hide()
        });
    },
```

Slika 37 Kod script dijela Profil.vue komponente 2

```
  created() {
    firebase.auth().onAuthStateChanged((user) => {
      if (user) {
        this.profile = user
        this.mojiApartmani = this.$store.getters['apartmani/apartmani'].find((apartman) => apartman.userId === user.uid);
        firebase.firestore().collection("apartmani").where("userId.id", "==", user.uid)
          .onSnapshot((querySnapshot) => {
            this.mojiApartmani = [];
              querySnapshot.forEach((doc) => {
                  const apartman = {
                      id: doc.id,
                    ...doc.data()
                  }
                  this.mojiApartmani.push(apartman)
              });
          })
      } else {
        this.profile = '',
        this.mojiApartmani = [],
        this.$router.replace('/apartmani');
        this.isLoggedIn = false
      }
    });
  }
}
</script>
```

Slika 38 Kod script dijela Profil.vue komponente 3

Script dio (slike 36, 37 i 38) je dosta kompleksan za ovu komponentu. Varijabla *apartman* na početku dok nije odabran specifičan apartman postavljena je na *default* vrijednost (vidi kôd linija 160 na slici 36). Nakon odabira apartmana, poziva se funkcija *editApartman* koja postavlja vrijednosti u objekt sukladno *id*-u unutar baze podataka.

Kada se pozove funkcija *editApartman*, prihvaća jedan parametar *apartman*. Funkcija sadrži varijablu *modal* koju definiramo u *data* funkciji. Nakon kreacije *modala* pozivamo ga te prikazujemo putem funkcije *show()*. Sljedeće, moraju se prikazati podatci o određenom apartmanu koje želimo uređivati. U tu svrhu poziva se varijablu *apartman* i pridružuje joj se parametar *apartman*. Kako bi se prikazalo koji je apartman aktivan, pozivamo varijablu *activeApartman* i pridružujemo joj vrijednost iz parametra *apartman.id*.

updateApartman funkcija se poziva nakon uređenja željenih podataka o apartmanu odnosno onda kada želimo spremiti promjene. Prvo kreiramo konstantu *state* kako bismo mogli dohvaćati varijable putem *this*. Definiramo *imageUrl* varijablu unutar koje ćemo kasnije pohraniti *url* slike. Zatim definiramo *apartmanRef* kojem pridružujemo *id* apartmana. Dolazimo do prvog većeg bloka u ovoj funkciji, a to je *if* uvjet koji provjerava postoji li neka spremljena slika u varijabli *image.* Ako je uvjet istinit nastavljamo s izvođenjem, a ako nije preskačemo uvjet i nastavlja se daljnji kôd. Unutar uvjeta nalazi se poziv prema *firestore* kolekciji *apartmani*. Dohvaćamo specifičan dokument pomoću prethodno definirane varijable *apartmanRef* i ažuriramo njenu vrijednost. Nakon ažuriranja nadovezujemo se s *then* funkcijom unutar koje definiramo ime *filea* kojeg ćemo *uploadati*. Potom definiramo gdje će se datoteka spremiti unutar *firebase storagea.* U nastavku je *handle* koji se izvršava ukoliko je sve bilo uspješno (linija 230 na slici 37). Dohvaćamo *downloadURL* iz *storage*a kako bismo sliku mogli pohraniti unutar baze podataka (linija 232 na slici 37). U nastavku kôd pozivamo *firestore*, točnije kolekciju *apartmani* i specifični dokument od aktivnog apartmana. Ovaj poziv služi za ažuriranje svih podataka koje smo izmijenili. Unutar *update* funkcije prosljeđujemo te podatke. Na prethodnu funkciju nadovezuje se *then* funkcija unutar koje postavljamo *isClicked* varijablu na *true* i postavljamo *setTimeout* kako bismo vratili *isClicked* varijablu na *false* nakon 2 sekunde. *isClicked* služi za prikaz obavijesti o uspješno ažuriranim podatcima (linija 12 na slici 33).

4.12.2. Obriši apartman funkcija

```
247     deleteApartman(apartmanId) {
248       const state = this
249       firebase.firestore().collection("apartmani").doc(apartmanId).delete()
250         .then(() => {
251           state.isClickedDel = true;
252           setTimeout(() => {
253             state.isClickedDel = false
254           }, 2000)
255         })
256       .catch((error) => {
257           console.error("Error removing document: ", error);
258       });
259     }
260   },
```

Slika 39 Kod script dijela Profil.vue komponente vezanog za brisanje apartmana

Funkcija *deleteApartman* (slika 39) se poziva klikom na *i* tag (32. linija na slici 33), odnosno klikom na ikonicu kante za smeće. Funkcija prihvaća jedan argument *apartmanId* kako bi se točno znalo o kojem se apartmanu radi. Nakon toga konstanti *state* pridružuje se *this* vrijednost kako bi se moglo pristupiti prethodno definiranim varijablama unutar dane funkcije. Koristimo *firebase firestore* poziv prema kolekciji *apartmani* gdje dohvaćamo specifičan dokument na način da u *.doc* prosljeđujemo *apartmanId* i kasnije koristimo *delete* funkciju. Nadovezujemo se na poziv s *.then* gdje mijenjamo vrijednost varijable *isClickedDel* na *true*, a koja će prikazati obavijest (linija 17 na slici 33). Nakon toga postavljamo *setTimeout* funkciju na 2000ms, kako bismo maknuli obavijest nakon isteka vremena. Za 2000ms promijenit će se vrijednost varijable *isClickedDel* na *false* te će obavijest nestati. Na kraju imamo *catch* funkciju koja unutar konzole prikazuje greške ukoliko dođe do njih.

4.13. Store

index.js

src > store > index.js > ...

```
import { createStore } from 'vuex'

import apartmaniModule from './apartmani/index.js'
import korisnikModule from './korisnik/index.js'

const store = createStore({
  modules: {
    apartmani: apartmaniModule,
    korisnik: korisnikModule
  },
  state() {
    return {
      apartmanId: '',
      userId: ''
    }
  },
  getters: {
    apartmanId(state) {
      return state.apartmanId;
    },
    userId(state) {
      return state.userId;
    }
  },
});

export default store
```

Slika 40 Kod index.js datoteke unutar store mape

Na početku kreiramo *store* (slika 40) tako da *import*-amo *createStore* funkciju. Nastavljamo s *import*-om komponenti podijeljenih na dva dokumenta - komponenta koja se odnosi isključivo na apartmane te druga za korisnika. Sljedeće, varijabli *store* dodijelimo funkciju *createStore* s kojom ćemo kasnije pristupati iz svih komponenti. Unutar funkcije definirani su *module*, *state* i *getters* za dohvaćanje određenih vrijednosti unutar *Vuex storea*. Unutar modula definiramo proizvoljan naziv za dvije komponente i nakon toga odredimo koju komponentu želimo pridružiti kojem nazivu. Naravno prethodno te komponente moraju bit *importane*. *State* služi za definiranje varijabli unutar kojih mogu, ali i ne moraju bit neke vrijednosti. Na početku su postavljene na prazan *string*. *Getters* funkcija se sastoji od *apartmanId* funkcije koja služi za dohvaćanje vrijednosti apartmana iz *statea*, dok *userId* ima istu funkciju samo za *dohvaćanje korisnika*. Na kraju dokumenta moramo *export*-ati store kako bi bio globalno dostupan za korištenje.

4.13.1. Apartmani

index.js M

src > store > apartmani > index.js > default > state > apartmani

```
import firebase from '../../firebase.js'

export default {
  namespaced: true,
  state() {
    return {
      apartmani: []
    }
  },
  actions: {
    noviApartman(context, data) {
      const apartmanId = firebase.firestore().collection('apartmani').doc();
      const podatciApartmana = {
          userId: data.userId,
          nazivObjekta: data.nazivObjekta,
          tipObjekta: data.tipObjekta,
          ulica: data.ulica,
          grad: data.grad,
          zupanija: data.zupanija,
          postanskiBroj: data.postanskiBroj,
          povrsinaApartmana: data.povrsinaApartmana,
          brojOsoba: data.brojOsoba,
          brojLezaja: data.brojLezaja,
          cijenaNocenja: data.cijenaNocenja,
          opis: data.opis,
          poveznica: data.poveznica
      }

      let imageUrl
      let apartmanRef = apartmanId.id

      apartmanId.set({
        ...podatciApartmana
      }).then(() => {
        const filename = data.image.name
        const ext = filename.slice(filename.lastIndexOf('.'))
        let uploadTask = firebase.storage().ref('apartmani/' + apartmanRef + ext).put(data.image)
        uploadTask.on('state_changed', () => {

        }, () => {
          // Handle unseccessful uploads
        }, () => {
          // Handle successful uploads on complete
          uploadTask.snapshot.ref.getDownloadURL().then((downloadURL) => {
            imageUrl = downloadURL
            firebase.firestore().collection('apartmani').doc(apartmanRef).update({imageUrl: imageUrl})
          })
        })
      }).then(() => {
        context.commit('noviApartman', {
          ...podatciApartmana,
          imageUrl: imageUrl,
          apartmanId: apartmanId
        });
      }).catch((error) => {
        console.log(error);
      })
    },
```

Slika 41 Kod index.js datoteke unutar apartmani mape 1

```
    async ucitajApartmane(context) {
      await firebase.firestore().collection("apartmani").get().then((querySnapshot) => {
        const apartmani = [];
        querySnapshot.forEach(doc => {
            const apartman = {
              id: doc.id,
              ...doc.data()
            };
            apartmani.push(apartman)

          });
          context.commit('postaviApartmane', apartmani);
        });
    }
  },
  mutations: {
    noviApartman(state, payload) {
      state.apartmani.push(payload);
    },
    postaviApartmane(state, payload) {
      state.apartmani = payload;
    }
  },
  getters: {
    apartmani(state) {
      return state.apartmani;
    },
    // Provjeravamo da li postoje apartmani u listi
    sadrziApartmane(state) {
      return state.apartmani && state.apartmani.length > 0;
    }
  }
}
```

Slika 42 Kod index.js datoteke unutar apartmani mape 2

Komponenta *apartmani* (slika 41) *impor*ta se u *store*. Unutar nje koristimo *import* za *firebase* za izradu poziva prema bazi. *Export default* vraća sve vrijednosti koje smo definirali. Unutar *export*-a se nalazi opcija *namespaced true*, što označava da se do ove komponente može doći pozivanjem imena definiranog u modulima unutar glavnog *storea*. Nakon toga dolazimo do *actions* funkcija unutar kojih su definirane funkcije za kreiranje novog apartmana te funkcije za učitavanje kreiranih apartmana.

noviApartman kao argumente prihvaća *context* i *data*. Na početku se definira *apartmanId* na način da pristupimo *firebase firestore* kolekciji *apartmani* i dohvaćamo id od dokumenta koji se kreirao ovim pozivom. Nakon toga kreiramo objekt *podatciApartmana* unutar kojeg biramo naziv za *key* (npr. *userId*), a *value* dobivamo tako da pristupimo objektu *data* i odaberemo *key* koji želimo (npr. *data.userId*). Zatim je definirana varijabla *imageUrl* i *apartmanRef* kojoj samo pridružili *id* apartmana. Nakon što smo sve definirali krećemo s postavljanjem podataka unutar dokumenta na način da pozovemo specifičan *id* apartmana tj. *apartmanId* i *set* funkciju unutar koje s *…podatciApartmana* proširimo vrijednosti objekta. Dovezujemo se s *.then* funkcijom unutar koje postavljamo sliku u *firebase storage*. Prvo definiramo *filename* i nastavak odnosno ekstenziju dokumenta.

Sljedeće izvršavamo *uploadTask* preko kojeg pozivamo *firebase storege* s referencom *'apartmani/'*, što znači da će sve fotografije biti pohranjene unutar mape, pod imenom *apartmanRef* + *ext*. Za kraj, *put* funkciju koja sadrži fotografiju u *data.image*. *uploadTask* pozivamo s *on* funkcijom koja služi za promatranje tijeka ukoliko je prijenos bio uspješan. U tom slučaju izvršit će se kod od 44. linije do 47. linije, odnosno vratit će se url slike iz *firebase storage*-a. Nadovezujemo se s *then* funkcijom unutar koje preko *contexta* šaljemo objekt s objektom *podatciApartmana*, *imageUrl*-om i *apartmanId* u funkciju *noviApartman*. Na kraju imamo *catch* funkciju koja ispisuje pogrešku unutar konzole.

ucitajApartmane kao argument ima *context*, a unutar sebe poziv prema *firebase*-u, točnije prema kolekciji *apartmani*. Poziv vraća sve vrijednosti koje se nalaze unutar kolekcije. Za prihvaćanje definiramo polje *apartmani* kako bismo ih pohranili i kasnije prikazali na stranici. Nakon toga definiramo svaki dokument (linija 62 na slici 42) - unutar kolekcije *apartmani* se kreira objekt *apartman* koji se kasnije „*push-a*“ u polje *apartmani*. Prije izlaska (linija 70 na slici 42) preko *context*-a šaljemo vrijednosti polja *apartmani* u funkciju *postaviApartmane*.

Od značajnijih izmjena, imamo funkciju *noviApartman* koja prihvaća argumente *state* i *payload* odnosno objekt. Unutar nje dohvaćamo varijablu *apartmani* preko *state.apartmani* te s *push* funkcijom u nju pohranjujemo sve vrijednosti iz *payload*-a.
postaviApartmane funkcionira na isti princip samo što je definirano da će vrijednost varijable *apartmani* u *state*-u biti jednaka *payload*-u.

Unutar *getters*-a imamo funkciju *apartmani* koja preko *return* vraća varijablu *apartmani*. *sadrziApartmane* vraća *true* ili *false* vrijednost jer se provjerava je li *state.apartmani* prazna varijabla i u drugom uvjetu je li ona veća od 0.

4.13.2. Korisnik

index.js M

src > store > korisnik > index.js > default > mutations > setUser

```
import firebase from '../../firebase.js';

export default {
  namespaced: true,
  state() {
    return {
      user: {},
      isLoggedIn: false
    }
  },
  actions: {
    login(context) {
      const provider = new firebase.auth.GoogleAuthProvider();
      firebase.auth().signInWithPopup(provider);

      firebase.auth().onAuthStateChanged(user => {
        if (user) {
          if(user.user) {
            user = user.user;
          }
          const setUser = {
            id: user.uid,
            name: user.displayName,
            email: user.email,
            image: user.photoURL,
            created_at: firebase.firestore.FieldValue.serverTimestamp()
          };
          firebase.firestore().collection('users').doc(setUser.id).set(setUser);
          // Korisnik je ulogiran
          context.commit('setUser', setUser)
        } else {
          // Korisnik nije ulogiran
          context.commit('setUser', null)
        }
      })
    },
    logout() {
      firebase.auth().signOut();
    },
  },
  mutations: {
    setUser(state, user) {
      if(user) {
        state.user = user;
        state.isLoggedIn = true;
      } else {
        state.user = {};
        state.isLoggedIn = false;
      }
    }
  },
  getters: {
    korisnik(state) {
      return state.korisnik;
    },
  }
}
```

Slika 43 Kod index.js datoteke unutar korisnik mape

Druga podkomponenta (slika 43) vazana je uz akcije koje su korisniku omogućene. Unutar nje se koristi *import* vezan uz *firebase* kako bi se mogao napraviti *login* korisnika i spremiti određene podatke o korisniku.

Export default dio služi, kao i u prethodno opisanoj podkomponenti, za dohvaćanje svih vrijednosti. Na početku se koristi *namespaced true* kako bi se moglo pristupati vrijednostima putem imena. *State* funkcija sadrži definiran *user* objekt unutar kojeg se pohranjuju svi korisnički podatci, te *isLoggedIn* varijablu koja je prvobitno postavljena na *false* dok korisnik nije ulogiran. Unutar *actions* funkcije nalazi se *login* funkcija koja prihvaća parametar *context*. Prvo varijabli *provider* dodjeljujemo novu instancu Google prijave, a zatim istu dodjeljujemo kao argument funkciji *signInWithPopup*. Slijedi funkcija koja provjerava je li došlo do promjene od prijave korisnika. Ukoliko je *if* uvjet zadovoljen odnosno postoje neki podatci unutar *usera* nastavljamo izvršavanje unutar uvjeta. *setUser* objekt se kreira sa svim potrebnim informacijama koje želimo koristiti. Ispod objekta slijedi poziv prema *firebase firestore*-u za kreiranje kolekcije *users* i dokument čiji je naziv jednak *setUser.id*-u. U dokument pohranjujemo vrijednosti objekta *setUser*. Preko *context*-a funkciji *setUser* šaljemo vrijednosti objekta *setUser*. Izlazimo iz *if* uvjeta i prelazimo u *else* koji se izvršava ako vrijednosti *usera* ne postoje. Isto tako preko *context*-a šalje se vrijednost u funkciju *setUser* i null vrijednost.

Pod *mutations* se nalazi se *setUser* funkcija koja prihvaća *state* i *user* argumente. Prvo se provjerava postoje li neki podatci unutar *user* varijable i ukoliko postoje, vrijednost u *state.user* postavlja se na podatke koji su prisutni u varijabli. Isto tako vrijedi i za *state.isLoggedIn* vrijednosti. Ukoliko za user ne postoje nikakve vrijednosti u uvjetu onda se izvršava *else* dio gdje se *state.user* postavlja na početnu vrijednost (prazan objekt) te *state.isLoggedIn* na *false* pošto korisnik više nije prijavljen.

Getters služi za dohvaćanje vrijednosti iz *state*-a - u ovom slučaju se dohvaća korisnik kako bi se u komponentama moglo vidjeti je li prijavljen. Unutar *korisnik* funkcije koja prihvaća argument *state* vraćamo s return *state.korisnik* vraćamo sve korisničke podatke.

5. Zaključak

U ovom je završnom radu detaljno opisan postupak nastajanja aplikacije *Svijet Apartmana* pomoću koje korisnici mogu pregledavati apartmane, sortirati ih i filtrirati po različitim kriterijama. Također, korisnici mogu unositi nove apartmane u aplikaciju, uređivati i brisati postojeće i pregledati svoje objavljene apartmane.

Prema navedenome, aplikacija je korisna iz perspektive kupca i prodavatelja. Ako se nalazimo u ulozi kupca, vrlo jednostavno možemo vidjeti koji su nam apartmani u ponudi. Kako bi se dodatno olakšalo kupcu, u aplikaciju su implementirani i filtri kako bi se dodatno suzio izbor apartmana ukoliko je izbor preširok.

Sa stajališta prodavatelja, korištenje aplikacije omogućava lakše iznajmljivanje apartman tako što je dostupan velikom broju ljudi. Unutar aplikacije *Svijet Apartmana* zasad se nudi besplatno oglašavanje apartmana, što bi trebalo privući mnoge prodavatelje. Sam postupak registracije i izrade oglasa apartmana je prilično jednostavan i ne zahtjeva posebna znanja.

Za kraj se želim osvrnuti na utiske koje sam imao prilikom izrade same aplikacije. Mogu napomenuti kako se nikada prije nisam susreo s ovakvim načinom izrade aplikacije i da sam jako puno naučio putem. Cijeli proces izrade me potaknuo na daljnje razmišljanje kako bi se aplikacija dodatno mogla popraviti ili poboljšati. Pa bi tako sljedeće što bi se moglo implementirati mogla biti komunikacija između iznajmljivača i zainteresirane osobe u vidu chata. Isto tako, mogla bi se poboljšati i sigurnost aplikacije implementiranjem raznih pravila unutar baze podataka. U svakom slučaju u budućem ću radu nastaviti raditi na unaprijeđenju aplikacije i poboljšanju njezine funkcionalnosti.

6. Literatura

1. I. S., Geek.hr, Što je web aplikacija
   https://geek.hr/pojmovnik/sto-je-web-aplikacija/
2. Andrew Bloomenthal, Investopedia, Electronic Commerce (e-commerce)
   https://www.investopedia.com/terms/e/ecommerce.asp
3. Ecommerce News, Ecommerce in Croatia
   https://ecommercenews.eu/ecommerce-in-europe/ecommerce-in-croatia/
4. Linda Fox, PhocusWire, How travel app usage paints another picture of travel's fate in 2020
   https://www.phocuswire.com/travel-mobile-app-growth-2020
5. Bojan Žarković, Medium, Šta su to „Single Page Applications"?
   https://medium.com/@bojanzarkovic/%C5%A1ta-su-to-single-page-applications-83a2297f884a
6. Firebase, Cloud Firestore
   https://firebase.google.com/docs/firestore
7. Firebase, Get started with Cloud Storage on Web
   https://firebase.google.com/docs/storage/web/start
8. Kevin Cooper, Hackernoon, Understanding Git (part 1) — Explain it Like I'm Five
   https://hackernoon.com/understanding-git-fcffd87c15a3
9. Brien Posey, WhatIs.com, What is a Server?
   https://whatis.techtarget.com/definition/server
10. Victoria Collins, Forbes, The Decline Of The Native App And The Rise Of The Web App
    https://www.forbes.com/sites/victoriacollins/2019/04/05/why-you-dont-need-to-make-an-app-a-guide-for-startups-who-want-to-make-an-app/?sh=2297fc0d6e63
11. Flavio Copes, Flavio Copes, Configuring VS Code for Vue Development
    https://flaviocopes.com/vue-vscode/
12. GitHub, About GitHub Pages
    https://docs.github.com/en/pages/getting-started-with-github-pages/about-github-pages
13. Bootstrap, Introduction
    https://getbootstrap.com/docs/5.1/getting-started/introduction/
14. Aditya Sridhar, freeCodeCamp, A quick introduction to Vue.js
    https://www.freecodecamp.org/news/a-quick-introduction-to-vue-js-72937ee8880d/
15. Bobby, Medium, Deploy Vue App to GitHub Pages
    https://medium.com/swlh/deploy-vue-app-to-github-pages-2ada48d7397e
16. Maximilian Schwarzmüller, Udemy, Vue - The Complete Guide (w/ Router, Vuex, Composition API)
    https://www.udemy.com/course/vuejs-2-the-complete-guide/

7. Popis priloga

Uz rad prilažem web adresu na aplikaciju *Svijet Apartmana*. Aplikaciju možete pronaći odlaskom na link: https://mihaelpavlakovic.github.io/svijet-apartmana-app/

8. Popis slika