VIŠEJEZIČNA APLIKACIJA ZA IZNAJMLJIVANJE APARTMANA

Blavicki, Sandro

Undergraduate thesis / Završni rad

2020

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:200:694256

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-07-14**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

VIŠEJEZIČNA APLIKACIJA ZA IZNAJMLJIVANJE APARTMANA

Završni rad

Sandro Blavicki

Osijek, 2020.

SADRŽAJ

1. UVOD

Hrvatski turizam jedna je od vodećih grana gospodarstva Republike Hrvatske. Sve većim porastom ponude u svijetu iznajmljivanja apartmana, vlasnicima apartmana je u cilju ostvariti prednost nad ostalima. Danas postoje razne stranice koje nude uslugu oglašavanja apartmana, a kako bi povećali šansu da iznajme apartman, vlasnici često objavljuju svoje apartmane na više takvih stranica. Takve stranice kao uslugu većinom nude samo stvaranje oglasa, a brigu o dostupnosti apartmana mora voditi sam vlasnik apartmana.

Rezultat ovoga rada je aplikacija čija je primarna svrha reklamiranje apartmana te njihovo iznajmljivanje. Osim iznajmljivanja, aplikacija također pruža mogućnost korisnicima da stupe u kontakt s vlasnikom, postave pitanja vezana za apartmane te vide povijest rezervacija koje su ostvarili. Vlasnici imaju mogućnost unositi rezervacije koje su ostvarene na drugim stranicama kako bi podaci o dostupnosti apartmana na stranici bili ispravni.

Rad je koncipiran u pet poglavlja kroz koja je opisan proces razvoja web aplikacije za iznajmljivanje apartmana. Nakon kratkog uvoda u rad te opisa zadatka završnog rada u prvom poglavlju, u drugom poglavlju su opisane tehnologije korištene za razvoj rada – Vue.js, Google Firebase, Javascript, HTML, CSS. U trećem poglavlju se opisuje proces izrade web aplikacije, tj. izrada njenog sučelja, stvaranje logike aplikacije i povezivanje sa bazom podataka. U četvrtom poglavlju je prikazano testiranje aplikacije i svih njenih mogućnosti, dok je u zadnjem poglavlju zaključak.

1.1. Zadatak završnog rada

Zadatak završnog rada je modernizirati već postojanu web stranicu kako bi bila responzivna i prilagođena za mobitel. Aplikacija prije svega treba imati mogućnost prevođenja na 6 jezika (hrvatski, češki, njemački, engleski, slovenski i talijanski). Aplikacija će nuditi korisnicima da se prijave, nakon čega mogu postaviti rezervaciju. Pri stvaranju upita za rezervaciju, korisnici moraju kalendaru, grafički odabrati dan dolaska i dan odlaska. Vlasniku treba biti pružen način kako da generira nove apartmane koji će se prikazivati na stranici. Vlasniku također treba pružiti opciju da

sam unese rezervacije koje su ostvarene na drugim web oglasima. Sve rezervacije treba pohraniti u bazu podataka, te pri upitu za rezervaciju vizualno prikazati korisnicima koji su dani već zauzeti.

2. Pregled postojećih rješenja i korištene tehnologije

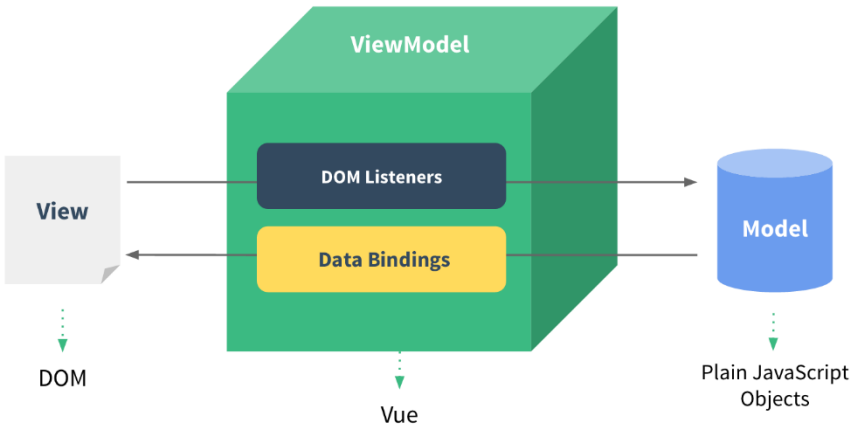
2.1. Pregled postojećih rješenja

Cilj ove aplikacije već je ostvaren na nekim sličnim aplikacijama koje se mogu pronaći na internetu. Web aplikacije poput Booking i Airbnb razvile su rješenje za problem iznajmljivanja nekretnina. [1][2] Njihova rješenja nude usluge iznajmljivanja na globalnoj razini tj. više korisnika može objaviti svoju nekretninu kako bi se iznajmila. U ovoj aplikaciji usluga iznajmljivanja apartmana će biti potrebna za samo jednog vlasnika, a kako se svi apartmani vlasnika nalaze u Novalji, na stranici će se nalaziti i informativni sadržaj s korisnim poveznicama Novalje i okolice.

2.2. Vue.js

Vue.js je softver otvorenog koda (engl. *open-source*) Javascript framework koji se koristi za izradu interaktivnih korisničkih sučelja i aplikacija na jednoj stranici (engl. *single-page applications* – SPA). SPA su web aplikacije koje dinamički mijenjaju trenutnu stranicu pomoću podataka dospjelih sa web servera, za razliku od standardnih metoda koje preglednik preusmjeravaju na potpuno druge stranice. SPA pregledniku predaju sav HTML, CSS i Javascript kod koji je potreban za prikaz web stranice na dva načina. Preglednik može preuzeti sav kod jednim učitavanjem stranice ili dinamički preuzimati i dodavati dijelove koda koji su mu potrebni za prikaz stranice. Osnovna Vue.js biblioteka usredotočena je samo na sloj prikaza, dok su naprednija svojstva poput preusmjeravanja i upravljanja stanjima pružena putem službenih i održavanih biblioteka. [3]

Glavne prednosti korištenja Vue.js frameworka su olakšano *bind*anje podataka (engl. *data-binding*) tj. povezivanje modela podataka s prezentacijskim slojem aplikacije (slika 2.1); te olakšana sinkronizacija modela i podataka.



Slika 2.1*Data-binding unutar Vue.js*

Vue.js koristi posebnu sintaksu unutar HTML koda kako bi omogućio povezivanje i sinkroniziranje podataka sa modelom čime se pruža jednostavnija struktura koda te se pospješuje njegova čitljivost i preglednost.

Još jedan bitan koncept Vue.jsa je sustav komponenti koji omogućava kreiranje aplikacija velikih razmjera. Takve aplikacije se sastoje od puno manjih komponenti koje su lakše za kreiranje te se najčešće mogu ponovno koristiti u drugim dijelovima aplikacije. Između komponenti je omogućen protok podataka (engl. *data-flow*) kao i komunikacija između događaja. [4]

2.3. Google Firebase

Firebase je razvojna platforma za mobilne i web aplikacije koju je razvio Google kako bi omogućio brže i lakše razvijanje aplikacija tj. integriranje raznih funkcionalnosti unutar tih aplikacija. Na početku je bila dostupna samo baza podataka za rad s podatcima na više različitih klijenata, a trenutno je dostupno puno više usluga poput *Cloud Storage*a, *Firebase Machine Learning*a, autentikacije, analitike, *Crash Reporting*a itd. Velika prednost korištenja Firebasea pri izradi aplikacija leži u Googleovoj infrastrukturi koja pruža podršku za sve veličine aplikacija te je lako skalabilna ovisno o potrebama aplikacije. Na ovaj način, programeri se mogu više fokusirati na krajnje korisnike i njihove zahtjeve, budući da Google Firebase nudi razne usluge od razvoja aplikacije do kontroliranja kvalitete. [5]

Pri izradi završnog rada za spremanje podataka je korištena *Firebase Realtime Database*. Ona predstavlja NoSQL bazu podataka koja sprema podatke u JSON formatu i to na *Firebase Cloud*u. Ova baza omogućava brzo i lako povezivanje s aplikacijom s programerske strane, no također omogućava sinkronizaciju podataka u stvarnom vremenu, kao i razmjenu podataka između svih njenih korisnika. Bitna značajka ove baze je to što podatci ostaju sačuvani čak i pri gubitku veze s internetom budući da ih Firebase sprema lokalno na uređaj, a zatim, pri ponovnoj uspostavi internetske veze, šalje i sprema u bazu na *cloud*u. [6]

Također je korišten Firebase Cloud Storage koji predstavlja uslugu za spremanje raznih sadržaja poput slika ili videa. Ovakav način spremanja sadržaja karakterizira robusnost pri prenošenju i preuzimanju sadržaja neovisno o kvaliteti internetske veze te pruža razne sigurnosne postavke s obzirom na parametre sadržaja (naziv, format, veličina itd.) [7]

2.4. JavaScript

JavaScript je skriptni jezik za klijentsku stranu što znači da pruža dinamičku funkcionalnost stranici i omogućava interakciju između korisnika i stranice. Budući da je skriptni jezik, za njegovo izvršavanje je potreban interpreter koji čita kod i prevodi ga u strojni kod pri svakom pokretanju. Interpreter za JavaScript je implementiran unutar svakog web preglednika što omogućava njegovo korištenje bez dodatnih implementacija. [8]

2.5. HTML

HTML (engl. *Hyper Text Markup Language*) predstavlja standardni označni (engl. *mark-up*) jezik za definiranje dizajna i strukture web stranica te se može prikazivati unutar bilo kojeg web preglednika. HTML dokument (*.html* ekstenzija) se sastoji od niza elemenata na temelju kojih se definira izgled web stranice.

HTML tehnologija zapravo povezuje pojedinačne dijelove stranice u jedno, bilo unutar jedne internetske stranice ili između više njih te je to moguće zahvaljujući hipervezama. Hiperveze su

jedan od osnovnih elemenata potrebnih za izgradnju i povezivanje internetskih stranica budući da omogućuju prelazak između različitih HTML datoteka. [9]

2.6. CSS

CSS (engl. *Cascading Style Sheets*) je jezik za uređivanje izgleda web stranice. Koristi se za opisivanje izgleda stranice te se njime određuju boje, fontovi, način kako su raspoređeni elementi na stranici itd. Postoje tri načina povezivanja CSS koda sa HTML dokumentom eksterni (engl. *external*), interni (engl. *internal*) i *inline*.

Eksterni način predstavlja potpuno drugu datoteku unutar koje se nalazi CSS kod te je potrebno unutar HTML datoteke specificirati koju se eksternu CSS datoteku želi koristiti. U slučaju da HTML kod sadrži i CSS kod unutar elemenata, korišten je interni način. Ovaj način je koristan ako na primjer postoji jedna stranica koja se izgledom razlikuje većinom od drugih. *Inline* način korištenja CSS koda se koriste tako da se CSS kod piše direktno unutar samog HTML koda. Ovakav način korištena znači da će napisani CSS kod biti primijenjen samo na liniju unutar koje je napisan.

Generalno najbolji način korištenja CSS koda je eksterni način korištenja, zato što je čitav kod pregledniji jer su CSS i HTML kod odvojeni unutar posebnih datoteka, dok su *inline* i interni načini prikladniji kod slučajeva gdje pojedini elementi moraju imati specifičan stil oblikovanja. [10]

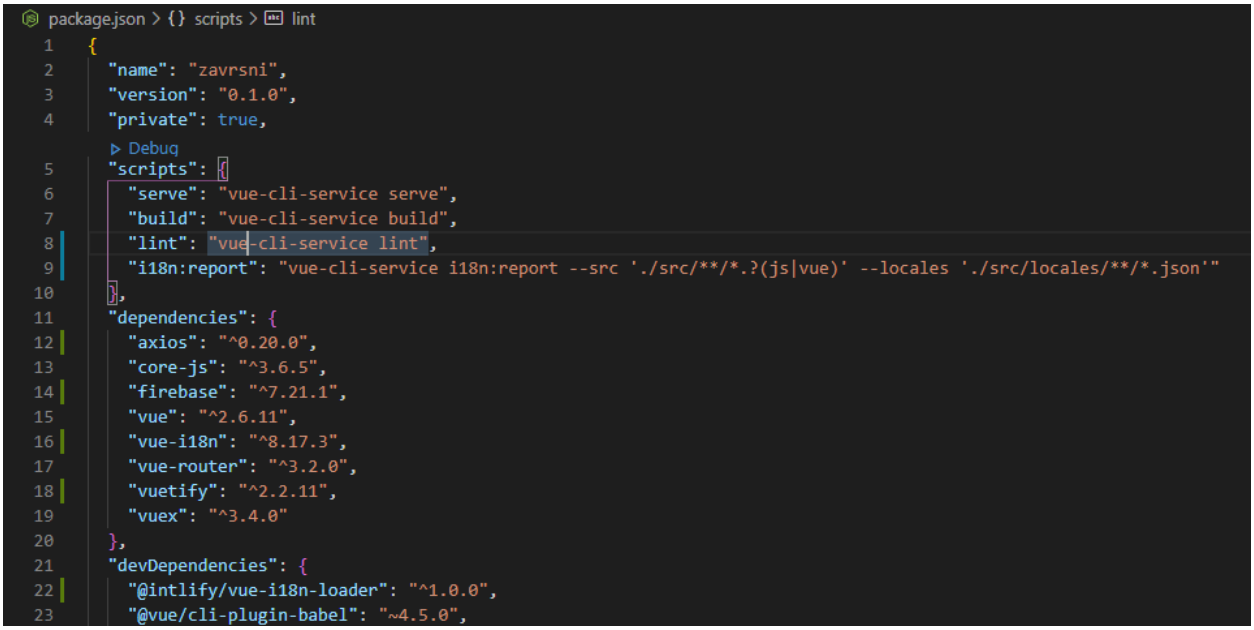
3. IZRADA APLIKACIJE

Unutar ovog poglavlja je opisan proces izrade aplikacije. Prvenstveno je opisan postupak pripreme svih potrebnih softvera kako bi se omogućila funkcionalna razvojna okolina. Zatim je opisana implementacija korisničkog sučelja koji ne ovisi o Vuex alatu, a unutar kojeg je promjenjiv jedino jezik na kojem se podaci prikazuju. Nadalje, navodi se implementacija korisničkog sučelja koji je ovisan o Vuex alatu. Uz jezik, kod ovih dijelova sučelja promjenjivi su i podaci. Podaci su generirani od strane korisnika/admina, dohvaćaju se iz baze ppodataka te ih nije moguće tvrdo kodirati (engl. Hard code)

3.1. Postavljanje razvojne okoline

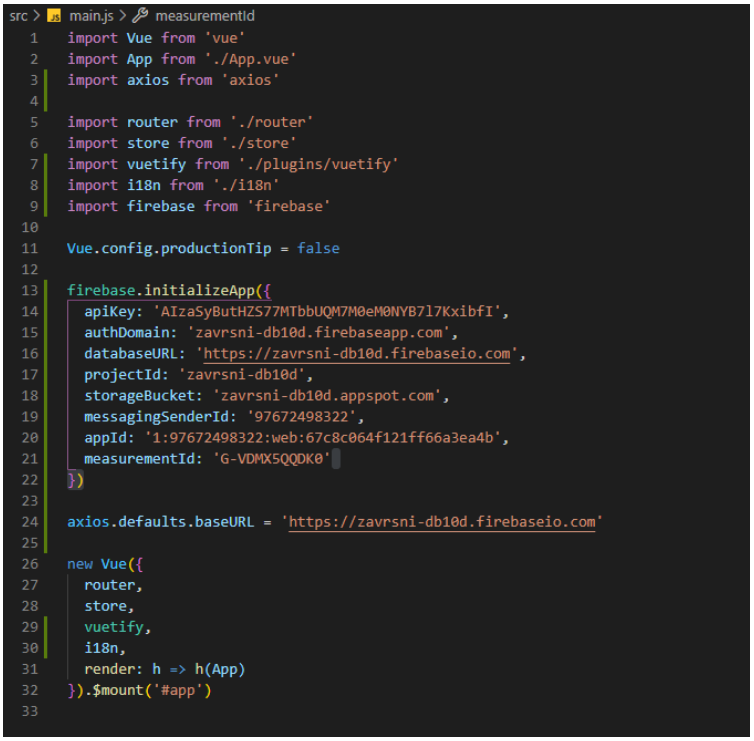
Koristeći Vue CLI naredbu *vue create zavrsni* putem naredbenog retka (engl. *Command prompt)* kreiran je Vue.js projekt sa odabranim imenom – zavrsni. *Create* naredba generira svu potrebnu početnu konfiguraciju kako bi aplikacija bila što optimiziranija. Odlaskom u stvoreni projekt i otvaranjem naredbenog retka, koristi se CLI naredba *vue add router vuex i18n* koja navedene dodatke omogućava unutar aplikacije i automatski ih integrira. Kako vuetify i axios nisu dio temeljnog Vue.js paketa, u projekt se uvode putem naredbe *npm install @nuxt/vuetify axios –save-d*.

Unutar *package.json* datoteke (slika 3.1.) se nalaze sve ovisnosti koje se koriste u projektu, kao i informacije o projektu, skripte korištene unutar projekta te sva ostala konfiguracija koja je potrebna kako bi Vue.js aplikacija ispravno radila.



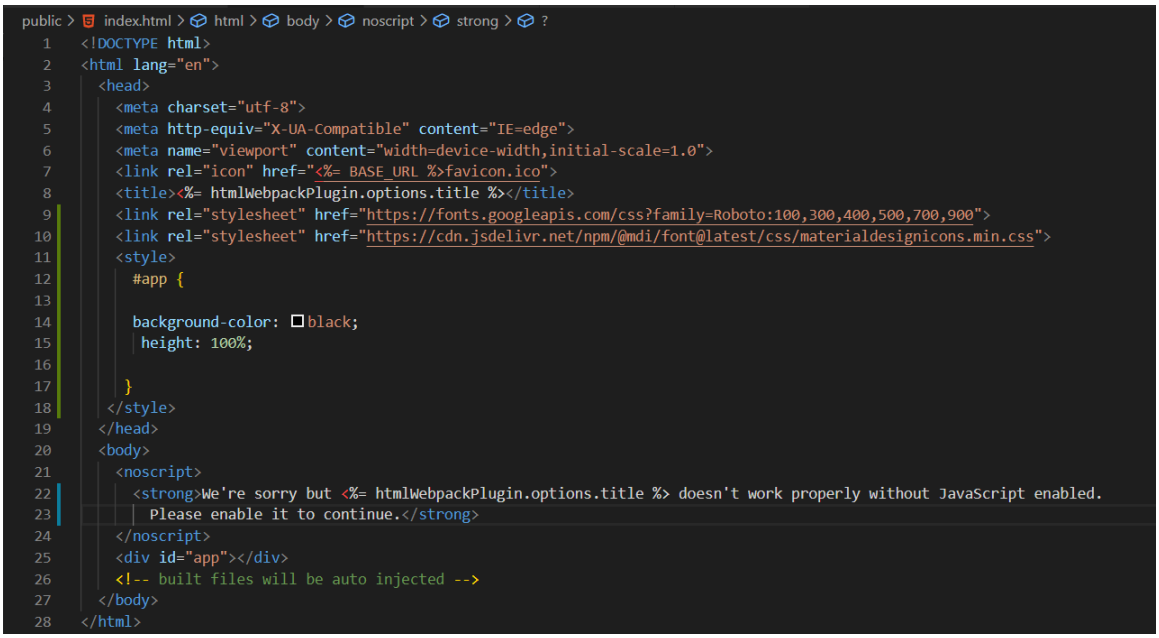
Slika 3.1. *Datoteka package.json*

Main.js je glavna datoteka (slika 3.2.) unutar koje se stvara Vue instanca i implementira sve ovisnosti koje se koriste unutar projekta, a generirana je automatski putem *vue create* naredbe i nalazi se unutar src direktorija.



Slika 3.2. *Datoteka Main.js*

Unutar *main.js* datoteke potrebno je uvesti sve ovisnosti koje su dodane projektu, kako bi se mogle navesti unutar konstruktora. Kreiranjem Vue instance unutar datoteke omogućava se korištenje svih alata koje Vue i implementirani dodaci se pružaju bilo gdje u aplikaciji tj. main.js služi kao korijenski element koji svojoj djeci pruža usluge. Također je moguće konfigurirati ovisnosti, čija je konfiguracija dostupna bilo gdje unutar aplikacije. Kako bi main.js datoteka bila pregledna, konfiguraciju usmjerivača moguće je izvršiti izvan main.js datoteke unutar druge datoteke *.js* formata, a konfiguraciju uvesti u main.js datoteku putem import naredbe. Navedeni pristup koristit će se za konfiguraciju Vue usjerivača, vuex trgovine (engl. *Store*) i i18n alata. U main.js aplikaciji moguće je navesti korijenski link *axios* alata, čime se osigurava da svaki http zahtjev izvršen korištenjem navedene *axios* instance ima zajednički korijenski link. Vue instancu potrebno je montirati na korijenski HTML element koji služi samo kao pristupna točka, a nalazi se unutar *index.html* datoteke smještene unutar javnog (engl. *public*) direktorija u projektu (slika 3.3.).



Slika 3.3. *Datoteka index.html*

Unutar *index.html* datoteke nalazi se korijenski div element s id oznakom *#app* u kojem će se injektirati Vue.js kod. U ovoj datoteci je također moguće navesti stilove koji će se odnositi na svaki dio Vue.js aplikacije.

3.1.1. Konfiguracija *VueRoutera*

Konfiguracija Vue usmjerivača smještena je unutar index.js datoteke koja se nalazi u *router* direktoriju smještenom u korijenskom src direktoriju. U *index.js* datoteci potrebno je naredbom import uvesti alate iz temeljnog Vue.js paketa kako bi se dobio očekivani rezultat. Korištenjem *Vue.use(VueRouter)* funkcije informira se korijenska Vue instanca da koristi alate pružene putem VueRouter dodatka (slika 3.4.). Na kraju je potrebno naredbom *export default router* izvesti kreirani VueRouter, kako bi se mogao uvesti i koristiti unutar Vue konstruktora u *main.js* datoteci.

```
52
53  const router = new VueRouter({
54    mode: 'history',
55    base: process.env.BASE_URL,
56    routes
57  })
58
59  export default router
```

Slika 3.4. *Definiranje routera*

Radi preglednosti koda, putanje korištene unutar stranice moguće je navesti unutar konstantnog polja nazvanog *routes*. Pružanjem konstante unutar VueRouter konstruktora injektiraju se putanje koje će se koristiti unutar aplikacije. Kada se putanje ne bi navodile u vanjskoj konstanti, sve putanje bi se morale navesti kao argument konstruktoru unutar *routes* argumenta kao objekt (routes: {...}). Zahvaljujući ES6 pravilima nije potrebno unutar konstruktora navoditi routes: routes, nego se može koristiti skraćena verzija i pisati samo *routes* ukoliko su argument konstruktora i objekt koji mu se pridjeljuje jednakih imena. Također je potrebno naredbom *import* uvesti sve Vue komponente koje se žele učitati na pojedinim putanjama. Sve komponente moguće je odmah na početku uvesti, no zahvaljujući VueRouter opciji lijenog učitavanja, pojedine komponente moguće je uvesti tek kada ih je potrebno otvoriti. U malim aplikacijama opcija lijenog učitavanja nema veliku ulogu, no ukoliko je aplikacija velika i ima velik broj komponenti, učitavanje svih komponenti odmah na početku može prouzrokovati neželjeno usporavanje aplikacije.

Svaki objekt koji se nalazi unutar *routes* polja je jedna putanja (slika 3.5.). Putanja kao argument prima *path*, tj. tekst koji se nalazi u linku nakon domene naše stranice. Putanja također može primiti argument *name* koji se koristi kada se želi pristupiti putanji putem *pushNamed()* funkcije i

argument *component* koji govori VueRouteru koju je komponentu potrebno učitati kada se pristupa nekoj putanji. Unutar argumenta *component* moguće je primjeniti VueRouter opciju lijenog učitavanja pružanjem funkcije kao vrijednosti argumenta *component*, koja kao rezultat vraća komponentu koja se želi koristiti. U argumentu *path* moguće je navesti i argumente koji će se dinamički injektirati u putanju. Navođenjem dvotočke unutar putanje informiramo VueRouter gdje je potrebno injektirati dinamički argument, a riječ nakon dvotočke gleda se kao ime dinamičkog argumenta. Dinamičke argumente prosljeđujemo VueRouteru unutar funkcije *push('/apartment, {id: 'Dinamički ID'})* kao objekt s ključ:vrijednost parovima, gdje je ključ odabrano ime dinamičke komponente.

Slika 3.5. *Konstanta routes sa svim rutama aplikacije*

3.1.2. Konfiguracija i18n

Radi preglednosti koda, konfiguracija i18n dodatka također je smještena u vanjskoj datoteci, koja je uvezena u *main.js* datoteku. Unutar i18n.js kreira se instanca VueI18n putem konstruktora. Argumente konstruktora čine *locale* i *fallbackLocale* koji imaju vrijednost *locale* kratica koje označavaju jezik tj. jezik koji će se koristiti ukoliko jezik nije podržan ili je došlo do greške. Vue CLI automatski generira *i18n.js* datoteku smještenu unutar *src* direktorija. Pri dodavanju i18n dodatka narednom *vue add* potrebno je navesti sve jezike koji će se koristiti u aplikaciji. Na temelju navedenih jezika, Vue CLI generira *locales* direktorij unutar kojeg se nalazi *.json* datoteku za svaki navedeni jezik. Konstruktor VueI18n instance prima i argument *messages* koji kao vrijednost ima funkciju *loadLocaleMessages()* (slika 3.6.).

src > i18n.js > loadLocaleMessages > forEach() callback > locale

```
1  import Vue from 'vue'
2  import VueI18n from 'vue-i18n'
3
4  Vue.use(VueI18n)
5
6  function loadLocaleMessages () {
7    const locales = require.context('./locales', true, /[A-Za-z0-9-_,\s]+\.json$/i)
8    const messages = {}
9    locales.keys().forEach(key => {
10     const matched = key.match(/([A-Za-z0-9-_]+)\./i)
11     if (matched && matched.length > 1) {
12       const locale = matched[1]
13       messages[locale] = locales(key)
14     }
15   })
16   return messages
17 }
18
19 export default new VueI18n({
20   locale: process.env.VUE_APP_I18N_LOCALE || 'en',
21   fallbackLocale: process.env.VUE_APP_I18N_FALLBACK_LOCALE || 'en',
22   messages: loadLocaleMessages()
23 })
```

Slika 3.6. *Funkcija loadLocaleMessages()*

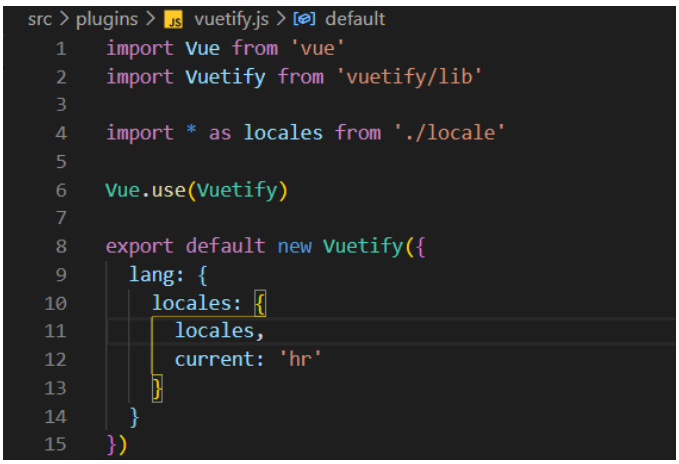
Ta funkcija na temelju stvorenih *.json* datoteka kreira objekt koji ima ključ:vrijednost strukturu, gdje su ključevi kratice jezika, a vrijednosti svi prijevodi za taj jezik (slika 3.7.). CLI također generira i lažne podatke (engl. *Dummy data*) koji su smješteni u svakoj od *.json* datoteka. Sve

prijevode je moguće smjestiti unutar generiranih *.json* datoteka, no radi lakšeg pristupanja i preglednijeg koda, u ovoj aplikaciji će se koristiti prevođenje na razini komponenata.

***Slika 3.7**. Primjer .json datoteke*

3.1.3. Konfiguracija *Vuetifya*

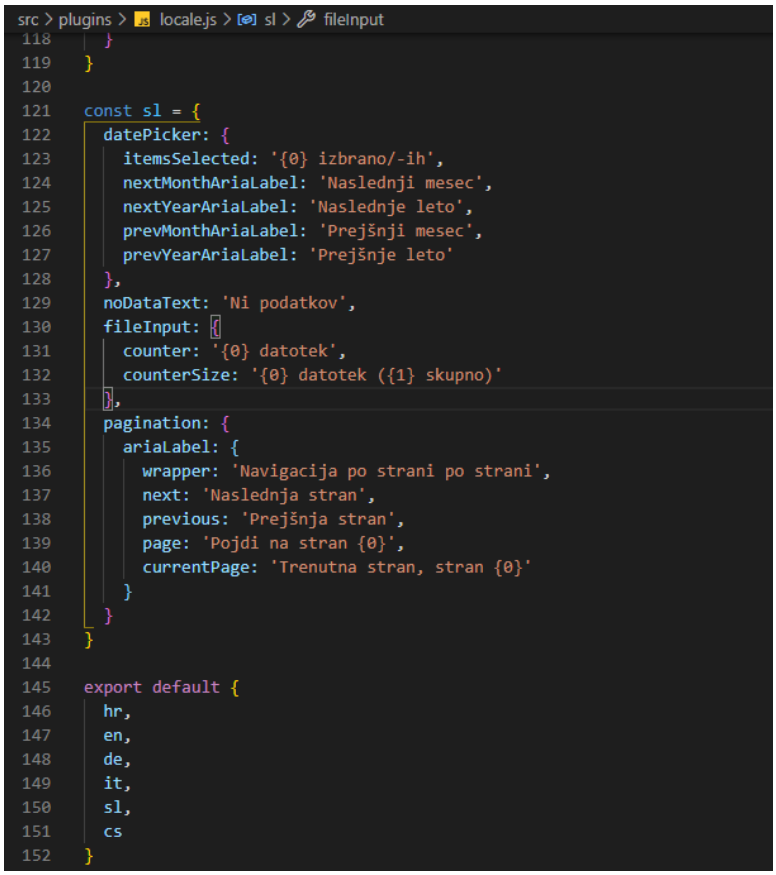
Kako bi u aplikaciji koristili Vuetify, isti je potrebno i kofigurirati. Naredbom *vue add Vue* CLI automatski generira konfiguraciju Vuetifya koja je smještena u *vuetify.js* datoteci, koja se nalazi unutar *plugins* direktorija. Na slici 3.8. prikazana je implementacija Vuetifyja.



***Slika 3.8**. Implementacija Vuetifyja*

Kako bi aplikacija mogla koristiti Vuetify alat i sve što on pruža, potrebno je nakon kreiranja instance pružiti glavnoj Vue instanci Vuetify instancu putem naredbe *Vue.use(Vuetify)*. Vuetify alat također ima i18n opciju, a njeno korištenje omogućeno je navođenjem *lang* argumenta unutar konstruktora. *Lang* argument je objekt koji prima objekt unutar kojeg se nalaze argumenti *current* (vrijednost trenutnog jezika koji se koristi) i *locales* (svi prijevodi koji se žele koristit unutar

aplikacije). Svi Vuetify prijevodi su kopija službenih Vuetify prijevoda, a smješteni su u *locale.js* datoteci koja je prikazana na slici 3.9. te se nalazi u istom direktoriju kao i *vuetify.js*.

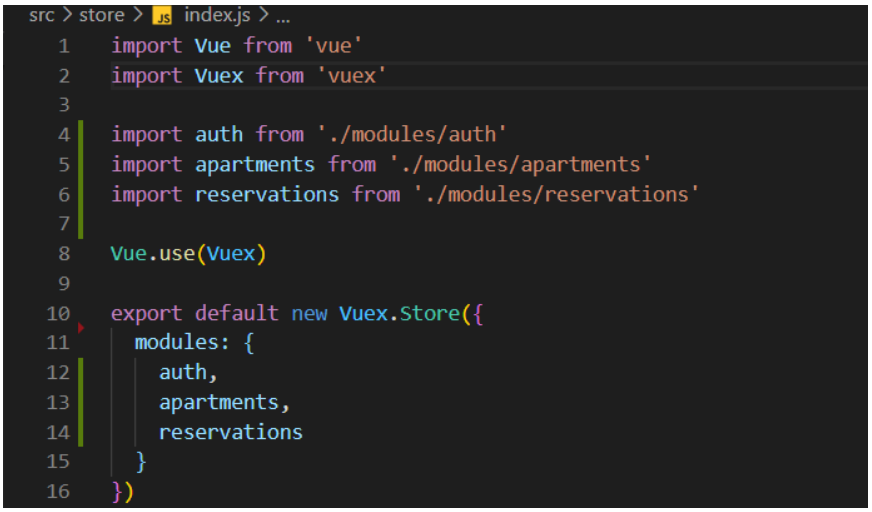


Slika 3.9. *Datoteka locales.js*

3.1.4. Konfiguracija Vuex trgovine

Vuex trgovina omogućava pristupanje podacima iz bilo koje točke aplikacije. Služi kao središnja točka za kontrolu nad zajedničkim podacima. Naredba *vue add Vue* CLI automatski generira direktorij *store* koji se nalazi unutar korijenskog src direktorija. Unutar store direktorija stvorena je *index.js* datoteka unutar koje se odvija instancirane i konfiguracija Vuex trgovine (engl. *Store*). Unutar *index.js* datoteke potrebno je uvesti Vue i Vuex iz osnovnog *Vue.js* paketa, nakon čega naredbom *Vue.use(Vuex)* omogućavamo korištenje Vuex alata. Kao i za VueRouter, potrebno je kreirati i izvesti instancu Vuex trgovine, kako bi se mogla uvesti unutar main.js datoteke i koristiti u aplikaciji. Vuex trgovinu instanciramo konstruktorom *Vuex.Store()* koji kao argumente može

primiti argumente state, mutations, actions, getters, modules. Kako bi aplikacija bila preglednija, konstruktor prima samo module koji su uvezeni, a unutar kojih se odvija sva logika skladištenja i upravljanja zajedničkih podataka Vue aplikacije. U ovom poglavlju definirat će se samo konfiguracija i instanciranje Vuex trgovine, a ostale module objasnit će se tokom rada kada njihova primjena bude potrebna. Opisani kod je prikazan na slici ispod.



Slika 3.10. *Implementacija Vuex trgovine*

3.2. Vuex nezavisne komponente

Unutar ovog potpoglavlja su detaljnije opisani načini implementacije statičkih komponenti koje ne ovise o Vuex trgovini. Podaci koje je potrebno prikazati unutar komponenata sučelja nalaze se lokalno, u aplikaciji, s obzirom da nikada neće biti promijenjeni.

3.2.1. App.vue

App.vue je prva *.vue* datoteka koja služi kao korijenska datoteka unutar koje su sve ostale komponente injektirane. Kako bi Header.vue i Footer.vue bili prikazani na svakoj stranici, potrebno ih je uvesti unutar *script* zaglavlja. Definiranjem ključa *components* i navođenjem *Header* i *Footer* unutar *components* objekta, omogućeno je korištenje navedenih komponenti. Kako bi ostale komponente uvijek došle između zaglavlja i podnožja, između komponenti

potrebno je staviti *router-view*. *Router-view* je element u koji će VueRouter injektirati komponente koje se nalaze na određenim putanjama. Taj element je smješten unutar *transition* oznaka čime je pružena animacija pri svakoj promjeni putanje. Na ispod (slika 3.11.) je prikazana App.vue datoteka.

src > App.vue > {} "App.vue" > template

```vue
<template>
  <v-app>
    <Header></Header>

    <v-main class="mb-5 mt-5">
      <transition name="slide" mode="out-in">
        <router-view></router-view>
      </transition>
    </v-main>

    <Footer></Footer>
  </v-app>
</template>

<script>
import Header from './components/header/Header.vue'
import Footer from './components/footer/Footer.vue'

export default {
  name: 'App',

  components: {
    Header,
    Footer
  },
  created () {
    this.$store.dispatch('attemptAutoLogin')
    this.$store.dispatch('fetchUsers')
    this.$store.dispatch('initApartments')
    this.$store.dispatch('initReservations')
  }
}
</script>
```

Slika 3.11. *App.vue*

3.2.2. Header.vue

Header.vue komponenta nalazi se na vrhu svake stranice, a njen izgled generiran je pomoću Vuetify *navigation-drawer* UI komponente (slika 3.12.). Predmeti koji uvijek trebaju biti prikazani generirani su pomoću *v-for* opcije na *v-list-item* komponenti. Prijevodi, ikone i putanje kojima se pristupa za svaku vezu unutar navigacije nalaze se unutar i18n atributa. Predmeti koji trebaju biti generirani s obzirom na to je li korisnik prijavljen ili ne, kreirani su uz pomoć *v-if* alata koja na temelju izračunate vrijednosti *auth* ih generira ili ne. Gumbu za odjavu korisnika pridijeljen je prisluškivač *@click* koji referencira na metodu *onLogout*. Unutar navigacije također se nalazi i *LanguagePicker* komponenta pomoću koje se mijenja jezik koji se koristi unutar aplikacije.

Slika 3.12. *Header.vue*

3.2.3. LanguagePicker.vue

LanguagePicker.vue jednostavna je komponenta koja uz pomoć HTML *select*a prikazuje popis svih dostupnih jezika. Svaka opcija unutar *select*a nalazi se unutar data objekta, a prikazuje se pomoću v-for naredbe. Kod komponente je prikazan na slici ispod (slika 3.13.).

src > components > header > LanguagePicker.vue > {} "LanguagePicker.vue" > script > default > data

```
1   <template>
2     <div class="locale-changer">
3       <select v-model="$i18n.locale">
4         <option v-for="(lang, i) in langs" :key="`Lang${i}`" :value="lang">{{ lang }}</option>
5       </select>
6     </div>
7   </template>
8
9   <script>
10  export default {
11    name: 'locale-changer',
12    data: () => ({
13      langs: ['hr', 'en', 'de', 'it', 'sl', 'cs']
14    })
15  }
16  </script>
```

Slika 3.13. *Datoteka LanguagePicker.vue*

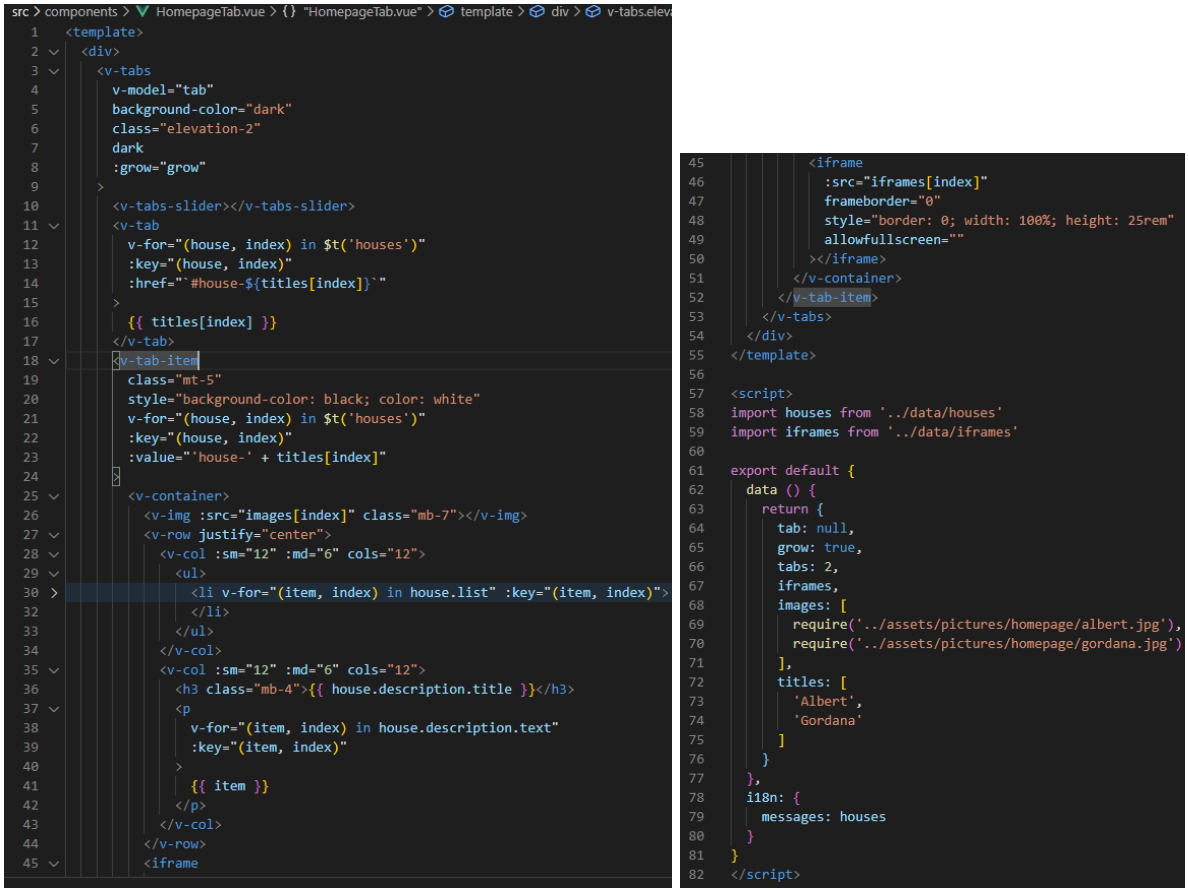
3.2.4. Footer.vue

Footer.Vue je komponenta koja je stvorena koristeći Vuetify *v-footer* komponente, čiji je sadržaj kreiran pomoću Vuetify sustava mreže. Pomoću *v-col* Footer.vue je podijeljen na dva dijela, gdje se prikazuju uvezene komponente *OwnersInfo* i *CreatorsInfo.vue* koje se sastoje od Vuetify liste.

Slika 3.14. *Datoteka Footer.vue*

3.2.5. Home.vue

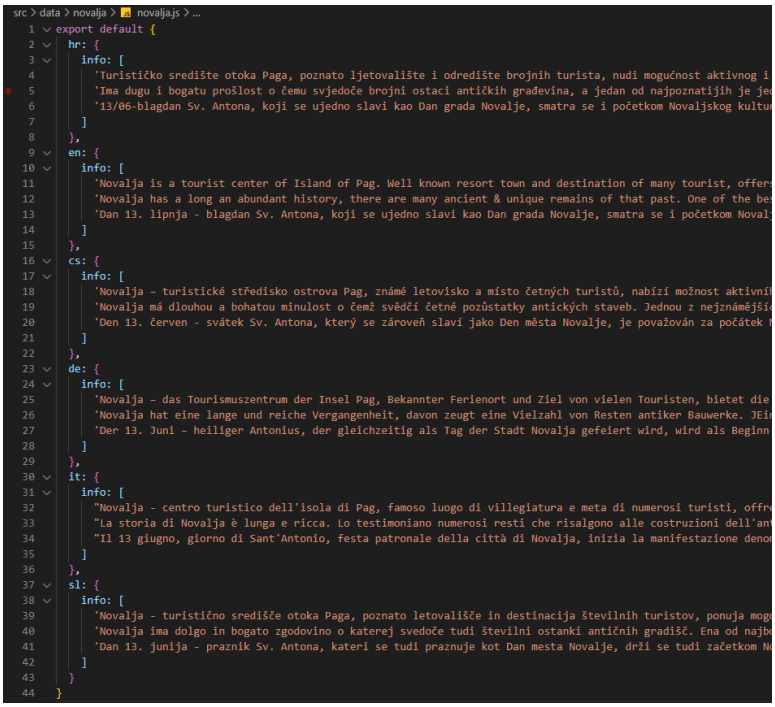
Home.vue je početna stranica aplikacije koja sadrži sliku Novalje sa svojstvom *parallax*a i *HomepageTab.vue* komponentom. Ta komponenta se sastoji od Vuetify *v-tabs* komponente u čijem se tabu prikazuje slika kuće, informacije o kući u listi i *iframe* s prikazom kuće na Google mapi. Komponenta je prikazana na slici ispod (slika 3.15.)



Slika 3.15. *Datoteka Home.vue*

3.2.6. Novalja.vue

Novalja.vue komponenta jednostavnog je dizajna unutar kojeg se nalaze komponente *Novalja.vue*, *Pag.Vue*, *Beaches.vue*, *Food.vue* i *FieldTrips.vue*. Novalja i Pag komponente služe kako bi prikazali opis o Novalji i Pagu, a sastoje se od naslova i liste informacija. Liste su generirane pomoću *v-for* a prijevodi su uvezeni iz vanjskih datoteka. Kako su podaci o Novalji i Pagu nepromjenjivi, prijevodi su skladišteni lokalno unutar data direktorija u *novalja.js* i *pag.js* datotekama. Na slici ispod (slika 3.16.) je prikazana datoteka novalja.js te je na isti princip kreirana i datoteka pag.js.



Slika 3.16. *Datoteka Novalja.js*

3.2.7. Beaches.vue, Food.vue i FieldTrips.vue

Pomoću v-for funkcije, za svaki podatak koji se nalazi unutar i18n prijevoda generira se Vuetify kartica koja u sebi sadržava sliku i ime. *Beaches* i *FieldTrips* kartice sadrže i *href* atribut (prikazane na slici 3.17.) pomoću kojeg se klikom na karticu usmjerava korisnika stranicu vezanu za stvar prikazanu na slici.

Slika 3.17. *Datoteka beaches.vue*

Kako su slike statičke tj. nikad se neće mijenjati, spremljene su lokalno unutar aplikacije u *assets* direktoriju, a njihovo učitavanje je prikazano na slici 3.18.

```
data () {
  return {
    images: [
      require('../../assets/pictures/food/cheese.jpg'),
      require('../../assets/pictures/food/goat.jpg')
    ]
  }
},
```

Slika 3.18. *Dohvaćanje slika iz assets direktorija*

Da bi kod bio pregledniji, podaci o plažama i izletima smješteni su u *beaches.js* i *fieldtrips.js* datotekama smještenim u *data* direktoriju. Na slici ispod (slika 3.19.) je prikazana datoteka *beaches.js* te je analogno njoj kreirana i *fieldtrips.js*.

src > data > novalja > beaches.js > beaches > image

```
const beaches = [
  {
    title: 'Zrće',
    link: 'https://www.zrce.com/',
    image: require('../../assets/pictures/beaches/zrce.jpg')
  },
  {
    title: 'Caska',
    link: 'https://visitnovalja.hr/uvala-caska/',
    image: require('../../assets/pictures/beaches/caska.jpg')
  },
  {
    title: 'Straško',
    link: 'https://www.campingstrasko.com/hr/',
    image: require('../../assets/pictures/beaches/strasko.jpg')
  },
  {
    title: 'Babe',
    link: 'https://www.otok-pag.hr/tours/plaza-babe/PG-TR-132',
    image: require('../../assets/pictures/beaches/babe.jpg')
  },
  {
    title: 'Trinćel',
    link: 'https://www.otok-pag.hr/tours/plaza-trincel/PG-TR-131',
    image: require('../../assets/pictures/beaches/trincel.jpg')
  },
  {
    title: 'Vrtić',
    link: 'https://www.otok-pag.hr/tours/gradska-plaza-vrtic/PG-TR-146',
    image: require('../../assets/pictures/beaches/vrtic.jpg')
  },
  {
    title: 'Braničevica',
    link: 'https://croatia.hr/hr-HR/doživljaji/plaze/plaza-branicevica',
    image: require('../../assets/pictures/beaches/branicevica.jpg')
  }
]

export default beaches
```

Slika 3.19. *Datoteka beaches.js*

3.2.8. About.Vue

About.vue komponenta (slika 3.20.) služi kako bi se prikazale informacije o tome kako doći do apartmana. Na početku sadrži dva *iframe*a unutar kojih se prikazuje Google karta za pojedini apartman, nakon čega slijedi *Transportations.vue* komponenta.

src > views > About.vue > {} "About.vue" > script > default > components

```vue
<template>
  <v-container>
    <v-row class="mb-10 text-center" style="color:white">
      <v-col v-for="(house, index) in $t('houses')" :key="(houses, index)">
        <h2>{{ house.title }}</h2>
        <iframe
          :src="house.src"
          frameborder="0"
          style="border: 0; width: 100%; height: 25rem"
          allowfullscreen=""
        ></iframe>
      </v-col>
    </v-row>
    <transportations class="mb-10"></transportations>

    <v-img src='../assets/pictures/findUs/novalja_map.jpg'></v-img>
  </v-container>
</template>

<script>
import Transportations from '../components/Transportations.vue'
import messages from '../data/houses'

export default {
  components: {
    Transportations
  },
  i18n: {
    messages: messages
  }
}
</script>
```

***Slika 3.20**. Datoteka About.vue*

Transportations.vue komponenta sadrži četiri opcije prijevoznih sredstava kojima se može doći do lokacije te kartu Novalje s ucrtanim putem do apartmana. Podaci o prijevoznim sredstvima nalaze se u vanjskoj datoteci *transportations.js* radi bolje preglednosti koda.

src > components > Transportations.vue > {} "Transportations.vue" > script > default >

```vue
<template>
  <v-row>
    <v-col
      v-for="(transportation, index) in transportations"
      :key="(transportation, index)"
      cols="3"
    >
      <v-card :href="transportation.link">
        <v-img :src="transportation.image" contain></v-img>
      </v-card>
    </v-col>
  </v-row>
</template>

<script>
import transportations from '../data/transportations'

export default {
  computed: {
    transportations () {
      return transportations
    }
  }
}
</script>
```

Slika 3.21. *Datoteka Transportations.vue*

3.3. Vuex zavisne komponente

U ovom potpoglavlju opisana je implementacija Vuex zavisnih komponenti tj. komponente koje prikazuju podatke od strane korisnika, a koje su skladištene unutar firebase baze podataka.

3.3.1. Signup.vue

Signup.vue komponenta sadrži formu pomoću koje se korisnik registrira na aplikaciji. Izgled forme postignut je pomoću Vuetify forme (slika 2.22.), a njena validacija se izvršava pomoću Vuetify *Vuelidate* paketa.

```
<v-form ref="form" v-model="valid" lazy-validation>
  <v-text-field
    :label="this.$t('email')"
    prepend-icon="mdi-account-circle"
    v-model="email"
    :rules="emailRules"
    required
  />
  <v-text-field
    :label="this.$t('password')"
    required
    :rules="passwordRules"
    :type="showPassword ? 'text' : 'password'"
    prepend-icon="mdi-lock"
    v-model="password"
    :append-icon="showPassword ? 'mdi-eye' : 'mdi-eye-off'"
    @click:append="showPassword = !showPassword"
  />
  <v-text-field
    :label="this.$t('confirmPassword')"
    required
    :rules="confirmPasswordRules"
    :type="showConfirmPassword ? 'text' : 'password'"
    prepend-icon="mdi-lock"
    v-model="confirmPassword"
    :append-icon="showConfirmPassword ? 'mdi-eye' : 'mdi-eye-off'"
    @click:append="showConfirmPassword = !showConfirmPassword"
  />
</v-form>
```

Slika 3.22. *Signup forma*

Prije nego što se podaci upisani u formu pošalju u bazu podataka i izvrši registracija korisnika, svaki podatak mora zadovoljiti određene kriterije koji se postavljaju kroz pravila. *Signup* forma ima tri seta pravila – pravila za email, lozinku i potvrdu lozinke. Svaki od ta tri seta sadrži u sebi pravilo da se mora unijeti podatak. Set pravila za email također sadrži pravilo koje provjerava je li uneseni podatak ispravan email, set pravila za lozinku sadrži provjeru je li uneseni podatak duži od 6 znakova, a potvrda lozinke provjerava je li potvrđena lozinka jednaka prethodno upisanoj lozinki. Na slici ispod (slika 3.23.) je prikazano postavljanje podataka za formu te spomenutih pravila za unos podataka.

```
<script>
import messages from '../../data/authRules'

export default {
  data () {
    return {
      valid: true,
      showPassword: false,
      showConfirmPassword: false,
      email: '',
      password: '',
      confirmPassword: ''
    }
  },
  computed: {
    emailRules () {
      return [
        (v) => /.+@.+\..+/.test(v) || this.$t('emailValid'),
        v => !!v || this.$t('req')
      ]
    },
    passwordRules () {
      return [
        (v) => !!v || this.$t('req'),
        (v) => (v && v.length <= 6) || this.$t('less')
      ]
    },
    confirmPasswordRules () {
      return [
        v => !!v || this.$t('req'),
        (v) => v === this.password || this.$t('same')
      ]
    }
  },
```

Slika 3.23. *Pravila za unos podataka u Signup.vue*

Ukoliko su sva pravila zadovoljena, omogućeno je slanje podataka u Firebase bazu podataka putem metode *onSubmit* koja u sebi sadrži naredbu *dispatch* putem koje se poziva akcija *signup* koja se nalazi unutar *vuex* trgovine (slika 3.24.).

```
onSubmit () {
  const formData = {
    email: this.email,
    password: this.password,
    confirmPassword: this.confirmPassword,
    admin: false
  }
  this.$store.dispatch('signup', formData)
}
```

Slika 3.24. *onSubmit metoda*

3.3.2. Login.vue

Login.vue sadrži formu pomoću koje se korisnik prijavljuje na stranicu te je forma kreirana analogno signup formi (slika 2.22.). U login formi je potrebno unijeti email i zaporku korisnika te postoji samo jedno pravilo – *required* pravilo. To pravilo služi za provjeru je li korisnik unio podatak. Ukoliko forma uspješno prođe validaciju, utoliko je omogućeno pozivanje *onSubmit* metode, koja unutar sebe poziva login akciju s podacima iz forme. Login akcija nalazi se u Vuex trgovini. Na slici ispod (slika 2.25.) je prikazano spomenuto *required* pravilo te *onSubmit* metoda za slanje podataka.

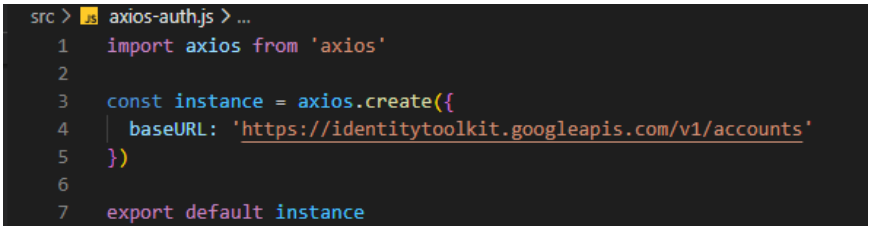
```
<script>
import messages from '../../data/authRules'

export default {
  data () {
    return {
      valid: true,
      showPassword: false,
      email: '',
      password: ''
    }
  },
  methods: {
    onSubmit () {
      const formData = {
        email: this.email,
        password: this.password
      }
      this.$store.dispatch('login', {
        email: formData.email,
        password: formData.password
      })
    }
  },
  computed: {
    required () {
      return [v => !!v || this.$t('req')]
    }
  },
  i18n: {
    messages
  }
}
</script>
```

Slika 3.25. *Datoteka Login.vue*

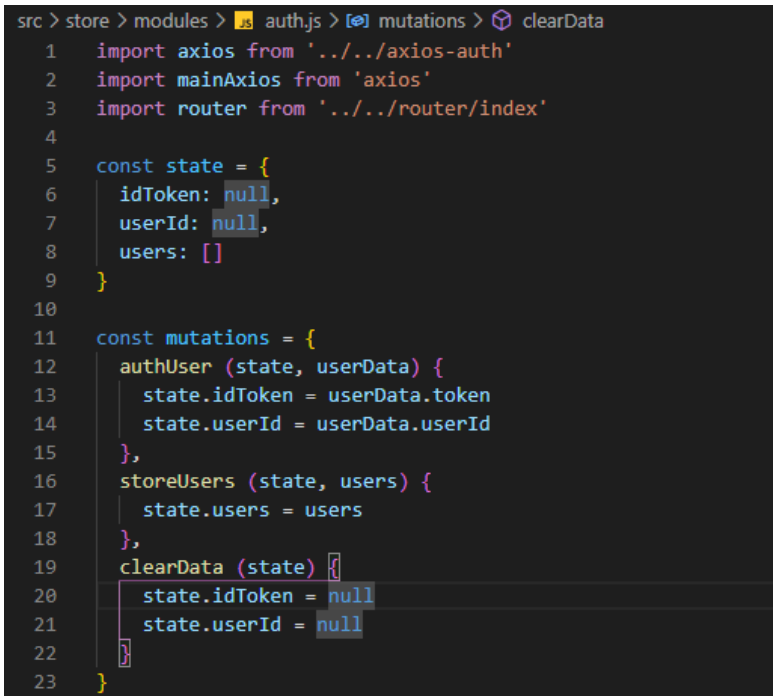
3.3.3. Vuex auth

Da bi izvršili pozvane akcije, prvo ih je potrebno definirati unutar Vuex trgovine. Kako bi se olakšao pristup korisnicima aplikacije, kreirana je nova axios instanca koja kao *baseURL* ima Firebase predefiniran URL za pristup korisničkih računa aplikacije (slika 3.26.).



Slika 3.26. *axios instanca za autentifikaciju*

Svi podaci i metode vezane za autentifikaciju smješteni su unutar modula nazvanog *auth.js* (slika 2.27.). Unutar auth.js definiran je *state* objekt unutar kojeg se spremaju podaci koji se mogu koristiti u cijeloj aplikaciji. Da bi se podaci u *state* objektu mijenjali, potrebno je definirati mutacije. *AuthUser* mutacija sprema podatke o prijavljenom korisniku, *storeUsers* mutacija sprema sve korisnike aplikacije, a *clearData* briše podatke o prijavljenom korisniku.



Slika 3.27. *Datoteka auth.js*

Unutar *actions* bloka definirane su akcije poput već objašnjenih login i signup akcija, no još jedna bitna akcija je *setLogoutTimer* prikazana na slici 3.28. Nakon što se korisnik prijavi, dodjeljuje se JWT (engl. *JSON web Token*) koji ističe nakon sat vremena. *SetLogoutTmer* akcija kao argument prima vrijednost vrijeme u sekundama nakon kojeg je potrebno korisnika automatski odjaviti. Nakon što vrijeme definirano u *setTimeout* funkciji istekne, poziva se *clearData* mutacija.

```
26    setLogoutTimer ({ commit }, expirationTime) {
27      setTimeout(() => {
28        commit('clearData')
29      }, expirationTime * 1000)
```

Slika 3.28. *setLogoutTimer akcija*

Signup akcija kao argument prima podatke iz *Signup* forme, koje putem *axios.post* metode uz predefiniran upit šalje na Firebase. Post funkcija kao rezultat vraća JWT i ID novostvorenog korisnika koji se spremaju u *state* pomoću *authUser* mutacije. JWT i ID se također spremaju u lokalu memoriju (*localStorage*) korištenog preglednika, kao i email prijavljenog korisnika i datum isteka JWT-a. Unutar *signup* akcije poziva se i *storeUser* akcija, *setLogoutTimer* i *router.replace('/')* funkcija koja signup putanju mijenja sa putanjom početnog zaslona. Ukoliko je signup akcija uspješno izvršena, *storeUser* akcija šalje podatke o novom korisniku i sprema ih unutar Firebase baze podataka kako bi se moglo pristupiti podacima o korisniku. Kod procesa signupa se nalazi na slici ispod (slika 3.29.)

```
signup ({ commit, dispatch }, authData) {
  axios.post(':signUp?key=AIzaSyButHZS77MTbbUQM7M0eM0NYB7l7KxibfI', {
    email: authData.email,
    password: authData.password,
    returnSecureToken: true
  })
    .then(res => {
      commit('authUser', {
        token: res.data.idToken,
        userId: res.data.localId
      })
      const now = new Date()
      const expirationDate = new Date(now.getTime() + res.data.expiresIn * 1000)
      localStorage.setItem('token', res.data.idToken)
      localStorage.setItem('userId', res.data.localId)
      localStorage.setItem('expirationDate', expirationDate)
      localStorage.setItem('email', authData.email)
      dispatch('storeUser', authData)
      dispatch('setLogoutTimer', res.data.expiresIn)
      router.replace('/')
    })
    .catch()
},
storeUser ({ commit, state }, userData) {
  if (!state.idToken) {
    return
  }
  mainAxios.post('/users.json' + '?auth=' + state.idToken, userData)
    .then()
    .catch()
},
```

Slika 3.29. *Signup akcija*

Login akcija jako nalikuje signup akciji, razlika je jedino u predefiniranom Firebase upitu koji se koristi u *axios.post* funkciji budući da u ovom slučaju nije potrebno spremiti novog korisnika u bazu.

Funkcija automatskog ponovnog prijavljivanja postignuta je *attemptAutoLogin* akcijom (slika 3.30.). Akcija provjerava postoji li u memoriji preglednika predmet s ključem token i je li rok isticanja tokena manji nego trenutno vrijeme tj. je li istekao. Ukoliko token postoji i vrijeme nije isteklo, iz memorije preglednika dohvaća se ID korisnika i pomoću mutacije *authUser*, dohvaćeni podaci spremaju se u Vuex state.

```
attemptAutoLogin ({ commit }) {
  const token = localStorage.getItem('token')
  if (!token) return
  const expirationDate = localStorage.getItem('expirationDate')
  const now = new Date()
  if (now >= expirationDate) {
    return
  }
  const userId = localStorage.getItem('userId')
  commit('authUser', {
    token: token,
    userId: userId
  })
},
```

Slika 3.30. *attemptAutoLogin akcija*

Akcija *logout* poziva mutaciju *clearData*, preusmjerava korisnika na login stranicu i oslobađa memoriju preglednika (slika 3.31).

```
98   logout ({ commit }) {
99     commit('clearData')
100    router.replace('/login')
101    localStorage.clear()
102  },
```

Slika 3.31. *logout akcija*

FetchUsers akcija, prikazana na slici ispod (slika 3.32.), dohvaća sve korisnike spremljene u Firebase bazi podataka te ih sprema unutar u *state*. Kako se svakom podatku koji se sprema u Firebase pridjeljuje Firebase generirani ključ, prolaskom kroz polje korisnika dobivenih kao rezultat post metode izvlači se ključ i dodjeljuje kao podatak svakom korisniku. Izvlačenje ključa rezultira sinkroniziranim ID vrijednostima u Vuex stateu i Firebase bazi podataka, što omogućuje upite i dohvaćanje određenog korisnika iz baze. Ažurirani podaci spremaju se ponovno u state.

```
103    fetchUsers ({ commit, state }) {
104      mainAxios.get('/users.json')
105        .then((res) => {
106          const data = res.data
107          const users = []
108          for (const key in data) {
109            const user = data[key]
110            user.id = key
111            users.push(user)
112          }
113          commit('storeUsers', users)
114        })
115        .catch()
116    }
117  }
```

Slika 3.32. *fetchUsers akcija*

Getteri su metode kojima dohvaćamo podatke unutar *state* objekta. Na slici 3.33. su prikazani svi implementirani getteri. Getter *user* kao argument prima email, a ukoliko je korisnik prijavljen i pronađen je korisnik sa proslijeđenim email u polju korisnika, getter vraća objekt pronađenog korisnika. U suprotnom vraća prazan objekt.

```
119  const getters = {
120    user: (state, getters) => (email) => {
121      if (!getters.isAuthenticated) return {}
122      const user = state.users.find(user => user.email === email)
123      if (!user) {
124        return {}
125      }
126      return user
127    },
128    userId (state) {
129      return state.userId
130    },
131    isAuthenticated (state) {
132      return state.idToken !== null
133    },
134    idToken (state) {
135      return state.idToken
136    }
137  }
```

Slika 3.33. *Getteri*

3.3.4. Apartments.vue

Apartments.vue je jednostavna komponenta koja služi kao korijenska komponenta unutar koje se nalaze *AddAparmentForm* komponenta i *Apartment* komponenta. Kako bi se dvije navedene komponente pravilno instancirale, potrebno je iz Vuex trgovine dohvatiti sve apartmane pomoću *apartments* gettera i provjeriti je li korisnik admin putem *admin* gettera.

```
18  computed: {
19    apartments () {
20      return this.$store.getters.apartments
21    },
22    admin () {
23      const res = this.$store.getters.user(localStorage.getItem('email')).admin
24      return res
25    }
26  },
27  components: {
28    Apartment,
29    AddApartmentFrom
30  }
```

Slika 3.34. *Instanciranje komponenti Apartment i AddApartmentForm*

AddApartmentForm.vue je komponenta koja se prikazuje samo ukoliko je korisnik *admin*, a ona omgućava stvaranje novih apartmana. Kako je u aplikaciji moguće koristiti 6 jezika, korisniku (adminu) je potrebno za svaki unos ponuditi 6 polja, jedno za svaki jezik. U formi se također nalazi i prostor za unos slike i mogućnost dinamičkog unosa broja elemenata u info objektu pomoću brojača *infoCounter*a (slika 3.35.) i gumbova za smanjivanje i povećavanje brojača.

```
105        <div v-show="showInfoForm">
106          <v-btn @click="infoCounter++">{{ $t("info.addInfo") }}</v-btn>
107          <v-btn v-if="infoCounter > 0" @click="infoCounter--" class="ml-3">{{
108            $t("info.reduceInfo")
109          }}</v-btn>
110          <div v-for="i in infoCounter" :key="i">
111            <h4>{{ $t("info.info") }} - {{ i }}.</h4>
112            <v-text-field
113              v-model="formData.info['hr'][i - 1]"
114              :rules="infoRules"
115              :label="'hr - ' + $t('info.info')"
116              required
117            ></v-text-field>
118 >          <v-text-field ···
123            ></v-text-field>
```

Slika 3.35. *Brojač za dodavanje elemenata*

Funkcija *saveData* (prikazana na slici 3.36.) sprema *formData* objekt u Firebase bazu podataka pomoću POST metode kojoj se predaje autentifikacijski token i podatci iz forme, a ukoliko su podatci uspješno spremljeni, slika se šalje i sprema na Firebase Storage.

```
230    saveData () {
231      axios
232        .post(
233          '/apartments.json' + '?auth=' + this.$store.getters.idToken,
234          this.formData
235        )
236        .then((res) => {
237          console.log(res.data)
238          this.picture = null
239          const storageRef = firebase.storage().ref(res.data.name).put(this.files)
240          storageRef.on('state_changed', snapshot => {}, error => console.log(error.message))
241        })
242        .catch((err) => console.log(err))
243    }
```

Slika 3.36. *Funkcija saveData*

Za korištenje Firebase Storagea, potrebno je konfigurirati njegovu instancu – navesti API ključ, domenu i URL baze, ID projekta što je prikazano na slici 3.37.

```
firebase.initializeApp({
  apiKey: 'AIzaSyButHZS77MTbbUQM7M0eM0NYB7l7KxibfI',
  authDomain: 'zavrsni-db10d.firebaseapp.com',
  databaseURL: 'https://zavrsni-db10d.firebaseio.com',
  projectId: 'zavrsni-db10d',
  storageBucket: 'zavrsni-db10d.appspot.com',
  messagingSenderId: '97672498322',
  appId: '1:97672498322:web:67c8c064f121ff66a3ea4b',
  measurementId: 'G-VDMX5QQDK0'
})
```

Slika 3.37. *Konfiguracija Firebase Storagea*

Kako bi se mogli prikazati apartmani pohranjeni u bazi podataka, potrebno je kreirati novi Vuex modul. Unutar *apartments.js* definiran je *state* objekt koji sadrži sve apartmane, mutacija *SET_APARTMENTS* koja sprema proslijeđene apartmane u state, akcija *initApartments* pomoću koje se pristupa bazi i dohvaća sve apartmane unutar nje i apartments i apartment(id) getteri pomoću kojih se dohvaćaju svih apartmani tj. apartman sa zadanim ID-jem.

Pomoću *Apartment.vue* komponente prikazuju se apartmani u *Apartments* komponenti. *Apartment.vue* kao svojstvo prima apartman koji se prikazuje, a unutar *created* metode dohvaća se slika apartmana sa Firebase Storagea (slika 3.38.).

```
created () {
  firebase.storage().ref(this.apartmentId).getDownloadURL().then(url => {
    this.src = url
  })
},
```

Slika 3.38. *Dohvaćanje slike s Firebase Storagea*

Kako su prijevodi vezani za apartman spremljeni u bazi podataka, nije moguće i18n objektu pružiti prijevode standardnim načinom putem *messages* objekta. Da bi se prikazali prijevodi potrebno je kreirati *dynamicLocalization mixin* (prikazan na slici 3.39.) pomoću kojeg dohvaćamo prijevode za svaki apartman.

```
src > mixins > dynamicLocalization.js > default > methods > $t
export default {
  methods: {
    $t: function (translate) {
      if (typeof translate === 'string') {
        return this.$i18n.t(translate)
      } else if (!translate) {
        return this.$i18n.t('loading')
      }
      return translate[this.$i18n.locale]
    }
  }
}
```

Slika 3.39. *Datoteka dynamicLocalization.js*

Pritiskom gumba za rezervaciju, VueRouter preusmjerava na Apartment.vue komponentu vezanu za odabrani apartman putem ID argumenta u putanji.

Apartment.vue komponenta koja je namijenjena samo za jedan apartman nalazi se unutar views direktorija, dok *Apartman.vue* komponenta pomoću koje se prikazuju apartmani na Apartments.vue nalazi unutar *components* direktorija. Na parametarskoj putanji */apartment:id* nalazi se slika apartmana s vrijednosti ID i Vuetify *v-date-picker*. Ukoliko je korisnik prijavljen, moguće je rezervirati apartman odabirom početnog i krajnjeg datuma rezervacije. *Date picker* kao pravila prima minimalni datum tj. trenutni datum i dozvoljene datume. Dozvoljene datume dohvaća se iz reservations.js Vuex modula, a njihova vrijednost dodjeljuje se *reservedDates* objektu unutar *created* metode. Dozvoljeni datumi se računaju pomoću *getAllowedDates* metode (slika 3.40.).

```
60  getAllowedDates (val) {
61    return !this.reservedDates.some((element) => {
62      const from = new Date(element.dates[0]).getTime()
63      const to = new Date(element.dates[1]).getTime()
64      const check = new Date(val).getTime()
65      return check >= from && check <= to
66    })
```

Slika 3.40. *Metoda getAllowedDates()*

Pomoću *saveReservation* metode, odabrana rezervacija sprema se u bazu podataka. Spremanje podataka je prikazano na slici 3.41.

```
saveReservation () {
  this.reservationData.apartmentId = this.$route.params.id
  this.reservationData.userEmail = localStorage.getItem('email')
  axios
    .post(
      '/reservations.json' + '?auth=' + this.$store.getters.idToken,
      this.reservationData
    )
    .then((res) => {
      this.reservationData.id = res.data.name
    })
    .catch((err) => console.log(err))

  this.$store.dispatch('storeReservation', this.reservationData)
  this.$router.replace('/')
}
```

Slika 3.41. *Metoda saveReservation() za spremanje rezervacija*

U *reservations.js* modulu definiran je state *reservations* u koji se spremaju sve rezervacije, mutacije pomoću kojih se spremaju proslijeđeni podatke u state, akcije koje dohvaćaju podatke iz baze podataka (slika 3.42.) i getteri *apartmentReservations(id)* i *userReservations(id)* pomoću kojih se dohvaćaju sve rezervacije apartmana tj. rezervacije korisnika (slika 3.43.).

```
initReservations ({ commit, state }) {
  mainAxios.get('/reservations.json')
    .then(response => {
      const data = response.data
      const reservations = []
      for (const key in data) {
        const reservation = data[key]
        reservation.id = key
        reservations.push(reservation)
      }
      commit('SET_RESERVATIONS', reservations)
    })
    .catch(err => console.log(err))
},
```

Slika 3.42. *Dohvaćanje svih rezervacija iz baze podataka*

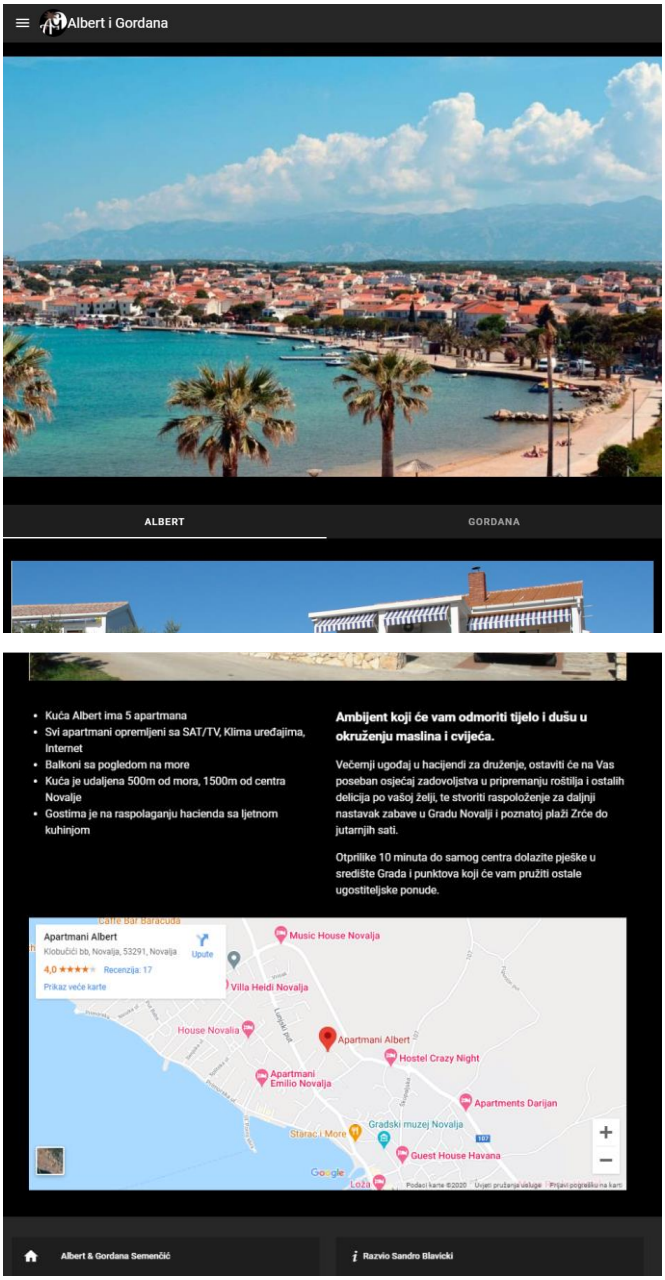
```
apartmentReservations: (state) => (id) => {
  var apartmentReservations = []
  apartmentReservations = state.reservations.filter(reservation => reservation.apartmentId === id)
  return apartmentReservations
},
reservations (state) {
  return state.reservations
}
```

Slika 3.43. *Getteri rezervacija*

4. TESTIRANJE APLIKACIJE

U ovom poglavlju prikazan je tijek korištenja aplikacije te je za svaki korak prikazana slika zaslona aplikacije te su objašnjenje sve funkcionalnosti na toj stranici.

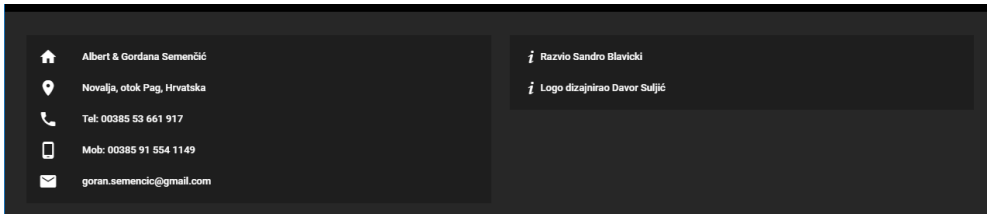
Pri otvaranju aplikacije, učitava se *HomePage.vue* komponenta koja je prikazana na slici ispod (slika 4.1.).



Slika 4.1. *Početni zaslon aplikacije - HomePage.vue*

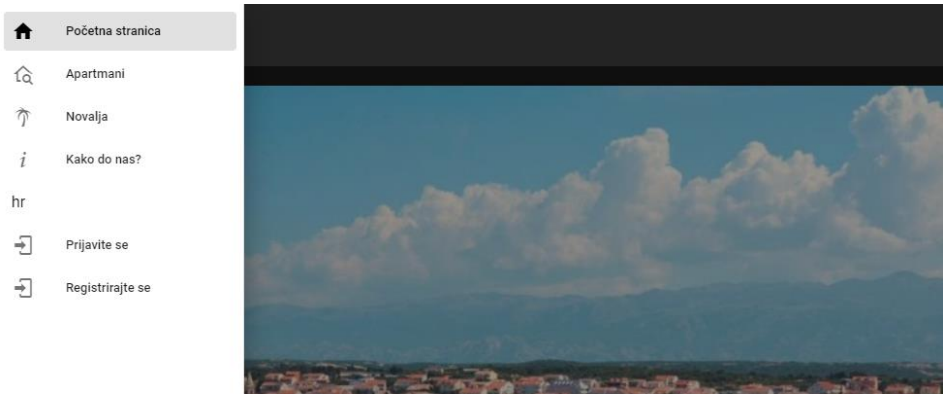
Zaglavlje (engl. *header*) (slika 4.2.) i footer (slika 4.3.) komponente vidljive su na svakoj stranici te se nalaze pri vrhu i pri dnu stranice.

Slika 4.2. *Izgled zaglavlja*



Slika 4.3. *Izgled footera*

Klikom na tri horizontalne crte smještene unutar zaglavlja (gornja lijeva strana) otvara se se navigacijska ladica u kojoj su smješteni svi linkove aplikacije. Ukoliko korisnik nije prijavljen, ladica će sadržavati gumb za prijavu i registraciju. Izgled navigacijske ladice je prikazan na slici 4.4.



Slika 4.4. *Izgled navigacijske ladice*

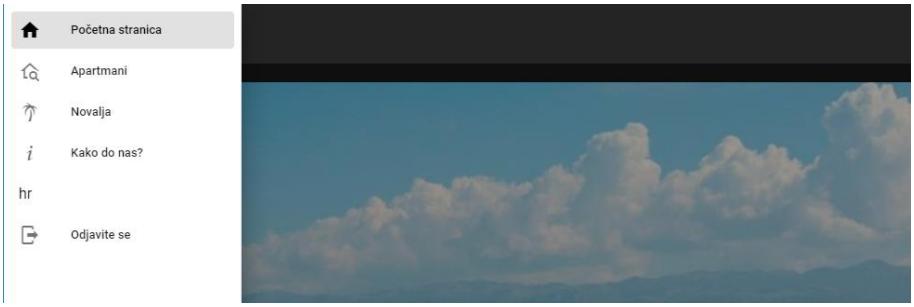
Pritiskom gumba za prijavu, učitava se zaslon za prijavu gdje korisnik mora unijeti svoj email i zaporku (slika 4.5.).

Slika 4.5. *Zaslon za prijavu*

U slučaju da korisnik nije registriran, potrebno je prvo unijeti podatke za registraciju kako bi se mogao prijaviti na stranicu. Korisnik mora unijeti email, zaporku te potvrdu zaporke kao što se može vidjeti na slici 4.6.

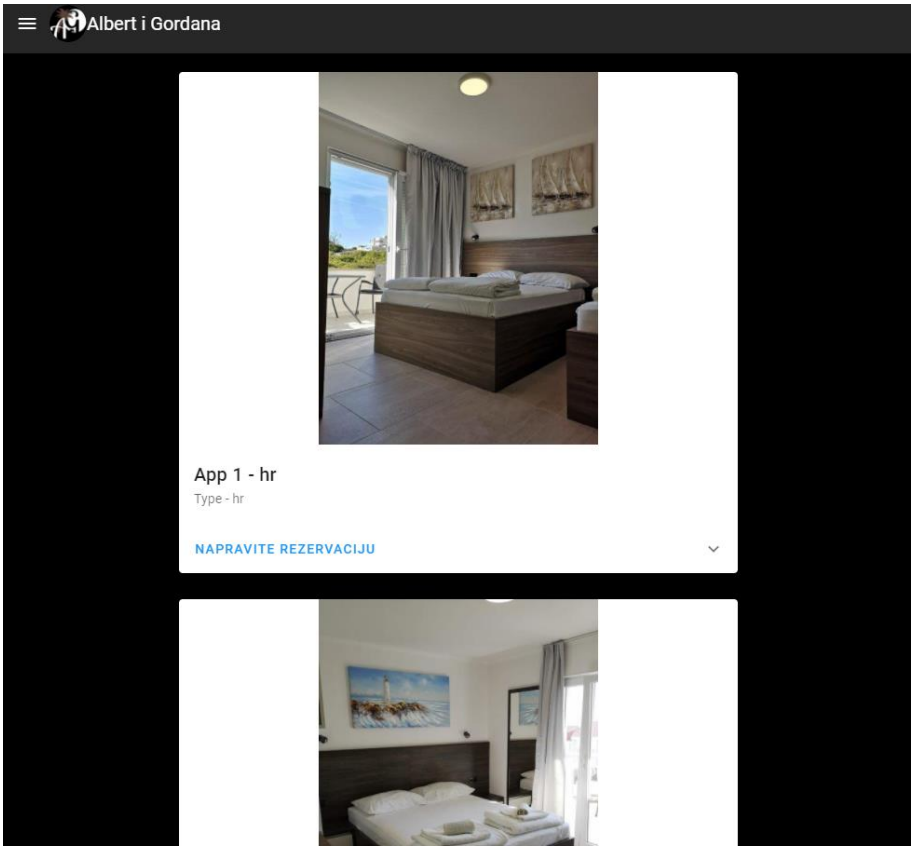
Slika 4.6. *Zaslon za registraciju*

Nakon registracije/prijave, unutar navigacijske ladice se prikazuje gumb za odjavu korisnika, gumbovi za registraciju i prijavu se skrivaju (slika 4.7.).

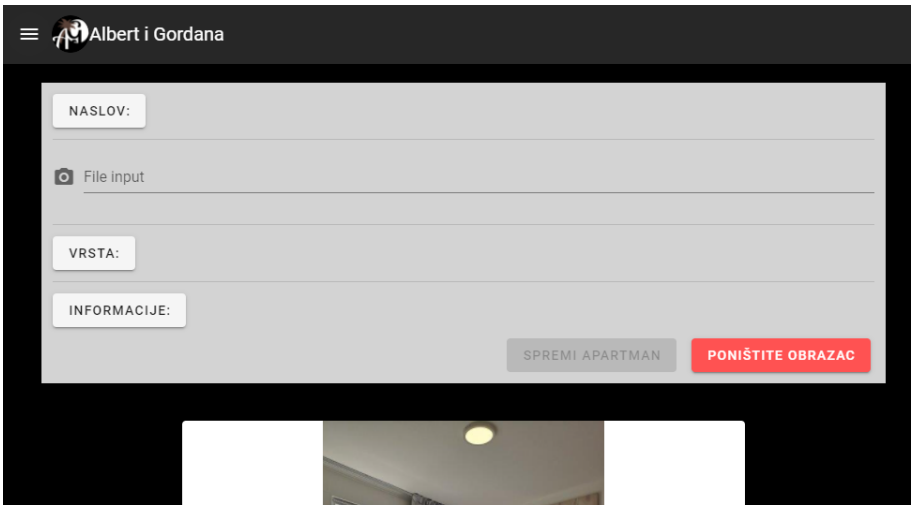


Slika 4.7. *Navigacijska ladica nakon prijave korisnika*

Odabirom opcije *Apartmani*, učitava se stranica koja sadržava sve dostupne apartmane (slika 4.8.), a u slučaju da korisnik ima svojstvo admina, prikazuje mu se i forma za dodavanje novih apartmana iznad popisa apartmana (slika 4.9.).

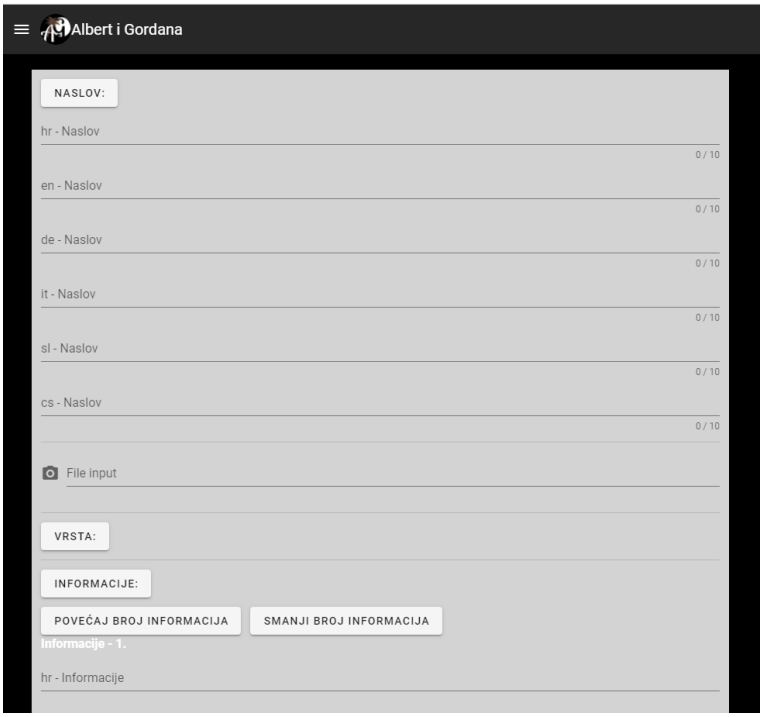


Slika 4.8. *Prikaz svih apartmana*



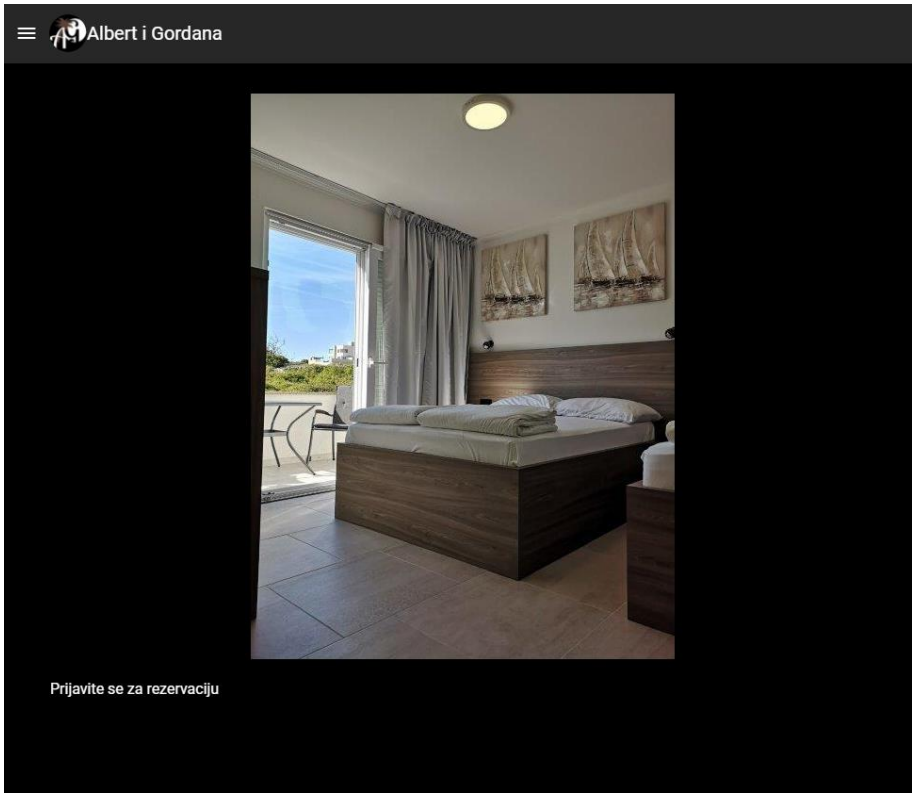
Slika 4.9. *Prikaz forme za dodavanje novih apartmana*

Klikom na gumb Naslov, Vrsta i Informacije prikazuju se mjesta za unos informacija. Kako aplikacija koristi 6 jezika, za svaki unos je potreno pružiti 6 prijevoda.



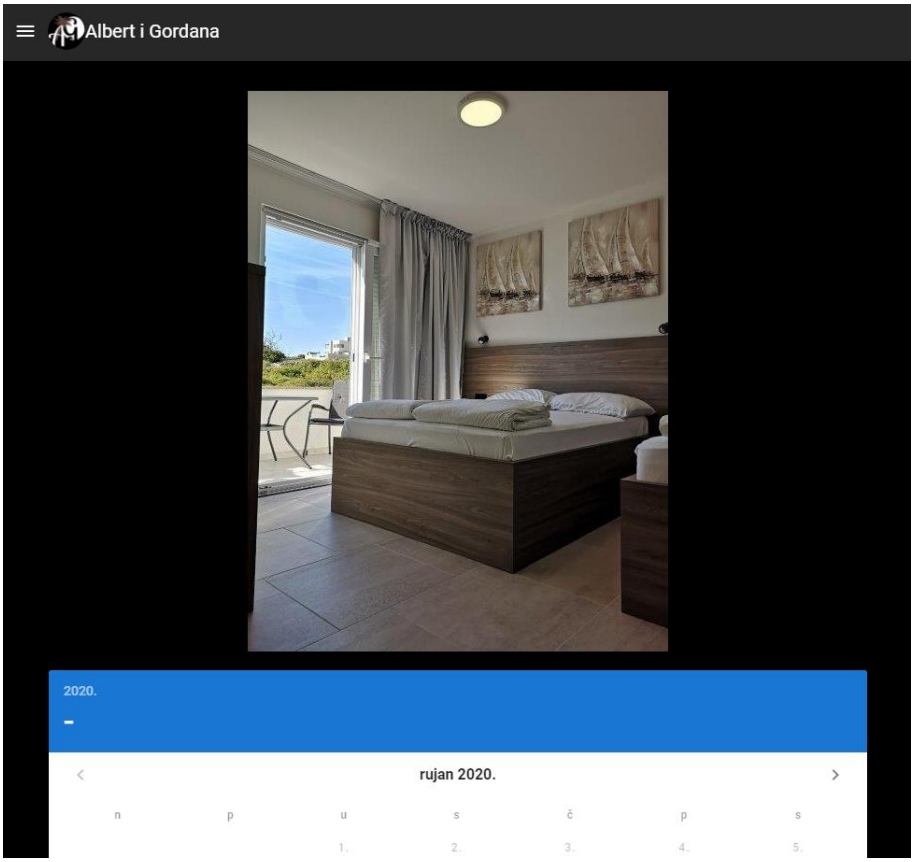
Slika 4.10. *Prikaz forme za unos informacija*

Klikom na gumb *Napravite rezervaciju*, aplikacija se usmjerava na stranicu koja prikazuje odabrani apartman. Ukoliko korisnik nije prijaveljn, u komponenti se prikazuje samo slika apartmana s napomenom da se korisnik treba prijaviti kako bi mogao rezervirati apartman.

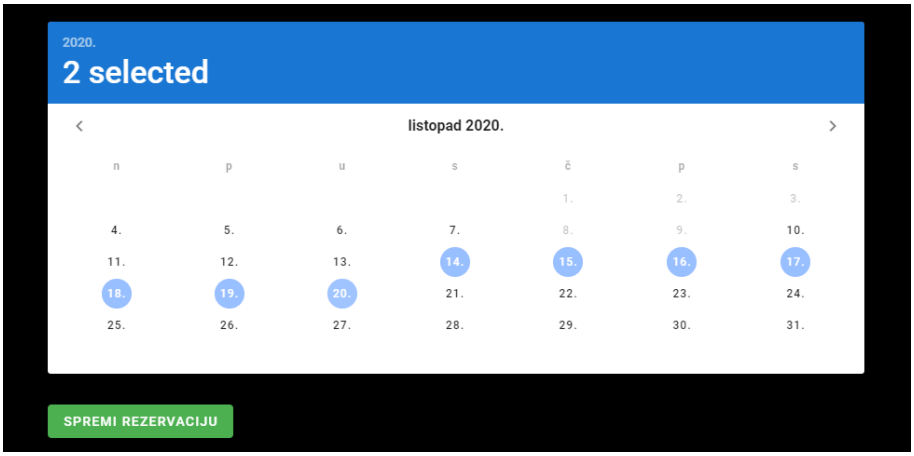


Slika 4.11. *Prikaz apartmana*

Na otvorenoj stranici (slika 4.12.) komponenti vide se slika apartmana i *date-picker* pomoću kojeg se odabire željeni datum za rezervaciju.

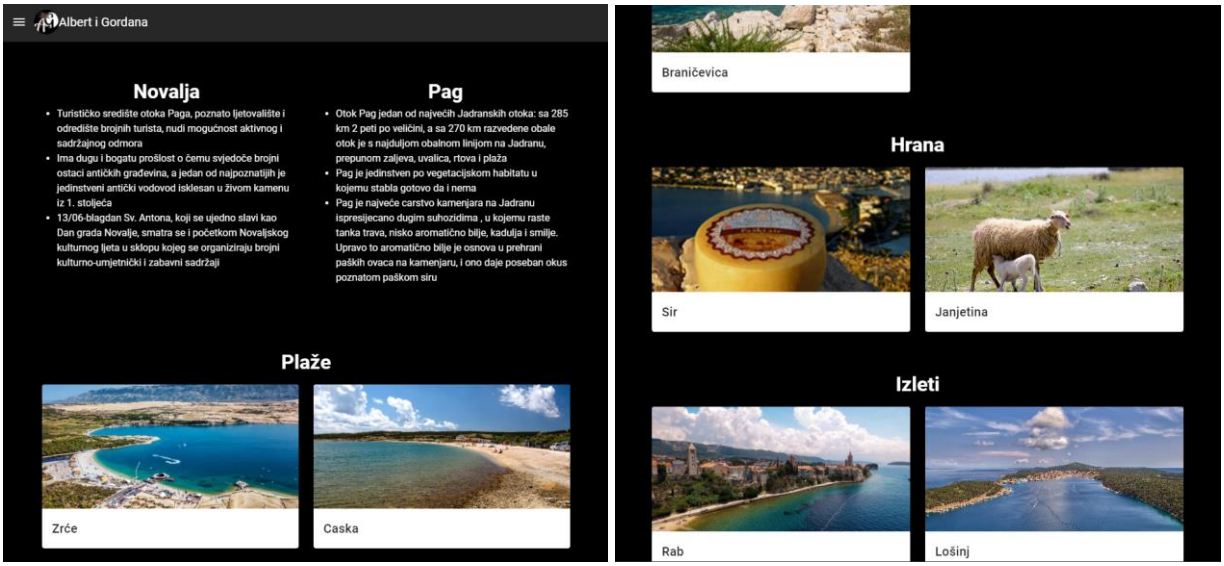


Na slici 4.12. je prikazan način biranja datuma rezervacije te je vidljivo da su neki datumi već onemogućeni (već postoji rezervacija za taj datum), a nakon odabiranja datuma dolaska i datuma odlaska, pojavljuje se *Spremi Rezervaciju* gumb.



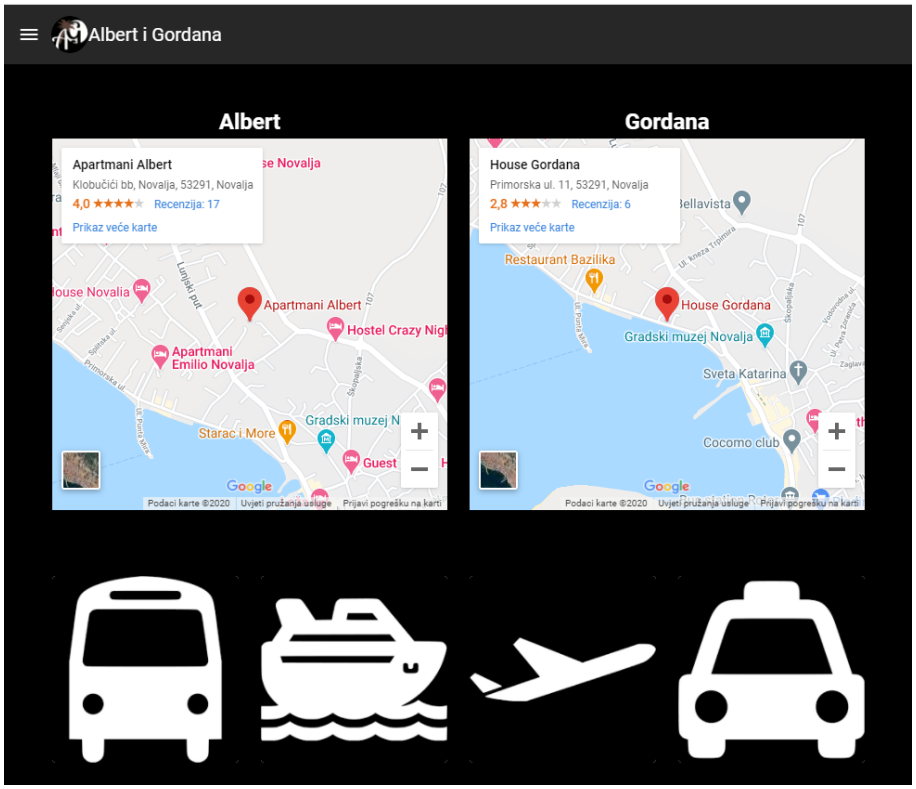
Slika 4.12. *Date-picker za odabir datuma rezervacije*

Ukoliko se unutar navigacijske ladice odabere *Novalja*, učitava se stranica, prikazana na slici 4.13., koja sadrži informacije o gradu Novalji, otoku Pagu, svim plažama koje se nalaze na otoku, ponudi hrane te su prikazani dostupni izleti u blizini.



Slika 4.13. *Informacije o Novalji*

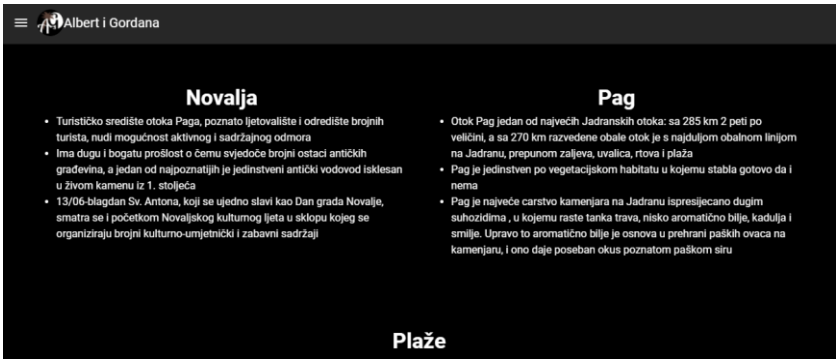
Odabirom *Kako do nas?* unutar navigacijske ladice otvara se stranica sa informacijama o ponuđenim smještajima. Prikazane su Google karte sa označenim lokacijama smještaja (slika 4.14.) te je također prikazana karta s ucrtanom putanjom do oba smještaja (slika 4.15.).



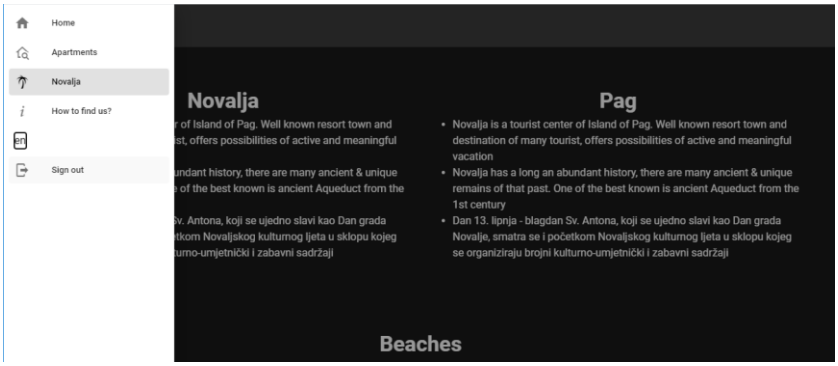
Slika 4.14. *Prikaz Google karti s označenim lokacijama smještaja*

Slika 4.15. *Karta s ucrtanom putanjom do smještaja*

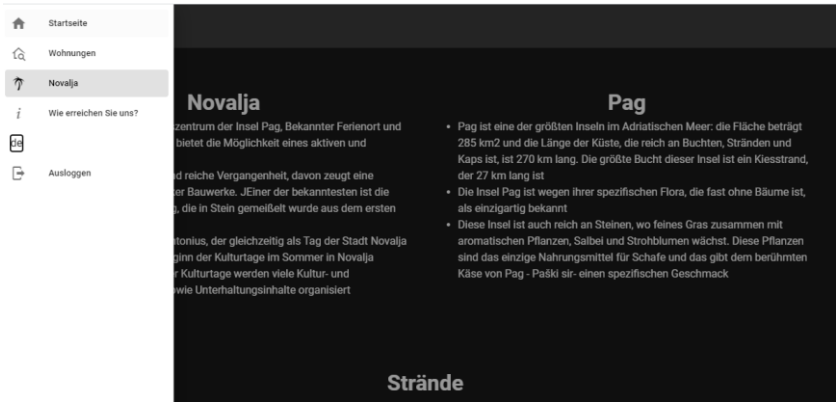
Kako bi se testiralo prevođenje na strane jezike, na slikama 4.16.-4.21. je prikazana stranica s informacijama o Novalji na temelju koje će se demonstrirati i18n alat za prevođenje. Za mijenjanje jezika, potrebno je u navigacijskoj ladici promijeniti jezik.



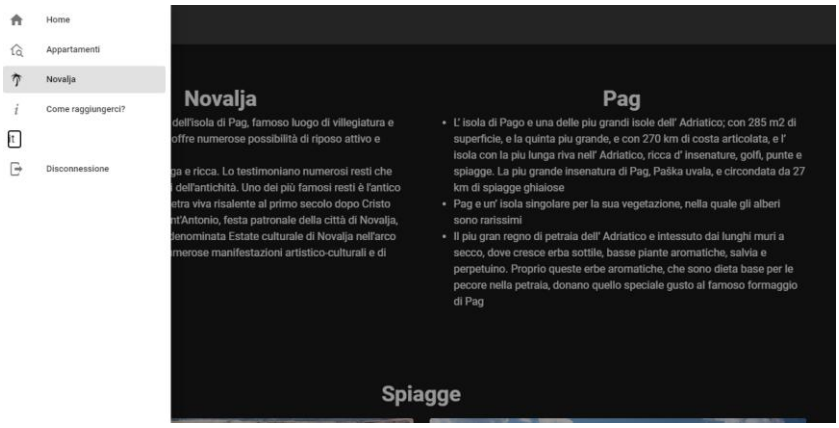
Slika 4.16. *Hrvatska verzija stranice*



Slika 4.17. *Engleska verzija stranice*



Slika 4.18. *Njemačka verzija stranice*



Slika 4.19. *Talijanska verzija stranice*

Slika 4.20. *Slovenska verzija stranice*

Slika 4.21. *Češka verzija stranice*

5. ZAKLJUČAK

Unatoč povećanju broja mobilnih aplikacija, potraga za web stranicama još uvijek je postojana. Izrada web aplikacija pomoću temeljnih tehnologija HTML-a i CSS-a mukotrpan je posao i nerijetko rezultira lošijem dizajnu stranice i performansama stranice. Ukoliko se radi o velikim stranicama, teško je održati kod čitljivim i preglednim. Prvi korak ka boljim stranicama je otkrivanje responzivnih frameworka poput Bootstrapa. Korištenje Bootstrapa uveliko ubrzava izradu web aplikacija i poboljšava njen dizajn. No, gotove UI komponente nisu dovoljne. Iako je bootstrap koristan alat, ukoliko želimo dinamički generirati sadržaj moramo koristiti javascript jezik. Temeljni JavaScript jezik i njegovo pristupanje i baratanje elementima nema najljepšu sintaksu, a nakon dosta izučavanja još uvijek imate osječaj da ništa ne znate. Pomoću bootstrapa i javascripta moguće je izraditi neke jednostavnije aplikacije, no ukoliko želimo skladištiti podatke generirane od strane korisnika, ni to nije dovoljno. Tada u priču ulazi svijet backenda i sve čari koje idu s njim. Tako programmer, nakon 3 godine prolaska kroz trnje dolazi do zvijezda i otkriva frontend framework, nema više izmišljanja tople vode! Zahvaljujući frontend frameworcima poput Vue.js, izrada aplikacija i njeno održavanje nikad nije bila jednostavnija i brža. Uvođenjem paketa poput vuetify-a koji koristi Material Design, korisnicima aplikacije pruža se poznati osjećaj i dizajn aplikacije koji se intuitivno koristi. Ukoliko je aplikaciju potrebno prevesti, nema potrebe duplicirati kod samo radi pružanja stranih prijevoda, Vue.js i18n paket omogućuje jednostavno prevođenje aplikacije. Korištenjem Vuex paket korisnicima se pruža dinamični user-friendly dizajn.

LITERATURA

[1] Booking.com, https://www.booking.com/index.hr.html (stranica posjećena 29.09.2020.)

[2] Airbnb.com, https://hr.airbnb.com/ (stranica posjećena 29.09.2020.)

[3] What is Vue.js?, https://vuejs.org/v2/guide/ (stranica posjećena 19.09.2020.)

[4] Vue.js Guide, https://v1.vuejs.org/guide/overview.html (stranica posjećena 19.09.2020.)

[5] Google Firebase, https://firebase.google.com/ (stranica posjećena 19.09.2020.)

[6] Firebase Realtime Database, https://firebase.google.com/docs/database (stranica posjećena 19.09.2020.)

[7] Firebase Cloud Storage, https://firebase.google.com/docs/storage (stranica posjećena 19.09.2020.)

[8] R., Nixon, **Learning PHP, MySQL, JavaScript, CSS & HTML5, 3rd Edition**, O'Reilly Media Inc., 1005 Gravenstein Highway North, Sebastopol, June 2014

[9] HTML: Hypertext Markup Language, https://developer.mozilla.org/en-US/docs/Web/HTML (stranica posjećena 19.09.2020.)

[10] TECH 101: The ultimate guide to CSS, https://skillcrush.com/blog/css/ (stranica posjećena 19.09.2020.)

SAŽETAK

Cilj završnog rada je bio realizirati modernizaciju web aplikacije koja nije responzivna. Kreirana je respozivna višejezična web aplikacija za iznajmljivanje apartmana koja korisnicima omogućuje uvid u dostupnost svakog apartmana, kao i postavljanje rezervacija za apartman. Za izradu aplikacije korišten je Vue.js framework te neki od njegovih paketa poput Vuex, i18n, VueRouter. Radi ljepšeg dizajna korišten je Vuetify paket koji je kreiran na temelju Bootstrapa, a koristi Material Design uzorke. Pristup Firebase bazi podataka ostvaren je putem Axios http alata. Primjenom aplikacije omogućavaju se razna poboljšanja u vidu olakšavanja posla i skraćenje vremena koje vlasnik mora uložiti da bi iznajmio apartman.

Ključne riječi: Axios, Firebase, I18n, Vue.js, VueRouter, Vuex, Vuetify

ABSTRACT

Multilingual application for renting apartments

The aim of the final work was to realize the modernization of a web application that is not responsive. A responsive multilingual web application for renting apartments has been created, which allows users to see the availability of each apartment, as well as set reservations for the apartment. During the implementation of the application, the Vue.js framework was used, as well as some of its packages such as Vuex, i18n, VueRouter. For a nicer design, the Vuetify package was used, which was created based on Bootstrap, and uses Material Design patterns. Access to the Firebase database was achieved through the Axios http tool. By applying the application, various improvements are enabled in the form of easier work and shortening of the time that the owner must invest in order to rent an apartment.

Keywords: Axios, Firebase, I18n, Vue.js, VueRouter, Vuex, Vuetify

PRILOZI

CD:

- Elektronička verzija rada (dokument u .docx i .pdf formatu)
- Vue.js projekt

ŽIVOTOPIS

Sandro Blavicki rođen je 3. Prosinca 1998. Godine u Slavonskom Brodu. Osnovnu školu je pohađao u Orašju, Bosna i Hercegovina, Montereyu, Kalifornija i završio u Slavonskom Brodu. Nakon završetka osnovne škole upisuje gimnaziju Matija Mesić u Slavonskom Brodu, smjer prirodoslovno-matematička gimnazija, koju završava 2017. godine te u istoj godini upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.

_____

Sandro Blavicki