Izrada progresivnih web aplikacija pomoću biblioteka Vue i Nuxt

Križ, Valentina

Master's thesis / Diplomski rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:168255

Izrada progresivnih web aplikacija pomoću biblioteka Vue i Nuxt

Križ, Valentina

Master's thesis / Diplomski rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:168255

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Valentina Križ

IZRADA PROGRESIVNIH WEB APLIKACIJA POMOĆU BIBLIOTEKA VUE I NUXT

Diplomski rad

Voditelj rada:
izv. prof. dr. sc. Zvonimir Bujanović

Zagreb, ožujak, 2021.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Svima onima koji nikad nisu sumnjali.

Sadržaj

Uvod

Razvojem mobilnog interneta i pametnih telefona raste i popularnost web-aplikacija. Osnovno obilježje web-aplikacija je njihova dostupnost na svim uređajima koji imaju pristup internetu te instaliran web-preglednik. Razvijaju se neovisno o platformi, što znatno smanjuje troškove njihovog razvoja, a budući da ne zahtijevaju instalaciju, web-aplikacije ne zauzimaju prostor na korisnikovom uređaju niti ih treba prilagođavati raznolikim konfiguracijama uređaja koje bi korisnici mogli imati.

Budući da se većina prednosti web-aplikacija odnosi na sami proces njihovog razvoja i distribucije, korisnici i dalje preferiraju nativne aplikacije kojima lako mogu pristupiti s početnog ekrana svojeg uređaja. Prosječnom korisniku aplikacije najbitnija je jednostavnost korištenja, mogućnost primanja obavijesti te mogućnost korištenja aplikacije u slučaju nepovezanosti s internetom, a sve to nude im nativne aplikacije.

Ideja razvoja nativnih aplikacija pomoću web-tehnologija pojavila se još 2007. godine kada je Apple pustio u prodaju svoj prvi iPhone. Prvobitna namjera je bila da se sve aplikacije pišu u HTML-u, no nakon što su shvatili da tehnologija ipak nije dovoljno razvijena, odlučeno je da će se aplikacije i dalje razvijati standardnim tehnologijama.

Više od desetljeća kasnije, web-tehnologije su doživjele značajan napredak. HTML, CSS i JavaScript nude mnogobrojne mogućnosti i gotovo da ne postoje ograničenja u razvoju aplikacija pomoću web-tehnologija. Također se pojavljuju mnogi aplikacijski okviri za razvoj JavaScript aplikacija koji znatno olakšavaju razvoj te tako dodatno smanjuju troškove njihove izrade. Upravo je napredak web-tehnologija otvorio vrata takozvanim progresivnim web-aplikacijama (PWA) koje predstavljaju spoj najboljih obilježja web i nativnih aplikacija.

U prvom dijelu rada pojasnit ćemo pojam PWA, navesti prednosti i mane te objasniti njihove glavne tehničke komponente. U drugom dijelu rada predstavit ćemo Vue.js i NuxtJS, dva aplikacijska okvira koja se mogu koristiti za razvoj PWA. U trećem dijelu ćemo na konkretnom primjeru prikazati kako pomoću tih okvira razviti progresivnu aplikaciju. Konačno, u četvrtom dijelu ćemo predstaviti rezultate analize razvijene aplikacije.

Poglavlje 1

Progresivne web-aplikacije (PWA)

1.1 Što su progresivne web-aplikacije

Progresivne web-aplikacije (PWA) predstavljaju skup strategija, tehnologija i aplikacijskih korisničkih sučelja (API-ja) za izradu modernih web-aplikacija koje korisniku pružaju doživljaj korištenja nativnih, odnosno desktop ili mobilnih aplikacija [18].

Pojam se prvi put pojavljuje 2015. godine u članku Alexa Russella, inženjera Google Chrome-a. Autor dotadašnje pokušaje razvoja nativnih aplikacija pomoću web tehnologijama ne smatra neuspješnima, ali predlaže novi način razmišljanja i korištenja dostupnih tehnologija. Budući da su web-preglednici doživjeli veliki napredak i nude nove mogućnosti, predlaže razvoj web-aplikacija koje će preglednik moći prikazati poput nativnih aplikacija. Popisuje očekivana svojstva takvih aplikacija i daje im naziv progresivne web-aplikacije. Neka od očekivanih svojstava [16] su:

- aplikacija je responzivna;
- aplikacija pruža osjećaj korištenja nativne aplikacije;
- aplikacija radi i pri slaboj povezanosti s internetom ili na sporim mrežama;
- aplikacija nudi određene funkcionalnosti i u slučaju nepovezanosti s internetom;
- aplikacija se može dodati na početni zaslon uređaja;
- aplikacija koristi TLS (Transport Layer Security) protokol.

Iako se riječ "progresivna" interpretira na nekoliko različitih načina, najčešće objašnjenje je da takva aplikacija *progresivno* postaje bolja na modernijim preglednicima, odnosno uređajima. Dok će na uređaju koji koristi stariju verziju preglednika Internet Explorer aplikacija nuditi najosnovnije funkcionalnosti, korisnici modernijih preglednika

uživat će u svim funkcionalnostima aplikacije. Od PWA se očekuje da će ponuditi najbolje moguće korisničko iskustvo s obzirom na korišteni preglednik i prisutnost ili brzinu internetske mreže.

1.2 Prednosti i mane

Progresivne web-aplikacije predstavljaju spoj najboljih karakteristika web i nativnih aplikacija. Dok nam web-aplikacije nude dostupnost na svim uređajima, jednostavnost ažuriranja i njihovog razvoja te mogućnost lakog dijeljenja aplikacije s drugim korisnicima, nativne aplikacije nam nude mogućnost slanja notifikacija, korištenja aplikacije bez povezanosti s internetom i brzog pristupa s početnog zaslona uređaja.

Naravno, PWA imaju i određene mane. Budući da je PWA i dalje web-aplikacija, za razliku od nativne aplikacije ima neka ograničenja na pristup hardveru uređaja. Uz to, mogućnosti aplikacije najviše će ovisiti o web-pregledniku koji korisnik ima instaliranog na svom uređaju, na što programer ne može utjecati.

1.3 Tehničke komponente

Datoteka manifest i service worker, uz HTTPS, predstavljaju ključne tehničke komponente PWA. Radi boljeg razumijevanja aplikacije koju ćemo razviti u nastavku rada, ukratko ćemo objasniti ta dva pojma.

Datoteka manifest

Izrada datoteke manifest je poprilično jednostavna, a upravo ona najviše utječe na nativni izgled svake PWA. Datoteka manifest je `JSON` datoteka koja pruža sve informacije o aplikaciji potrebne za instalaciju na početni zaslon uređaja i što bogatije korisničko iskustvo. U datoteci može biti zadano ime i opis aplikacije, pozadinska boja i ikone raznih dimenzija za prikaz na početnom zaslonu, pa čak i poveznica na nativnu verziju aplikacije ako ona postoji.

Primjer datoteke manifest:

```
{
  "name": "Tracking App",
  "short_name": "Tracking App",
  "description": "## Build Setup",
  "icons": [
    {
      "src": "/_nuxt/icons/icon_64x64.3dfe72.png",
```

```
      "sizes": "64x64",
      "type": "image/png",
      "purpose": "any maskable"
    },
    {
      "src": "/_nuxt/icons/icon_120x120.3dfe72.png",
      "sizes": "120x120",
      "type": "image/png",
      "purpose": "any maskable"
    },
    ... // popis ikona raznih dimenzija
    {
      "src": "/_nuxt/icons/icon_512x512.3dfe72.png",
      "sizes": "512x512",
      "type": "image/png",
      "purpose": "any maskable"
    }
  ],
  "start_url": "/?standalone=true",
  "display": "standalone",
  "background_color": "#eee",
  "theme_color": "#160",
  "lang": "en-EN"
}
```

Datoteka manifest se lako uključuje u svaku web-stranicu pomoću tag-a <link>:

```
<link rel="manifest" href="manifest.json">
```

Nakon uspješnog učitavanja datoteke manifest, ukoliko web-preglednik ima podršku za PWA, u adresnoj traci prikazat će se gumb za instalaciju aplikacije na uređaj. Većina preglednika također će u određenom trenutku korisniku prikazati obavijest da se aplikacija može dodati na zaslon uređaja.

Noviji preglednici na temelju informacija u datoteci manifest (ime aplikacije, boja pozadine i ikona) automatski generiraju i takozvani *splash screen* koji se umjesto prazne stranice prikazuje prilikom pokretanja aplikacije s početnog zaslona uređaja, što dodatno pridonosi osjećaju korištenja nativne aplikacije [8].

Service worker

Korištenjem datoteke manifest naša web-aplikacija će poprimiti izgled nativne aplikacije, no u imitaciji funkcionalnosti nativne aplikacije najviše će nam pomoći service wor-

ker. Service worker je JavaScript datoteka koja se izvršava odvojeno od glavne dretve preglednika, a zadužena je za presretanje svih mrežnih zahtjeva, predmemoriranje i dohvaćanje podataka iz predmemorije te slanje takozvanih push notifikacija (poruka koje se isporučuju na klijentov uređaj, a da pritom klijent to ne mora zahtijevati). Budući da se izvršava u zasebnoj dretvi, ne može direktno pristupiti elementima web-stranice (DOM-a) niti `localStorage`-u. Korištenjem predmemorije za spremanje mrežnih zahtjeva može se postići offline način rada aplikacije, a korištenjem postojećih API-ja mogu se implementirati i ostale funkcionalnosti koje nude nativne aplikacije. Primjer takvih API-ja su Notification API i Push API koje možemo koristiti za slanje notifikacija korisniku i za pretplatu na razne servise za slanje push poruka [17].

Valja napomenuti kako web-aplikacija ne mora nužno pružati sve navedene funkcionalnost nativne aplikacije kako bi se smatrala progresivnom, no mora koristiti barem minimalnu service worker datoteku. Za uspješno korištenje service worker-a u aplikaciji potrebno ga je registrirati, instalirati i aktivirati.

Primjer service worker datoteke `sw.js` koja pruža minimalnu offline funkcionalnost predmemoriranjem `index.html` stranice:

```
1  self.addEventListener('install', (event) => {
2    event.waitUntil(
3      caches.open('sw-cache').then(function(cache) {
4        return cache.add('index.html');
5      })
6    );
7  });
8
9  self.addEventListener('activate', (event) => {
10   console.log('Service worker is now ready to receive events.');
11 });
12
13 self.addEventListener('fetch', (event) => {
14   event.respondWith(
15     caches.match(event.request).then(function(response) {
16       return response || fetch(event.request);
17     })
18   );
19 });
```

U service worker-u se možemo pretplatiti na primanje raznih događaja emitiranih od strane preglednika. Nakon uspješnog preuzimanja i instalacije service worker-a, preglednik emitira događaj `install`. U prvoj liniji našeg primjera možemo vidjeti kako se pretplatiti na taj događaj i kako u tom slučaju otvoriti predmemoriju pod imenom `sw-cache` i spre-

miti datoteku `index.html` u nju. Nakon uspješne instalacije, pokreće se proces aktivacije service worker-a. U slučaju uspješne aktivacije preglednik emitira događaj `activate` koji označava da je service worker spreman za korištenje. U ovom koraku preporuča se čišćenje predmemorije nastale u eventualnoj starijoj verziji service worker-a, no u našem primjeru samo ispisujemo prigodnu poruku u konzolu preglednika. Konačno, dodajemo pretplatu na događaj `fetch` koji se emitira u slučaju slanja mrežnog zahtjeva. U liniji 15 provjeravamo nalazi li se tražena datoteka u predmemoriji i vraćamo predmemoriranu verziju datoteke ako ona postoji. Ukoliko se datoteka ne nalazi u predmemoriji, service worker propušta originalni zahtjev i vraća pripadni odgovor. U linijama 3 i 15 koristimo tzv. *promise* objekt koji predstavlja eventualni završetak ili neuspjeh neke asinkrone operacije. U metodi `then` zadajemo metodu koja se treba izvršiti u slučaju uspješnog završetka operacije. Također je moguće zadati metodu koja se izvršava u slučaju neuspjeha ili onu koja se izvršava neovisno o ishodu, korištenjem metoda `catch` i `finally`.

Područje rada service worker-a ovisi o lokaciji datoteke. Ukoliko želimo omogućiti presretanje svih mrežnih zahtjeva, datoteku ćemo smjestiti u korijenski direktorij web-aplikacije. Nakon toga potrebno je registrirati service worker-a unutar `index.html`:

```
if('serviceWorker' in navigator) {
    navigator.serviceWorker.register('sw.js');
};
```

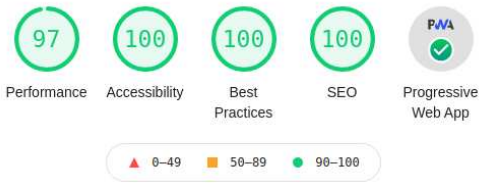
Prije pozivanja registracije potrebno je provjeriti sadrži li objekt `navigator` svojstvo `serviceWorker`, odnosno podržava li preglednik korištenje service worker-a.

1.4 Lighthouse

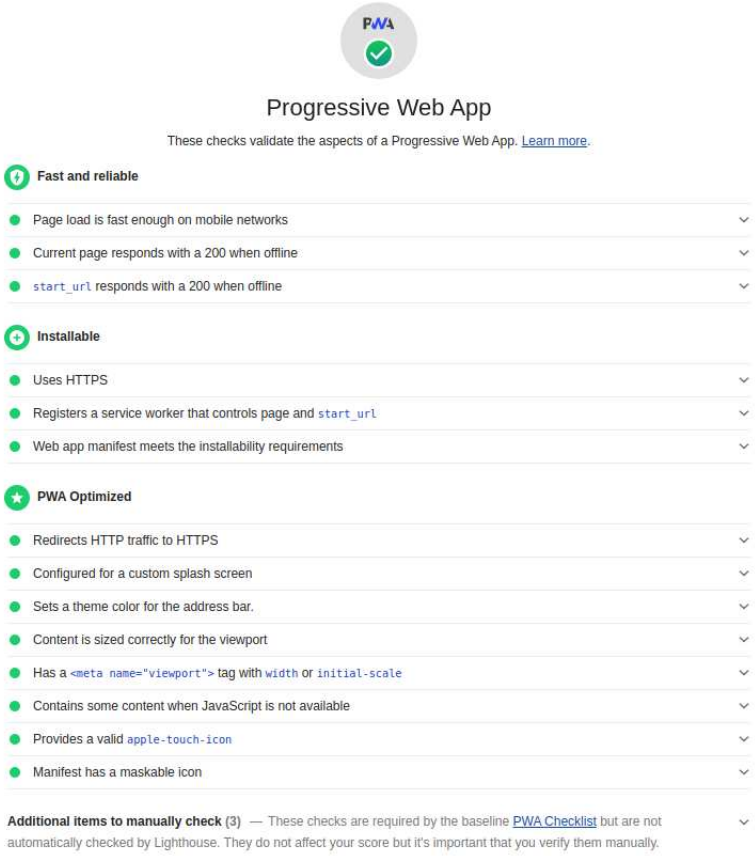
Za provjeru zadovoljava li stranica PWA kriterije često se koristi alat Lighthouse. Lighthouse je alat otvorenog koda (eng. *open-source*) za poboljšanje kvalitete web-stranica, a uz provjeru PWA kriterija nudi mogućnost provjere performansi, pristupačnosti, korištenja najboljih praksi te optimiziranosti stranice za pretraživače (eng. *search engine optimization* - SEO) [6].

Može se koristiti na više načina, iz komandne linije, kao modul za Node.js ili unutar Chrome DevTools. Mi ćemo ga koristiti unutar Chrome DevTools jer ne zahtjeva dodatnu instalaciju, a potpuni izvještaj o kvaliteti naše web-stranice dobivamo u nekoliko klikova mišem.

Na slikama 1.1 i 1.2 vidimo izvješće o kvaliteti web-stranice izrađene kao primjer PWA u nastavku rada. Detaljniju analizu rezultata napravit ćemo u poglavlju 4.



Slika 1.1: Sažetak rezultata koje daje Lighthouse



Slika 1.2: PWA kriteriji

Poglavlje 2

Alati za razvoj progresivnih web-aplikacija

2.1 Uvod

Rastom popularnosti PWA raste i broj tehnologija i aplikacijskih okvira za njihov razvoj. Gotovo svi aplikacijski okviri za razvoj web-aplikacija nude alate za razvoj PWA ili trenutno rade na njihovom razvoju. U ovome radu posvetit ćemo se bibliotekama Vue.js i NuxtJS i pokazati kako pomoću njih možemo lako izraditi web-aplikaciju koja zadovoljava kriterije PWA.

2.2 Vue.js

Vue.js je progresivni JavaScript aplikacijski okvir za izradu korisničkih sučelja (tzv. frontend). Pored drugih frontend aplikacijskih okvira ističe se svojom brzinom, malom veličinom i detaljnom dokumentacijom. Vue.js je u svojoj srži fokusiran na prezentacijski sloj aplikacije, te ga je samim time lako integrirati s drugim bibliotekama i postojećim projektima. Koristeći Vue.js u kombinaciji s drugim bibliotekama dobivamo moćan alat za izradu takozvanih *single page* aplikacija (SPA). SPA aplikacije su web-aplikacije koje se sastoje od jedne stranice po kojoj korisnik dinamički navigira i mijenja njen sadržaj. Lokacija web-aplikacije se gotovo nikada ne osvježava, a osjećaj prave navigacije postiže se pomoću biblioteke za usmjeravanje (eng. *routing*).

Glavna obilježja Vue.js-a su:

- korištenje virtualnog `DOM`-a i HTML predložaka;
- komponente i direktive;

- animacije i tranzicije;
- upravljanje stanjima i usmjeravanje.

U primjerima koji slijede te u aplikaciji koju smo razvili u nastavku rada koristili smo Vue.js verziju 2.6.12.

Komponente

Komponente su Vue instance predviđene za višestruko korištenje unutar aplikacije. Svaka komponenta se sastoji od tri dijela - HTML predloška, JavaScript koda i CSS stila. Izgled i najbitnije mogućnosti Vue komponenti prikazujemo na primjeru:

```
<template>
  <div class="background-blue padding-15">
    <p v-if="userMessage">
      User message: {{ userMessage }}
    </p>
    <p v-else>
      No message
    </p>

    <p class="padding-15">
      <userComponent username="marc" v-on:messageSent="getMessage" />
    </p>
  </div>
</template>

<script>

import UserComponent from './User'

export default {
  name: 'ParentComponent',

  components: {
    UserComponent
  },

  data () {
    return {
      userMessage: ''
```

```
    }
  },

  methods: {
    getMessage (event) {
      this.userMessage = event.userMessage
    }
  }
}

</script>

<style scoped>

.background-blue {
  background-color: blue;
}

.padding-15 {
  padding: 15px;
}

</style>
```

U primjeru vidimo kako stvoriti komponentu pod nazivom `ParentComponent` koja unutar svog HTML predloška dodatno koristi komponentu `UserComponent`. Sve komponente koje želimo koristiti unutar predloška potrebno je registrirati pomoću svojstva `components`. Komponentu zatim možemo dodati u predložak pomoću tag-a koji odgovara imenu komponente. U komponenti `ParentComponent` koristimo varijablu `userMessage` koju je potrebno definirati unutar metode `data`. Pomoću direktiva `v-if` i `v-else` možemo uvjetno dodavati elemente u `DOM`. Ovisno o vrijednosti varijable `userMessage` prikazujemo paragraf s porukom ili paragraf s pojašnjenjem da poruka ne postoji. Sadržaj poruke spremljen u varijabli `userMessage` možemo prikazati unutar paragrafa interpolacijom stringa koju označavamo dvostrukim vitičastim zagradama. U slučaju promjene vrijednosti varijable `userMessage`, automatski će se promijeniti i sadržaj paragrafa s porukom. Vrijednost varijable mijenjamo prilikom primanja događaja `messageSent` koji se emitira unutar komponente `UserComponent`, a na koji smo se pretplatili korištenjem direktive `v-on`. Pretplatom na događaje možemo primati podatke iz komponente koja se nalazi unutar roditeljske komponente. Ukoliko želimo ostvariti komunikaciju u drugom smjeru, koristimo svojstvo `props` unutar komponente `UserComponent`:

```
<template>
  <div class="background-white padding-15">
    <p class="text-blue">
      Hi {{ username }}!
    </p>

    <p>
      Please write a message for your parent component.
    </p>

    <input v-model="message" type="text">
    <button v-on:click="onButtonClick">
      Emit message
    </button>
  </div>
</template>

<script>

export default {
  name: 'UserComponent',

  props: {
    username: {
      type: String,
      required: true,
      default: null
    }
  },

  data () {
    return {
      message: ''
    }
  },

  methods: {
    onButtonClick () {
      this.$emit('messageSent', {
        userMessage: this.message
      })
    }
```

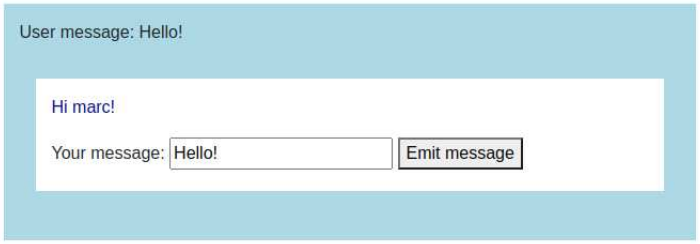
```
  }
}

</script>

<style scoped>

.text-blue {
  color: blue;
}

.background-white {
  background-color: white;
}

.padding-15 {
  padding: 15px;
}

</style>
```

Podatak pod nazivom `username` sada možemo poslati u komponentu `UserComponent` i koristiti ga kao varijablu unutar te komponente. Događaj `messageSent` emitiramo nakon klika na gumb pomoću metode `this.$emit` kojoj predajemo ime događaja i vrijednost koju želimo poslati.

Prethodno opisani način komunikacije između dvije komponente naziva se *two-way data binding* i predstavlja najčešći način komunikacije unutar jednostavnijih aplikacija. Ulančavanjem događaja i `props` vrijednosti možemo ostvariti komunikaciju i između komponenti koje se ne nalaze jedna unutar druge.

Na slikama 2.1 i 2.2 vidimo izgled komponente `ParentComponent` prije i nakon unosa poruke te klika na gumb za emitiranje događaja unutar komponente `UserComponent`.

Slika 2.1: Komponenta `ParentComponent` prije unosa poruke



Slika 2.2: Komponenta `ParentComponent` nakon unosa poruke

Vue Router

Zbog lake integracije najčešći izbor biblioteke za usmjeravanje unutar aplikacije je službena Vue biblioteka Vue Router. Lako se dodaje u postojeću Vue aplikaciju pomoću nekog od upravitelja paketima kao što je `npm`. Sve željene putanje potrebno je popisati u niz koji se predaje kao parametar pri registraciji router-a u Vue instanci. Primjer takvog niza:

```
const routes = [
  {
    path: '/',
    name: 'home',
    component: Home
  },
  {
    path: '/about',
    name: 'about',
    component: About
  },
  {
    path: '/user/:userName',
    name: 'user',
    component: User
  }
]
```

Kreiranje instance router-a i registracija u Vue instanci:

```
const router = new VueRouter({
  routes
})

const app = new Vue({
```

```
  router
}).$mount('#app')
```

Uz konfiguraciju iz primjera, Vue Router će generirati tri putanje. U slučaju pristupa početnoj stranici aplikacije prikazat će se komponenta `Home`, a u slučaju pristupa putanji */about* prikazat će se komponenta `About`. U definiciji putanja možemo dodati i parametre koje je potrebno označiti dvotočkom. U našem primjeru Vue Router će za svaku putanju koja počinje s */user/* prikazati komponentu `User`, a sve što slijedi nakon toga proslijediti će kao parametar putanje (korisničko ime).

Vuex

Iako se za komunikaciju između komponenti mogu koristiti već spomenute Vue funkcionalnosti (`props`, događaji ili tzv. *event bus*), u većim aplikacijama se javlja potreba za centraliziranim skladištem čijim se korištenjem izbjegava nepotrebna komunikacija između više komponenti. Službena biblioteka za upravljanje stanjima naziva se Vuex i također se lako instalira pomoću `npm`-a. Skladište se sastoji od stanja (eng. *state*), gettera, mutacija (eng. *mutations*) i akcija (eng. *actions*). Stanje možemo shvatiti kao globalni objekt koji je dostupan u svim komponentama unutar aplikacije. Mutacije se koriste za promjenu stanja, a ukoliko uz promjenu stanja želimo izvršiti i neku asinkronu metodu, tada moramo koristiti akcije jer one mogu biti i asinkrone.

Preporuča se modularizacija skladišta - raspoređivanje stanja, mutacija i akcija u više datoteka, ovisno o funkcionalnosti za koju su vezane, posebice u slučaju skladišta s većim stanjem i mnogobrojnim mutacijama i akcijama.

2.3 NuxtJS

NuxtJS je aplikacijski okvir u sklopu ekosustava Node.js za izradu Vue aplikacija. Aplikacije se mogu razvijati u dva različita modula: univerzalnom (SSR) ili single page (SPA). Puna snaga Nuxt-a leži u univerzalnom načinu rada koji omogućava izvršavanje na serveru (eng. *Server Side Rendering*), no Nuxt olakšava i razvoj aplikacija koje se izvršavaju samo na klijentu u SPA modulu. U ovom radu posvetit ćemo se upravo SPA modulu jer nam za razvoj naše aplikacije nije potreban SSR, već je naglasak na izradi PWA za koje je bitno da mogu funkcionirati i offline, tj. u slučaju nedostupnosti servera. Osim SSR-a, neke od prednosti Nuxt-a koje dolaze do izražaja i pri izradi SPA aplikacija su:

- automatsko prevođenje ES6+ koda u JavaScript kod koji se može izvršavati i na starijim preglednicima;
- pisanje CSS-a s pretprocesorom (Sass, Less, Stylus);

- jednostavna nadogradnja pomoću proširenja i modula;
- upravljanje elementom <head>;
- podjela većih datoteka u više manjih i minifikacija koda pomoću alata webpack;
- automatska konfiguracija Vue Router-a i Vuex-a;
- jasno definirana struktura projekta.

Struktura Nuxt projekta

Programer pri razvoju Nuxt aplikacije ne mora brinuti o strukturi projekta jer je ona strogo definirana. Svaki novostvoreni Nuxt projekt se sastoji od sljedećih direktorija i datoteka:

- direktorij `/assets/` služi za spremanje slika, fontova te Less, Stylus ili Sass datoteka koje će se prevesti pomoću webpack-a;
- direktorij `/static/` služi za pohranu datoteka koje se ne mogu ili ne trebaju prevoditi pomoću webpack-a, kao što su `.ico` datoteke;
- direktorij `/pages/` sadrži sve tzv. view-ove i putanje aplikacije (u obliku Vue datoteka) na temelju kojih se automatski generira router;
- direktorij `/layouts/` sadrži sve tzv. layout-e aplikacije (uključujući i automatski generirani layout `default.vue`);
- direktorij `/components/` služi za pohranu svih Vue komponenti;
- direktorij `/plugins/` služi za pohranu proširenja (eng. *plugins*);
- direktorij `/store/` sadrži sve Vuex datoteke;
- direktorij `/middleware/` sadrži JavaScript funkcije koje želimo izvršiti prije prikazivanja stranice ili grupe stranica;
- datoteka `package.json` služi za dodavanje ovisnosti (eng. *dependencies*) i skripti;
- datoteka `nuxt.config.js` služi za konfiguraciju aplikacije (npr. meta tag-ovi, naziv aplikacije).

Konfiguracija Vue Router-a i Vuex-a

Nuxt projekt dolazi s već instaliranim i spremnim za uporabu bibliotekama Vue Router i Vuex. Biblioteke nije potrebno dodatno registrirati za uporabu niti ručno konfigurirati. Kao što smo već spomenuli, router se automatski generira na temelju datoteka koje se nalaze u direktoriju `/pages/`. Podržano je korištenje parametara, tj. dinamičkih putanja i kreiranje ugniježđenih putanja. Ukoliko imamo sljedeću strukturu:

```
pages/
--| about.vue
--| users /
----| _userName.vue
--| index.vue
```

generirat će se router sa sljedećom konfiguracijom:

```
const routes = [
  {
    path: '/',
    name: 'index',
    component: 'pages/index.vue'
  },
  {
    path: '/about',
    name: 'about',
    component: 'pages/about.vue'
  },
  {
    path: '/user/:userName',
    name: 'users-userName',
    component: 'pages/users/_userName.vue'
  }
]
```

Također je moguće dodati stranicu koja će se prikazivati u slučaju neispravne putanje. Dovoljno je u direktorij `/pages/` dodati datoteku naziva `_.vue`.

Valja napomenuti kako je Nuxt stranica upravo Vue komponenta uz dodatak atributa i metoda specifičnih za Nuxt (npr. metoda `asyncData`, metoda `fetch`, atributi `layout`, `loading` ili `middleware`).

Na sličan način automatski se generira i centralno skladište. Dovoljno je u direktorij `/store/` dodati barem jednu JavaScript datoteku iz koje stanje izvozimo kao funkciju, a gettere, mutacije i akcije kao objekte. U Nuxt-u je obavezna modularizacija skladišta, no

nije potrebno sve module objediniti u jednoj datoteci, to će Nuxt učiniti umjesto nas. U slučaju većih datoteka preporuča se podjela modula na više datoteka - `state.js`, `getters.js`, `mutations.js` i `actions.js`. Također, nije potrebno objediniti sve datoteke u jednu, već ih je dovoljno smjestiti u jedan direktorij s imenom modula. Primjer strukture direktorija `/store/`:

```
store/
--| module1 /
----| state.js
----| mutations.js
----| actions.js
--| module2 /
----| state.js
----| mutations.js
----| actions.js
```

Primjer datoteke `state.js` u modulu `module1`:

```
export const state = () => ({
  userId: null,
  userName: false
})
```

Varijable `userId` i `userName` bit će dostupne unutar cijele aplikacije, a unutar komponente mogu se koristiti pomoću svojstva `computed`:

```
computed: {
  userId () {
    return this.$store.state.module1.userId
  }
}
```

Primjer pripadne datoteke `mutations.js`:

```
export const mutations = {
  SET_USER_ID (state, payload) {
    state.userId = payload
  },

  SET_USER_NAME (state, payload) {
    state.userName = payload
  }
}
```

Mutacije se također mogu koristiti unutar cijele aplikacije, a u komponentama poput standardnih metoda pomoću svojstva `methods`:

```
methods: {
  setUserName (value) {
    this.$store.commit('module1/SET_USER_NAME', value)
  }
}
```

Na sličan način definiramo i akcije te ih koristimo unutar aplikacije.

Korištenje proširenja i modula

Nuxt je izgrađen s modularnom arhitekturom, što znači da pruža mogućnost lake nadogradnje uporabom raznih proširenja i modula. Nuxt i Vue nude službena proširenja i module, no tu je također i pozamašan broj onih razvijenih od strane neovisnih programera.

Proširenje je globalna JavaScript funkcija enkapsulirana u `.js` datoteku tako da se može instalirati u Vue ili Nuxt aplikaciju. Za korištenje gotovog proširenja u Nuxt projektu potrebno ga je instalirati pomoću `npm`-a, a zatim ga pomoću `Vue.use(pluginName)` uvesti u Vue instancu te dodati ime proširenja u opciju `plugins` u `nuxt.config.json`. U slučaju pisanja vlastitog proširenja, dovoljno je datoteku koja sadrži proširenje dodati u direktorij `/plugins/` te ga kao i ranije registrirati za korištenje. Unutar Nuxt projekta možemo koristiti mnogobrojna Vue proširenja kao što su `vue-notifications` i `vue-loaders`.

Moduli su također JavaScript funkcije enkapsulirane u `.js` datoteke, ali se one izvršavaju prilikom pokretanja Nuxt aplikacije, odnosno prije pozivanja Vue instance, proširenja i globalnih funkcija. Zbog vremena njihovog pozivanja module možemo koristiti za dodavanje CSS biblioteka, konfiguraciju webpack loader-a i obavljanje ostalih zadataka koji su potrebni za uspješan rad aplikacije. U slučaju korištenja gotovog modula, dovoljno je isti instalirati pomoću `npm`-a i njegovo ime navesti u opciji `modules` u datoteci `nuxt.config.json`. Ukoliko želimo napisati vlastiti modul, potrebno je datoteku s njegovim sadržajem smjestiti u direktorij `/modules/` i dodati putanju do datoteke u opciju `modules`. Primjer modula je `@nuxtjs/pwa` koji ćemo koristiti za izradu PWA u nastavku rada.

Poglavlje 3

Aplikacija Tracking App

3.1 Opis aplikacije

Aplikacija Tracking App korisnicima nudi mogućnost praćenja i analize svojeg kretanja. Prije korištenja aplikacije korisnik se mora prijaviti pomoću e-mail adrese i lozinke, a moguća je i registracija novih korisnika.

Nakon uspješne prijave, na početnom zaslonu prikazuje se prikladna pozdravna poruka i statistika kretanja za tekući tjedan i mjesec (ukupan broj zapisa, prijeđena udaljenost, utrošeno vrijeme, promjena nadmorske visine te staza s najbržom prosječnom brzinom). Također je moguć pregled statistike za proizvoljan tjedan ili mjesec.

Korisnik nove informacije o kretanju unosi učitavanjem `.gpx` datoteka. Datoteke `gpx` formata moguće je preuzeti s bilo kojeg uređaja koji ima mogućnost praćenja lokacije korisnika poput pametnog telefona ili pametnog sata. Također, postoje aplikacije za njihovo generiranje ručnim označavanjem položaja na karti.

Primjer `.gpx` datoteke:

```
<?xml version="1.0"?>
<gpx version="1.1" creator="gpxgenerator.com">
    <trk>
        <trkpt lat="45.7777051147612" lon="15.981758654128617">
            <ele>111.80</ele>
            <time>2020-12-17T22:32:54Z</time>
        </trkpt>
        <trkpt lat="45.77776263508261" lon="15.983850283982934">
            <ele>111.91</ele>
            <time>2020-12-17T22:36:08Z</time>
        </trkpt>
        <trkpt lat="45.77761297978649" lon="15.98668269670266">
```

```
            <ele>114.48</ele>
            <time>2020-12-17T22:40:32Z</time>
        </trkpt>
    </trk>
</gpx>
```

Korisniku se prilikom učitavanja datoteke na karti prikazuje pregled kretanja, a potom je potrebno unijeti ime staze i datum nastanka ukoliko on nije zapisan u samoj datoteci. Datoteka se zatim sprema u Firebase Storage kako bi ju korisnik kasnije ponovno mogao pregledati.

Nadalje, korisniku se nudi pregled svih zapisa o kretanju po mjesecima uz mogućnost brisanja i detaljnog pregleda svakog od zapisa. U detaljnom pregledu zapisa na karti se prikazuje staza i ispisuju se informacije o vremenima početka i završetka kretanja, prijeđenoj udaljenosti, prosječnoj brzini i promjeni nadmorske visine.

Za svakog korisnika prati se ukupna prijeđena udaljenost, ukupno vrijeme, ukupna promjena nadmorske visine te prosječna brzina. Dostupna je rang lista korisnika po svakoj od tih kategorija, a korisniku s najboljim rezultatom u pojedinoj kategoriji se na početnoj stranici prikazuje prigodni trofej.

Ukoliko korisnik ima ovlasti administratora prikazuje mu se i lista svih korisnika aplikacije te mogućnost uvida u zapise o kretanju svakog od njih.

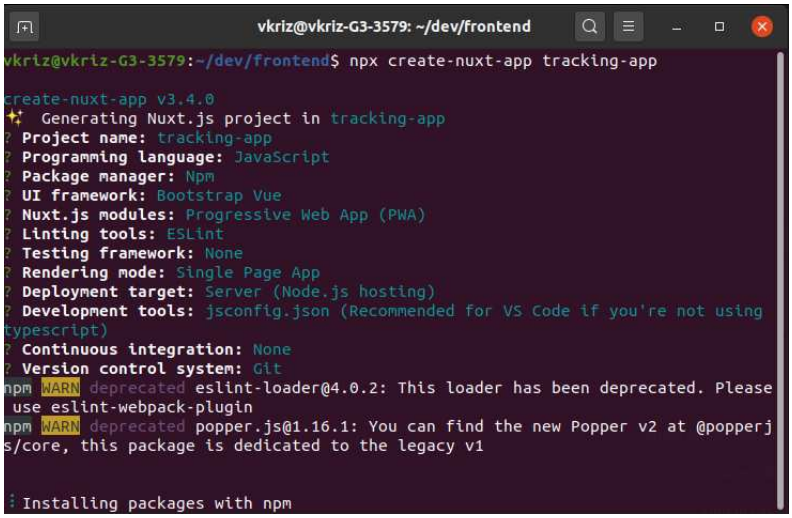
Izgled aplikacije je prilagođen za korištenje na ekranima svih veličina, od stolnih računala do mobilnih uređaja. Aplikacija se može dodati na početni zaslon uređaja, radi pri slaboj povezanosti ili bez pristupa internetu te nudi mogućnost slanja push obavijesti. Budući da se nalazi na Firebase poslužitelju, aplikacija koristi HTTPS.

3.2 Kreiranje Nuxt projekta

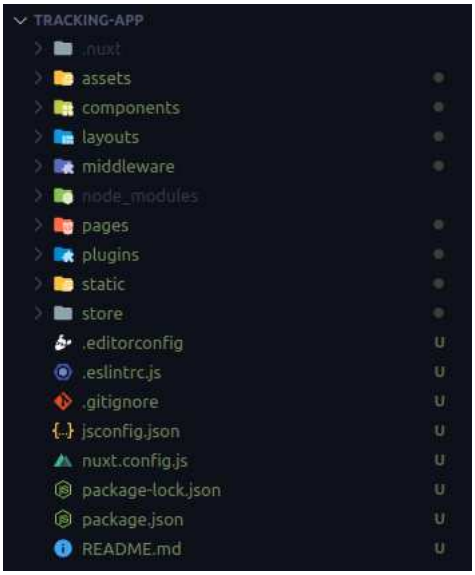
Za jednostavno kreiranje Nuxt projekta moguće je koristiti alat `npx` za izvršavanje `npm` paketa. Dovoljno je pokrenuti naredbu `npx create-nuxt-app <project-name>` i pratiti čarobnjak za izradu projekta koji vidimo na slici 3.1. Nakon odabira svih opcija pokreće se instalacija svih potrebnih paketa i unutar nekoliko minuta imamo Nuxt aplikaciju spremnu za daljnji razvoj. Na slici 3.2 vidimo strukturu stvorenog projekta koja odgovara onoj opisanoj u poglavlju 2.3.

Aplikaciju možemo pokrenuti u dva različita modula - *development* i *production*, a u oba slučaja aplikacija će biti dostupna na adresi `http://localhost:3000`. Aplikaciju u *development* modulu uz tzv. *hot reloading* pokrećemo pomoću naredbe `npm run dev`. *Hot reloading* omogućava da sve promjene u kodu budu vidljive bez ponovnog pokretanja ili učitavanja aplikacije. Prije pokretanja aplikacije u *production* modulu potrebno je pokrenuti naredbu `npm run build` koja će stvoriti direktorij `.nuxt`. U njemu će se nalaziti

sav kod potreban za deploy (postavljanje aplikacije na poslužitelj na kojem će biti dostupna), a naredbom `npm run start` taj kod će se pokrenuti i vidjet ćemo produkcijsku verziju aplikacije.



Slika 3.1: Kreiranje Nuxt aplikacije



Slika 3.2: Struktura Nuxt projekta

3.3 Pomoćne biblioteke

`nuxt-leaflet` i `leaflet-gpx`

Leaflet je JavaScript biblioteka otvorenog koda za prikaz interaktivnih karata. Zbog male veličine i vrhunske dokumentacije, idealno je rješenje za rad s kartama u našoj aplikaciji [5]. Koristimo ga pomoću modula `nuxt-leaflet` koji nudi gotove Vue komponente za korištenje unutar Nuxt aplikacije [11].

Sama biblioteka Leaflet nudi osnovne mogućnosti, kao što su prikaz karte, postavljanje markera i dodavanje slojeva, no lako se nadograđuje korištenjem drugih proširenja i modula. U našoj aplikaciji koristimo proširenje `leaflet-gpx` koje nudi parsiranje i analizu `gpx` datoteka i njihov prikaz u obliku sloja na Leaflet karti [4].

Biblioteke instaliramo koristeći naredbe `npm install nuxt-leaflet` te `npm install leaflet-gpx`. Prije korištenja biblioteke `leaflet-gpx`, istu je potrebno učitati unutar aplikacije. To činimo u komponenti `Map` unutar metode `mounted` koja se poziva u trenutku kad je `DOM` koji komponenta generira spreman za korištenje:

```
mounted () {
  this.L = require('leaflet-gpx')
  this.initLoader()
}
```

Leaflet kartu s besplatnim slojem OpenStreetMap [15] lako prikazujemo pomoću komponenti `LMap` i `LTileLayer` iz modula `nuxt-leaflet`:

```
<div id="map-wrap" class="map-wrap mt-3">
  <l-map :zoom="13" :center="[47.313, -1.319]" v-on:ready="onReady">
    <l-tile-layer url="http://{s}.tile.osm.org/{z}/{x}/{y}.png" />
  </l-map>
</div>
```

Komponenti `Map` pomoću opcije `props` predajemo `gpx` datoteku koju treba parsirati i prikazati ju kao sloj na karti. U metodi `initLoader` inicijaliziramo `FileReader` za čitanje datoteke te dodajemo pretplatu na njegov događaj `onload` koji se emitira u trenutku uspješnog učitavanja datoteke. U liniji 11 pomoću objekta `GPX` stvaramo novi Leaflet sloj na temelju podataka iz učitane datoteke, a u liniji 42 taj sloj dodajemo na kartu koju smo prethodno stvorili pomoću komponentne `LMap`. Objekt `GPX` nudi događaj `loaded` pomoću kojeg možemo dohvatiti izračunate podatke o kretanju koristeći odgovarajuće metode. U liniji 32 emitiramo događaj `calculated` kako bi izračunate podatke mogli spremiti u meta podatke datoteke prilikom njenog spremanja u Firebase Storage u komponenti `FileUpload`.

```
1  initLoader () {
2    this.loader = new FileReader()
```

```
3     this.loader.onload = (loadEvent) => {
4       if (loadEvent.target.readyState !== 2) {
5         return
6       }
7       if (loadEvent.target.error) {
8         return
9       }
10      const gpx = loadEvent.target.result
11      this.gpxLayer = new this.L.GPX(gpx,
12        {
13          async: true,
14          marker_options: {
15            startIconUrl: '../marker.png',
16            endIconUrl: '../marker.png',
17            shadowUrl: null,
18            wptIconUrls: {
19              '': '../marker.png'
20            }
21          }
22        })
23        .on('loaded', (e) => {
24          this.map.fitBounds(e.target.getBounds())
25          this.startTime = e.target.get_start_time()
26          this.endTime = e.target.get_end_time()
27          this.totalTime = e.target.get_total_time()
28          this.distance = e.target.get_distance()
29          this.elevationGain = e.target.get_elevation_gain()
30          this.speed = e.target.get_total_speed()
31
32          this.$emit('calculated', {
33            customMetadata: {
34              startTime: this.startTime,
35              totalTime: this.totalTime,
36              distance: this.distance,
37              elevationGain: this.elevationGain,
38              speed: this.speed
39            }
40          })
41        })
42        .addTo(this.map)
43    }
44  }
```

Modul @nuxtjs/firebase

Firebase je Google-ova platforma za razvoj mobilnih i web-aplikacija. Nudi mnogobrojne mogućnosti, uključujući autentikaciju, korištenje baze podataka, skladištenje datoteka te hosting (posluživanje web-aplikacije) [1].

Dodavanje i upravljanje aplikacijom je jednostavno zahvaljujući Firebase konzoli kojoj pristupamo iz preglednika. Dovoljno je prijaviti se svojim Google računom, kliknuti na izradu novog projekta i odabrati funkcionalnosti koje želimo koristiti. U našoj aplikaciji koristit ćemo autentikaciju korisnika, bazu podataka Realtime Database za spremanje osnovnih informacija o korisnicima te Storage za spremanje `gpx` datoteka učitanih od strane korisnika.

Nuxt Firebase je modul za integraciju Firebase-a u Nuxt aplikaciju. Instalacija i postavljanje su jednostavni, a svim metodama koje modul nudi pristupamo pomoću objekta `this.$fire`.

Uz dodavanje vrijednosti `@nuxtjs/firebase` u opciju `modules` unutar datoteke `nuxt.config.js`, potrebno je u istu datoteku dodati i opciju `firebase` sa željenim postavkama:

```
1  firebase: {
2    config: firebaseConfig,
3    lazy: true,
4    services: {
5      auth: true,
6      storage: true,
7      database: true
8    }
9  },
```

U liniji 2 postavljamo konfiguraciju za povezivanje naše aplikacije s postojećim projektom u Firebase konzoli. Konfiguraciju čitamo iz posebne datoteke `firebaseConfig` koju možemo preuzeti iz Firebase konzole:

```
export const firebaseConfig = {
  apiKey: 'api-key',
  authDomain: 'trackingapp-e6315.firebaseapp.com',
  projectId: 'trackingapp-e6315',
  storageBucket: 'trackingapp-e6315.appspot.com',
  messagingSenderId: '283374867098',
  appId: '1:283374867098:web:ac2693e727a4c0803d8523',
  measurementId: 'G-YLL2ZZM2KN'
}
```

U liniji 3 postavljamo vrijednost opcije `lazy` na `true`. Na taj način modulu govorimo da servise ne želimo učitati prilikom učitavanja aplikacije, već želimo manualno pokrenuti njihovo učitavanje. Razlog tomu je konflikt koji nastaje prilikom istovremenog korištenja modula Workbox unutar modula `@nuxt/pwa` te Firebase servisa za autentikaciju. Oba modula generiraju service worker datoteke koje nisu usklađene za zajedničko korištenje te dolazi do ignoriranja jedne od njih. Ukoliko ne koristimo service worker servisa za autentikaciju, prilikom pokretanja aplikacije u offline načinu rada dolazi do greške koja zaustavlja daljnje učitavanje aplikacije. Kako bi izbjegli pojavu greške te omogućili korištenje aplikacije u offline načinu rada, servis za autentikaciju učitavamo samo u slučaju povezanosti s internetom u trenutku pokretanja aplikacije.

U aplikaciju dodajemo proširenje `nuxt-client-init.js` u kojem pozivamo Vuex akciju koju želimo izvršiti nakon pokretanja aplikacije:

```
export default (context) => {
  context.store.dispatch('nuxtClientInit', context)
}
```

Pozivamo asinkronu akciju `nuxtClientInit` u kojoj provjeravamo da li je korisnik online te učitavamo servis za autentikaciju u slučaju potvrdnog odgovora:

```
1  export const actions = {
2    async nuxtClientInit ({ dispatch, commit }) {
3      if (navigator.onLine) {
4        await this.$fire.authReady()
5
6        const unsubscribe = this.$fire.auth.onAuthStateChanged((user) => {
7          dispatch('auth/onAuthStateChangedAction', user, { root: true })
8          unsubscribe()
9        })
10     }
11
12     commit('SET_CHECKED_CONNECTION', true)
13   }
14 }
```

U liniji 6 definiramo koju Vuex akciju ili mutaciju želimo izvršiti prilikom promjene stanja autentikacije. Koristimo akciju `onAuthStateChangedAction`:

```
async onAuthStateChangedAction ({ commit }, value) {
  if (value) {
    commit('SET_USER_ID', value.uid)
    localStorage.setItem('userId', value.uid)
```

```
    commit('SET_IS_AUTHORIZED', true)

    try {
      const userData = await this.$fire.database.ref('users/' + value.uid)
                        .once('value')
      commit('SET_USER_NAME', {
        firstName: userData.val().firstName,
        lastName: userData.val().lastName
      })
      localStorage.setItem('userFirstName', userData.val().firstName)
      localStorage.setItem('userLastName', userData.val().lastName)
      commit('SET_IS_ADMIN', userData.val().isAdmin)

      this.$router.push({ path: '/' })
    } catch (error) {
      commit('SET_ERROR', error)
    }
  }
}
```

U slučaju prijave korisnika, dohvaćamo njegovo ime i prezime iz baze podataka te ih, zajedno s njegovim identifikatorom, spremamo u stanje koje kasnije koristimo za uvjetni prikaz stranica u navigaciji te ispis pozdravne poruke. Stanje mijenjamo pomoću metode `commit` kojoj predajemo ime mutacije koju želimo izvršiti.

Servis za autentikaciju nudi mogućnost postavljanja opcije `persistence` pomoću koje određujemo način perzistiranja stanja autentikacije. Zadana vrijednost opcije u konfiguraciji je `local`, a ona govori da je stanje potrebno spremiti u `localStorage` kako se ono ne bi izgubilo prilikom izlaska iz preglednika ili ponovnog učitavanja aplikacije. Time postižemo da se korisnik ne mora prijavljivati prilikom svakog otvaranja aplikacije, već je dovoljno prijaviti se jednom, a prilikom svakog drugog pristupa podaci će se automatski pročitati iz `localStorage`-a. Uočimo kako smo u akciji `onAuthStateChangedAction` identifikator korisnika te korisničko ime dodatno spremili u `localStorage`. To činimo kako bi omogućili korištenje aplikacije u offline načinu rada, odnosno kada ne možemo koristiti servis za autentikaciju. Ukoliko korisnik učita aplikaciju bez pristupa internetu, aplikacija će pročitati vrijednost identifikatora iz `localStorage`-a. Ako postoji spremljeni identifikator, postavit ćemo varijablu `userId` unutar Vuex stanja na njegovu vrijednost i tako omogućiti da se aplikacija koristi uz resurse spremljene u predmemoriji.

Korisnik se u aplikaciju prijavljuje pomoću e-mail adrese i lozinke. Uneseni podaci se provjeravaju i u slučaju uspješne prijave korisnik se preusmjerava na početnu stranicu, a u slučaju neuspješne prijave ispisuje se poruka s greškom. Prijava se izvršava pomoću akcije `login`:

```
async login ({ commit, dispatch }, credentials) {
  dispatch('resetAll')

  try {
    const data = await this.$fire.auth.signInWithEmailAndPassword(
                  credentials.email, credentials.password)
    commit('SET_USER_ID', data.user.uid)
    try {
      const userData = await this.$fire.database
                       .ref('users/' + data.user.uid)
                       .once('value')
      commit('SET_USER_NAME', {
        firstName: userData.val().firstName,
        lastName: userData.val().lastName
      })
      commit('SET_IS_ADMIN', userData.val().isAdmin)
    } catch (error) {
      commit('SET_ERROR', error)
      return
    }
    commit('SET_SUCCESS', true)
  } catch (error) {
    commit('SET_ERROR', error)
  }
}
```

Za samu prijavu korisnika koristimo metodu `signInWithEmailAndPassword` koja je dostupna unutar modula Nuxt Firebase.

Na sličan način omogućavamo i registraciju novih korisnika. Koristimo metodu `createUserWithEmailAndPassword` te dodatno spremamo unesene podatke o korisniku u bazu podataka:

```
await this.$fire.database.ref('users/' + data.user.uid)
        .set({
          firstName: credentials.firstName,
          lastName: credentials.lastName,
          isAdmin: false
        })
```

Prilikom registracije, svim korisnicima se zastavica `isAdmin` postavlja na `false`, a njenu vrijednost je moguće promijeniti u Firebase konzoli. Zastavica se provjerava u middleware metodi za čuvanje ruta. Iako vrijednost zastavice pohranjujemo u Vuex stanju, njenu vrijednost dodatno provjeravamo u trenutku pokušaja pristupa ruti `/userList` na kojoj se

prikazuje lista svih korisnika aplikacije. To činimo kako bismo izbjegli neovlašten pristup navedenoj ruti u slučaju mijenjanja stanja pomoću konzole web-preglednika. Vrijednost zastavice u Firebase bazi podataka dohvaćamo pomoću Vuex akcije `checkAdminFlag`. Neprijavljenim korisnicima brani se pristup svim stranicama osim onima za prijavu i registraciju:

```
export default async function ({ store, redirect, route }) {
  if (route.fullPath.startsWith('/login')
        || route.fullPath.startsWith('/signup')) {
    if (store.state.auth.userId) {
      return redirect('/')
    }
    return
  }

  if (!store.state.auth.userId) {
    return redirect('/login')
  }

  if (route.fullPath.startsWith('/userList')) {
    const isAdmin = await store.dispatch('auth/checkAdminFlag')

    if (!isAdmin) {
      return redirect('/')
    }
  }
}
```

Kako bi se navedena metoda izvršila prilikom svake promjene rute, dovoljno je u opciju `router` unutar datoteke `nuxt.config.js` dodati opciju `middleware` s imenom datoteke u kojoj se nalazi metoda:

```
router: {
  middleware: 'auth'
}
```

Firebase Storage koristimo u komponentama za učitavanje i dohvaćanje staza. Staze pri spremanju u Storage organiziramo u mape radi lakšeg pronalaska željene staze, ali i računanja statistike za svakog korisnika. Struktura Storage-a je sljedeća:

```
  userId/
  --| year /
```

```
----| month
------| day-track_name.gpx
```

Na ime svake staze dodajemo prefiks s danom na koji je nastala.

U komponenti `FileUpload` nakon obrade datoteke unutar komponente `Map` učitavamo datoteku na Storage:

```
const ref = this.$fire.storage.ref(this.userId + '/'
             + trackDate.getFullYear() + '/'
             + this.months[trackDate.getMonth()] + '/'
             + day + '-' + this.trackName.replaceAll(' ', '-') + '.gpx')
ref.getDownloadURL().then((url) => {
  this.error = 'Track with given name already exists.'
    this.loading = false
  }).catch((error) => {
    if (error.code === 'storage/object-not-found') {
      ref.put(this.file).then(() => {
        if (this.metadata) {
          ref.updateMetadata(this.metadata).then(() => {
            this.success = true
            this.loading = false
          }).catch(() => {
            this.error = 'An error occured while saving file metadata.'
            ref.delete()
          })
        }
      }).catch((error) => {
        this.error = error
      })
    }
  })
```

U slučaju da datoteka sa zadanim imenom već postoji korisniku se prikazuje prigodna poruka te se zahtijeva promjena imena datoteke. Izračunate podatke o kretanju spremamo u meta podatke datoteke kako ih ne bismo morali ponovno računati prilikom svakog računanja statistike.

Detaljan prikaz učitane staze dostupan je na stranici `tracks/:path` na kojoj se putanja do odgovarajuće datoteke u Storage-u čita iz parametra rute pomoću objekta `this.$route.params` te se odabrana datoteka preuzima:

```
downloadFile () {
  this.$fire.storage.ref(this.$route.params.path).getDownloadURL()
```

```
    .then((url) => {
      const xhr = new XMLHttpRequest()
      xhr.responseType = 'blob'
      xhr.onload = (event) => {
        this.blob = xhr.response
      }
      xhr.open('GET', url)
      xhr.send()
    })
}
```

Staza se zatim prikazuje na karti pomoću već spomenute komponente `Map`.

Hosting

Za jednostavan deploy naše aplikacije na besplatni Firebase Hosting koristimo alat Firebase CLI. Potrebno ga je instalirati pomoću naredbe `npm install -g firebase-tools`, a zatim se prijaviti u Firebase konzolu pomoću naredbe `firebase login` koja formu za prijavu otvara u pregledniku. Nakon uspješne prijave potrebno je pokrenuti naredbu `firebase init` unutar korijenskog direktorija aplikacije. Prilikom inicijalizacije moramo odabrati direktorij koji će se deploy-ati, a u našem slučaju odabiremo direktorij `dist/`.

Svaki put kada želimo deploy-ati novu verziju aplikacije dovoljno je pokrenuti naredbe `npm run build`, a zatim `firebase deploy`. Te dvije naredbe možemo objediniti u jednu skriptu koju dodajemo u opciju `scripts` unutar datoteke `package.json`:

```
{ "deploy": "npm run build && firebase deploy" }
```

Deploy tada možemo izvršiti koristeći naredbu `npm run deploy`. Aplikacija se nalazi na lokaciji oblika `https://project-id.web.app/`, a u Firebase konzoli dostupan je pregled svih verzija aplikacije.

3.4 Modul @nuxtjs/pwa

Najvažniji modul u našoj aplikaciji je Nuxt PWA koji automatski registrira service worker, generira datoteku manifest, dodaje SEO meta podatke, generira ikone raznih dimenzija te omogućava slanje push obavijesti [12].

Modul se može instalirati prilikom izrade novog Nuxt projekta u čarobnjaku `create-nuxt-app` ili naknadno koristeći naredbu `npm install @nuxtjs/pwa`. U slučaju korištenja drugog načina instalacije potrebno je u datoteku `nuxt.config.js` dodati:

```
buildModules: [
  '@nuxtjs/pwa'
]
```

Modul Nuxt PWA sastoji se od pet manjih modula čijom se kombinacijom osigurava da naša aplikacija uistinu bude PWA.

Modul Icon

Modul Icon automatski generira ikone raznih veličina i dodaje ih u datoteku manifest. Ukoliko nije drugačije definirano, modul će ikone izraditi na temelju datoteke `icon.png` u direktoriju `static/` te će izraditi ikone širine i visine 64, 120, 144, 152, 192, 384 i 512 piksela. Zbog toga je poželjno da `icon.png` bude kvadratna, minimalnih dimenzija 512x512 piksela. Zadane postavke mogu se promijeniti u datoteci `nuxt.config.js` unutar opcije `pwa.icon`:

```
icon: {
  source: './static/track.png'
}
```

Modul Meta

Modul Meta omogućava lako dodavanje meta tag-ova. Nudi mogućnost postavljanja 22 različita tag-a, a neki od njih su `charset`, `viewport`, `description`, `author` i `lang`. Ukoliko nije drugačije zadano u opciji `pwa.meta`, modul će koristiti postavke iz opcije `head.meta` u datoteci `nuxt.config.js`. Primjer postavljanja opisa aplikacije pomoću opcije `pwa.meta`:

```
meta: {
  description: 'Web app for tracking your tracks.',
  og: {
    description: 'Web app for tracking your tracks.'
  }
}
```

Na slici 3.3 vidimo postavljene meta tag-ove u našoj aplikaciji.

Slika 3.3: Meta podaci

Modul Manifest

Modul Manifest generira datoteku manifest i dodaje ju u aplikaciju. Sve opcije datoteke manifest imaju zadane vrijednosti, a moguće ih je promijeniti pomoću opcije `pwa.manifest`. Nakon pokretanja naredbe `npm run build`, u direktoriju `dist/_nuxt/` dobivamo datoteku sljedećeg sadržaja u kojoj možemo uočiti postavke za ikone generirane pomoću modula Icon:

```
{
  "name": "Tracking App",
  "short_name": "Tracking App",
  "description": "## Build Setup",
  "icons": [
    {
      "src": "/_nuxt/icons/icon_64x64.3dfe72.png",
      "sizes": "64x64",
      "type": "image/png",
      "purpose": "any maskable"
    },
    {
      "src": "/_nuxt/icons/icon_120x120.3dfe72.png",
      "sizes": "120x120",
      "type": "image/png",
      "purpose": "any maskable"
    },
    ... // popis ikona raznih dimenzija
    {
      "src": "/_nuxt/icons/icon_512x512.3dfe72.png",
```

```
      "sizes": "512x512",
      "type": "image/png",
      "purpose": "any maskable"
    }
  ],
  "start_url": "/?standalone=true",
  "display": "standalone",
  "background_color": "#eee",
  "theme_color": "#160",
  "lang": "en-EN"
}
```

Modul Workbox

Modul Workbox koristi skup JavaScript biblioteka za dodavanje service worker-a u aplikaciju. U postavkama modula zadano je automatsko predmemoriranje svih zahtjeva prema lokacijama `/_nuxt/` te `/`. To znači da će sve rute, zajedno s datotekama koje sadrže JavaScript kod potreban za njihovo izvršavanje, biti spremljene u predmemoriju i biti dostupne u trenutku kada korisnik ne bude imao pristup internetu.

Budući da naša aplikacija uvelike ovisi o podacima i datotekama koje dohvaćamo iz Firebase servisa, potrebno se pobrinuti i o predmemoriranju zahtjeva prema tim servisima. Modul nuxt-firebase nudi automatsko predmemoriranje zahtjeva prema bazi, no ne i prema Storage-u gdje su spremljeni podaci o kretanju korisnika. Također je potrebno predmemorirati i zahtjeve prema OpenStreetMap serveru kako bi se staze ispravno prikazivale na karti. Kako bi omogućili predmemoriranje zahtjeva prema Storage-u i OpenStreetMap-u, u datoteku `nuxt.config.js` dodajemo sljedeće postavke:

```
workbox: {
  runtimeCaching: [
    {
      urlPattern: 'https://firebasestorage.googleapis.com/v0/b/' +
                  'trackingapp-e6315.appspot.com/.*',
      strategyOptions: {
        cacheName: 'storage-cache'
      },
      strategyPlugins: [{
        use: 'Expiration',
          config: {
            maxEntries: 50,
            maxAgeSeconds: 604800 // one week
          }
```

```
      }]
    },

    {
      urlPattern: 'http://b.tile.osm.org/.*',
      strategyOptions: {
        cacheName: 'openstreet-cache'
      },
      strategyPlugins: [{
        use: 'Expiration',
        config: {
          maxEntries: 50,
          maxAgeSeconds: 604800 // one week
        }
      }]
    }
  ]
}
```

Opciju `urlPattern` potrebno je postaviti na lokaciju na kojoj se nalazi Firebase Storage za našu aplikaciju, odnosno na lokaciju OpenStreetMap servera. Postavljanjem opcije `maxEntries` možemo ograničiti broj zahtjeva koji će u svakom trenutku biti spremljen u predmemoriji, a pomoću opcije `maxAgeSeconds` možemo ograničiti koliko dugo svaki zahtjev može biti spremljen u predmemoriji.

Nakon pokretanja naredbe `npm run build` u direktoriju `dist/` dobivamo datoteku `sw.js`. Uočimo da je bitno jednostavnije pripremiti gore opisani objekt `workbox` no manualno implementirati datoteku `sw.js` (pogotovo koristeći samo Service Worker API), iz čega se vidi prednost korištenja Nuxt-a, odnosno modula Nuxt PWA. Iz generirane datoteke ističemo objekt `options` te dvije metode zadužene za predmemoriranje:

```
const options = {
  "workboxURL": "https://cdn.jsdelivr.net/npm/workbox-cdn@5.1.3/" +
                "workbox/workbox-sw.js",
  "importScripts": [],
  "config": { "debug": false },
  "clientsClaim": true,
  "skipWaiting": true,
  "cleanupOutdatedCaches": true,
  "offlineAnalytics": false,
  "preCaching": ["/?standalone=true"],
  "runtimeCaching": [
    {
```

```
      "urlPattern": "https://firebasestorage.googleapis.com/v0/b/" +
                    "trackingapp-e6315.appspot.com/.*",
      "strategyOptions": { "cacheName": "storage-cache" },
      "strategyPlugins": [
        {
          "use": "expiration.ExpirationPlugin",
          "config": [{ "maxEntries": 50, "maxAgeSeconds": 604800 }]
         }
      ],
      "handler": "NetworkFirst",
      "method":"GET"
    },
    {
      "urlPattern": "http://b.tile.osm.org/.*",
      "strategyOptions": { "cacheName": "openstreet-cache" },
      "strategyPlugins": [
        {
          "use": "expiration.ExpirationPlugin",
          "config": [{ "maxEntries": 50, "maxAgeSeconds": 604800 }]
         }
      ],
      "handler": "NetworkFirst",
      "method":"GET"
    },
    {
      "urlPattern": "/_nuxt/",
      "handler": "CacheFirst",
      "method": "GET",
      "strategyPlugins":[]
    },
    {
      "urlPattern": "/",
      "handler": "NetworkFirst",
      "method": "GET",
      "strategyPlugins":[]
     }
  ],
  "offlinePage": null,
  "pagesURLPattern": "/",
  "offlineStrategy": "NetworkFirst"
}
```

```
function precacheAssets(workbox, options) {
  if (options.preCaching.length) {
    workbox.precaching.precacheAndRoute(options.preCaching,
                                        options.cacheOptions)
  }
}

function runtimeCaching(workbox, options) {
  for (const entry of options.runtimeCaching) {
    const urlPattern = new RegExp(entry.urlPattern)
    const method = entry.method || 'GET'
    const plugins = (entry.strategyPlugins || [])
      .map(p => new (getProp(workbox, p.use))(...p.config))
    const strategyOptions = { ...entry.strategyOptions, plugins }
    const strategy = new workbox.strategies[entry.handler]
                     (strategyOptions)
    workbox.routing.registerRoute(urlPattern, strategy, method)
  }
}
```

Postoje dva načina predmemoriranja resursa. Metoda `precacheAssets` zadužena je za tzv. *precaching* koji služi za predmemoriranje statičkih resursa. Za razliku od *precaching*-a u kojemu se svi potrebni resursi predmemoriraju odjednom, *runtime caching* sprema resurse u predmemoriju u trenutku slanja mrežnog zahtjeva prema određenom resursu. Za *runtime caching* zadužena je metoda `runtimeCaching`.

Uočimo da se u *runtime caching*-u pojavljuju dvije različite vrijednosti za opciju `handler` - `NetworkFirst` i `CacheFirst`. U slučaju postavljene vrijednosti `CacheFirst`, preglednik će prvo provjeriti nalazi li se željeni resurs u predmemoriji. Ukoliko se ne nalazi, izvršit će se mrežni zahtjev prema lokaciji na kojoj se resurs nalazi. U slučaju postavljene vrijednosti `NetworkFirst` prvo će se pokušati izvršiti zahtjev prema lokaciji resursa. Ukoliko dođe do greške pri dohvaćanju resursa, isti će se pokušati pronaći u predmemoriji. `CacheFirst` se koristi za resurse koji se rijetko mijenjaju, a `NetworkFirst` za one koji se često mijenjaju, ali su također bitni za offline rad aplikacije. Uz prethodno navedene, moguće je koristiti i sljedeće vrijednosti: `CacheOnly`, `NetworkOnly` te `StaleWhileRevalidate`.

Modul OneSignal

Modul OneSignal omogućava slanje push obavijesti korisnicima aplikacije. OneSignal je jedan od mnogih vanjskih servisa za slanje push obavijesti za web ili nativne aplikacije. Podržava gotovo sve web i nativne platforme, a jednostavno slanje obavijesti omogućeno

je putem konzole kojoj pristupamo unutar web-preglednika. Budući da modul u aplikaciju dodaje dodatni service worker, njegovo postavljanje je nešto kompliciranije. Modul nije instaliran u sklopu modula Nuxt PWA, nego ga je potrebno zasebno instalirati i dodati u opciju `modules` ispred modula `@nuxtjs/pwa`. Zatim je potrebno dodati opciju `oneSignal` u datoteku `nuxt.config.js`:

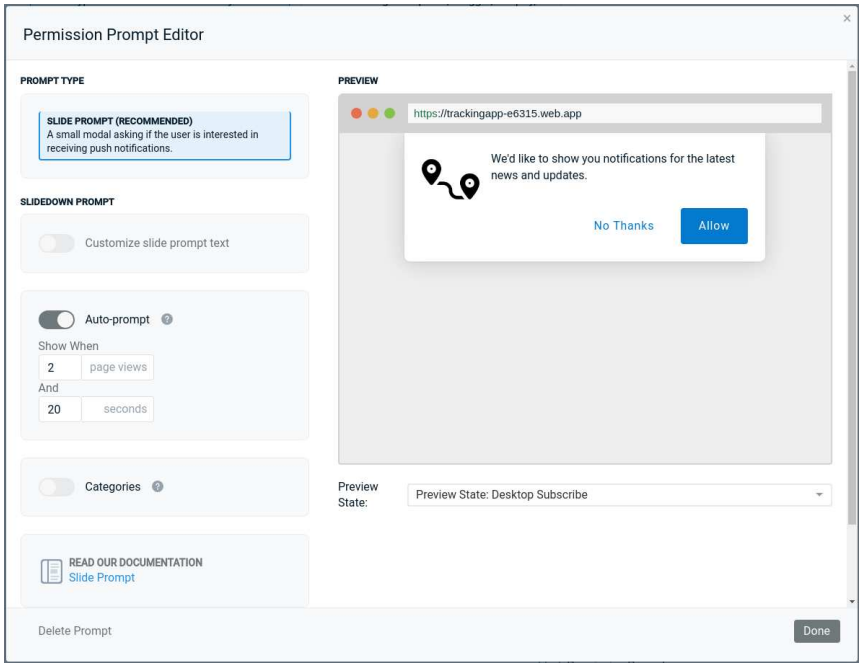
```
oneSignal: {
  init: {
    appId: 'YOUR_APP_ID',
    welcomeNotification: {
      disable: true
    }
  }
}
```

Vrijednost `appId` moguće je pronaći u OneSignal konzoli u kojoj je potrebno registrirati našu aplikaciju kako bi mogli slati push obavijesti [14]. Između ostalog, potrebno je zadati ime aplikacije, lokaciju na kojoj se nalazi te ikonu koju želimo prikazati prilikom primitka obavijesti. OneSignal ne šalje direktno obavijest na uređaje korisnika, već ih šalje prema Google i Apple serverima koji su zaduženi za njihovu daljnju distribuciju. Budući da korisnik mora pristati na primanje push obavijesti, korisniku je potrebno prikazati dijalog u kojemu može potvrditi ili odbiti primanje obavijesti prije njihovog slanja. Na slici 3.4 vidimo kako pomoću OneSignal konzole možemo jednostavno postaviti izgled i vrijeme prikazivanja dijaloga. U našem slučaju, dijalog se prikazuje nakon što korisnik pregleda barem 2 različite stranice te u aplikaciji boravi barem 20 sekundi.

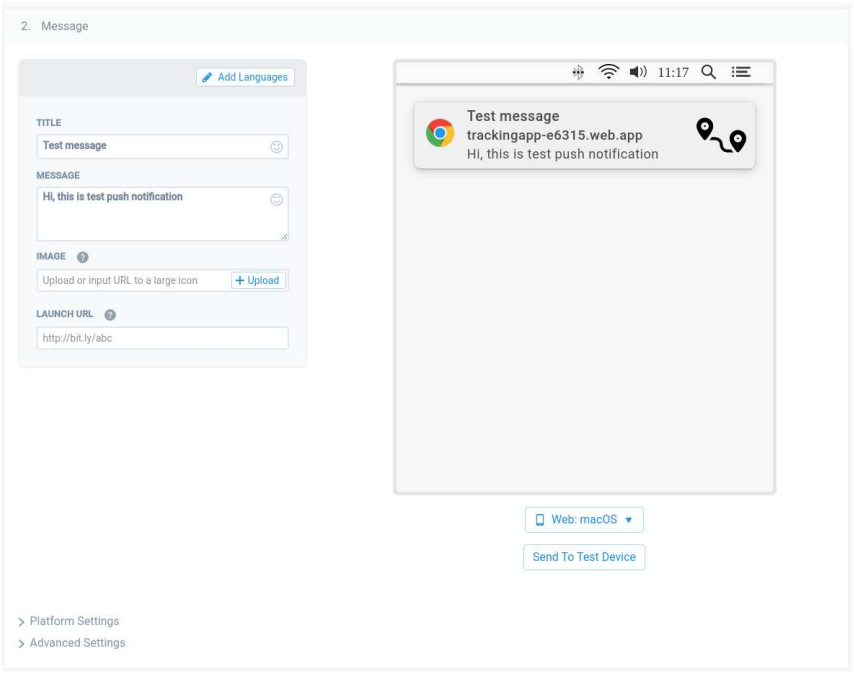
Obavijesti se mogu slati izravno iz OneSignal konzole ili pomoću OneSignal REST API-ja. Budući da je OneSignal REST API predviđen za korištenje na serverskoj strani aplikacije, a naša aplikacija sastoji se samo od klijentskog koda, obavijesti ćemo slati pomoću konzole. Primjer slanja obavijesti iz konzole vidimo na slici 3.5.

Uočimo da korisnik neće primiti obavijest ukoliko u trenutku njenog slanja nije povezan s internetom. Prilikom slanja obavijesti iz OneSignal konzole moguće je postaviti opciju Time To Live koja određuje koliko dugo će obavijest biti spremljena na Google i Apple serverima prije nego ju oni pošalju korisnicima koji su offline. Zadana vrijednost opcije je tri dana, što znači da korisnik neće primiti obavijest ukoliko unutar tri dana od slanja obavijesti ne uspostavi vezu s internetom.

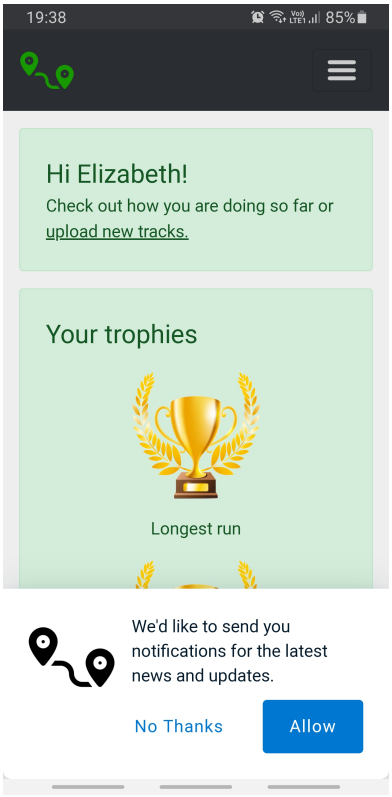
Na slici 3.6 vidimo izgled dijaloga za prihvaćanje obavijesti na mobilnom uređaju, a na slici 3.7 primjer primljene obavijesti.



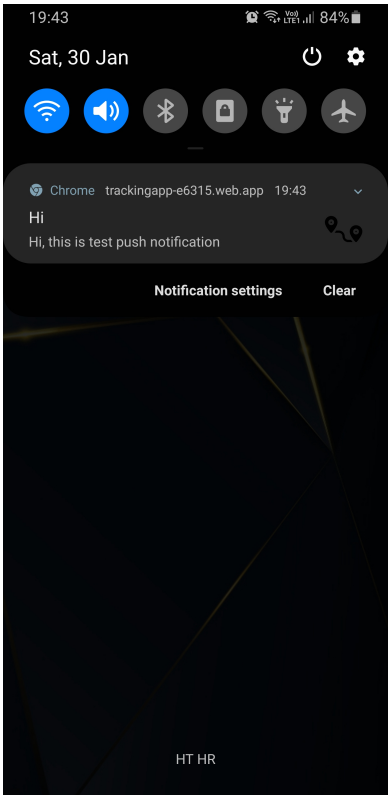
Slika 3.4: Postavljanje dijaloga za prihvaćanje push obavijesti



Slika 3.5: Slanje push obavijesti iz OneSignal konzole



Slika 3.6: Dijalog za prihvaćanje obavijesti



Slika 3.7: Prikaz primljene obavijesti

3.5 Konačan izgled

Nakon opisa implementacije, ukratko ćemo prezentirati i konačan izgled aplikacije. Za korištenje aplikacije potrebno se prijaviti ili registrirati ukoliko korisnik ne posjeduje korisnički račun. Na slici 3.8 vidimo izgled forme za registraciju novih korisnika.

Nakon uspješne prijave, prikazuje se početna stranica aplikacije na kojoj je vidljiva pozdravna poruka, statistika za tekući tjedan i mjesec te osvojeni trofeji. Izgled početne stranice vidimo na slici 3.9. Na svim stranicama prikazuje se navigacijska traka koja nudi pristup svim stranicama te odjavu korisnika.

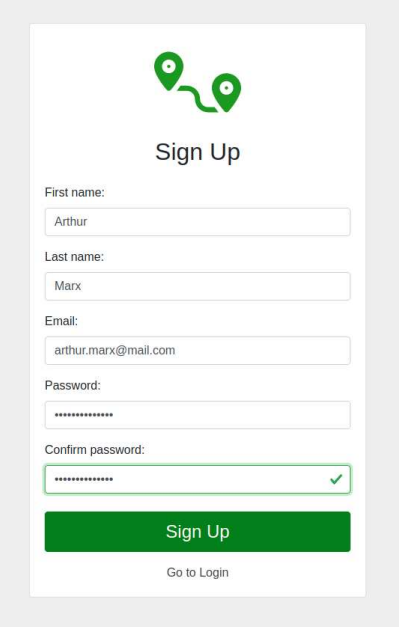
Učitavanje nove `gpx` datoteke moguće je na stranici Upload. Nakon odabira datoteke s uređaja prikazuje se pregled staze te forma koju je potrebno ispuniti prije spremanja datoteke u Firebase Storage. Izgled stranice za učitavanje datoteke vidimo na slici 3.10.

Pregled svih staza po godinama i mjesecima omogućen je na stranici My Tracks čiji izgled vidimo na slici 3.11. Također je omogućen prikaz statistike o kretanju na tjednoj ili mjesečnoj bazi, a nalazi se na stranici Statistics koju možemo vidjeti na slici 3.12.

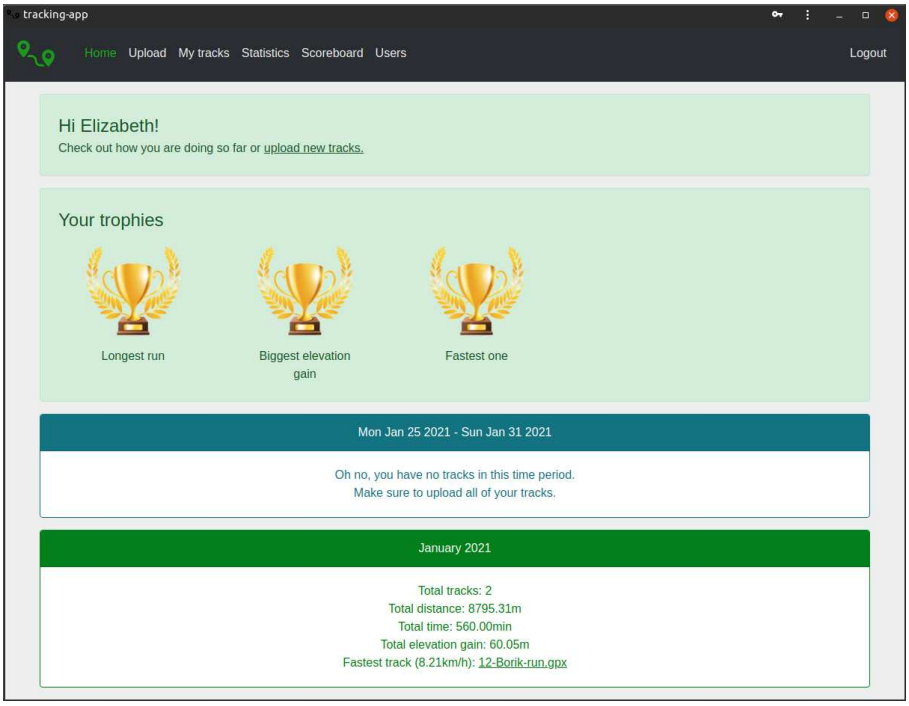
Konačno, na stranici Scoreboard prikazuje se top lista korisnika po raznim kriterijima. Ukoliko prijavljeni korisnik ima administratorske ovlasti, može pristupiti i stranici Users na kojoj je prikazana lista svih korisnika. Klikom na ime svakog od korisnika otvara se stranica s pregledom njegovih staza. Izgled stranica Users i Scoreboard vidimo na slikama 3.13 i 3.14.

Budući da su mogućnosti aplikacije ograničene ukoliko uređaj nije povezan s internetom, u navigacijskoj traci prikazujemo ikonu kako bi korisnik bio svjestan da aplikaciju koristi u offline načinu rada. Na slici 3.15 vidimo izgled aplikacije u offline načinu rada. Zbog predmemoriranja zahtjeva prema OpenStreetMap serveru te Firebase servisima, korisnik može ponovno pregledavati sve stranice koje je posjetio prije gubitka veze.

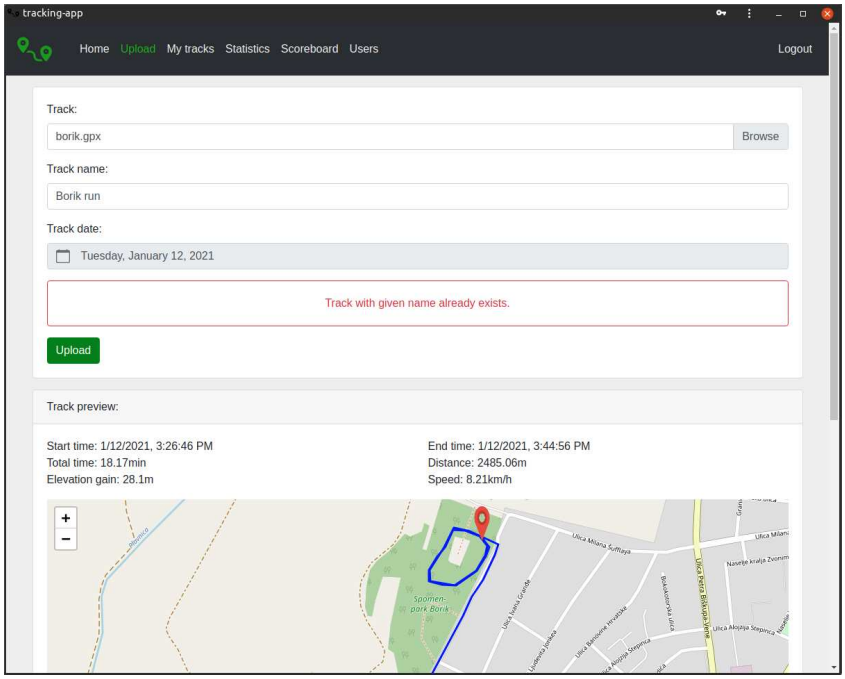
Kao što smo već pojasnili, u slučaju nepovezanosti s internetom u trenutku pokretanja aplikacije, identifikator korisnika čita se iz `localStorage`-a kako bi aplikacija mogla koristiti predmemorirane resurse. Ukoliko prilikom korištenja aplikacije uređaj ostvari vezu s internetom, korisniku se ispisuje poruka o uspostavi veze u kojoj se korisnik moli da ponovno učita aplikaciju radi procesa automatske autentikacije koji će omogućiti neometano daljnje korištenje aplikacije. Izgled poruke vidimo na slici 3.16.



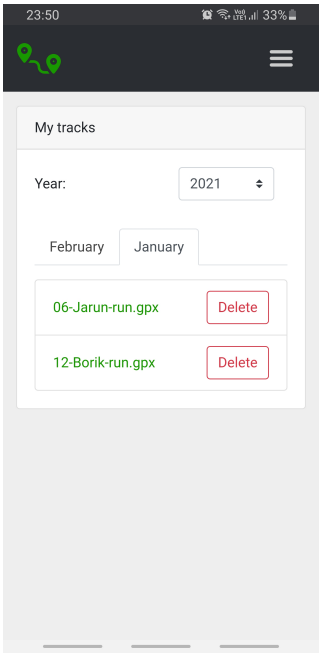
Slika 3.8: Registracija novog korisnika



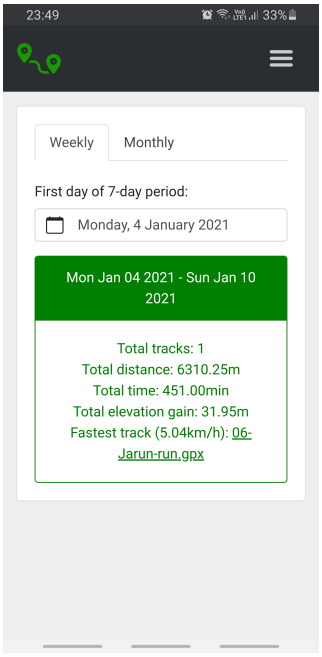
Slika 3.9: Početna stranica



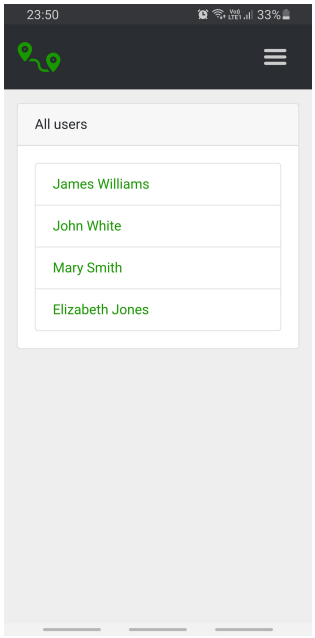
Slika 3.10: Učitavanje nove staze



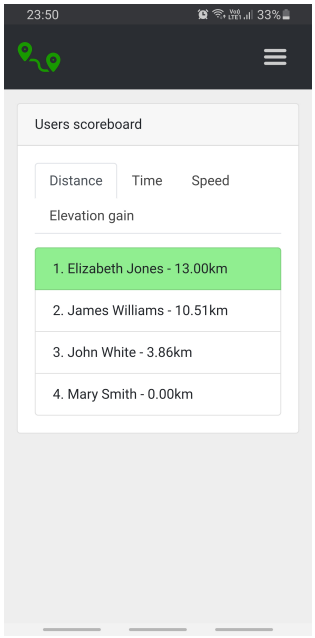
Slika 3.11: Pregled svih staza



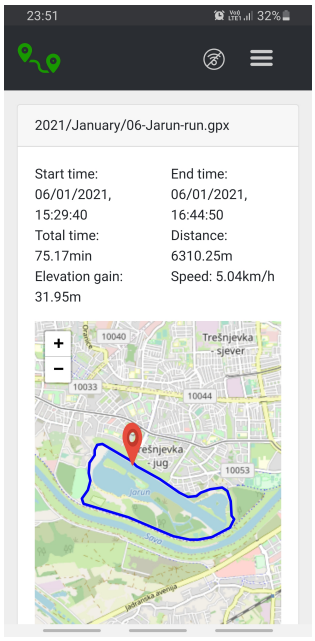
Slika 3.12: Pregled statistike



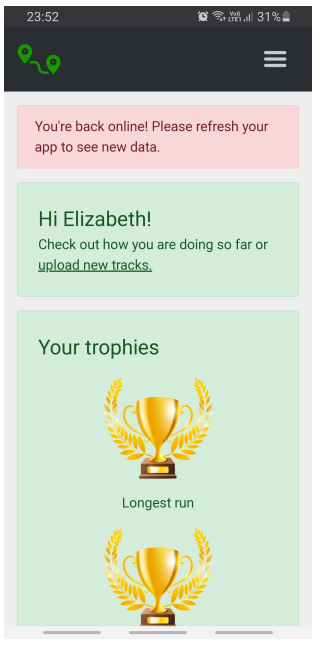
Slika 3.13: Popis svih korisnika aplikacije



Slika 3.14: Top lista korisnika



Slika 3.15: Offline način rada



Slika 3.16: Poruka o uspostavi veze

Poglavlje 4

Rezultati

U odjeljku 1.4 predstavili smo alat Lighthouse za provjeru kvalitete web-aplikacije i prikazali rezultate dobivene analizom naše aplikacije. Sada ćemo detaljnije analizirati svaki od dobivenih rezultata.

Performanse aplikacije

Prilikom ocjenjivanja performansi aplikacije, Lighthouse u obzir uzima razne čimbenike, a naglasak je na vremenu potrebnom za prikaz prvog sadržaja na stranici te vremena do potpune interaktivnosti aplikacije. Dobiveni rezultat od 97 bodova možemo smatrati odličnim, no valja napomenuti da je teško pravilno ispitati performanse aplikacija koje koriste autentikaciju. Budući da su podaci o prijavljenom korisniku spremljeni unutar preglednika, a prije pokretanja analize se svi podaci brišu, Lighthouse će analizirati samo stranicu za prijavu korisnika. Početnu stranicu, koja se prikazuje prijavljenim korisnicima, možemo analizirati tako da se prvo prijavimo s proizvoljnim korisnikom, a zatim ručno u pregledniku izbrišemo sve podatke osim onih koji sadrže podatke o prijavljenom korisniku. Tada u postavkama Lighthouse-a dodatno trebamo označiti da ne želimo brisati spremljene podatke.

Pristupačnost

Lighthouse također ocjenjuje pristupačnost web-aplikacije, odnosno da li aplikacija omogućava istu dostupnost i iste mogućnosti svim korisnicima, osobito osobama s invaliditetom. Kako bi postigli rezultat od 100 bodova bilo je potrebno podesiti boje pozadine i teksta tako da imaju dovoljan kontrast te dodati atribute za čitače ekrana.

Korištenje najboljih praksi

Prilikom ocjenjivanja korištenja najboljih praksi između ostalog se provjerava koristi li aplikacija HTTPS, mogu li korisnici lijepiti tekst u polje za unos lozinke i ispisuje li aplikacija greške u konzolu preglednika. Prilikom prvog pokretanja analize naša aplikacija prošla je sve provjere osim ispisa u konzolu. Za ispis greške bio je odgovoran modul Nuxt PWA zbog pogrešne konfiguracije modula Workbox. Konfiguracija je prepravljena u novijoj verziji modula, pa je nakon njegovog ažuriranja aplikacija dobila svih 100 bodova.

Optimiziranosti za pretraživače

Za prolazak provjere optimiziranosti aplikacije za pretraživače se također provjeravaju mnogi faktori, a neki od njih su postavljeni meta podaci, sadržavaju li slike atribut `alt` te mogu li se stranice aplikacije indeksirati. Već nakon generiranja novog Nuxt projekta bila je zadovoljena većina ovih uvjeta, a ono što je nedostajalo dodali smo pomoću modula Meta te tako postigli svih 100 bodova.

PWA kriteriji

Ono što nas najviše zanima je da li naša aplikacija zadovoljava sve kriterije progresivnosti. Na slici 1.2 možemo vidjeti popis svih kriterija i uvjeriti se da naša aplikacija zadovoljava sve navedene. Važno je napomenuti da je za ispunjavanje gotovo svih kriterija zaslužan upravo modul Nuxt PWA uz minimalnu dodatnu konfiguraciju. Već nakon same instalacije modula aplikacija je zadovoljavala većinu uvjeta, a dodatno smo morali osigurati da veličina sadržaja aplikacije odgovara veličini ekrana, dodati sliku koja će se koristiti za generiranje svih potrebnih ikona te postaviti sadržaj koji će se prikazivati u slučaju da preglednik ne podržava korištenje JavaScript-a.

Vidimo da uz korištenje modernih aplikacijskih okvira kao što je NuxtJS te pripadnih modula možemo vrlo jednostavno izraditi aplikaciju koja zadovoljava sve PWA kriterije te ima zadovoljavajuće performanse. Budući da programer ne mora brinuti o pisanju service worker-a za offline rad i slanje notifikacija, datoteke manifest te postavljanju svih meta podataka, može se u potpunosti posvetiti funkcionalnostima aplikacije i pisanju kvalitetnog koda. Upravo to je najveća prednost korištenja aplikacijskih okvira te dobro testiranih proširenja i modula.

Bibliografija

[1] *Firebase*, https://firebase.google.com/, pristupljeno u veljači 2021.

[2] V. K. Garg, *Pro Vue.js 2*, Apress, 2018.

[3] L. T. Kok, *Hands-on Nuxt.js Web Development*, Packt, 2020.

[4] *leaflet-gpx*, https://github.com/mpetazzoni/leaflet-gpx, pristupljeno u veljači 2021.

[5] *Leaflet*, https://leafletjs.com/, pristupljeno u veljači 2021.

[6] *Lighthouse*, https://developers.google.com/web/tools/lighthouse, pristupljeno u veljači 2021.

[7] C. Love, *Progressive Web Application Development by Example*, Packt, 2018.

[8] *Web app manifests*, https://developer.mozilla.org/en-US/docs/Web/Manifest, pristupljeno u veljači 2021.

[9] B. Nelson, *Getting to Know Vue.js*, Apress, 2018.

[10] *Nuxt Firebase*, https://firebase.nuxtjs.org/, pristupljeno u veljači 2021.

[11] *nuxt-leaflet*, https://github.com/schlunsen/nuxt-leaflet, pristupljeno u veljači 2021.

[12] *Nuxt PWA*, https://pwa.nuxtjs.org/, pristupljeno u veljači 2021.

[13] *NuxtJS*, https://nuxtjs.org, pristupljeno u veljači 2021.

[14] *One Signal*, https://app.onesignal.com/, pristupljeno u veljači 2021.

[15] *OpenStreetMap*, https://www.openstreetmap.org/, pristupljeno u veljači 2021.

[16] A. Russell, *Progressive Web Apps: Escaping Tabs Without Losing Our Soul*, https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/, pristupljeno u veljači 2021.

[17] *Introduction to Service Worker*, https://developers.google.com/web/ilt/pwa/introduction-to-service-worker, pristupljeno u veljači 2021.

[18] D. Sheppard, *Beginning Progressive Web App Development: Creating a Native App Experience on the Web*, Apress, 2017.

[19] *Progressive Web Apps*, https://web.dev/progressive-web-apps/, pristupljeno u veljači 2021.

Sažetak

U ovom radu predstavili smo pojam progresivnih web-aplikacija (PWA), naveli smo prednosti i mane te objasnili glavna obilježja takvih aplikacija. Istaknuli smo dva JavaScript aplikacijska okvira koja se mogu koristiti za izradu PWA - Vue.js i NuxtJS. Ukratko smo objasnili proces izrade aplikacija pomoću okvira Vue.js, a potom prikazali kako okvir NuxtJS, zahvaljujući automatskoj konfiguraciji, olakšava izradu Vue aplikacija. Tijek izrade aplikacije koja zadovoljava sve uvjete PWA prikazali smo na primjeru aplikacije Tracking App. Aplikaciju smo izradili pomoću aplikacijskog okvira NuxtJS, a svi uvjeti progresivnosti ispunjeni su uz pomoć modula Nuxt PWA.

Summary

In this thesis we introduced the concept of progressive web-applications (PWAs), presented advantages and disadvantages and core features of these applications. We pointed out two JavaScript frameworks which can be used for PWA development - Vue.js and NuxtJS. We briefly explained the process of application development with Vue.js, and then presented how NuxtJS, thanks to its automatic configuration, makes Vue application development easier. The flow of PWA development was presented by building Tracking App application. We built this application using NuxtJS framework, and all PWA requirements were satisfied with help of Nuxt PWA module.

Životopis

Rođena sam 03.03.1996. u Bjelovaru gdje sam završila III. osnovnu školu Bjelovar te opći smjer Gimnazije Bjelovar.

Nakon završene srednje škole, 2015. godine upisala sam preddiplomski sveučilišni studij Matematike na Prirodoslovno-matematičkom fakultetu u Zagrebu. 2018. godine upisala sam diplomski sveučilišni studij Računarstva i matematike na istom fakultetu.

2019. godine počela sam raditi kao frontend developer, a trenutno radim kao fullstack developer.