Web aplikacija za praćenje pozicije nestacionarnih objekata

Vrbić, Antonio

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

WEB APLIKACIJA ZA PRAĆENJE POZICIJE NESTACIONARNIH OBJEKATA

Završni rad

Antonio Vrbić

Osijek, 2017.

Sadržaj

1. UVOD 1

1.1. Zadatak završnog rada 1

2. MODEL WEB APLIKACIJE 2

2.1. Pregled postojećih rješenja 2

2.2. Specifikacije zahtjeva vlastitog rješenja 3

2.3. Funkcionalni model aplikacije 4

2.3.1. Poslužiteljski model aplikacije 5

2.3.2. Model korisničkog sučelja 7

3. PROGRAMSKO RJEŠENJE WEB APLIKACIJE 8

3.1. Korištene tehnologije i alati 8

3.1.1. Programski jezici 8

3.1.2. Izvedeni programski jezici 9

3.1.3. Protokoli i tipovi podataka 10

3.1.4. Programski okviri i biblioteke 11

3.2. Implementacija autoriziranja i autentificiranja 11

3.2.1. Autentificiranja 11

3.2.2. Autoriziranje 13

3.3. Implementacija aplikacijskog programskog sučelja 14

3.4. Implementacija korisničkog sučelja 16

3.4.1. Komponente 18

3.4.2. Putanje 20

3.4.3. Usluge 20

3.4.4. Upravljanje stanjem aplikacije 21

4. UPOTREBA I TESTIRANJE WEB APLIKACIJE 23

4.1. Upute za korištenje web aplikacije 23

4.2. Provjera specifikacije zahtjeva 24
4.2.1. Prilagodljivost sučelja različitim uređajima 25
4.3. Testiranje brzine 26
4.3.1. Vrijeme pristupa web aplikaciji 26
4.3.2. Usporedba veličina datoteka web aplikacije 27
5. ZAKLJUČAK 29
LITERATURA 30
SAŽETAK 32
ŽIVOTOPIS 33
PRILOZI (na CD-u) 34

1. UVOD

Glavni cilj ovog rada je ostvarenje web aplikacije koja prikazuje trenutne i prijašnje lokacije praćenih objekata (automobili, ljudi, životinje i sl.) na karti. Pod tim ciljem podrazumijeva se dizajniranje prilagodljivog i intuitivnog korisničkog sučelja koje omogućuje pregled svih praćenih uređaja te dodavanje i prilagodbu novih. Potrebno je prikazati implementaciju sustava upotrebom najnovijih tehnologija koje trenutno postoje. Pri implementaciji, najveći izazov predstavlja spajanje više odvojenih usluga u jednu funkcionalnu cjelinu.

Drugo poglavlje opisuje model web aplikacije te strukturu od koje se ona sastoji. Predstavljena je podjela aplikacije koja se sastoji od tri glavna dijela koji su međusobno neovisni, ali zajedno čine funkcionalnu aplikaciju. U trećem poglavlju opisane su tehnologije i alati koji su korišteni pri izradi aplikacije, gdje najveću ulogu predstavljaju programski okviri *Angular* i *ASP.NET Core* te usluga za iscrtavanje karte i usluga za prijavu korisnika. Osim toga, treće poglavlje sadrži implementaciju prijave korisnika u sustav te provjere njegovih prava pri pokušaju dohvaćanja zaštićenih resursa. Opisana je implementacija pozadinskog aplikacijskog sučelja koje komunicira s bazom podataka te je opisana implementacija korisničkog sučelja koje prikazuje potrebne informacije korisniku aplikacije. Četvrto poglavlje bavi se testiranjem korisničkog sučelja te testiranjem brzine dohvaćanja aplikacije i provjerom korisničkog sučelja.

1.1. Zadatak završnog rada

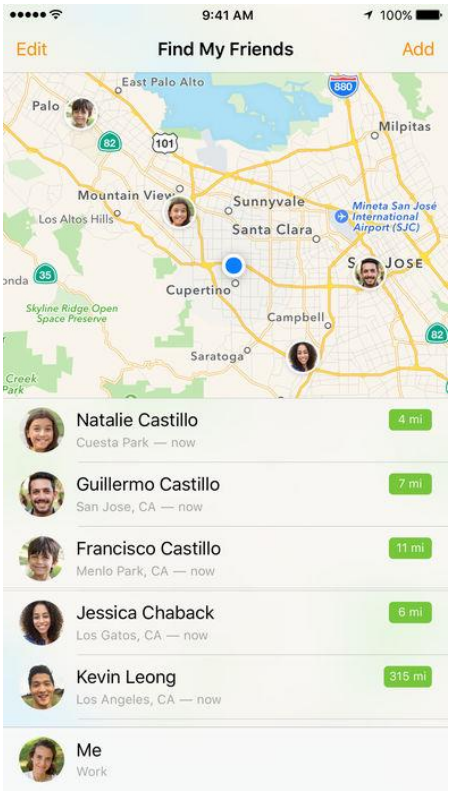
Potrebno je napraviti web aplikaciju koja omogućuje prikaz nestacionarnih objekata (automobili, motocikli, bicikli, ljudi, životinje) na karti te prikaz povijesti njihovih pozicija i kretanja. Kroz aplikaciju također treba biti omogućeno dodavanje i prilagođavanje pojedinih objekata (mijenjanje imena i slike). Aplikacija mora koristi ASP.Net Web API za dobivanje i slanje podataka, tj. za razmjenu podataka. Korisničko sučelje mora biti prilagodljivo, dok je sama tehnologija izrade sučelja proizvoljna.

2. MODEL WEB APLIKACIJE

2.1. Pregled postojećih rješenja

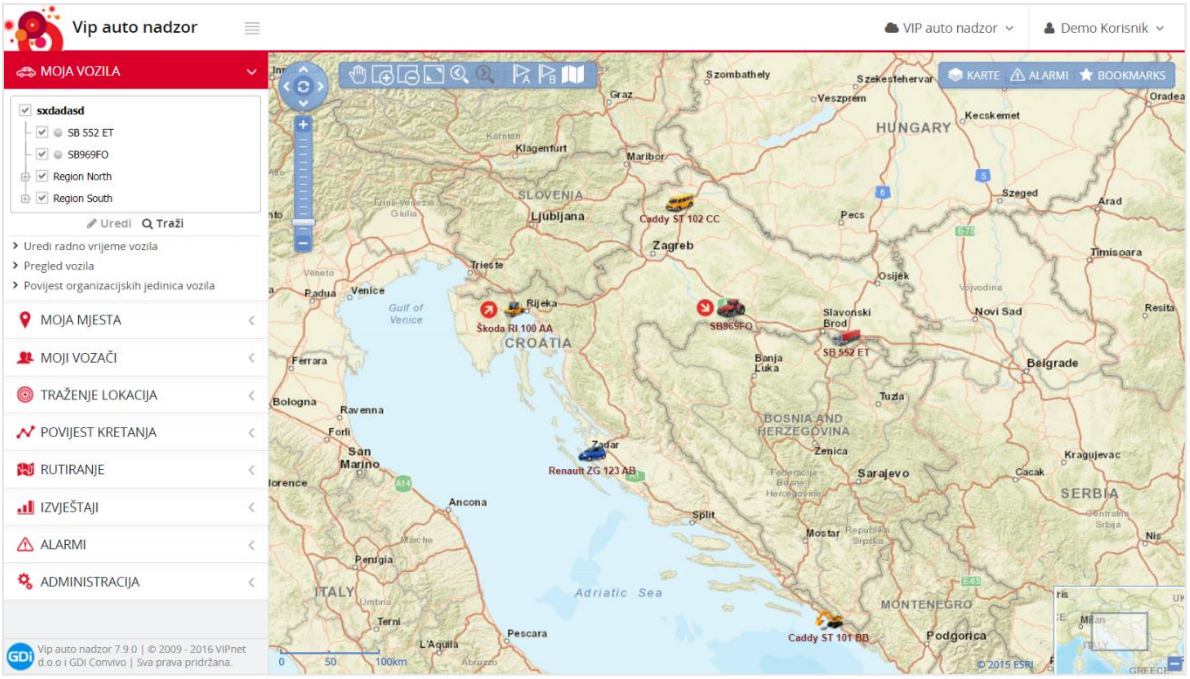
U ovom poglavlju navedena su postojeća rješenja koja nude lokacijske usluge.

Slikom 2.1 prikazano je sučelje *Find My Friends* [1] aplikacije koja je dostupna na operacijskom sustavu iOS. Aplikacija nudi praćenje lokacije prijatelja u stvarnom vremenu i postavljanje lokacijskih alarma koji mogu obavijestiti korisnika ukoliko se neki od njegovih prijatelja nalazi u blizini, napustio ili ušao u određeno područje i sl. Uz navedeno, kako ne bi u potpunosti bila narušena privatnost korisnika, aplikacija nudi zabranu praćenja lokacije na određeno vrijeme ili u potpunosti. Za korištenje aplikacije i korisnik i prijatelji moraju posjedovati iOS uređaja, dok je sama aplikacija u potpunosti besplatna.



Sl. 2.1 Korisničko sučelje *Find My Friends* aplikacije [1]

Lokacijska usluga *Vip auto nadzor* [2] omogućuje praćenje vozila u stvarnom vremenu te prikazivanje lokacija na karti. Aplikacija osim nadzora lokacija vozila nudi i lokacijske alarme, povijest kretanja, usmjeravanje, generiranje izvještaja te mnoge druge funkcionalnosti. Aplikacija je dostupna preko Internet preglednika i mobilnih uređaja, a praćenje vozila se ostvaruje pomoću posebnih uređaja koji se naplaćuju. Na slici 2.2 je prikazano korisničko sučelje aplikacije s praćenim vozilima prikazanim na karti.



Sl. 2.2 Korisničko sučelje *Vip auto nadzor* aplikacije

Slično prethodnim uslugama, ali za kućne ljubimce, usluga *Whistle* [3] omogućuje praćenje lokacije kućnih ljubimaca, pregled povijesti kretanja te analizu podataka. Usluga je dostupna samo unutar SAD-a te za praćenje potrebno je posjedovati poseban uređaj. Praćenje je moguće vršiti preko web aplikacije i mobilnih aplikacija za Android i iOS.

Prema navedenim primjerima, za rad je izabrana aplikacija koja će objediniti sve usluge i ponuditi jedno rješenje. Aplikacija mora omogućiti sučelje za korisnika te sučelje za podatke na koje će se primati lokacijski podatci. Sam odabir uređaja koji će pratiti je ostavljen na izbor korisniku aplikacije (korisnik može pratiti preko mobilnog uređaja, posebnog GPS uređaja). Korisnička aplikacija mora biti web aplikacija koja je dostupna i funkcionalna na svim uređajima od mobilnih uređaja do stolnih računala.

2.2. Specifikacije zahtjeva vlastitog rješenja

Poslije funkcionalnosti, najvažniji zahtjev koji aplikacija mora zadovoljiti je korisničko iskustvo. Pod korisničkim iskustvom podrazumijeva se intuitivno i jednostavno korištenje aplikacije, brzo reagiranje aplikacije na određene događaje te dohvaćanje ključnih podataka u kratkom vremenskom intervalu. Osim toga, aplikacija mora biti prilagodljiva različitim veličinama uređaja, od mobitela pa do stolnog računala.

Tablicom 2.1 prikazani su svi zahtjevi koje aplikacija mora zadovoljiti kako bi se implementacija mogla nazvati uspješnom.

Tab. 2.1 Specifikacije zahtjeva na aplikaciju

Redni broj	Opis zahtjeva
1	Prilagodljivost različitim veličinama uređaja
2	Prikazivanje zadnje lokacije svih praćenih uređaja na karti
3	Mogućnost dodavanja novih uređaja za praćenje
4	Mogućnost mijenjanja svojstava uređaja
5	Mogućnost spremanja podataka o praćenim uređajima u bazi podataka
6	Mogućnost pregleda povijesti praćenih uređaja

2.3. Funkcionalni model aplikacije

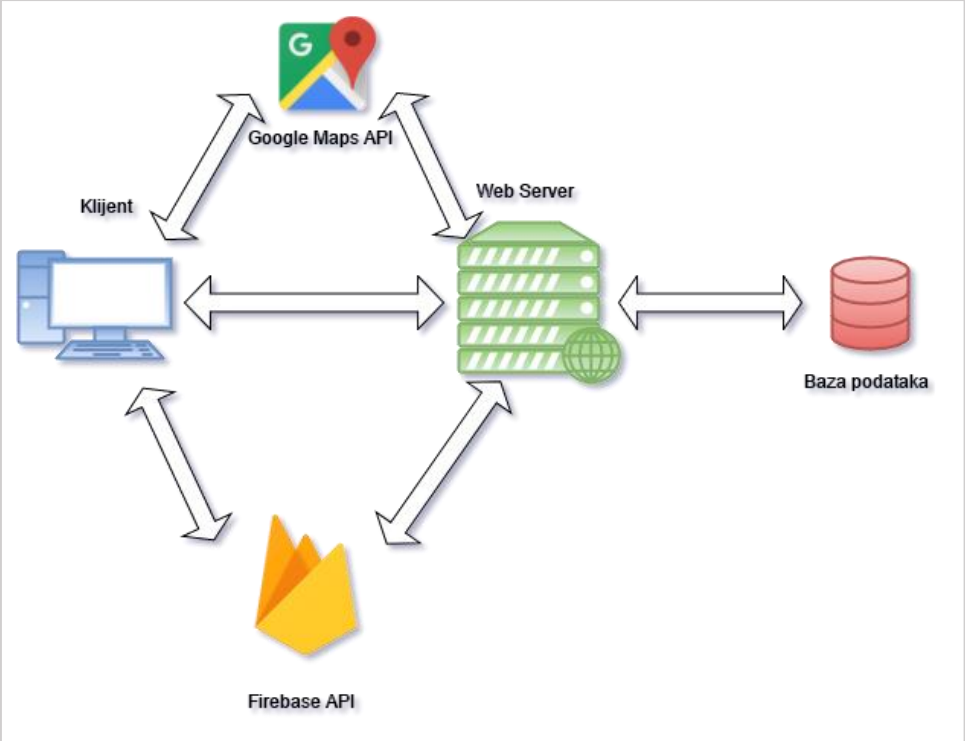
Funkcionalni model web aplikacije za praćenje nestacionarnih objekata sastoji se od korisničkog sučelja (engl. *frontend) i* pozadinskog dijela (engl. *backend*), koji se nalazi na poslužitelju i pruža aplikacijsko sučelje preko kojeg aplikacija može dohvaćati, mijenjati, dodavati i brisati podatke. Uz navedene dijelove, funkcionalni model se sastoji i od usluga koje pružaju treće stranke (Sl. 2.1).

Klijentska strana tj. korisničko sučelje ostvareno je modelom aplikacije s jednom stranicom (engl. *single page application*, SPA) gdje se sva logika prikaza podataka i interakcija ostvaruje kod korisnika u pregledniku. U SPA modelu ne postoji poslužiteljska strana koja generira HTML kôd, već se on generira na klijentskoj strani. Takav princip omogućuje izradu interaktivnih, a prije svega, osjetno bržih aplikacija kod kojih su svi potrebni resursi već unaprijed preuzeti zbog čega ih nije potrebno preuzimati pri posjeti neke nove putanje unutar web aplikacije.

Poslužiteljska strana pruža sučelje preko kojeg je moguće dohvaćati i manipulirati podatcima koji su pohranjeni u bazi podataka. Od krajnjeg korisnika skriven je model baze podataka te sve što on može vidjeti su rezultati poziva na sučelje poslužitelja koji vraća za njega točno određene podatke. Naime, na poslužitelju se provjerava koji korisnik želi pristupiti podatcima te ima li on pravo pristupiti.

Među glavnim dijelovima aplikacije su usluge koje nam nude treće stranke. Za upravljanje korisnicima, njihovu ovjeru autentičnosti i dozvolu pristupa resursima koristi se *Firebase* [4, 5] platforma. Ona se koristi i na klijentskoj i na poslužiteljskoj strani. Na klijentskoj strani omogućuje prijavu korisnika, dok na poslužiteljskoj strani omogućuje ili onemogućuje pristup aplikacijskom sučelju. Druga glavna usluga koji se koristi je *Google Maps JavaScript API* koja omogućuje prikaz

karte i iscrtavanje objekata na karti. Treća usluga koji se koristi je *Google Maps Geocoding API* koja se koristi na poslužiteljskoj strani i omogućuje dobivanje adrese nekog mjesta, ako znamo njegovu geografsku lokaciju (geografsku duljinu i širinu).



Sl. 2.1 Model komunikacije dijelova aplikacije

2.3.1. Poslužiteljski model aplikacije

Poslužiteljsku stranu aplikacije čine aplikacijsko sučelje i baza podataka, koji su usko povezani. Baza podataka izvedena je *code-first* metodologijom, gdje umjesto ručnog kreiranja baze podataka, definiranja tablica, stupaca i atributa, oni se automatski kreiraju iz modela koji je definiran u aplikacijskom kôdu. Taj princip omogućuje bržu izradu aplikacije, manju brigu o samim podatcima, lakši pristup podatcima, mogućnost promjene modela bez gubljenja podataka itd.

U aplikaciji za praćenje nestacionarnih objekata postoje dva glavna tipa podataka: uređaji i lokacije. Uređaji (engl. *devices*) predstavljaju uređaje koji vrše određivanje lokacije i šalju je prema poslužitelju. Oni su sinonim za ono što se prati koje može biti auto, bicikl, čovjek, životinja i sl. Lokacije (engl. *locations*) je tip podataka koji predstavlja vrijeme i mjesto na kojem je određeni uređaj lociran.

Prema ta dva tipa podataka kreirani su modeli *Devices* i *Locations* s atributima zadanim u tablicama 2.2 i 2.3.

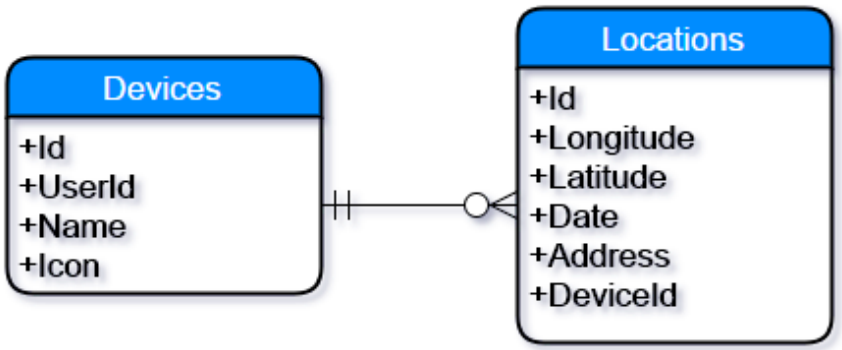
Tab. 2.2 Atributi modela *Devices*

Naziv atributa	Opis atributa
Id	Jedinstveni ključ koji definira uređaj
UserId	ID korisnika kojem pripada uređaj
Name	Ime uređaja
Icon	Ikona koja predstavlja uređaj u korisničkom sučelju

Tab. 2.3 Atributi modela *Locations*

Naziv atributa	Opis atributa
Id	Jedinstveni ključ koji definira lokaciju
Longitude	Geografska duljina na kojoj je izvršeno lociranje
Latitude	Geografska širina na kojoj je izvršeno lociranje
Date	Vrijeme pri kojem je izvršeno lociranje
Address	Adresa gdje je lociran uređaj
DeviceId	ID uređaja koji je lociran

Atribut *UserID* je identifikacijski broj koji definira korisnika kojem pripadaju uređaji tj. osobu koja prati uređaje (ljude, aute, životinje…). On se dobiva iz autoriziranog poziva aplikacijskog sučelja tako što se iz žetona (engl. *token*) izvuku vrijednosti o korisniku koji pristupa sučelju. Pošto se prijava korisnika u sustav obavlja preko *Firebase* usluge, nije potreban model o korisniku, zbog čega je model baze podataka pojednostavljen i ne sadrži mnogo entiteta (UML dijagram baze podataka prikazan je slikom 2.2).



Sl. 2.2 UML dijagram baze podataka

Aplikacijsko sučelje koje nudi poslužitelj klijentu bazira se na dva osnovna podatka (uređaje i lokacije), koji se spremaju u bazu podataka. Ono nudi operacije stvaranja, čitanja, mijenjanja i brisanja tih podataka preko HTTP poziva krajnjih točaka (engl. *endpoints*). Krajnje točke nalaze se pod putanjama „/api/devices“ i „/api/locations“. Pozivi se vrše preko HTTP metoda GET (služi za dobivanje podataka tj. čitanje), POST (kreiranje), PUT (mijenjanje sadržaja) i DELETE (brisanje podatka). Tablica 2.4 prikazuje sve metode koje je moguće pozvati te potrebne parametre za krajnju točku modela *Devices* (isto vrijedi i za krajnju točku *Locations*).

Tab. 2.4 HTTP pozivi za Devices model

Krajnja točka	HTTP metoda	Parametri	Opis
/api/devices	GET	-	Dohvaća sve uređaje koji se prate
/api/devices/{id}	GET	id – jedinstveni broj uređaja	Dohvaća samo jedan uređaj koji odgovara jedinstvenom broju
/api/devices/{id}	DELETE	id – jedinstveni broj uređaja	Briše uređaj koji odgovara jedinstvenom broju
/api/devices	POST	-	Stvara novi uređaj prema svojstvima koje se predaju u tijelu zahtjeva
/api/devices/{id}	PUT	id – jedinstveni broj uređaja	Mijenja svojstva uređaja koji odgovara jedinstvenom broju prema svojstvima koja se predaju u tijelu zahtjeva

2.3.2. Model korisničkog sučelja

Model korisničkog sučelja web aplikacije za praćenje nestacionarnih objekata može se predstaviti na više načina. Jedan od tih načina je u obliku stranica tj. putanja od kojih se sastoji aplikacija. One predstavljaju određeni dio aplikacije u kojem se odrađuje specifična radnja.

Te putanje su:

- / – (početna stranica) - omogućuje pregled praćenih objekata na karti
- /edit/{id} – stranica za promjene svojstava praćenog objekta tj. uređaja (id predstavlja identifikacijski broj uređaja koji se mijenja)
- /add – stranica za dodavanje novog objekta (uređaja) za praćenje
- /login – stranica za prijavu postojećeg korisnika ili stvaranje novog korisnika
- /history/{id} – stranica za pregleda povijesti kretanja (id predstavlja identifikacijski broj uređaja čija se povijest kretanja promatra)
- /about – stranica s informacijama o aplikaciji

3. PROGRAMSKO RJEŠENJE WEB APLIKACIJE

U ovom poglavlju obrađeno je programsko rješenje web aplikacije za praćenje nestacionarnih objekata tj. detaljno je objašnjena implementacija autorizacije, programskog i korisničkog sučelja te su navedene tehnologije, programski alati i jezici koji su se korišteni za izradu aplikacije.

3.1. Korištene tehnologije i alati

Za izradu aplikacije korišteni su razni jezici, protokoli, tipovi podataka, programski okviri i biblioteke (Tab. 3.1).

Tab. 3.1 Korištene tehnologije i alati u web aplikaciji

Programski jezici	• HTML • CSS • JavaScript • C#
Izvedeni programski jezici	• Sass • TypeScript
Protokoli	• HTTP • OAuth 2.0
Tipovi podataka	• JSON • JWT
Programski okviri i biblioteke	• Angular • ASP.NET Core • Angular Material

3.1.1. Programski jezici

Programski jezici koji se koriste mogu se podijeliti u dvije skupine: programski jezici korisničkog sučelja i programski jezik pozadinske usluge. U programske jezike korisničkog sučelja spadaju standardni jezici koji se koriste za izradu stranica i aplikacija na internetu, a oni su HTML, CSS i JavaScript, dok je programski jezik za izradu pozadinske usluge (koji će komunicirati s bazom podataka i klijentskom aplikacijom) C#.

HTML (engl. *Hyper Text Markup Language*) je prezentacijski jezik koji omogućuje prikazivanje dokumenata u internetskim preglednicima. Pomoću njega definiraju se elementi i struktura dokumenta tj. internetske stranice. Baziran je na principima XML-a (engl. *Extensible Markup*

Language), s jasno definiranim oznakama (engl. *tags*) koje se mogu koristiti za prikazivanje i određivanje rasporeda elemenata korisničkog sučelja.

CSS (engl. *Cascading Style Sheets*) je jezik koji opisuje izgled HTML dokumenta. Omogućuje definiranje svojstava (visina, širina, boja, transparentnost, pozicija i mnogi drugi) izgleda elemenata HTML dokumenta.

JavaScript je dinamičan viši programski jezik koji se interpretira pri izvršavanju [6]. To je programski jezik koji podržava više paradigmi pisanja programa (objektno-orijentiran, funkcionalan, imperativan programski jezik). Najčešće se koristi na klijentskoj strani, gdje omogućuje aplikacijama (stranicama) reagiranje na događaje koji se događaju, te omogućuje interakciju i modifikaciju elemenata HTML dokumenta. Osim na klijentskoj strani, JavaScript se koristi i na poslužiteljskoj strani. Baziran je na specifikacijama ECMAScript jezika.

C# je programski jezik koji je dizajniran za aplikacije koje se izvršavaju na .NET programskom okviru (engl. *.NET Framework*) [7]. To je objektno orijentirani programski jezik opće namjene, razvijen od strane Microsofta i baziran na sintaksi i principima koje sadrži programski jezik C++. Osim objektno-orijentirane paradigme podržava i funkcionalno, deklarativno i imperativno programiranje. Prva verzija C# je objavljena 2000. godine.

3.1.2. Izvedeni programski jezici

Izvedeni programski jezici su programski jezici koji se prevode u osnovni jezik od kojeg su nastali. Stvoreni su kako bi popunili nedostatke osnovnom jeziku te dodali nove mogućnosti. Oni su nadogradnja na osnovni jezik pa se može reći da je svaki kôd napisan u osnovnom jeziku u potpunosti kompatibilan s izvedenim jezikom, dok za kôd napisan u izvedenom jeziku ne vrijedi kompatibilnost s osnovnim (kompatibilnost se postiže prevođenjem u osnovni programski jezik).

Sass (engl. *Syntactically Awesome Style Sheets*) je programski jezik koji nadopunjuje CSS. Sass kao jezik podržava dva oblika sintakse. Jedan oblik je sličan izvornoj CSS sintaksi pa dokumenti pisani tim oblikom imaju ekstenziju *.scss*. Drugi oblik je uvučeni oblik i on ne sadrži vitičaste zagrade kako bi se odvojili različiti dijelovi kôda, već se oni odvajaju uvlačenjem kôda pod nadređeni element (ekstenzija je *.sass*). Sass, u odnosu na CSS, omogućuje varijable, gniježđenje elemenata, aritmetičke operacije, funkcije, podjelu kôda, nasljeđivanje i još mnoge druge mogućnosti.

TypeScript je programski jezik koji nadopunjuje JavaScript [8]. To je programski jezik otvorenog kôda koji je razvijan od strane Microsofta koji je postao javno dostupan u listopadu 2012. godine.

Na JavaScript dodaje opcionalne statičke tipove podataka i klasno bazirano objektno-orijentirano programiranje. Najveća prednost pri korištenju TypeScripta, u odnosu na JavaScript, su alati koji omogućuju bolje predviđanje kôda te bolju provjeru i dojavu grešaka koje se pojavljuju u kôdu [9]. Ako napisani TypeScript program nema nikakvih grešaka onda se on uspješno procesom prevođenja prevodi u normalni JavaScript.

3.1.3. Protokoli i tipovi podataka

Protokoli su definirani standardi koji određuju komunikaciju između dva ili više uređaja. U aplikaciji za praćenje nestacionarnih objekata koristi se HTTP protokol za razmjenu podataka tj. komunikaciju između klijenta i poslužitelja te se koristi OAuth 2.0 protokol za autorizaciju korisnika koji koristi aplikaciju.

HTTP (engl. *Hyper Text Transfer Protocol*) je najčešće korišteni protokol na internetu. Koristi se pri razmjeni podataka između poslužitelja i klijenta. Svaka internetska stranica koja se učitava preko web preglednika dohvaćena je HTTP protokolom tako što se napravi zahtjev (engl. *request*) za stranicom, a dobije se odgovor (engl. *response*). Najnovija verzija HTTP protokola je 2.0, a trenutno najkorištenija je verzija 1.1. Protokol funkcionira na principu zahtjeva i odgovora, gdje zahtjevi mogu biti od dobivanja sadržaja (GET) do brisanja sadržaja (DELETE). Na poslužitelju HTTP zahtjevi dolaze na ulaz (engl. *port*) 80, ako se radi o nesigurnoj vezi, dok sigurni zahtjevi dolaze na ulaz 443.

OAuth 2.0 protokol je protokol koji je baziran na HTTP protokolu i određuje načine na koje se razmjenjuju podatci o korisničkom računu [10, 11]. Koristi se kada želimo na siguran način prijaviti se na neku aplikaciju ili stranicu koristeći korisnički račun od treće strane (Facebook, Google, Twitter). Protokol definira više tokova (engl. *flows*) kojima je moguće dobiti podatke o korisniku. Glavni dio komunikacije čine žetoni koji mogu sadržavati informacije o korisniku, životnom vijeku žetona, tko je izdao žeton i na koji način su ti podatci zaštićeni.

JSON (engl. *JavaScript Object Notation*) je tip podatka za razmjenu podataka koji je baziran na tipu podatka za objekt iz programskog jezika JavaScript. Odlikuje ga to jednostavan oblik koji je jednostavan čovjeku za pisanje i čitanje, a uz to lagan računalu za generiranje i raščlanjivanje (engl. *parsing*). U odnosu na ostale tipove podataka koji se koriste u komunikaciji na internetu kao što je XML, JSON je dosta brži jer ne sadrži elemente za opis koji čine podatak većim pa time i sporijim za prijenos.

JWT (engl. *JSON Web Token*) tip je podatka koji predstavlja žeton i koristi se u razmjeni korisničkih informacija u OAuth protokolu [12]. Baziran je na JSON tipu podatka i sadrži tri dijela:

zaglavlje (engl. *header*), tijelo (engl. *payload*) i potpis (engl. *signature*). U zaglavlju nalaze se informacije o tipu žetona te tip funkcije za izračunavanje sažetka (engl. *hashing function*) pri generiranju potpisa. U tijelu žetona se nalaze informacije o korisniku, životnom vijeku žetona te tko je napravio žeton. Potpis je proizvod funkcije za izračunavanje sažetka kojoj su inputi zaglavlje, tijelo i tajna lozinka.

3.1.4. Programski okviri i biblioteke

Za jednostavniju izradu koriste se razni programski okviri koji imaju svoje načine kako se stvaraju aplikacije. Tako se u aplikaciji za praćenje nestacionarnih objekata za poslužiteljsku stranu koristi ASP.NET Core programski okvir, a za klijentsku stranu Angular.

ASP.NET Core programski je okvir namijenjen za stvaranje modernih Internet aplikacija poput web aplikacija i aplikacije za Internet objekata koje je moguće koristiti na različitim operacijskim sustavima (Windows, Linux, Mac OS) [13]. ASP.NET Core aplikacije se pokreću na .NET Core programskom okviru (nasljednik .NET programskog okvira) ili na .NET programskom okviru. Glavna prednost ASP.NET Core aplikacija njihova je modularna struktura i optimiziranost za rad kao usluga u oblaku. ASP.NET Core programski je okvir otvorenog kôda kojeg razvija Microsoft.

Angular programski je okvir koji služi za stvaranje kompleksnih klijentskih aplikacija na internetu [14]. Odlikuje ga struktura koja je bazirana na komponentima koje omogućuju ponovnu upotrebljivost određenog dijela korisničkog sučelja bez potrebe za pisanjem novog kôda. Sa svojom sintaksom, koja nadopunjuje HTML, omogućuje vezivanje HTML elemenata s programskom logikom koja je definirana u JavaScriptu. Angular, kao i TypeScript na kojem je baziran, otvorenog je kôda, a primarno razvijan od strane Googleovih inženjera.

Angular Material je biblioteka koja pruža skup komponenti za Angular baziranih na *Material Design* komponentama [15]. Među tim komponentama su komponente za prikaz korisničkog sučelja kao što su gumbi, polja za unos teksta, polja za odabir i slično. Angular Material pruža bržu izradu aplikacija s komponentama koje imaju unificirani izgled.

3.2. Implementacija autoriziranja i autentificiranja

3.2.1. Autentificiranja

Pod pojmom autentificiranje podrazumijeva se određivanje korisnika koji je prijavljen u sustav. Autentikacija se obavlja preko Firebase usluge koju je potrebno konfigurirati. Naime, potrebno je prijaviti se u Firebase Console, napraviti novi projekt, uključiti autentificiranje te dodati Firebase

konfiguraciju u aplikaciju. Na slici 3.1 prikazan je konfiguracijski objekt potreban za korištenje Firebase sustava.

```
firebase: {
  apiKey: 'AIzaSyD-R3PbukFn_T4zNSSRkKUaRZyGxYFYRMY',
  authDomain: 'maps-e8b5a.firebaseapp.com',
  databaseURL: 'https://maps-e8b5a.firebaseio.com',
  projectId: 'maps-e8b5a',
  storageBucket: 'maps-e8b5a.appspot.com',
  messagingSenderId: '599497429884'
}
```

Sl. 3.1 Firebase konfiguracija

Sustav Firebase potrebno je inicijalizirati pri pokretanju aplikacije. To se radi u datoteci *app.module.ts*, gdje se u glavni modul Angular aplikacije dodaje Firebase modul i inicijalizira se s Firebase konfiguracijom koju smo dobili u Firebase Console pri kreiranju projekta (Sl. 3.2).

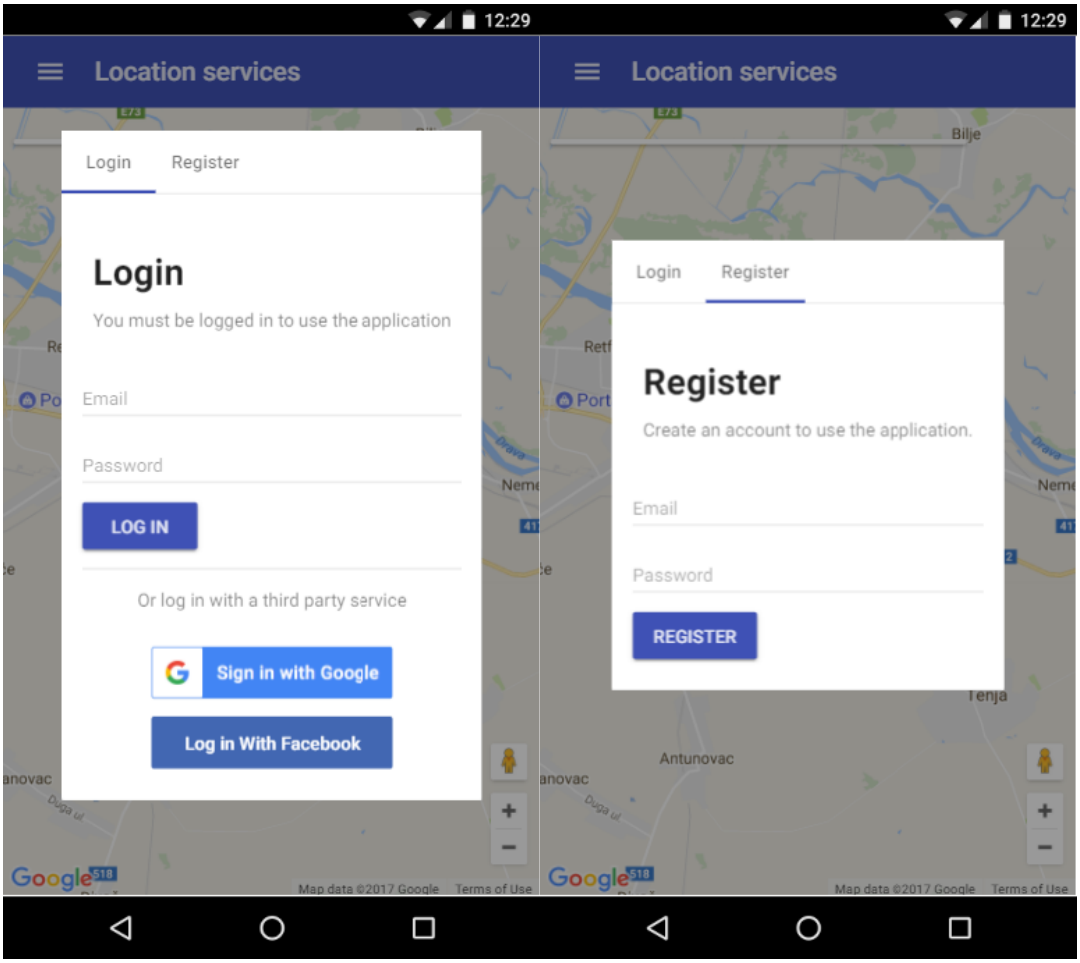
```
import { AngularFireModule } from 'angularfire2';
import { AngularFireAuthModule } from 'angularfire2/auth';

@NgModule({
  imports: [
    AngularFireModule.initializeApp(environment.firebase),
    AngularFireAuthModule
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Sl. 3.2 Inicijalizacija Firebase modula

Sustav Firebase pruža uslugu za upravljanje prijava korisnika koja je bazirana na OAuth 2.0 protokolu i *OpenID Connect* standardu. Korištenjem sustava Firebase za autentifikaciju omogućena je prijava u sustav s više opskrbljivača (engl. *providers*), kao što su Google, Facebook, Twitter i Github, te je, osim njih, omogućena je prijava putem računa e-pošte, brojem mobilnog telefona i anonimna prijava, gdje korisnik može koristiti sustav dok ne odluči napraviti pravi korisnički račun. Iako je prijavu preko trećih opskrbljivača moguće napraviti bez Firebase sustava, on je ipak korišten jer objedinjuje više opskrbljivača, nudi centralizirano mjesto za pohranu korisničkih računa i pruža jednostavno sučelje za prijavu postojećih i registraciju novih korisnika.

Prijava u aplikaciju omogućena je na tri načina: Google prijava, Facebook prijava i prijava preko korisničkog računa koji je napravljen s email računom i lozinkom. Logika za prijavu korisnika i

korisničko sučelje (Sl. 3.3) je implementirana komponentom *LoginComponent*. Komponenta sadrži četiri metode (*loginWithGooogle*, *loginWithFacebook*, *loginAccount* i *registerAccount*) koje se pozivaju pritiskom na dugmad za prijavu ili registraciju korisnika. U slučaju greške pri prijavi ili registraciji ispisuje se odgovarajuća poruka koja obavještava korisnika o problemu (već postojeći račun, nevaljana lozinka i sl.).



Sl. 3.3 Korisničko sučelje za prijavu i registraciju korisnika

Nakon prijave korisnika, aplikacija dobiva informacije o korisniku koje se prikazuju na korisničkom sučelju. Ukoliko među dobivenim informacijama ne nalazi se ime korisnika prikazuje se njegov email, a u slučaju ne postojanja profilne slike prikazuje se zadana slika koja označava korisnika.

3.2.2. Autoriziranje

Autorizacija je postupak provjere pristupa određenom resursu tj. ima li adekvatna prava. Komponenta *AppComponent* sadrži autorizacijsku logiku koja onemogućuje pristup neovlaštenim putanjama aplikacije u slučaju kada korisnik nije prijavljen tako što ga preusmjeri na stranicu za prijavu.

Kada aplikacija želi pristupiti resursima, kao što je lista praćenih uređaja, koji se nalaze na poslužitelju, potrebno je uz zahtjev priložiti žeton preko kojeg se on autorizira. Prilikom slanja HTTP zahtjeva potrebno je u *Authorization* zaglavlje postaviti *idToken* žeton trenutno prijavljenog korisnika. Taj žeton dobiva se pozivajući metodu *getToken* Firebase biblioteke, koja će dohvatiti žeton iz memorije ili u slučaju da je on istekao pozvati Firebase uslugu i zatražiti novi žeton. Autorizacijsko zaglavlje treba sadržavati ključnu riječ *Bearer*, tako da u konačnici zaglavlje ima oblik: „Bearer token“, gdje je *token* tekst koji sadrži žeton.

Nakon slanja zahtjeva s autorizacijskim zaglavljem, na poslužitelju je potrebno provjeriti valjanost žetona. Žeton je ispravan ako nije istekao, ako je ispravan izdavatelj i primatelj te ako je potpis ispravan. Proces preko kojeg se provjerava ispravnost je definiran *OpenID Connect* standardom, a svodi se na to da onaj koji provjerava žeton od izdavatelja žetona zatraži ključ preko kojeg može generirati potpis ili on poziva uslugu izdavatelja žetona koja odgovara je li žeton ispravan.

Na slici 3.4 prikazan je kôd koji će zatražiti ključ od izdavatelja žetona pri pokretanju poslužiteljske aplikacije te ga kasnije koristiti za provjeru ispravnosti žetona.

```
app.UseJwtBearerAuthentication(new JwtBearerOptions
{
    AutomaticAuthenticate = true,
    IncludeErrorDetails = true,
    Authority = "https://securetoken.google.com/maps-e8b5a",
    TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuer = true,
        ValidIssuer = "https://securetoken.google.com/maps-e8b5a",
        ValidateAudience = true,
        ValidAudience = "maps-e8b5a",
        ValidateLifetime = true,
    },
});
```

Sl. 3.4 Provjera ispravnosti žetona

Krajnje točke (u ASP.NET Core okruženju nazivaju se kontroleri) koje imaju atribut *Authorize* će biti podvrgnute provjeri ispravnosti žetona te u slučaju neispravnosti vratit će HTTP statusni kôd 401 *Unauthorized*.

3.3. Implementacija aplikacijskog programskog sučelja

Implementacija programskog sučelja se vrši ASP.NET Core programskim okvirom i programskim jezikom C#. Implementacija čine dva glavna tipa komponenti: modeli i kontroleri.

Modeli se definiraju kao klasa te iz njenih svojstava će biti napravljen model baze podataka s pripadajućim svojstvima. Na slici 3.5 je prikazana klasa *Location* koja definira model lokacije na kojoj se nalazi uređaj.

```
public class Location
{
    public int Id { get; set; }

    [Required]
    public float Longitude { get; set; }

    [Required]
    public float Latitude { get; set; }

    public DateTime Date { get; set; }

    public string Address { get; set; }

    [Required]
    public int DeviceId { get; set; }

    public virtual Device Device { get; set; }
}
```

Sl. 3.5 C# klasa koja definira model lokacije

Kako bi ASP.NET Core programski okvir znao od kojih modela je potrebno napraviti tablice u bazi podataka potrebno ih je definirati, a to je ostvareno u datoteci pod nazivom *Context.cs* (glavni sadržaj je prikazan na slici 3.6). U toj datoteci su definirana imena tablica koje će biti stvorene na osnovu modela i definiran je konstruktor koji za parametar prima *DbContextOptions* preko kojeg se određuje kojoj se bazi podataka pristupa.

```
public class Context : DbContext
{
    public Context(DbContextOptions<Context> options) : base(options)
    {
    }
    public virtual DbSet<Device> Devices { get; set; }
    public virtual DbSet<Location> Locations { get; set; }
}
```

Sl. 3.6 Definiranje modela baze podataka

Nakon toga potrebno je aplikaciji koja će pružati programsko sučelje reći koju bazu podataka će koristiti (Sl. 3.7.). To se vrši u datoteci *Startup.cs* koja se prva pokreće pri pokretanju aplikacije.

```
  services.AddDbContext<Context>(options ⇒
      options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));
```

Sl. 3.7 Definiranje koja baza podataka će se koristiti

Drugi glavni dio koji čini programsko sučelje su kontroleri. Aplikacija sadrži tri kontrolera, a oni su *DevicesController*, *LocationsController* i *HistoryController*. To su C# klase s metodama koje predstavljaju krajnje točke pozadinskog aplikacijskog sučelja. Te metode imaju parametre i atribute koji odgovaraju metodama HTTP poziva. Na slici 3.8 su prikazane definicije metoda *DevicesController* kontrolera.

```
// GET: api/Devices
[HttpGet]
public IEnumerable<Device> GetDevices()
{…
}

// GET: api/Devices/5
[HttpGet("{id}")]
public async Task<IActionResult> GetDevice([FromRoute] int id)
{…
}

// PUT: api/Devices/5
[HttpPut("{id}")]
public async Task<IActionResult> PutDevice([FromRoute] int id, [FromBody] Device device)
{…
}

// POST: api/Devices
[HttpPost]
public async Task<IActionResult> PostDevice([FromBody] Device device)
{…
}

// DELETE: api/Devices/5
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteDevice([FromRoute] int id)
{…
}
```

Sl. 3.8 Metode koje sadrži *DevicesController*

Unutar svake metode postoji odgovarajuća logika koja se brine za pravilan rad s podatcima, pa tako postoji logika za dohvaćanje jednog ili svih podataka, mijenjanje, brisanje i dodavanje podataka.

3.4. Implementacija korisničkog sučelja

Korisničko sučelje je izvedeno korištenjem programskog sučelja Angular (verzija 4) i programskim jezikom TypeScript. Za izgled korisničkog sučelja prema smjernicama *Material Design* korištena ja biblioteka Angular Material, koja nudi pregršt u potpunosti izrađenih komponenti kao što su dugmadi, elementi za unos podataka, padajuće liste, indikatori progresa i mnogi drugi.

Arhitektura Angular programskog okvira bazirana je na komponentima, odnosno na međusobno neovisnim cjelinama kôda koje obavljaju točno jednu radnju. Takva arhitektura omogućuje ubrzan razvoj aplikacija jer smanjuje potrebu za pisanjem nepotrebnog kôda. Nakon što se jednom napravi komponenta koja obavlja određenu funkciju, nju je moguće na jednostavan način koristiti u drugim komponentama.

Angular projekt je napravljen uz pomoć Angular CLI (engl. *Command Line Interface*) sučelja. Kako bi se mogao koristiti Angular CLI na računalu treba biti instalirana *NodeJS* platforma. Za stvaranje novog Angular projekta u komandnoj liniji je potrebno pokrenuti naredbu „ng new“, koja će stvoriti novi projekt unutar direktorija gdje je pokrenuta naredba. Osim naredbe za stvaranje novog projekta, Angular CLI nudi i naredbe za pokretanje, testiranje i kreiranje konačne verzije te naredbe za stvaranje komponenti, usluga, direktiva, klasa i modula.

Ulazna točka Angular projekta je *main.ts* datoteka gdje se pokreće glavni modul aplikacije (Sl. 3.9).

```
platformBrowserDynamic().bootstrapModule(AppModule);
```

Sl. 3.9 Pokretanje glavnog modula aplikacije

Glavni modul, *AppModule*, definira sve komponente, usluge i module koji se koriste unutar aplikacije. Osim toga, u njemu se proglašava korijenska komponenta (*AppComponent*) aplikacije od koje će se sve ostale komponente granati. Na slici 3.10 je prikazan kôd koji definira modul *AppModule*.

```
@NgModule({
  declarations: [
    AppComponent,
    LoginComponent,
    AddDeviceComponent,
    ListDevicesComponent,
    EditDeviceComponent,
    MainComponent,
    HistoryComponent,
    DeleteDialogComponent
  ],
  entryComponents: [DeleteDialogComponent],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    MaterialModule,
    AppRoutingModule,
    BrowserAnimationsModule,
    AgmCoreModule.forRoot({
      apiKey: 'AIzaSyCKb-hVu7N2XQHpq2dR-mWOTaqSxfOjP7U'
    }),
    AngularFireModule.initializeApp(environment.firebase),
    AngularFireAuthModule,
    StoreModule.provideStore({ 'app': appReducer, 'devices': devicesReducer })
  ],
  providers: [DevicesService, HistoryService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Sl. 3.10 Definicija modula *AppModule*

3.4.1. Komponente

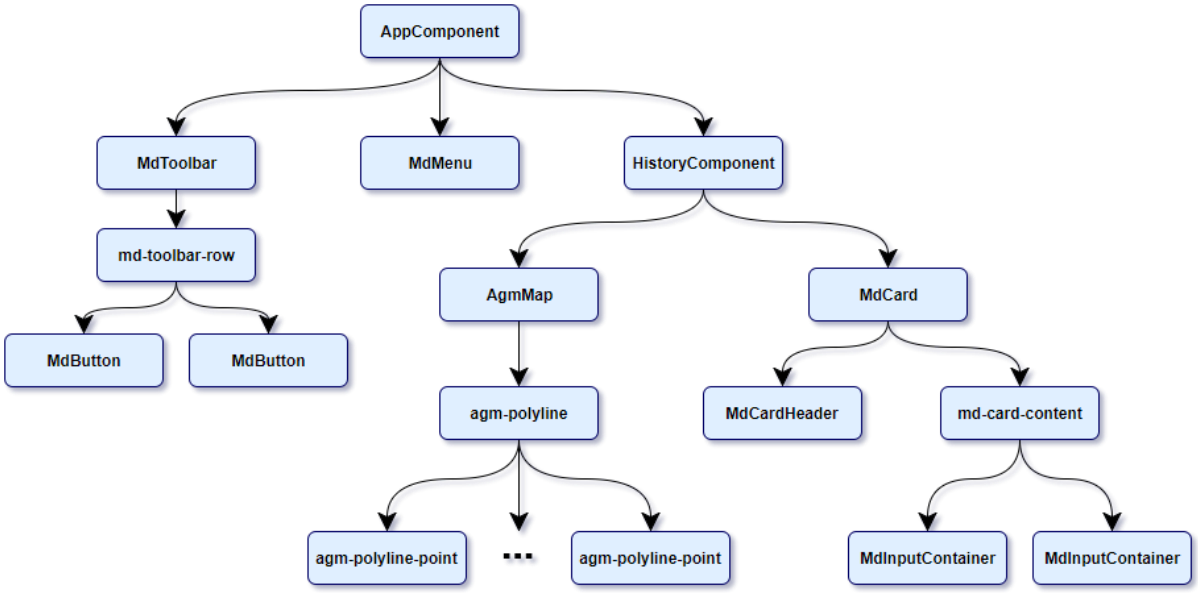
Komponente su osnovne građevinske jedinice Angular aplikacije. One spajaju izgled i funkcionalnost određenog dijela korisničkog sučelja u jednu cjelinu. Jedna komponenta može sadržavati više drugih komponenti, ali ne može sadržavati sama sebe. Komponente mogu primati podatke iz drugih komponenti preko definiranih ulaznih atributa. Slanje informacija prema vani komponente obavljaju odašiljanjem događaja (druge komponente moraju pratiti događaj za primanje podataka).

Osnovnu strukturu aplikacije čine aplikacijske komponente te komponente koje nude biblioteke Angular Material i Angular Google Maps. Aplikacijske komponente su:

- *AboutComponent* – komponenta koja prikazuje informacije o aplikaciji
- *AddDeviceComponent* – komponenta koja služi za dodavanje novog uređaja za praćenje
- *AppComponent* – korijenska komponenta koja sadrži alatnu traku, logiku o prijavi i odjavi korisnika te omogućuje prikaz elemenata ovisno o putanji aplikacije

- *DeleteDialogComponent* – komponenta koja se pokazuje pri pokušaju brisanja uređaja, upozorava korisnika na akciju brisanja
- *EditDeviceComponent* – komponenta koja služi za mijenjanje podataka o uređaju
- *HistoryComponent* – komponenta koja na karti prikazuje povijest kretanja uređaja, omogućuje odabir perioda vremena unutar kojega se praćeni objekt kretao
- *ListDevicesComponent* – komponenta koja prikazuje sve uređaje koji se prati, omogućuje akcije brisanja i mijenjanja nad uređajima te dodavanje novog uređaja
- *LoginComponent* – komponenta koja se prikazuje ukoliko korisnik nije prijavljen, omogućuje korisniku prijavu preko emaila i lozinke ili preko Google ili Facebook korisničkog računa, omogućuje i registraciju korisnika preko emaila i lozinke
- *MainComponent* – komponenta koja prikazuje kartu i listu uređaja te omogućuje prikaz komponenti za dodavanje i mijenjanje uređaja

Gniježđenjem komponenti unutar aplikacije nastaje stablo komponenti koje počinje od korijenske komponente. Na slici 3.11 prikazano je stablo komponenti za putanju koja prikazuje povijest kretanja praćenog uređaja. Vidljivo je grananje korijenske komponente na naslovnu traku i komponentu za prikaz povijesti. *HistoryComponent* dalje se grana na komponentu za prikaz karte te komponentu koja prikazuje informacije o praćenom uređaju i vremenu unutar kojeg je prikazana povijest.

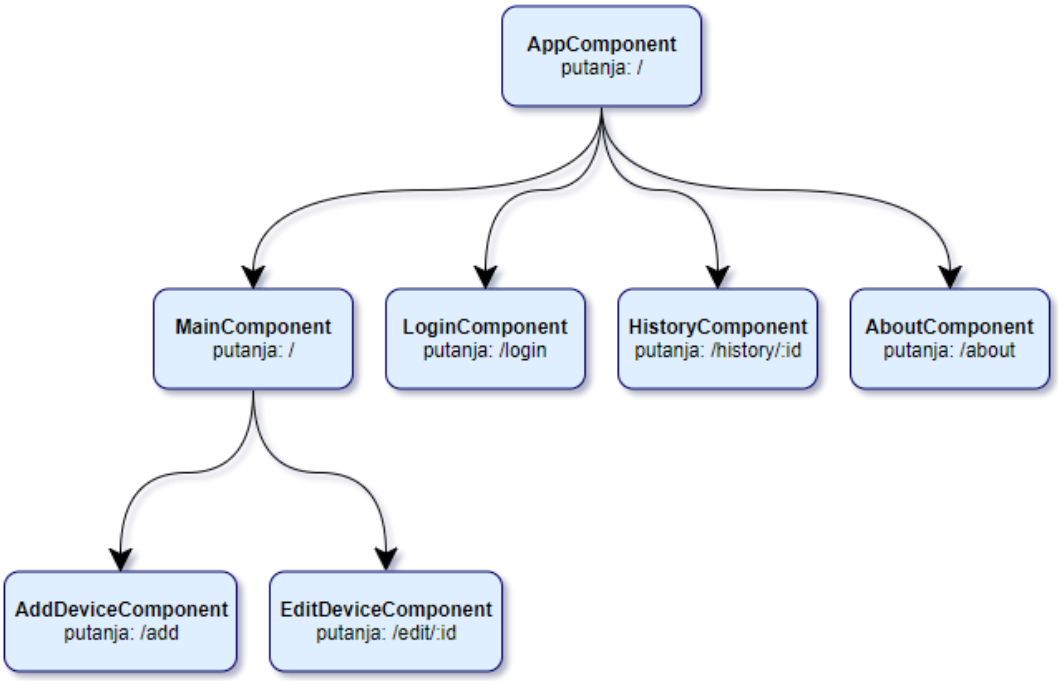


Sl. 3.11 Stablo komponenti za prikaz povijesti kretanja

3.4.2. Putanje

Zbog toga što jedna komponenta može imati više komponenti nastaje stablo komponenti, koje započinje od korijenske komponente. Podgrupa stabla komponenti je stablo putanja, koje sadrži samo one komponente koje se mijenjaju pri promjeni putanje.

Na slici 3.12 prikazano je stablo putanja aplikacije iz koje se jasno vidi hijerarhija komponenata i putanja. *AppComponent* i *MainComponent* imaju istu putanju što znači da će pri toj putanji biti prikazan *MainComponent* tj. sučelje koje prikazuje kartu sa uređajima koje se prate.



Sl. 3.12 Stablo putanja aplikacije

Putanje su definirane u datoteci *app-routing.module.ts* u modulu pod nazivom *AppRoutingModule* koji je uvezen u modul *AppModule* kako bi aplikacija znala o putanjama koje se nalaze unutar aplikacije. U slučaju da korisnik zatraži putanju koja ne postoji biti će preusmjeren na početnu putanju.

3.4.3. Usluge

Usluge u *Angular* okruženju su obične klase koje imaju svrhu obavljati jednu specifičnu radnju. To su najčešće kompliciranije radnje koje ne bi trebale biti u komponentima, već komponenti ovise o njima, pa zbog toga su komponente najčešći korisnici usluga. Oblici usluga koji mogu biti su usluge za razmjenu podataka, usluge za zapis informacija, usluge za konfiguraciju aplikacije i sl.

Ne uzimajući u obzir usluge koje dolaze s programskim okvirom Angular i ostalim bibliotekama koje se koriste u završnom radu, preostaju dvi usluge za razmjenu podataka: *DevicesService* i *HistoryService*. *DevicesService* usluga sadrži metode za dohvaćanje, brisanje i mijenjanje uređaja, a koriste ga komponente *AppComponent, AddDeviceComponent* i *EditDeviceComponent*. S druge strane, *HistoryService* usluga ima samo jednu metodu koja dohvaća povijest kretanja za određeni uređaj i njega koristi komponenta za prikazivanje povijesti.

Neke od važnijih usluga koja je potrebno spomenuti su *Http* (usluga pomoću koje se vrši komunikacija s pozadinskim sučeljem), *AngularFireAuth* (usluga koju pruža biblioteka Angular Firebase te ona omogućuje razne operacije potrebne za prijavu, odjavu i provjeru korisnika) te usluga *Store* (usluga koja omogućuje spremanje i dohvaćanje podataka korištenih unutar aplikacije).

3.4.4. Upravljanje stanjem aplikacije

Za upravljanje stanjem u aplikaciji potrebno je značajne podatke koji se koriste na više mjesta unutar aplikacije napraviti dostupnim s jednog centraliziranog mjesta. Nadalje, potrebno je omogućiti jednostavnu promjenu stanja i ažuriranje komponenti koji ga koriste. Za upravljanje stanjem odabrana je biblioteka Store [16] koja je bazirana na konceptima koji su popularizirani bibliotekom Redux. Osim toga, Store je baziran na načelima biblioteke RxJS (engl. *Reactive Extensions for JavaScript*) koji omogućuju jednostavno korištenje i obradu podataka unutar asinkronih programskih okruženja i okruženja baziranih na događajima.

Store se odlikuje po tome što podatci o stanju putuju u samo jednom i ne postoji mogućnost nastajanja kolizija prilikom upisivanja podataka. Podatci teku kružno unutar aplikacije, od mjesta gdje se mijenjanju do mjesta gdje se koriste. Za to su zaslužna tri ključna dijela biblioteke: skladište (engl. *store*, može se smatrati klijentskom bazom podataka gdje su pohranjeni i dostupni svi podatci na jednom mjestu), reduktori (engl. *reducers*, to su čiste funkcije, odnosno funkcije bez posljedica, koje kao parametre primaju prošlo stanje i akciju koju je potrebno napraviti nad trenutnim, a kao rezultat vraćaju novo stanje u kojem će se aplikacija nalaziti) i akcije (engl. *actions*, akcijom definiramo koja promjena će biti napravljena nad trenutnim stanjem). Odlike takvog načina upravljanja stanja aplikacije su centralizirano mjesto podataka, brže i pravilnije ažuriranje podataka jer komponente dobivaju obavijest o promijeni te olakšan proces razvoja aplikacije koji smanjuje potrebu za razmišljanjem u kojem dijelu aplikacije se nalaze potrebni podatci.

Za ovaj završni rad potrebno je pamtiti dva stanja unutar aplikacije: poziciju s geografskom širinom i dužinom pri kojoj je centrirana karta te kolekciju podataka o svim uređajima koji su praćeni. Sukladno tome, definirana su dva stanja unutar modula *AppModule* s nazivima *app* i *devices* koji koriste reduktore *appReducer* i *devicesReducer* (Sl. 3.13).

```
StoreModule.provideStore({ 'app': appReducer, 'devices': devicesReducer })
```

Sl. 3.13 Definiranje stanja aplikacije u modulu *AppModule*

Biblioteka Store pruža dvije ključne metode: *select* (omogućuje dohvaćanje podataka iz skladišta, kao parametar prima naziv stanja koje se želi dohvatiti) i *dispatch* (pokreće akciju koja će biti izvršena u reduktorima s opcionalnim dodatnim podatcima). Akcije koje se mogu izvršiti su postavljene ključnim riječima *SET_POSITION* i *SET_DEVICES* te obje akcije primaju dodatni sadržaj (engl. *payload*). Na slici 3.14 je prikazan dio kôda koji šalje novu listu uređaja u skladište.

```
this.store.dispatch({ type: SET_DEVICES, payload: devices });
```

Sl. 3.14 Postavljanje nove liste uređaja

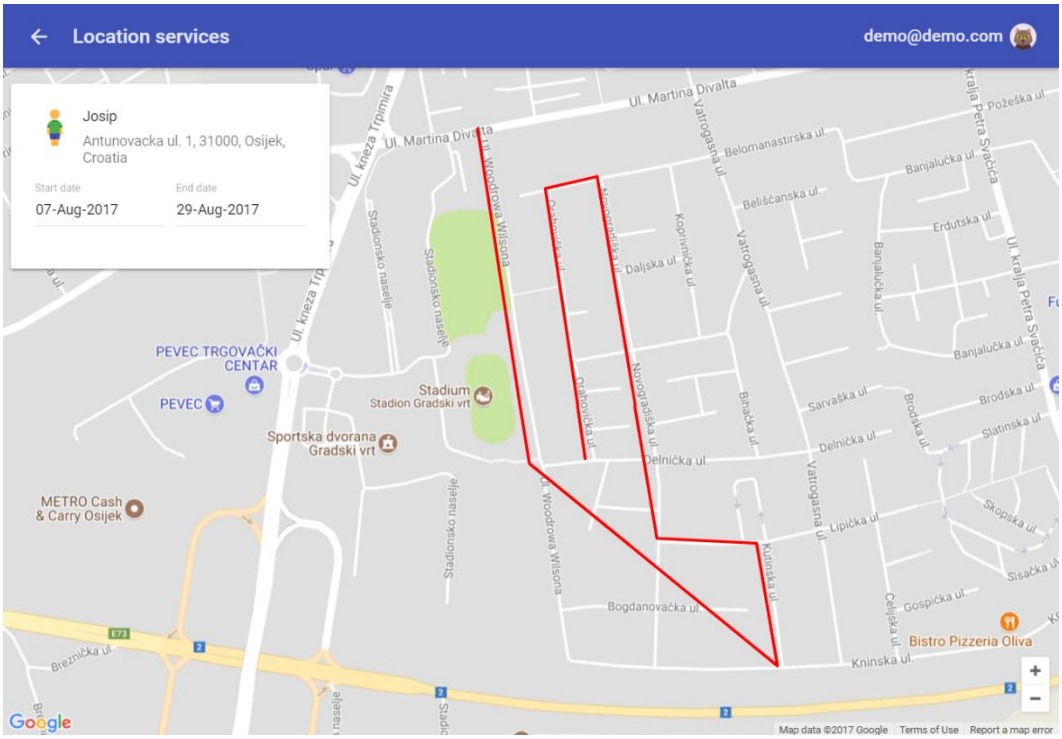
4. UPOTREBA I TESTIRANJE WEB APLIKACIJE

U ovom poglavlju je opisano na koji način je provedeno testiranje web aplikacije te koji su rezultati testiranja. Testirane su značajke korisničkog sučelja poput prilagodljivosti veličini ekrana i ispravnosti podataka te je testirana brzina učitavanja aplikacije i uspoređene veličine datoteka konačne produkcijske klijentske aplikacije. Osim testiranja, navedene su i upute za korištenje web aplikacije.

4.1. Upute za korištenje web aplikacije

Nakon što se korisnik prijavi u sustav, prezentiran je sa sučeljem koje omogućuje pregled pozicija praćenih uređaja. Sučelje je popunjeno podatcima kojima korisnik ima pristup, nad kojima može činiti razne akcije. Na karti su prikazani svi praćeni objekti s trenutačnim pozicijama, predstavljeni ikonom koja je određena pri definiranju uređaja. Sučelje se sastoji od tri glavna dijela: lista praćenih uređaja, karta i naslovna traka.

Pritiskom na uređaj unutar liste praćenih uređaja vrši se centriranje karte na trenutnu poziciju tog uređaja. Svaki od uređaja na listi ima i dugme predstavljeno s tri vertikalne točkice koje otvara padajući izbornik za odabrani uređaj. Padajući izbornik sadrži dugmad „*Show history*“, „*Edit*“ i „*Delete*“ koji prikazuju povijest uređaja, otvaraju prozore za mijenjanje i brisanje uređaja. Povijest kretanja (Sl. 4.1) označena je crvenom bojom na karti i predstavlja putanju kojom se praćeni objekt kretao.



Sl. 4.1 Prikaz povijesti kretanja

Prikaz povijesti kretanja, osim karte, sadrži karticu na kojoj piše za koji se uređaj promatra povijest kretanja i unutar kojeg razdoblja. Kartica sadrži dva elementa za unos vremena koji se odnose na početni i krajnji datum razdoblja povijesti kretanja. Mijenjanjem datuma karta će se automatski osvježiti s novom putanjom za određeno vremensko razdoblje.

Lista praćenih uređaja pri vrhu sadrži traku na kojoj se nalazi dugme za dodavanje novog uređaja i dugme za skrivanje ili prikazivanje liste. Pritiskom na dugme „*Add device*“ otvara se prozor koji omogućuje dodavanje novog uređaja ukoliko se unesu potrebni podatci (sastoji se od elemenata za unos podataka te dugmadi za spremanje uređaja i odustajanja od radnje). Prozor za mijenjanje uređaja sadrži u potpunosti iste elemente kao i prozor za dodavanje uređaja, a razlikuju se samo u funkcionalnosti.

Naslovna traka sastoji se od naslova i ikone aplikacije s lijeve strane te imena korisnika i njegove profilne slike s desne strane. Ikona aplikacije mijenja se u dugme sa strelicom prema lijevo ukoliko se aplikacija ne nalazi na početnoj putanji i omogućuje vraćanje nazad. Na desnoj strani, prilikom pritiska na profilnu sliku otvara se padajući izbornik s akcijama „*About*“ i „*Log out*“. Pritiskom na „*Log out*“ korisnik se odjavljuje iz aplikacije i vraća na stranicu za prijavu.

4.2. Provjera specifikacije zahtjeva

U poglavlju 2.1. pod tablicom 2.1 su navedene specifikacije koje web aplikacija mora ostvariti kako bi se smatrala u potpunosti gotovom. Nakon testiranja i provjere svih elemenata specifikacije utvrđeno je da su svi zahtjevi specifikacije ostvareni.

Tab. 4.1 Provedeno testiranje specifikacije zahtjeva

Redni broj	Opis zahtjeva	Rezultat testiranja
1	Prilagodljivost različitim veličinama uređaja	DA
2	Prikazivanje zadnje lokacije svih praćenih uređaja na karti	DA
3	Mogućnost dodavanja novih uređaja za praćenje	DA
4	Mogućnost mijenjanja svojstava uređaja	DA
5	Mogućnost spremanja podataka o praćenim uređajima u bazi podataka	DA
6	Mogućnost pregleda povijesti praćenih uređaja	DA

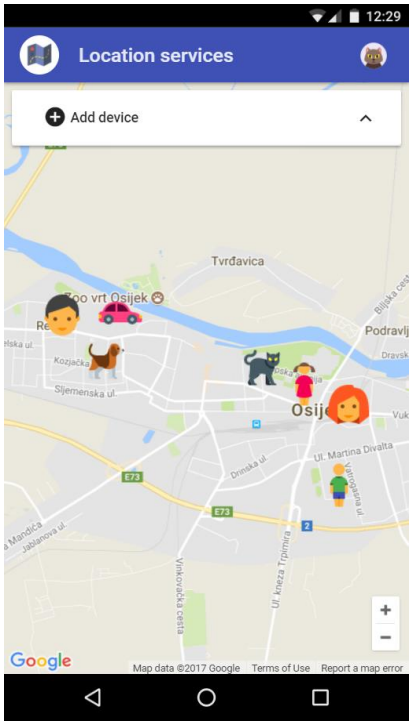
Svih šest testova provedeno je na sljedeće načine:

1. Povećavanjem i smanjivanjem prozora preglednika

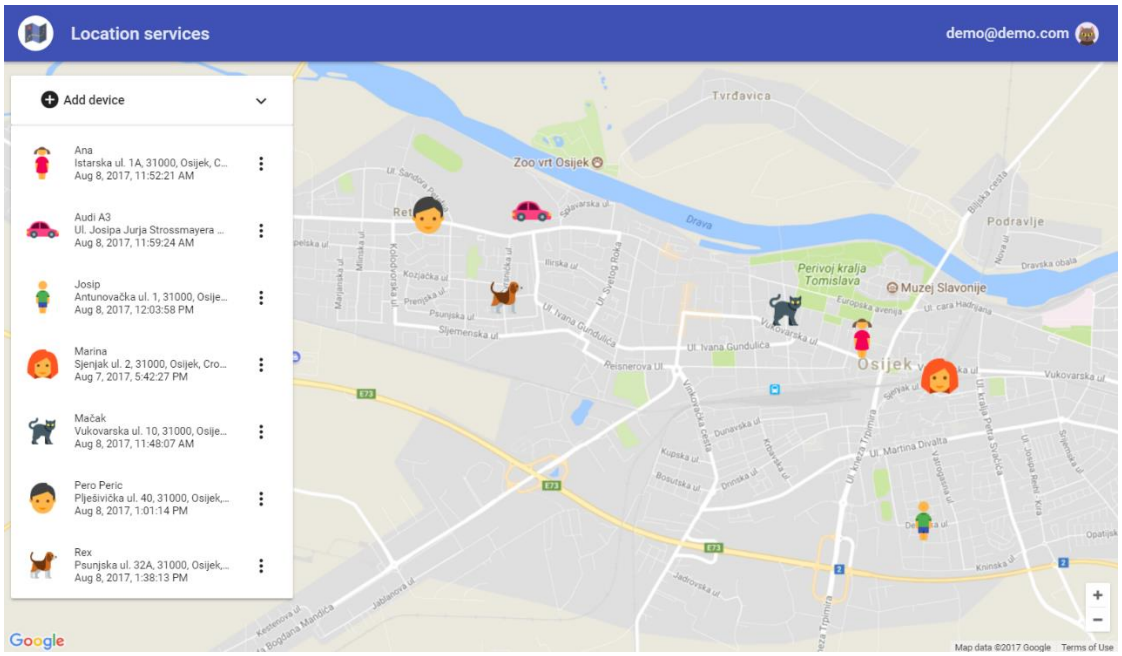
2. Dodana je nova lokacija za jedan uređaj te je provjereno da li je se pojavila na listi
3. Popunjavanjem i slanjem forme za dodavanje novog uređaja
4. Mijenjanjem svojstava unutar forme za promjenu
5. Provjerena ažurnost podataka u bazi podataka
6. Provjeren prikaz povijesti kretanja na karti

4.2.1. Prilagodljivost sučelja različitim uređajima

U današnjem svijetu gdje većina posjeta internetskim stranicama dolazi s mobilnih uređaja jako je važno imati korisničko sučelje koje je intuitivno i praktično za korištenje na uređajima s manjim zaslonima. Na slikama 4.2 i 4.3 prikazana su korisnička sučelja za mobilni uređaj i za uređaje koji su veličinom veći od mobilnih uređaja. Iz slika je vidljiva prilagodljivost sučelja različitim veličinama, lista praćenih uređaja na većim ekranima se pomjera na lijevu stranu kako ne bi zauzimala previše prostora i omogućila intuitivnije korištenje. Također, lista se može i sakriti ili prikazati što je jako korisno na mobilnim uređajima. Nadalje, na mobilnim uređajima na naslovnoj traci se ne prikazuje ime korisnika koji je prijavljen već samo njegova slika zbog smanjene količine prostora. To je omogućeno postavljanjem prijelomnih točki u CSS dokumentima koje definiraju izgled korisničkog sučelja pri različitim veličinama zaslona.



Sl. 4.2 Korisničko sučelje za mobilne uređaje



Sl. 4.3 Korisničko sučelje za računala

4.3. Testiranje brzine

Brzina učitavanja web aplikacije ovisi o stanju mreže (brzina interneta, kašnjenje, gubitci podataka i sl.) i veličini web aplikacije (broj bitova koje je potrebno poslati preko mreže).

4.3.1. Vrijeme pristupa web aplikaciji

Kako bi odredili brzinu, mjeri se vrijeme učitavanja aplikacije pri prvoj posjeti i pri ponovljenoj posjeti kada već imamo spremljene resurse aplikacije u predmemoriju Internet preglednika. Testovi su izvršeni u Internet pregledniku Chrome 60 s postavljenim usporenjem dohvaćanja sadržaja na 3G za simulaciju stvarnih uvjeta.

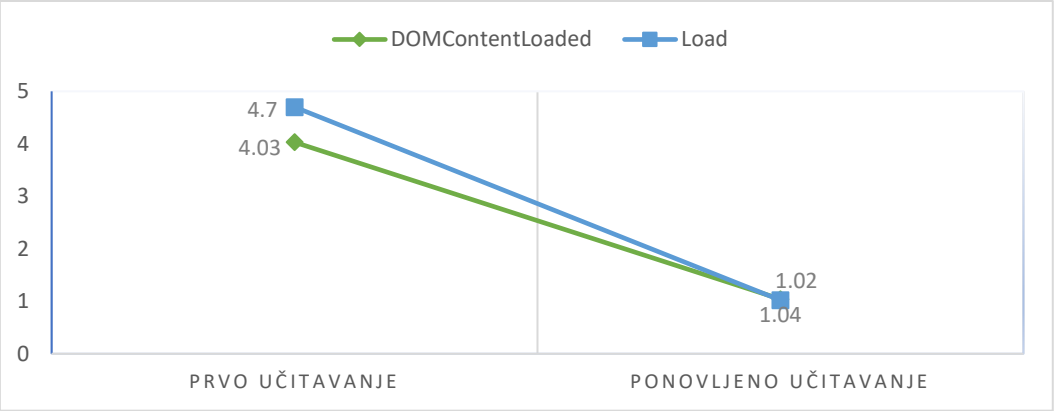
Pri testiranju se promatraju vremena dva *JavaScript* događaja:

1. *DOMContentLoaded* – događaj koji se okida kada se inicijalni HTML dokument u potpunosti učita i raščlani
2. *Load* – događaj koji se okida kada se svi resursi učitaju

Testovi brzine učitavanja su napravljeni na stranici za prijavu. Iz tablice 4.2 i slike 4.4 vidljivo je drastično smanjenje brzine učitavanja za sa 4,03 sekunde pri prvoj posjeti na 1,04 sekunde pri ponovljenoj posjeti što je približno smanjenje od 75%. Dobiveni rezultati ukazuju na jako brzo učitavanje aplikacije pri prvoj posjeti, dok pri ponovljenoj posjeti s učitavanjem od jedne sekunde aplikacija konkurira aplikacijama direktno napravljenim za operacijski sustav. Rezultat je postignut agresivnim spremanjem resursa u predmemoriju koje su potrebne za rad aplikacije.

Tab. 4.2 Brzine učitavanja web aplikacije

	DOMContentLoaded	Load
Prvo učitavanje	4,03 s	4,7 s
Ponovljeno učitavanje	1,04 s	1,02 s



Sl. 4.4 Graf učitavanja web aplikacije

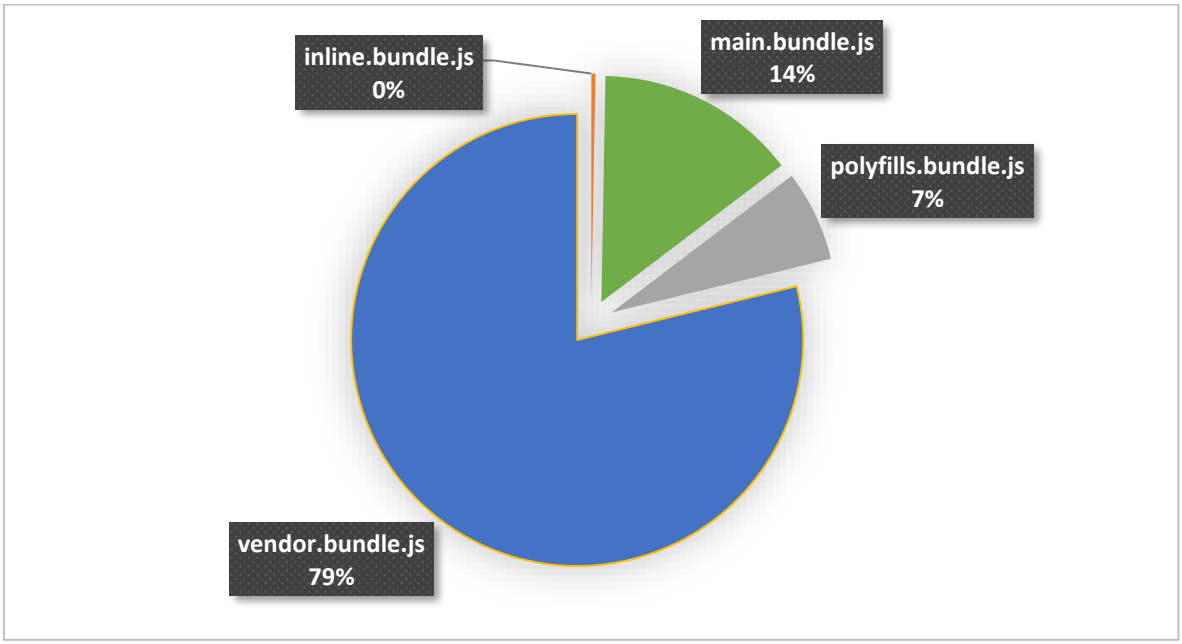
4.3.2. Usporedba veličina datoteka web aplikacije

Kako web aplikacija radila što je brže moguće, potrebno je što više moguće smanjiti veličinu datoteka koje se šalju. S obzirom da aplikaciju najvećim dijelom čine JavaScript datoteke, nad njima se vrši proces minificiranja (engl. *minification*) i sažimanja. Pod minificiranjem se podrazumijeva uklanjanje komentara, uklanjanje nepotrebnih funkcija i varijabli, mijenjanje naziva imena i varijabli tako da sadrže manje znakova te pojednostavljenje naredbi. Oblik sažimanja koji se koristi nad datotekama je gzip kompresija koja je najkorištenija na Internetu te je podržavaju svi Internet preglednici. Sažimanjem se veličina datoteke dodatno smanjuje tako što se nizovi znakova koji se ponavljaju zamjenjuju referencom na taj niz znakova koji se zapiše samo jednom. Iz tablice 4.3 vidljivo je smanjenje ukupne veličine JavaScript datoteka nakon sažimanja s 4490kB na 1406kB, što je smanjenje za 68%.

Tab. 4.3 Veličine JavaScript datoteka

Naziv datoteke	Veličina u kB prije sažimanja	Veličina u kB nakon sažimanja	Veličina u kB nakon gzip kompresije
inline.bundle.js	5,62	1,45	0,927
main.bundle.js	74,2	291	45,8
polyfills.bundle.js	183	63,6	20,7
vendor.bundle.js	4230,93	1050	251
Ukupno:	**4493,75**	**1406,05**	**318,43**

U konačnici, primjenom gzip komprimiranja s poslužitelja prema klijentu se šalje samo 318kB (dodatno smanjenje veličine za 77,4%). Slika 4.5 prikazuje udio pojedinačnih JavaScript resursa nakon gzip kompresije. Najveći udio u ukupnim resursima čini datoteka *vendor.bundle.js* koja sadrži sve biblioteke i programske okvire koji su korišteni unutar web aplikacije.



Sl. 4.5 Udio JavaScript resursa nakon gzip kompresije

5. ZAKLJUČAK

Kroz ovaj završni rad je obrađena implementacija web aplikacije za praćenje nestacionarnih objekata. Cilj je bio napraviti aplikaciju koja će na intuitivan, koristan i jednostavan način prikazivati lokacije praćenih objekata, koji mogu biti ljudi, auti, životinje i sl. Prikazan je način na koji se kompleksna aplikacija može rastaviti na više jednostavnih dijelova te kako te dijelove spojiti i ukomponirati u jednu funkcionalnu cjelinu.

Implementirano je korisničko sučelje koje je povezano s poslužiteljem od kojeg prima potrebne podatke. Prikazan je postupak spajanja korisničkog i poslužiteljskog dijela upotrebom sustava za prijavu korisnika i autorizaciju. Napravljeno je sučelje preko kojeg korisnik može vidjeti gdje se svi praćeni objekti nalaze. Također, izrađen je pregled povijesti kretanja praćenih objekata prikazom putanje kretanja na karti.

Testiranjem je utvrđena funkcionalnost aplikacije te pravilno ponašanje kako je navedenom prema specifikacijskim zahtjevima. Utvrđeno je jako brzo učitavanje i pokretanje aplikacije, koje je ostvareno minificiranjem, kompresijom i spremanjem potrebnih resursa. Utvrđena je prilagodljivost korisničkog sučelja različitim veličinama zaslona na kojima je pokrenuta aplikacija.

LITERATURA

[1] Anonimno, Find My Friends, Apple, dostupno na: https://itunes.apple.com/us/app/find-my-friends/id466122094 (28. kolovoza 2017.)

[2] Anonimno, Vip auto nadzor, Vipnet, dostupno na: http://autonadzor.vip.hr/home (28. kolovoza 2017.)

[3] Anonimno, Whistle, Whistle Labs, dostupno na: https://www.whistle.com/ (28. kolovoza 2017.)

[4] Anonimno, Firebase Documentation, Google, dostupno na: https://firebase.google.com/docs/ (20. lipnja 2017.)

[5] A. Taherkordi, F. Eliassen, G. Horn, From IoT Big Data to IoT Big Services, SAC '17 Proceedings of the Symposium on Applied Computing, str. 485-491, Marrakech, Maroko, 2017

[6] Anonimno, Javascript, Mozilla, dostupno na: https://developer.mozilla.org/en-US/docs/Web/JavaScript (20. lipnja 2017.)

[7] Anonimno, C#, Microsoft, dostupno na: https://docs.microsoft.com/en-us/dotnet/csharp/csharp (20. lipnja 2017.)

[8] Anonimno, TypeScript, Wikipedia, dostupno na: https://en.wikipedia.org/wiki/TypeScript (20. lipnja 2017.)

[9] Z. Ghao, C. Bird, E. T. Barr, To type or not to type: quantifying detectable bugs in JavaScript, ICSE '17 Proceedings of the 39th International Conference on Software Engineering, str. 758-767, Buenos Aires, Argentina, 2017

[10] Aaron Parecki, OAuth 2 Simplified, dostupno na: https://aaronparecki.com/oauth-2-simplified/ (20. lipnja 2017.)

[11] H. Wang, Y. Zhang, J. Li, D. Gu, The Achilles' Heel of OAuth: A Multi-Platform Study of OAuth-based Authentication, ACSAC '16 Proceedings of the 32nd Annual Conference on Computer Security Applications, str. 167-169, Los Angeles, SAD, 2016

[12] Anonimno, Introduction to JSON Web Tokens, Auth0, dostupno na: https://jwt.io/introduction/ (20. lipnja 2017.)

[13] Anonimno, Introduction to ASP.NET Core, Microsoft, dostupno na: https://docs.microsoft.com/en-us/aspnet/core/ (20. lipnja 2017.)

[14] Anonimno, What is Angular?, Google, dostupno na: https://angular.io/docs (29. lipnja 2017.)

[15] Anonimno, Material Design components for Angular, Google, dostupno na: https://github.com/angular/material2 (29. lipnja 2017.)

[16] Brian Troncone, Comprehensive Introduction to @ngrx/store, https://gist.github.com/btroncone/a6e4347326749f938510 (16. kolovoza 2017.)

SAŽETAK

Subjekt ovog završnog rada je web aplikacija za praćenje pozicija nestacionarnih objekata koja prikazuje njihove pozicije na karti. Glavni cilj rada je ostvarivanje mogućnosti prikaza lokacija praćenih objekata na uređajima različitih veličina zaslona upotrebom već postojećih usluga koje je potrebno ukomponirati u jednu cjelinu. Aplikaciju čine tri međusobno neovisne komponente: korisničko sučelje koje je izrađeno koristeći Angular programski okvir, pozadinsko sučelje izrađeno sa ASP.NET Core programskim okvirom te usluge Google Maps API i Firebase. Nakon testiranja, utvrđena je prilagodljivost aplikacije na različite uređaje te ispravno prikazivanje pozicija praćenih objekata na karti.

Ključne riječi: Angular, Firebase, karta, lokacijska usluga, praćenje, web aplikacija

ABSTRACT

WEB APPLICATION FOR TRACKING NONSTATIONARY OBJECTS

The subject of this final paper is a web application for tracking the position of nonstationary objects that displays their position on a map. The main goal of this paper is the creation of an implementation that displays the locations on different devices with different screen sizes by using already existing third-party services and making them work together. The application consists of three independent components: a user interface that is implemented using the Angular framework, a backend service that is implement with the ASP.Net Core framework and third-party services Firebase and Google Maps API. After successful testing, the web application was determined to be responsive to different screen sizes and that it is showing the correct locations on the map.

Keywords: Angular, Firebase, map, location service, tracking, web application

ŽIVOTOPIS

Antonio Vrbić rođen je 8. lipnja 1995. godine u Žepču. Pohađao je osnovnu školu u Žepču u razdoblju od 2002. do 2010. godine. Nakon osnovne škole pohađa Katolički školski centar „Don Bosco“ Žepče, smjer Tehničar za mehatroniku, u razdoblju od 2010. do 2014. godine. Nakon završene srednje škole, 2014. godine upisuje sveučilišni preddiplomski studij Računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Tijekom preddiplomskog studija ujedno je i zaposlen u tvrtki GDi kao *software developer*.

PRILOZI (na CD-u)

Prilog 1. Datoteke pisanog dijela rada (docx i pdf)

Prilog 2. Programski kod