Razvoj web aplikacija za pomoć u planiranju putovanja

Šaravanja, Matija

Sveučilište u Zadru

Odjel za informacijske znanosti
Preddiplomski stručni studij Informacijske tehnologije

Matija Šaravanja

RAZVOJ WEB APLIKACIJE ZA POMOĆ U PLANIRANJU PUTOVANJA

Završni rad

Zadar, 2023.

Sveučilište u Zadru

Odjel za informacijske znanosti

Preddiplomski stručni studij Informacijske tehnologije

RAZVOJ WEB APLIKACIJE ZA POMOĆ U PLANIRANJU PUTOVANJA

Završni rad

Student:

Matija Šaravanja

Mentor:

Doc. dr. sc. Ante Panjkota

Komentor:

Dr. sc. Neven Pintarić

Zadar, 2023.

Izjava o akademskoj čestitosti

Ja, **Matija Šaravanja,** ovime izjavljujem da je moj **završni** rad pod naslovom **Razvoj web aplikacije za pomoć u planiranju putovanja** rezultat mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na izvore i radove navedene u bilješkama i popisu literature. Ni jedan dio mojega rada nije napisan na nedopušten način, odnosno nije prepisan iz necitiranih radova i ne krši bilo čija autorska prava.

Izjavljujem da ni jedan dio ovoga rada nije iskorišten u kojem drugom radu pri bilo kojoj drugoj visokoškolskoj, znanstvenoj, obrazovnoj ili inoj ustanovi.

Sadržaj mojega rada u potpunosti odgovara sadržaju obranjenoga i nakon obrane uređenoga rada.

Zadar, 2. listopada 2023.

SAŽETAK

Rad opisuje izradu web aplikacije za pomoć pri planiranju putovanja koja će korisniku pružati informacije o više područja putovanja te prikaz lokacija na interaktivnim kartama. Definiran i opisan je problem koji proizlazi iz malog broja funkcionalnosti drugih web aplikacija namijenjenih za putovanja. Dan je pregled područja tehnoloških i programskih rješenja koja mogu pomoći u rješavanju problema. Identificirani su i opisani zahtjevi za aplikaciju. Prikazana je izrada web aplikacije pomoću opisanih tehnologija i rješenja te su navedena moguća unaprjeđenja aplikacije.

Ključne riječi: web aplikacija za planiranje putovanja, ReactJS, sučelje za programiranje aplikacija, interaktivne karte

POPIS OZNAKA I KRATICA

AJAX engl. Asynchronous JavaScript And XML = skup tehnika za web razvoj koje koriste razne web tehnologije na klijentskoj strani

API engl. Application Programming Interface = kod koji omogućuje komunikaciju između dvaju softvera

Backend Poslužiteljski dio aplikacije koji upravlja podacima i osigurava funkcioniranje na klijentskom dijelu

CSS engl. Cascading Style Sheets = stilski jezik za opis prezentacije dokumenta napisanog pomoću HTML jezika

DOM engl. Document Object Model = programsko sučelje za web dokumente

Frontend Klijentski dio aplikacije kojeg korisnik vidi i s kojim izravno komunicira

HTML engl. HyperText Markup Language = prezentacijski jezik za izradu web stranica

HTTP engl. Hypertext Transfer Protocol = protokol za prijenos informacija na Webu

Img engl. Image = HTML oznaka za sliku

JS JavaScript = programski jezik

JSON engl. JavaScript Object Notation = format za razmjenu podataka

JSX engl. JavaScript Syntax Extension (JavaScript XML) = služi za pisanje HTML-a u React-u

Leaflet JavaScript biblioteka otvorenog koda za interaktivne karte

NPM engl. Node Package Manager = naredbeni alat koji instalira, ažurira i deinstalira Node.js pakete

Props engl. Properties = atributi za prijenos JavaScript vrijednosti sa komponente roditelja na komponentu dijete

REST engl. Representational State Transfer = arhitektonski stil za pružanje standarda među računalnim sustavima na webu

URL engl. Uniform Resource Locator = jedinstvena lokacija resursa

XML engl. Extensible Markup Language = proširivi jezik za označavanje podataka i dokumenata

SADRŽAJ

1. UVOD

Živimo u modernom vremenu u kojem se nove tehnologije svakodnevno razvijaju. Prilikom korištenja tehnologija korisnik želi što veći broj funkcionalnosti, korisničko sučelje koje mu omogućava potrebne podatke i informacije kao te njihov grafički prikaz. Prije su ljudi morali potrošiti sate i sate kako bi došli do nekog rješenja, dok sada zbog razvoja tehnologije i brojnih dostupnih programskih rješenja mogu do potrebnog rješenja doći preko nekoliko klikova miša. Razvile su se mnoge tehnologije koje pomažu ljudima kako bi što lakše obavili svoj posao. Kroz posljednjih desetak godina nastali su mnogi popularni razvojni okviri i biblioteke koji programerima uvelike olakšavaju razvoj traženih funkcionalnosti u aplikacijama. Jedan od razvojnih okvira za izradu web aplikacija je *ReactJS* i on će se koristiti u ovome radu.

Cilj ovog rada je identificirati i opisati problem, definirati tehnologije i programska rješenja kojim se može riješiti problem te dati njihov prikaz, identificirati i opisati zahtjeve za web aplikaciju te primijeniti identificiranu tehnologiju i rješenja za izradu web aplikacije.

Svrha rada je omogućiti korisniku planiranje cijelog putovanja putem samo jedne aplikacije.

Završni rad podijeljen je u pet dijelova. U prvom dijelu daje se pregled područja koji je potreban za razumijevanje razvoja navedene web aplikacije. U drugom dijelu rada identificirani su i opisani zahtjevi web aplikacije, dok je najveći fokus stavljen na treći dio rada u kojem je opisan razvoj web aplikacije s korištenim tehnologijama i primjerima iz praktičnog dijela. U četvrtom dijelu dotaknuto je pitanje mogućih nadogradnji web aplikacije, a u zadnjem dijelu zaokružen je cijeli rad.

1.1. Opis problema

Problem od kojeg ovaj rad polazi je taj što većina web aplikacija za pomoć u putovanju služi za pretraživanje samo jednog područja putovanja. Tako je *Airbnb*[1] web aplikacija koja služi samo za traženje smještaja, a *RentalCars.com*[2] je web aplikacija samo za pretraživanje automobila za iznajmljivanje. Primjer web aplikacije koja nudi više područja za pretraživanje je *Putovnica.net*[3], ali njezin nedostatak je što ona samo preusmjerava korisnika na druge web aplikacije prilikom pretraživanja pojedinih područja. Na primjer ako korisnik želi pretražiti smještaje na određenoj lokaciji, preusmjeren je na web stranicu *Booking.com*[4], ako ga zanimaju

[1] Airbnb - https://hr.airbnb.com/
[2] RentalCars - https://www.rentalcars.com/
[3] Putovnica.net - https://www.putovnica.net/
[4] Booking.com - https://www.booking.com/index.en-gb.html

automobili za iznajmljivanje, preusmjeren je na web stranicu *Skyscanner*[5] itd. Rješenjem ovog problema želi se postići dostupnost podataka vezanih za sva područja putovanja za korisnika na jednome mjestu tj. unutar jedne web aplikacije. Primjer takve web aplikacije koja pokriva nekoliko načina putovanja je *Wanderu*[6], ali njezin nedostatak je što ne sadrži interaktivne karte na kojima se lokacije i rute dinamički prikazuju. U izradi web aplikacije koristit će se tehnološka i programska rješenja koja će joj omogućiti pretraživanje većeg broja načina putovanja te grafički prikaz lokacija i ruta putem interaktivnih karti.

[5] Skyscanner - https://www.skyscanner.net/
[6] Wanderu - https://www.wanderu.com/

2. PREGLED PODRUČJA

U ovoj cjelini dan je pregled i opis tehnoloških i aplikacijskih rješenja koja omogućavaju: rješavanje postavljenog problema i ostvarenje funkcionalnosti za web aplikaciju. Za početak će biti objašnjene osnovne tehnologije na kojima se temelji Web te kako je moguće postići responzivnost tj. preglednost web aplikacije na svim vrstama ekrana. Radi bržeg razvoja koristit će se *JavaScript* razvojni okvir. Moderniji izgled korisničkog sučelja postići će se primjenom *CSS* razvojnog okvira. Poželjno je brže usmjeravanje među podstranicama, a to zahtjeva primjenu navigacije koja neće za svaku novu podstranicu slati zahtjev na server. Uz to, u web aplikaciju integrirat će se interaktivne karte na kojim će se dinamički izmjenjivati lokacije koje korisnik pretražuje.

2.1. Frontend

Korisniku je frontend najbitniji dio web aplikacije. Sve ono vizualno što korisnik čita, vidi i čini u izravnoj interakciji s web aplikacijom je zapravo frontend. Najveće uloge frontenda su vjerodostojno prikazivanje nečije ideje, jednostavno korištenje te mora biti oku ugodan. Frontend mogu činiti neki elementi ili komponente koji će se prikazivati na korisničkom ekranu, a to može biti pojedini tekst, forme koje traže unos određenih podataka od korisnika, slike, tablice ili ostalo. Na uspješnost frontenda utječe i vrijeme koje je potrebno za učitavanje stranice, jednostavnost ili složenost korisnikovog usmjeravanja među sadržajem ili podstranicama te pristupačnost ljudima koji imaju invaliditet.

Važno svojstvo frontenda je responzivnost što omogućuje prilagođen izgled web stranici na različitim širinama zaslona. Smisao je da korisnik dobije jednako kvalitetno iskustvo bez obzira s kojeg uređaja posjećuje web stranicu ili aplikaciju. Web aplikacija mora dobro izgledati, nebitno gleda li korisnik tu stranicu na računalu, na tabletu ili na mobitelu. To je moguće postići pomoću tri ključna svojstva: fleksibilnog mrežnog (engl. grid) rasporeda, fleksibilnih slika te korištenjem *CSS*-a kako bi se moglo usmjeriti ponašanje web stranice ovisno o uređaju.[7]

Responzivnost na različitim zaslonima i uređajima je moguće postići pomoću provjere medijskih upita (engl. media queries). To je *CSS* tehnika koja koristi *@media* pravila kako bi se uključili blokovi *CSS* svojstava ako je postavljeni uvjet istinit. Pomoću medijskih upita omogućena je provjera karakteristika uređaja na kojem je web aplikacija pokrenuta. Upit se sastoji od dvije komponente: vrste medija (zaslona) i zapravo upita smještenog unutar zagrada

[7] Ethan Marcotte, „Responsive Web Design“, A List Apart, 25. svibanj 2010., od. 7, https://alistapart.com/article/responsive-web-design/.

koji sadrži pojedino medijsko svojstvo s ciljanom vrijednosti koje će se ispitati.[8] Postoje dva načina prilagođavanja aplikacije kako bi bila funkcionalna na svim širinama zaslona. Jedan način je kad se prvo izradi aplikacija koja je namijenjena za pregled na računalu, pa je se onda prilagođava prema užim zaslonima i tad se koristi uvjet maksimalne širine (engl. *max-width*). Dok je drugi način kad se prvo izradi aplikacija za mobitele, pa se onda ista prilagođava za šire zaslone, tad je potrebno koristiti uvjet minimalne širine (engl. *min-width*).
Osnovne frontend tehnologije su *HTML*, *CSS* i *JavaScript*.

2.1.1. Prezentacijski jezik - HTML

Prezentacijski jezik HTML koristi oznake kako bi naznačio web pregledniku na koji način bi autor želio prezentirati sadržaj. Osmislio ga je Barners-Lee krajem 1991. godine. Verzija *HTML-5* koja ima veći broj oznaka postoji od 2012. godine. Osmišljen je kako bi definirao strukturu dokumenata kroz naslove, paragrafe i ostalo. To je prezentacijski jezik koji čini kostur gotovo svake web stranice, pruža sadržaj koji je vidljiv u web pregledniku te je velika većina svih web stranica izrađena pomoću *HTML*-a. *HTML* dokument je jednostavno tekstualna datoteka koja sadrži informacije koje se trebaju objaviti i odgovarajuće instrukcije o označavanju koje upućuju na to kako bi preglednik trebao strukturirati ili prezentirati dokument.[9] Na vrhu *HTML* dokumenta nalazi se *<!DOCTYPE>* deklaracija koja označava verziju korištenog *HTML*-a. Unutar ishodišnog (engl. root) *HTML* elementa, prema klasičnoj strukturi dokumenta, nalaze se elementi glava (engl. head) i tijelo (engl. body). Element glava sadrži informacije i oznake koje opisuju dokument, kao što je npr. naslovni element (engl. title), dok element tijelo sadrži sami dokument s pridruženim oznakama potrebnim za specificirati njegovu strukturu.

Informacije koje se nalaze unutar glave dokumenta najčešće su one koje se odnose na stranicu koja je korištena za vizualno stiliziranje, definiranje interaktivnosti, postavljanje naslova stranice i pružanje ostalih korisnih informacija koje opisuju ili kontroliraju dokument.[10] Bitni elementi koji se koriste unutar glave su naslovni element i element s metapodacima. Naslovni element se koristi kako bi se postavio tekst koji je prikazivan u pregledniku na traci s naslovima. Ta vrijednost unutar naslovnog elementa također se koristi u sustavu povijesti

[8] Marcotte, od. 4.
[9] Thomas A. Powell i Thomas A. Powell, *HTML & CSS: The Complete Reference*, 5th ed (New York: McGraw-Hill, 2010), 4.
[10] Powell i Powell, 23.

preglednika, pa je bitno da taj naslov ima smišljeno, opisno i primjereno ime. Metapodaci su jako korisni. Mogu se koristiti za specificiranje vrijednosti koje su ekvivalentne HTTP zaglavljima odgovora. Također, se koriste za definiranje skupine znakova te za postavljanje proizvoljnih parova ime-sadržaj za pružanje meta informacija o dokumentu u svrhe kao što je optimizacija tražilice.

Unutar tijela dokumenta dostupni su razni tipovi elemenata. Postoje elementi blok razine kao što su paragrafi, naslovi (npr. h1 i h2 itd.). To su elementi koji definiraju strukturalne blokove sadržaja. Unutar elemenata koji nisu prazni mogu se pronaći brojni linijski elementi kao što su: *bold* (b), *italic* (i), *strong* (strong) i ostali. Unutar blokova i linijskih elemenata nalazit će se tekstualni sadržaj.[11]

Postoje četiri glavna atributa koji su zajednički za gotovo sve elemente i imaju isto značenje za sve elemente. To su atributi: klasa (engl. class), identifikator (engl. id), stil (engl. style) i naslov (engl. title). Atribut klasa odnosi se na klasu ili klase kojima određeni element pripada. Ime klase može biti korišteno od strane lista stila (engl. style sheet) za povezivanje stilskih pravila s većim brojem elemenata ili za pristup skripti koristeći metodu *getElementsByClassName()*. Moguće je imati nekoliko vrijednosti za atribut klasu odvojenih prazninom. Atribut identifikator određuje jedinstveni alfanumerički identifikator koji je pridružen elementu. To je važno kako bi se elementu moglo pristupiti s listom stila, poveznicom ili skriptnim jezikom. Jednom kad element sadrži identifikatorski atribut, postaje jednostavno manipulirati njime pomoću skriptnog jezika. Stil atribut specificira linijski stil povezan za element koji određuje prikazivanje elementa na kojeg utječe. Naslovni atribut pruža savjetodavni tekst o elementu ili njegovu sadržaju. Preglednici inače prikazuju taj tekst kao naputak iznad elementa kad se prođe pokazivačem preko. Još neki opći atributi su: *accesskey*, *align*, *disabled*, *height*, *contenteditable*, itd.[12]

2.1.2. Stilski jezik - CSS

Stilski jezik *CSS* opisuje prezentaciju *HTML* dokumenta i njegovih elemenata te je jedna od osnovnih tehnologija na kojima se temelji današnji web. Nastao je 1995. godine s idejom da se dizajn odvoji od *HTML*-a u posebne datoteke i kako bi se kroz jednostavna pravila isti dizajn mogao primjenjivati na veći broj elemenata istovremeno. *CSS* daje osobnost svakoj stranici.

[11] Powell i Powell, 29.
[12] Powell i Powell, 139.

Pravila *CSS*-a definiraju se pomoću selektora koji određuju na koje *HTML* elemente će se određeni stilovi odnositi.

Selektori su srce *CSS*-a. Mogu se podijeljeni su u dvije kategorije. Prvi su oni selektori koji djeluju direktno na elemente definirane u stablu dokumenta (engl. document tree), to su na primjer, *p* elementi ili *href* atributi. Ta kategorija sadrži selektore: klasu, vrstu i atribut. U drugu kategoriju spadaju *pseudo* selektori koji djeluju na elemente koji se nalaze izvan stabla dokumenta, a to može biti prvo slovo paragrafa ili zadnje dijete od elementa roditelja.[13]

```
h1 { color: blue; text-align: center; }
```

Programski kod 1.: CSS pravilo
Izvor: autor

U programskom kodu br. 1 prikazan je primjer *CSS* pravila. Prikazano pravilo odnosi se na *h1* elemente tj. elemente koji su odabrani pomoću selektora. Drugi dio pravila sastoji se od dijela koji se nalazi unutar vitičastih zagrada, a to je deklaracijski blok u kojem se nalaze deklaracije. Deklaracije se sastoje od svojstva i vrijednosti koja će biti pridodana *h1* elementu. Prema prikazanom primjeru, naslov *h1* će imati slova plave boje i bit će centralno poravnat.

Stilski jezik funkcionira tako što preglednik prvo učitava *HTML* kojeg pretvara u objektni model dokumenta (engl. Document Object Model, skraćeno DOM) koji predstavlja dokument u memoriji računala. Zatim, preglednik dohvaća ostale resurse koji su vezani na *HTML* dokument kao što su slike, videozapisi te povezani *CSS*. Preglednik obrađuje taj *CSS*, svrstava različita pravila po tipovima selektora. Ovisno o selektorima koje nađe, otkriva koja pravila treba dodati kojim čvorovima u *DOM*-u te im tek onda pridodaje pripadni stil.

Valja spomenuti relativne jedinice r*em* i *em*. To su jedinice koje uobičajeno u pregledniku imaju vrijednost od 16 točaka (engl. pixels), a relativne su vršnoj (engl. root) vrijednosti.[14]

2.1.3. Programski jezik - JavaScript

U ovom radu koristit će se skriptni programski jezik *JavaScript* koji web stranici omogućuje interaktivnost. U početku web-a, web preglednici prikazivali su samo statične web

[13] Devon Young, Craig Buckler, i Optimalworks Ltd, „Praise for the First Edition of The Book of CSS3“, 2015., 21.
[14] Devon Young, Craig Buckler, i Optimalworks Ltd, „Praise for the First Edition of The Book of CSS3“, 2015., 200.

stranice. Brendan Eich predstavio je *JavaScript* 1995. godine u Netscape pregledniku s ciljem uvođenja interaktivnosti.[15] *JavaScript* djeluje u kombinaciji s *HTML*-om i *CSS*-om.

Pomoću *HTML*-a i *CSS*-a web aplikaciji je dan sadržaj i izgled, a *JavaScript* joj daje njene funkcionalnosti. *JavaScript* je interpretirani jezik prilikom čije upotrebe se kod pokreće od vrha do dna i rezultat koda se odmah vraća. Nije potrebno pretvaranje koda u drugu formu prije pokretanja u pregledniku. Iako je *JavaScript* najpoznatiji po tome što je skriptni jezik za izradu web stranica, korišten je i u mnogim okruženjima bez preglednika kao što su *Node.js*, *Apache CouchDB* i *Adobe Acrobat*.[16]

Što se tiče funkcija, postoje obične i tzv. streličaste (engl. arrow) funkcije. Streličaste funkcije dodane su u *JavaScript* iz dva razloga: pružaju sažetiji način kreiranja funkcija i funkcioniraju bolje kao prave funkcije unutar metoda – streličasta funkcija može pristupiti specijalnoj varijabli *this* od okolne metode, što obične funkcije ne mogu.[17] U nastavku je prikazan jednostavan primjer korištenja obične i streličaste funkcije u *JavaScript*-u.

```
// obična anonimna funkcija
(function (a, b) {
  return a + b + 10;
});
// streličasta funkcija
(a, b) => a + b + 10;

const a = 4;
const b = 2;

// obična anonimna funkcija (bez parametara)
(function () {
  return a + b + 10;
});

// streličasta funkcija (bez parametara)
() => a + b + 10;
```

Programski kod 2.: Anonimne i streličaste funkcije
Izvor: autor

[15] Axel Rauschmayer, *JavaScript for Impatient Programmers* (s.l.: s.n., 2019), 19.
[16] Mozilla, „JavaScript | MDN“, pristupljeno 08. rujan 2023., https://developer.mozilla.org/en-US/docs/Web/JavaScript.
[17] Axel Rauschmayer, *JavaScript for Impatient Programmers* (s.l.: s.n., 2019), 239.

Iz primjera programskog koda br. 2 je vidljivo kako streličasta funkcija ima sažetiju sintaksu. Prilikom korištenja streličaste funkcije nije potrebno pisati ključne riječi *function*, *return* i čak vitičaste zagrade nisu potrebne ako se radi o jednoj liniji koda.

Danas se stručnjaci u području oslanjaju na tzv. razvojne okvire (engl. frameworks) za lakšu izradu web aplikacija. Razvojni okviri čine rad s *JavaScript*-om jednostavnijim tako što omogućavaju velik broj gotovih dijelova koda i dodatnih mogućnosti za aplikacije. To su strukture na kojima se može graditi aplikacija te omogućuju uštedu vremena i smanjenje rizika od pogrešaka u kodu. Većina razvojnih okvira je usvojila principe jednostraničnih aplikacija (engl. single-page application), a to su web aplikacije koje ne učitavaju potpuno nove stranice u pregledniku, već u interakciji s korisnikom dinamično prerađuju trenutnu stranicu s novim podacima. Neke od glavnih prednosti takvih jednostraničnih aplikacija su: brzo vrijeme odgovora, razdvajanje prezentacije od poslovne logike, brži i lakši transfer podataka te mogućnost izvan mrežne podrške.[18]

Postoje razlike između razvojnog okvira i biblioteke. Razvojni okvir je strukturalni uzorak (engl. pattern) koji definira dizajn aplikacije, a biblioteka je skup funkcija čije korištenje olakšava izradu aplikacije. Kod za razvojni okvir se piše kako bi se izradila aplikacija, a biblioteka se koristi kako bi se napisao kod kojim će se moći izraditi aplikacija, pa je to i ključna razlika između razvojnog okvira i biblioteke.[19]

Dvije bitne tehnologije vezane za *JavaScript* su: *Node.js* i *NPM*.

2.1.3.1. Preduvjeti za razvojni okvir

Glavni cilj *Node.js*-a je pružanje sigurnog i lakog načina za izgradnju mrežnih aplikacija visokih performansi u *JavaScript*-u. Razvijen je od strane Ryan Dahl-a i njegovih kolega 2009. godine. Dvije godine nakon postaje projekt otvorenog koda. Budući da se *JavaScript* izvršava u pregledniku, javila se potreba za pokretanjem na poslužiteljskoj razini i *Node.js* se pokazao najpraktičnijim rješenjem. *Node.js* je platforma izrađena na *JavaScript* pogonu (engl. engine) koji služi za razvoj aplikacija za Web. Neke od prednosti *Node.js*-a su: velika brzina, ima

[18] Marin Kaluža i Bernard Vukelic, „Comparison of front-end frameworks for web applications development“, *Zbornik Veleučilišta u Rijeci* 6 (01. siječanj 2018.): 263, https://doi.org/10.31784/zvr.6.1.19.
[19] Lorena Škalac, „ReactJS“ (Diplomski rad, Sveučilište Josipa Jurja Strossmayera, Odjel za matematiku, 2017), 7, https://urn.nsk.hr/urn:nbn:hr:126:915147.

licencu otvorenog koda, jednonitan je, može se koristiti na svim operacijskim sustavima te nudi višeplatformsko okruženje za vrijeme izvođenja.[20]

Uz *Node.js* mora se spomenuti alat *Node Package Manager* (*NPM*) koji dolazi sa svakom *Node.js* instalacijom. *NPM* je upravitelj paketa ili modula za *Node.js*. Ukoliko se koristi neki vanjski kod, *NPM* provjerava postoji li nadogradnja ili novija verzija istog, te preuzima tu nadogradnju.

Paketi ili moduli su dijelovi koda koji se mogu koristiti više puta. Paket je mapa unutar koje se nalazi jedna ili više datoteka i ima paket *package.json* s metapodacima vezanim za taj paket. Prosječna web aplikacija ovisi o stotinama paketa. A za razliku od paketa, modul je dio koda koji se može učitati u *Node.js* aplikaciji pomoću naredbe *require*.

2.2. Razvojni okvir - ReactJS

Jedan od razvojnih okvira za izradu web aplikacija u *JavaScript*-u je *ReactJS*. To je besplatni *JavaScript* razvojni okvir otvorenog koda temeljen na komponentama i namijenjen je za lakšu i bržu izradu interaktivnih korisničkih sučelja. Objavljen je 2013. godine od strane *Meta*-e i zajednice individualnih programera. Najveći razlog korištenja *React*-a je taj što povećava brzinu aplikacija. Koristi virtualni *DOM* koji poboljšava performanse aplikacije tj. virtualni *DOM* brži je od regularnog *DOM*-a.[21]

Svaka *React* aplikacija se temelji na komponentama. Komponente se dijele na one visoke razine koje određuju strukturu aplikacije i na one komponente koje su niže razine i nalaze se unutar većih komponenti. One su zapravo funkcije koje vraćaju manje dijelove *HTML* koda, neovisne su o ostalim komponentama i mogu se ponovno koristiti na drugom mjestu u aplikaciji. Česta je pojava ponavljanje većeg broja identičnih struktura u aplikacijama, ali s drukčijim podacima. Upravo to je jedno od bitnijih svojstva *React* komponenti – ponovna upotreba iste strukture.[22]

Ekstenzija *JSX* je vrlo bitna za *React* programere. To je ekstenzija *JavaScript* jezika koja pruža strukturiranje renderiranja komponenti koristeći sintaksu sličnu *HTML*-u. *JSX* omogućuje pisanje *HTML* elemenata unutar JavaScript koda na način da se postavljaju u *DOM* pretvarajući

[20] Dhruti-Shah, *Node .Js* (BPB Publications, 2018), 10.
[21] Artemij Fedosejev, *React.Js Essentials* (Packt Publishing Ltd, 2015), 19.
[22] Alex Banks i Eve Porcello, „Learning React, 2nd Edition [Book]“, lipanj 2020., 66, https://sd.blackball.lv/library/Learning_React_(2020).pdf.

HTML oznake u *React* elemente bez potrebe za drugim metodama poput *createElement()*. Takva kombinacija *HTML*-a i *JavaScript*-a omogućuje aplikacije s moćnijim učinkom. Bitna stvar kod *JSX*-a je što se taj kod uvijek mora sastojati od samo jednog vršnog elementa unutar kojeg su učahureni ostali elementi, u suprotnom *JSX* neće funkcionirati. [23]

Iako je *JSX* veoma sličan *HTML*-u - postoje određene razlike. Kao na primjer, *JSX* omogućuje dodavanje komponenti kao djecu drugim komponentama. Budući da je *class* rezervirana riječ u *JavaScript*-u, umjesto njega koristi se *className* atribut. *JavaScript* izrazi stavljaju se u *JSX*-u unutar vitičastih zagrada koje indiciraju promjenu varijabli i vraćanje njihovih vrijednosti. Također, *JSX* je *JavaScript*, pa je moguće ugraditi *JSX* direktno unutar *JavaScript* funkcija. Tako je moguće mapirati niz u *JSX* elemente.[24]

Potrebno je opisati još jedan bitan alat *React*-a, a to su *React Hooks*.

2.2.1. Specijalne funkcije - React Hooks

React Hooks su specijalne funkcije koje omogućuju korištenje različitih *React* svojstava iz komponenti tako što se „zakvače“ za njih, pa se tako može koristiti *React* funkcionalnost bez potrebe za klasnim komponentama. Prednosti *React Hooks* su: čitljivost, manje koda, cjelokupna optimizacija komponente, jednostavnije kodiranje kompleksnih komponenti, baratanje događajima i logikom u funkcionalnoj komponenti te poboljšani učinak s funkcionalnim komponentama.[25] Neke od češće korištenih ugrađenih *React Hooks*-a su: *useState*, *useEffect* i *useRef*, pa su iste objašnjene u nastavku.

Uloga *useState* je pamćenje podatka kao što je neki korisnikov unos. *UseState* uzima inicijalno stanje i vraća dvije vrijednosti: trenutno stanje i funkciju koja ažurira stanje. Tako se sadržaj na zaslonu može dinamično mijenjati.

U*seRef Hook* služi za direktno kreiranje reference na element u funkcionalnoj komponenti. Korištenje *useRef* umjesto običnih referenci je efikasnije jer *useRef* kroz renderiranja čuva svoju vrijednost i ne uzrokuju nova renderiranja kad se vrijednost promijeni, pa to čini aplikaciju bržom.

[23] Stephen Arancio, „What is JSX?. A look at what JSX is, what it is used… | Medium“, 06. listopad 2021., https://medium.com/@sjarancio/what-is-jsx-e3dda0af3490.
[24] Alex Banks i Eve Porcello, „Learning React, 2nd Edition [Book]“, lipanj 2020., 72, 73 https://sd.blackball.lv/library/Learning_React_(2020).pdf.
[25] Daniel Bugl, *Learn React Hooks: Build and Refactor Modern React.Js Applications Using Hooks* (Packt Publishing Ltd, 2019), 25, 26.

UseEffect Hook rješava nuspojave u funkcionalnoj komponenti. Primjeri nuspojava su dohvaćanje podataka ili osvježavanje *DOM*-a. Ovaj *hook* pokreće se nakon svakog render-a, ali može se koristiti niz sa zavisnostima koji može kontrolirati efekt renderiranja. *UseEffect* prima dva argumenta, s tim da je drugi argument opcionalni niz sa zavisnostima. Ako nije dodana nijedna zavisnost onda se efekt pokreće nakon svakog renderiranja. Ako je dodan prazan niz, onda će se efekt pokrenuti samo jednom, nakon prvog renderiranja. Ako niz sa zavisnostima sadrži neke vrijednosti, onda će se efekt pokrenuti nakon početnog renderiranja i svaki put kad se neka od zavisnosti promijeni.

React nudi mogućnost izrade prilagođenih *hooks-a*. To su funkcije koje na početku naziva imaju riječ '*use*', baš kao i kod klasičnih *hooks*-a. Kako ne bi u nekim slučajevima dolazilo do dupliciranja koda, tad se kreira prilagođeni *hook* koja se može koristiti u bilo kojoj komponenti. Još jedna prednost prilagođenih *hook*-ova je to što se stvara čist kod jer se logika izdvaja iz komponente u *hook*.[26]

2.3. CSS razvojni okvir stiliziranih formi i elemenata

Razvojni okviri pomažu programerima u izradi web aplikacija. Prednosti korištenja *CSS* razvojnih okvira su: implementacija raznih naprednih svojstava i vizualnih elemenata na web stranici kao što su forme, navigacijske trake i sl., čine stvaranje web stranica kompatibilnim s više preglednika, imaju stilove spremne za korištenje što čini kodiranje bržim i brzo generiranje korisničkih sučelja koja su laka za uporabu i vizualno privlačna.

React Bootstrap je *CSS* razvojni okvir koji nudi velik broj stiliziranih formi, elemenata i sustav za pozicioniranje ili raspored elemenata. Služi za izgradnju responzivnih web stranica i web aplikacija. To je efikasan način za razvijanje moćnih korisničkih sučelja. *React Bootstrap* nudi čišći kod za komponente jer prilikom uvoza je moguće uvesti individualnu komponentu iz *React Bootstrap*-a umjesto uvoza cijele biblioteke. Ne postoje nikakve zavisnosti bez kojih ne bi mogao funkcionirati.

Bitno svojstvo *React Bootstrap*-a je njegov mrežni sustav za raspored elemenata. Mrežni sustav koristi serije kontejnera, redove i stupce za razmještaj i poravnanje sadržaja.

[26] Meta Open Source, „Reusing Logic with Custom Hooks – React“, pristupljeno 21. rujan 2023., https://react.dev/learn/reusing-logic-with-custom-hooks.

2.4. Usmjeravanje

U tradicionalnim web stranicama, kad korisnik pritisne na poveznicu u navigaciji, preglednik učitava novi HTML dokument što inače zahtjeva nešto više vremena. Biblioteka kojom se omogućava usmjeravanje na klijentskoj strani je *React Router*. Usmjeravanje na klijentskoj strani omogućava aplikaciji ažuriranje *URL*-a nakon klika na poveznicu bez potrebe za učitavanjem novog dokumenta. To utječe na pozitivno korisničko iskustvo jer omogućava brže izvođenje. *React Router* najčešće se koristi pri izradi jednostraničnih (engl. single page) web aplikacija. Pomoću njega definiraju se rute unutar aplikacije. Ako korisnik unese *URL* u pregledniku, a ta *URL* putanja se poklapa s nekom od ruta iz aplikacije, bit će preusmjeren na tu rutu.

2.5. Upravljanje podacima

Biblioteka koja služi za lakše upravljanje podacima u *React-u*, pogotovo kad su u pitanju velike aplikacije, naziva se *React Redux*. *React store* je skladište stanja aplikacije, nalazi se najčešće unutar *React* projekta u zasebnoj mapi u posebnoj *JavaScript* datoteci gdje je kreiran, a moguće mu je pristupiti iz bilo koje komponente u aplikaciji jer se cijela aplikacija omota s komponentom *Provider* koja im omogućuje pristup *store*-u. *Redux* omogućuje *React* komponentama učitavanje stanja iz *React store*-a, te isporuku akcija *Store*-u za ažuriranje stanja.

Prije početka rada s *Redux*-om, prvo je potrebno instalirati *Redux Toolkit* i *React Redux* pakete. Zatim je kreirana *Redux store* datoteka u aplikaciji. Potrebno je u vršnoj datoteci *index.js* omotati *App.js* datoteku s *Provider* komponentom iz Redux-a kojoj dajemo skladište (engl. *store)* kao svojstvo (engl. prop(erty)). Unutar skladišta možemo kreirati koliko želimo odsječaka (engl. slice) koji moraju sadržavati ime, inicijalno stanje i jedan ili više reduktora (engl. reducers). Reduktori su funkcije koje ažuriraju stanja. U *React* komponentama mogu se učitati podaci pomoću *useSelector Hook*-a, a isporučiti akcije pomoću *useDispatch Hook*-a i *dispatch* metode.

2.6. Interaktivne karte

Velik broj web ili mobilnih aplikacija u današnje vrijeme koristi neki od servisa za interaktivne karte kao što je *Google Maps*. Jedna od alternativa za *Google Maps* koja je besplatna za uporabu je *React Leaflet*. *React Leaflet* je samo *LeafletJS* primijenjen za rad s *React* komponentama. *Leaflet* je *JavaScript* biblioteka otvorenog koda za kreiranje web aplikacija za

prikazivanje geografskih karti. On koristi najnovije značajke *JavaScript*-a te *HTML-5* i *CSS3*, te je jednostavan za korištenje.

Za prikaz podataka na ekranu *Leaflet* koristi skupine podataka koji se nazivaju slojevima (engl. layers). Svaki sloj je geografski referenciran i pomiče se tako što korisnik pomiče kartu, a slojevi se međusobno sinkroniziraju. Osnovni tipovi slojeva su slojevi korisničkog sučelja, rasterski, vektorski i ostali. U korisničkim slojevima razlikuju se oznake (engl. marker), skočni prozori (engl. popup) te informativni oblačići.

Glavni *Leaflet* objekti rasterskog tipa su *TileLayer* i *ImageOverlay* za prikaz rasterskih slika na karti, vektorskog tipa *Path, Polygon* i *Circle* koji omogućavaju crtanje linija po karti, grupiranog tipa *LayerGroup, FeatureGroup* i *GeoJSON*, kontrole karte *Zoom* i *Layers*.[27]

2.7. Aplikacijsko programsko sučelje

Vrsta programskog koda koji omogućuje klijentu, upravljanom od strane korisnika ili automatiziranom, pristup resursima koji modeliraju podatke i funkcije sustava naziva se aplikacijsko programsko sučelje (engl. Application Programming Interface, API). Ono omogućava komunikaciju i razmjenu podataka koristeći skup definicija i protokola. *API* dokumentacija sadrži informacije za razvojne programere o tome kako strukturirati zahtjeve i odgovore. Uobičajeno, *API*-ji koriste *JSON* (engl. JavaScript Object Notation) kao glavni jezik za pružanje metoda dizajniranih za izvlačenje i manipulaciju podataka pohranjenih u *HTML* dokumentu. *JSON* je način reprezentiranja podataka koji izgledaju poput *JavaScript* objekata. Arhitektura *API*-ja objašnjena je kroz termine klijent i poslužitelj. Računalo ili program koji šalje zahtjev naziva se klijent, a namjensko računalo ili program koji vraća odgovor zove se poslužitelj.

Najpopularniji i najfleksibilniji *API*-ji na webu danas su *REST API*-ji. Oni definiraju set funkcija kao što su *GET, PUT, DELETE* i ostale koje klijenti mogu koristiti kako bi pristupili podacima s poslužitelja. Klijenti i poslužitelji koriste *HTTP* protokol za razmjenu podataka. Načini pristupanja *API*-ju su preko autentikacijskih tokena ili pomoću *API* ključeva. Koraci za implementaciju novog *API*-ja su: dobivanje *API* ključa, postavljanje *HTTP API client*-a ili ako ne posjedujete *API client*-a, onda možete sami pokušati strukturirati zahtjev u pregledniku

[27] Volodymyr Agafonkin, „Documentation - Leaflet - a JavaScript library for interactive maps“, pristupljeno 21. rujan 2023., https://leafletjs.com/reference.html.

pomoću *API* dokumentacije, te jednom kad je sintaksa *API*-ja jasna, može se početi koristiti u kodu.

REST API-ji koriste statuse od *HTTP* poruka odgovora za informiranje korisnika o njihovom sveobuhvatnom odgovoru na zahtjev. Postoji 40 standardnih statusnih kodova *HTTP*-a koji mogu prenijeti rezultate korisnikovih zahtjeva. Statusni kodovi mogu se podijeliti u 5 skupina. Kodovi koji započinju s brojem „1“ predstavljaju informativni status što znači da je poslužitelj primio zahtjev i nastavlja s procesom. Oni koji započinju s brojem „2“ predstavljaju uspješan korisnikov zahtjev. Zatim, kodovi koji započinju s brojem „3“ indiciraju na to da korisnik mora obaviti neku dodatnu radnju kako bi se zahtjev dovršio. Kodovi koji počinju s brojem „4“ predstavljaju korisnikovu pogrešku, dok kodovi s početnim brojem „5“ označavaju da poslužitelj poprima odgovornost za statusne kodove o pogrešci. [28]

[28] Mark H. Massé i Mark Massé, *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces* (Beijing Köln: O'Reilly, 2012), 28.

3. IDENTIFIKACIJA ZAHTJEVA ZA APLIKACIJU

U ovoj cjelini opisani su konkretni zahtjevi za web aplikaciju. Cjelina daje idejni opis rješenja s funkcionalnim zahtjevima aplikacije, pogled na implementirane *API*-je i interaktivne karte.

3.1. Idejni opis i funkcionalna specifikacija aplikacije

Ideja web aplikacije je omogućiti korisniku dobivanje više različitih podataka i informacija vezanih za izabrano putovanje na jednom mjestu. Nastala je ideja za web aplikaciju koja će pružati informacije o autobusnim linijama, letovima, smještajima i automobilima za iznajmljivanje. Korisnik će moći date ponude za automobile i smještaj zaprimiti i pregledati te ih po želji sortirati po cijeni, odnosno po ocjeni. Radi bolje vizualizacije putovanja, rute i lokacije će se prikazivati na interaktivnim kartama. Nakon što korisnik ispuni upit u vidu forme, zahtjev će se trebati slati na određeni *API* ovisno o kojem polju putovanja ili načinu putovanja se radi te ako *API* vrati pozitivan odgovor s poslužitelja, u aplikaciji se pojavljuje lista s danim ponudama i informacijama među kojima korisnik može odabrati ponudu koja mu je zanimljiva i dodati je u karticu imena *Favorites*. S liste „favorita“ korisnik će moći brisati dodane ponude od prije. U tablici br. 1 definirano je 12 funkcionalnih zahtjeva za aplikaciju.

Funkcionalni zahtjev	Opis funkcionalnosti
F1	Omogućiti korisniku grafičko korisničko sučelje s formama za unos podataka
F2	Omogućiti korisniku usmjeravanje među podstranicama
F3	Pružanje podataka o autobusnim linijama
F4	Pružanje podataka o letovima
F5	Pružanje podataka o automobilima za najam
F6	Pružanje podataka o dostupnom smještaju
F7	Omogućiti korisniku dodavanje ponuda u „Favorite“
F8	Omogućiti korisniku uklanjanje ponuda s liste favorita
F9	Omogućiti prikaz traženih lokacija i ruta na interaktivnim kartama
F10	Omogućiti korisniku sortiranje liste
F11	Omogućiti upravljanje pogreškama
F12	Omogućiti pristup aplikaciji na bilo kojem uređaju

Tablica 1.: Funkcionalna specifikacija
Izvor: autor

Dalje u radu kad se budu spominjali pojedini funkcionalni zahtjevi, koristit će se njihovi nazivi iz tablice: F1, F2, F3 itd.

Osnovni zahtjevi koji služe za rješavanje našeg definiranog problema su F1, F3, F4, F5, F6 i F9, a ostali funkcionalni zahtjevi su definirani kako bi se omogućile dodatne korisne funkcionalnosti za aplikaciju.

3.2. Izvori podataka za aplikaciju

Za web aplikaciju će se koristiti sljedeći izvori: *Flixbus API* za dohvat podataka o autobusnim linijama, *AeroDataBox API* za dohvat podataka o letovima, *Priceline Com Provider API* za dohvat podataka o automobilima za najam, *Booking API* i *Airbnb API* za dohvat podataka o dostupnim smještajima, *Address From To Latitude Longitude API* za dohvat podataka o lokacijama. Za korištenje navedenih *API*-ja potrebno se prethodno registrirati na stranicu *Rapid API Hub*. Svi navedeni *API*-ji su besplatni za korištenje za manji broj zahtjeva, a ako će se na *API* slati veći broj zahtjeva, tad je potrebno nadograditi pretplatu i plaćati istu.

4. RAZVOJ APLIKACIJE

U ovoj cjelini detaljnije je opisan proces izrade web aplikacije za pomoć pri planiranju putovanja. Cijeli kod web aplikacije javno je dostupan na sljedećoj poveznici: *https://doi.org/10.5281/zenodo.8397123*. Ova cjelina dotaknut će se korištenih razvojnih okvira i biblioteka te pokazati konkretne primjere njihove primjene. Razvoj kreće od postavljanja *React* aplikacije pomoću *Create React App* alata, nakon čega se može početi s pisanjem koda.

Funkcionalni zahtjev	Primijenjene tehnologije
F1	React Bootstrap
F2	React Router
F3	React Redux + Fetch API + izvor podataka Flixbus API
F4	React Redux + Fetch API + izvor podataka AeroDataBox API
F5	React Redux + Fetch API + izvor podataka Priceline Com Provider API
F6	React Redux + Fetch API + izvori podataka Booking API i Airbnb API
F7	React Redux
F8	React Redux
F9	React Leaflet + React Redux + izvor podataka Address From To Latitude Longitude API
F10	JS sort() method + React Redux
F11	Catch metoda + React Bootstrap
F12	Media queries + React Bootstrap

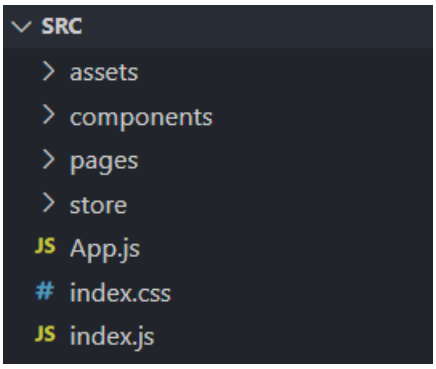
Tablica 2.: Primijenjene tehnologije za realizaciju funkcionalnih zahtjeva

U tablici br. 2 prikazane su tehnologije pomoću kojih će se realizirati određeni funkcionalni zahtjevi za aplikaciju.

4.1. Organizacija aplikacije

Najprije je potrebno instalirati *NPM* (upravitelj paketima za *JS* programski jezik) i *Node.js* (serversko okruženje otvorenog koda). Najjednostavniji način za kreirati *React* projekt je putem naredbenog retka i „*create-react-app*“ naredbe. *Create React App* omogućuje kreiranje novog *React* projekta i stvaranje projektne strukture za razvojne programere s jednom naredbenom linijom.

Kad je projekt kreiran, u njemu se nalazi */src* (source) direktorij. To je direktorij koji je za programere najvažniji jer se u njemu nalaze *JavaScript* i *CSS* datoteke. Struktura predmetnog *React* projekta prikazana je na prikazu br. 2.



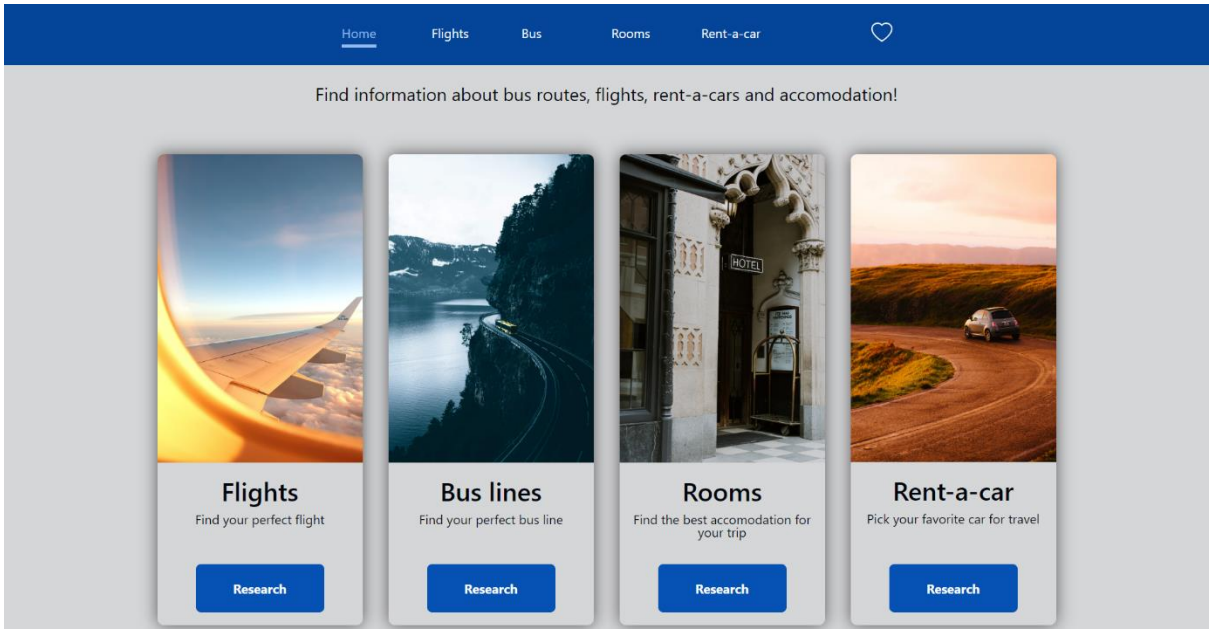
Slika 1.: Struktura *source* direktorija unutar web aplikacije
Izvor: autor

Iz prikaza br. 1 vidljiv je *source* direktorij koji sadrži nekoliko poddirektorija s *JavaScript* i *CSS* datotekama. U poddirektoriju */assets* nalaze se fotografije koje su korištene u aplikaciji. Poddirektorij */components* sastoji se od svih *React* komponenti od kojih je sastavljena web aplikacija. U poddirektoriju */pages* nalaze se komponente visoke razine tj. komponente koje predstavljaju zasebne stranice (npr. *Home* stranica, *Favorites* stranica, stranica za pronalazak smještaja itd.) u web aplikaciji među kojima korisnik može navigirati. Unutar *JSX* koda od *pages* komponenti ugniježđene su ostale komponente iz poddirektorija */components*. Unutar poddirektorija */store* nalazi se *store.js* datoteka koja predstavlja „*Redux store*“ što je veliki kontejner u kojemu se čuvaju podaci za aplikaciju. *Redux* je detaljnije opisan u cjelini o razradi razvoja aplikacije. U *source* direktoriju ostale su još tri datoteke koje treba objasniti. Datoteka *index.css* obična je *CSS* datoteka za stiliziranje *index.js* datoteke, zbog čega i imaju isti naziv kako bi se znalo na izgled koje komponente se odnosi. *App* komponenta korištena je unutar *index.js* datoteke koja je polazišna točka aplikacije. *Index.js* sadrži kod koji treba renderirati. *App.js* je izvorna komponenta u *React* aplikaciji tj. najviša komponenta u hijerarhiji.

4.2. Korisničko sučelje i funkcionalnosti

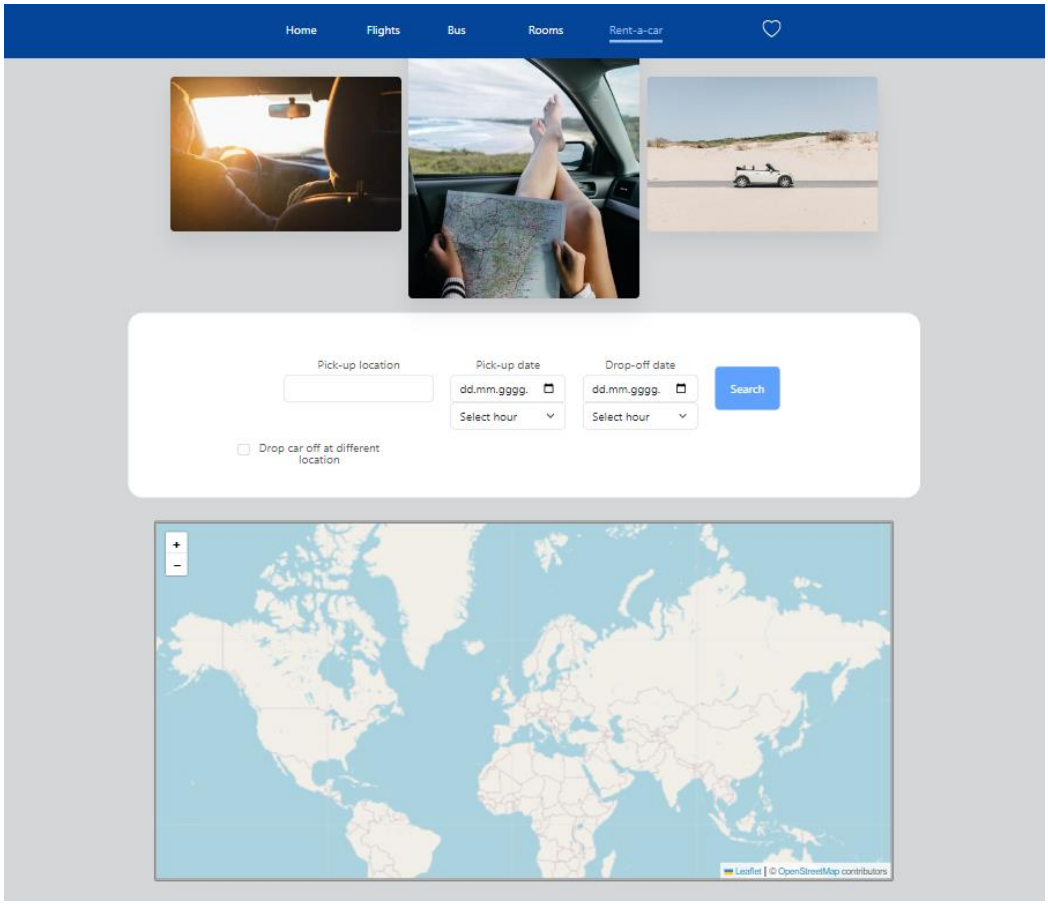
Korisničko sučelje je lice frontend aplikacije s kojim korisnik izravno vrši interakciju kako bi koristio aplikativno rješenje. U izradi korisničkog sučelja aplikacije, uz obični *HTML* i *CSS*, korišten je i razvojni okvir *React Bootstrap* koji ima velik broj gotovih stiliziranih komponenti koje je jednostavno ubaciti u kod. Time će se realizirati funkcionalni zahtjev F1. Prilikom

pokretanja aplikacije otvara se početna stranica koja sadrži četiri kartice među kojima korisnik može izabrati jednu od ponuđenih opcija koje se odnose na putovanje. Na vrhu početne stranice, kao i na svakoj drugoj stranici, prikvačena je navigacijska traka preko koje korisnik također može pristupiti ostalim stranicama. Iz prikaza br. 2 može se vidjeti izgled početne stranice. Prilikom pritiska na poveznicu na navigacijskoj traci ili na *Research* gumb, korisnik se preusmjerava na odabranu stranicu u web aplikaciji. Sve fotografije koje su korištene u aplikaciji su preuzete s web stranice *Unsplash* koja nudi besplatne fotografije koje se smiju preuzeti i koristiti u bilo kojem projektu.



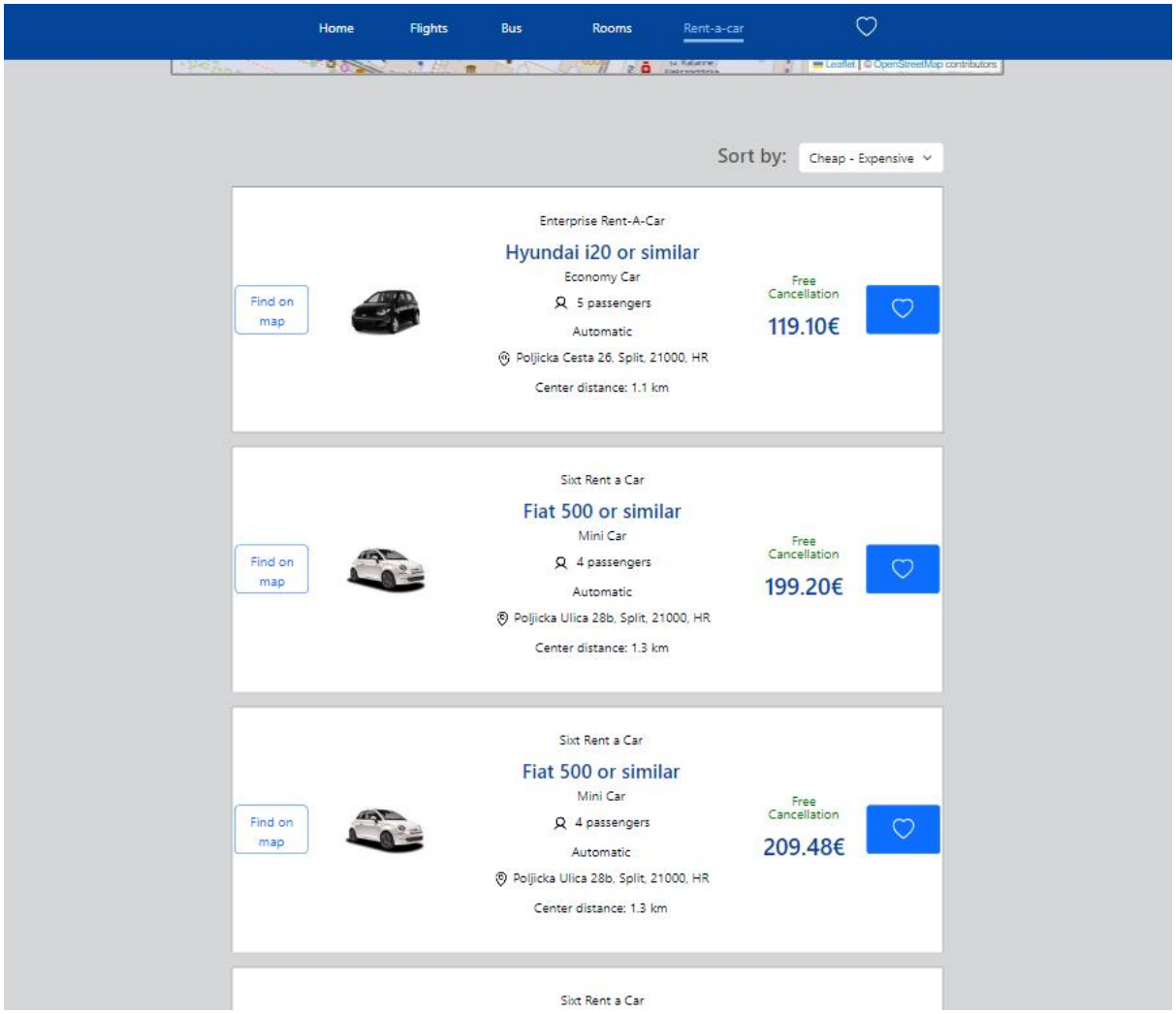
Slika 2.: Slika zaslona početne stranice u aplikaciji
Izvor: autor

U prikazu br. 3 vidljiva je stranica za pretraživanje automobila za iznajmljivanje tzv. *rent-a-car*.



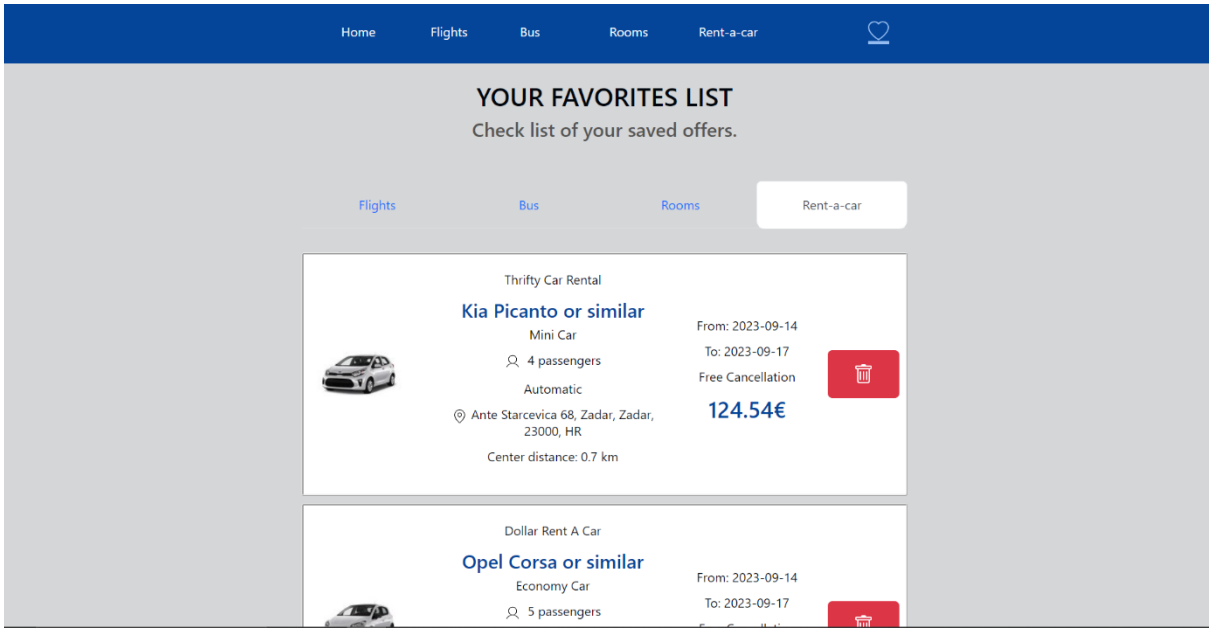
Slika 3.: Slika zaslona *Rent-a-car* stranice iz aplikacije
Izvor: autor

Pri vrhu *Rent-a-car* stranice u prikazu br. 3 definirane su tri fotografije. Ispod fotografija nalazi se forma za unos podataka. Od korisnika se traži upisivanje naziva lokacije s koje želi preuzeti automobil i odabir datuma i točnog vremena preuzimanja i povrata automobila, te ukoliko želi vratiti automobil na nekoj drugoj lokaciji može izabrati i tu opciju. *Search* gumb je neaktivan dok se ne ispune sve polja u formi, pa ga je nemoguće stisnuti. Na dnu stranice integrirana je *Leaflet* karta svijeta na kojoj se prikazuje tražena lokacija nakon pritiska gumba u formi. Prilikom unosa novih lokacija i pritiska gumba, karta s lokacijom se ažurira, kao što se ažurira i lista s ponudama automobila od različitih agencija na dnu stranice koja se pojavi nakon što *API* vrati pozitivan odgovor sa servera. Izgled liste s automobilima moguće je vidjeti na prikazu br. 4.



Slika 4.: Slika zaslona iz aplikacije s listom ponuđenih automobila
Izvor: autor

Lista sadrži podatke o automobilima i cijenama za najam. Listu je moguće sortirati po cijeni od najjeftinije do najskuplje ili od najskuplje do najjeftinije ponude čime je omogućen funkcionalni zahtjev F10. Pritiskom *Find on map* gumba na karti svijeta prikazuje se lokacija agencije koja daje ponudu za automobil, dok se pritiskom na plavi gumb ponuda sprema na *Favorites* stranicu koja je zamišljena kao korisnikovo spremište ponuda koje smatra zanimljivima tj. ponudama „favoritima“ te je tako omogućen funkcionalni zahtjev F7. Po sličnom principu napravljene su i stranice za pretraživanje letova, autobusa i smještaja, pa ovdje nisu pojedinačno prikazane slike zaslona tih stranica. Još ćemo samo proanalizirati stranicu *Favorites* na kojoj se nalaze spremljene ponude. Izgled *Favorites* stranice možete vidjeti na prikazu br. 5.



Slika 5.: Slika zaslona *Favorites* stranice
Izvor: autor

Pretpostavimo da smo prije odlaska na stranicu spremili nekoliko ponuda za najam automobila s prošle stranice. U prikazu br. 5 vidi se kako stranica sadrži liste sačuvanih ponuda pod pripadnim karticama (*Flights*, *Bus*, *Rooms* i *Rent-a-car*). Komponente spremljenih ponuda slične su komponentama ponuda s prijašnjih stranica na kojima se ponude pretražuju. Jedine su razlike što su ovim karticama dodani datumi „*From*“ (od kad) i „*To*“ (do kad) te što imaju drukčiji crveni gumb koji služi za brisanje komponente s liste favorita pomoću kojeg je realiziran funkcionalni zahtjev F8.

4.3. React Bootstrap

Za potrebe izrade web aplikacije, korišten je *CSS* razvojni okvir *React Bootstrap*.

```
Import Carousel from „react-bootstrap/Carousel“;
import bus1Img from „../assets/rsz_2bus1.jpg“;
import bus2Img from „../assets/rsz_1bus2.jpg“;
import bus3Img from „../assets/rsz_bus3.jpg“;
import classes from „./BusCarousel.module.css“;

const BusCarousel = () => {
  return (
    <Carousel className={classes.busCarousel}>
      <Carousel.Item interval={4000}>
        <img src={bus1Img} alt=“First slide“ />
        <Carousel.Caption className={classes.textStyle}>
          <h3>Find your ideal bus line </h3>

          <p>Travel cheap and stressless</p>
```

```
      </Carousel.Caption>
    </Carousel.Item>
    <Carousel.Item interval={4000}>
      <img src={bus2Img} alt="Second slide" />
      <Carousel.Caption className={classes.textStyle}>
        <h3>Comfortable seats</h3>
        <p>The best, modern Flixbus vehicles</p>
      </Carousel.Caption>
    </Carousel.Item>
    <Carousel.Item interval={4000}>
      <img src={bus3Img} alt="Third slide" />
      <Carousel.Caption className={classes.textStyle}>
        <h3>Enjoy the world around you</h3>
        <p>Notice all beautiful nature and arhitecture while travelling</p>
      </Carousel.Caption>
    </Carousel.Item>
  </Carousel>
 );
};

export default BusCarousel;
```

Programski kod 3.: Primjer karusel komponente
Izvor: autor

U programskom kod br. 3 prikazana je karusel komponentu koja sadrži tri fotografije s tekstom. Najprije je potrebno uvesti *Carousel* komponentu iz *React Bootstrap*-a što je vidljivo u prvoj liniji koda. Funkcionalna komponenta koja je kreirana, nazvana je *BusCarousel* jer se govori o karuselu koji je korišten na stranici za pretraživanje autobusnih linija. To je jednostavna komponenta koja vraća *JSX* kod s vršnim elementom *Carousel*. Karusel se sastoji od tri *React Bootstrap* komponente ili stavke (engl. items) koje se izmjenjuju svake 4 sekunde (4000 milisekunda) i koje sadrže fotografije koje su uvezene iz direktorija a*ssets* u kojem se nalaze sve fotografije iz web aplikacije. Unutar stavki koriste se još i *React Bootstrap* komponente *Caption* (dijete od *Carousel*) koje mogu sadržati tekstualni sadržaj koji će se prikazati preko fotografija. Primjerom je prikazana jednostavnost korištenja stiliziranih *React Bootstrap* komponenti.

4.4. Navigacija

Tradicionalna metoda navigacije je navigacijska traka i gotovo svaka web aplikacija je sadrži. Navigacijska traka sadrži poveznice koje korisnika vode na druga mjesta ili stranice na web aplikaciji. Može se implementirati na više načina, vodoravno ili okomito, fiksno ili dinamično. Ovdje će se opisati realizacija funkcionalnog zahtjeva F2 opisanog u tablici br. 1.

U konkretnoj aplikaciji implementirana je vodoravna navigacijska traka koja se fiksno nalazi uvijek na vrhu zaslona i pomoću nje je omogućeno korisniku kretanje među stranicama. Na

užim zaslonima (mobiteli) poveznice nestaju s trake i pojavljuje se gumb za izbornik. Pritiskom na gumb otvara se i zatvara izbornik s vertikalno poredanim poveznicama. Komponenta *MainHeader.js* predstavlja navigacijsku traku s poveznicama (*NavLink* elementi čiji atribut *to* prima relativnu putanju stranice koja je definirana pomoću *React Router*-a te preusmjerava korisnika na tu stranicu) u aplikaciji prikazana je u programskom kodu br. 4.

```
import { NavLink } from „react-router-dom“;
import classes from „./MainHeader.module.css“;
import { Button } from „react-bootstrap“;
import { useState } from „react“;

const MainHeader = () => {

 return (
  <div>
   <header className={classes.header}>
    <nav>
     <ul className={classes.desktop}>
      <li>
       <NavLink activeClassName={classes.active} to=“/“ exact>
        Home
       </NavLink>
      </li>
      <li>
       <NavLink activeClassName={classes.active} to=“/airlines“>
        Flights
       </NavLink>
      </li>
      <li>
       <NavLink activeClassName={classes.active} to=“/buses“>
        Bus
       </NavLink>
      </li>
      <li>
       <NavLink activeClassName={classes.active} to=“/booking“>
        Rooms
       </NavLink>
      </li>
      <li>
       <NavLink activeClassName={classes.active} to=“/rentals“>
        Rent-a-car
       </NavLink>
      </li>
      <li className={classes.last}>
       <NavLink activeClassName={classes.active} to=“/favorites“>
        <svg
         xmlns=“http://www.w3.org/2000/svg“
         width=“26“
         height=“26“
         fill=“currentColor“
         className=“bi bi-heart“
         viewBox=“0 0 16 16“
        >
         <path d=“m8 2.748-.717-.737C5.6.281 2.514.878 1.4 3.053c-.523 1.023-.641 2.5.314
4.385.92 1.815 2.834 3.989 6.286 6.357 3.452-2.368 5.365-4.542 6.286-6.357.955-1.886.838-3.362.314-
```

```
4.385C13.486.878 10.4.28 8.717 2.01L8 2.748zM8 15C-7.333 4.868 3.279-3.04 7.824
1.143c.06.055.119.112.176.171a3.12 3.12 0 0 1 .176-.17C12.72-3.042 23.333 4.867 8 15z“ />
                    </svg>
                  </NavLink>
                </li>
              </ul>
                . . .
```

Programski kod 4.: Primjer navigacijske trake s linkovima
Izvor: autor

Za funkcioniranje navigacije zaslužan je *React Router* čija primjena je prikazana u nastavku.

4.4.1. React Router

U web aplikaciji je korišten *BrowserRouter* koji unutar *index.js root* datoteke omata *App* komponentu tj. cijelu aplikaciju što je prikazano kroz programski kod br. 5.

```
<BrowserRouter>
    <App />
  </BrowserRouter>
```

Programski kod 5.: BrowserRouter omata aplikaciju
Izvor: autor

Zatim je unutar *App.js* komponente definirana *Switch* komponenta koja sadrži po jednu rutu (engl. route) za svaku stranicu u aplikaciji. Svaka ruta sadrži relativnu putanju za određenu stranicu i omata komponentu stranice do koje ta ruta vodi. Unutar navigacijske trake koriste se *NavLink*-ovi čiji atribut *to* prima relativnu putanju stranice koja je ranije definirana te preusmjerava korisnika na tu stranicu, dok sve ostale putanje koje nisu definirane vode na *NotFound* stranicu. Taj primjer definiranja ruta prikazan je u nastavku kroz programski kod br. 6.

```
import { Route, Switch } from "react-router-dom";
import MainHeader from "./components/MainHeader";
import Buses from "./pages/Buses";
import Booking from "./pages/Booking";
import Airlines from "./pages/Airlines";
import Rentals from "./pages/Rentals";
import Home from "./pages/Home";
import NotFound from "./pages/NotFound";
import Favorites from "./pages/Favorites";

function App() {
  return (
    <div>
      <MainHeader />
      <main>
```

```
        <Switch>
          <Route path="/" exact>
            <Home />
          </Route>
          <Route path="/buses">
            <Buses />
          </Route>
          <Route path="/booking">
            <Booking />
          </Route>
          <Route path="/airlines">
            <Airlines />
          </Route>
          <Route path="/rentals">
            <Rentals />
          </Route>
          <Route path="/favorites">
            <Favorites />
          </Route>
          <Route path="*">
            <NotFound />
          </Route>
        </Switch>
      </main>
    </div>
  );
}

export default App;
```

Programski kod 6.: Definiranje ruta unutar aplikacije
Izvor: autor

4.5. Načini upravljanja podacima unutar aplikacije

Jedan od načina upravljanja podacima u aplikaciji je pomoću *useState Hook*-a koji je već opisan u točki 2.2.1., a još jedan način je preko koncepta *props* koji omogućuje podacima ili vrijednostima prelazak iz komponente roditelja u komponentu dijete.

```
import styles from "./Card.module.css";
import { Link } from "react-router-dom";

const Card = (props) => {
  return (
    <div className={styles.card}>
      <div className={styles["card__body"]}>
        <img src={props.img} alt={props.alt} />
        <h2 className={styles["card__title"]}>{props.title}</h2>
        <p className={styles["card__description"]}>{props.description}</p>
      </div>
      <button className={styles["card__btn"]}>
        <Link className={styles["link"]} to={props.route}>
          Research
        </Link>
      </button>
    </div>
  );
};
```

```
export default Card;
```

Programski kod 7.: Korištenje props-a
Izvor: autor

U prethodnom, programskom kodu br. 7 prikazana je komponenta *Card* koja putem *props*-a prima sadržaj od komponente roditelja. Komponenta pretpostavlja primitak slike *img*, opisa slike a*lt*, naslova *title*, opisa *description* i putanje *route* za preusmjeravanje na drugu stranicu.

```
import Card from "../components/Card";
import styles from "./Home.module.css";

import airplaneImg from "../assets/airplane_img.jpg";
import busImg from "../assets/bus_img.jpg";
import carImg from "../assets/car_img.jpg";
import hotelImg from "../assets/hotel_img.jpg";

const Home = () => {
  return (
    <div>
      <h1 className={styles["heading-welcome"]}>Welcome to my travel app!</h1>
      <p className={styles["paragraph_intro"]}>
        Find information about bus routes, flights, rent-a-cars and
        accomodation!
      </p>
      <div className={styles.wrapper}>
        <Card
          img={airplaneImg}
          alt="Airplane"
          title="Flights"
          description="Find your perfect flight"
          route="/airlines"
        />
        <Card
          img={busImg}
          alt="Bus"
          title="Bus lines"
          description="Find your perfect bus line"
          route="/buses"
        />
        <Card
          img={hotelImg}
          alt="Hotel"
          title="Rooms"
          description="Find the best accomodation for your trip"
          route="/booking"
        />
        <Card
          img={carImg}
          alt="Car"
          title="Rent-a-car"
          description="Pick your favorite car for travel"
          route="/rentals"
        />
      </div>
    </div>
```

```
  );
};

export default Home;
```

Programski kod 8.: Card komponente primaju sadržaj preko props-a
Izvor: autor

Iz prikazanog primjera u programskom kodu br. 8 možemo vidjeti kako se unutar *Home* komponente (roditelj) nalaze *Card* komponente (djeca) koje putem svojstava (engl. props) primaju sadržaj koji onda koriste u svom kodu.

4.5.1. React Redux

React Redux korišten je u web aplikaciji za skladištenje vanjskih podataka koji se dohvaćaju s *API*-ja i zatim prikazuju u web aplikaciji, pa je na taj način uključen u realizaciji funkcionalnih zahtjeva F3, F4, F5 i F6. Za lakše korištenje *Redux*-a, instaliran je *Redux-Toolkit* uz *react-redux* paket pomoću sljedeće naredbe u terminalu:

```
npm install @reduxjs/toolkit react-redux
```

Zatim je kreirana mapa s*tore* unutar /source direktorija. U toj mapi je onda kreirana index.js datoteka koja predstavlja skladište za stanja. U toj datoteci je uvezen *configureStore API* iz *Redux Toolkit*-a pomoću kojeg se stvara skladište (engl. store). Također je uvezen i *createSlice API* za stvaranje odsječka (engl. slice). Primjer kreiranja jednog odsječka prikazan je kroz programski kod br. 9.

```
import { createSlice, configureStore } from "@reduxjs/toolkit";

const rentalState = {
  location: [],
  isLocationChanged: false,
  cars: [],
};

const rentalSlice = createSlice({
  name: "rental",
  initialState: rentalState,
  reducers: {
    uploadLocation(state, action) {
      state.location = action.payload.location;
      state.isLocationChanged = true;
    },
    uploadCars(state, action) {
      state.cars = action.payload.cars;
    },
    uploadLocationChanged(state, action) {
      state.isLocationChanged = action.payload.isLocationChanged;
    },
```

```
  },
});
```

Programski kod 9.: Kreiranje Redux odsječka
Izvor: autor

Odsječak je kreiran pomoću *createSlice API*-ja na način da se mora definirati njegovo ime, inicijalno stanje i reduktori. Reduktori omogućuju opisivanje logike promjene koju želimo postići. Kroz definirane promjene stanje se ne mijenja nego se po tim promjenama kreira novo nepromjenjivo stanje. Sad se može kreirati skladište i mogu mu se dodati jedan ili više pripadnih reduktora (engl. reducers) te se mogu definirati izvozi za daljnji rad, a to je prikazano u programskom kodu br. 10.

```
const store = configureStore({
  reducer:
    rental: rentalSlice.reducer,
  },
});

export const rentalActions = rentalSlice.actions;
export const rentalLocation = rentalSlice.getInitialState.location;
export const rentalCars = rentalSlice.getInitialState.cars;
export const rentalIsChanged = rentalSlice.getInitialState.isLocationChanged;
export default store;
```

Programski kod 10.: Kreiranje Redux skladišta
Izvor: autor

Provider komponenta omata cijelu aplikaciju kako bi skladište (engl. store) bilo dostupno svugdje i to je prikazano u programskom kodu br. 11.

```
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <Provider store={store}>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </Provider>
);
```

Programski kod 11.: Provider komponenta omata aplikaciju
Izvor: autor

Nakon toga je moguće slati akcije u skladište na razini cijele web aplikacije pomoću *Redux hook*-a *useDispatch()*. Unutar komponente za dohvat podataka o automobilima za najam, prilikom dohvata podataka, podaci će se slati putem akcije u skladište. Najprije treba uvesti potrebne akcije i *hook*-a, te inicijalizirati *hook*-ove što je prikazano u programskom kodu br. 12.

```
import { rentalActions } from "../store";
import { useDispatch } from "react-redux";
const dispatch = useDispatch();
```

Programski kod 12.: Uvoz akcija i useDispatch Hook-a te njena inicijalizacija
Izvor: autor

Nakon što su automobili dohvaćeni s *API*-ja i pospremljeni u niz, taj niz se može poslati (engl. dispatch) u skladište pomoću akcije *uploadCars()*. Korištenje *dispatch* funkcije prikazano je u kodu br. 13.

```
dispatch(rentalActions.uploadCars({ cars: carsArr }));
```

Programski kod 13.: Primjer korištenja dispatch funkcije
Izvor: autor

Sad se ti podaci o automobilima mogu koristiti unutar bilo koje komponente pomoću *useSelector Hook-a* na sljedeći način što je prikazano u programskom kodu br. 14.

```
import { useSelector } from "react-redux";
const carsCheap = useSelector((state) => state.rental.cars);
```

Programski kod 14.: Primjer uvoza useSelector Hook-a i njeno korištenje
Izvor: autor

4.6. Dohvaćanje podataka s API-ja

Za slanje zahtjeva na API uvijek je korištena fetch metoda. Fetch API moćna je i jednostavna za korištenje alternativa AJAX-u i jQuery-u. Fetch metoda uzima jedan obavezan argument, a to je putanja do resursa kojeg želi dohvatiti te vraća „obećanje“ (engl. promise) koje se razrješava u odgovor na taj zahtjev. Primjer kreiranja zahtjeva detaljno je opisan u nastavku te su tako realizirani funkcionalni zahtjevi F3, F4, F5 i F6.

```
const searchFlixTrips = () => {
  fetch(
    `https://flixbus.p.rapidapi.com/v1/search-
trips?to_id=${cityIdTo}&from_id=${cityIdFrom}&currency=EUR&departure_date=${tripDate}&number_ad
ult=1&search_by=cities`,
    optionsSearchTrips
  )
    .then((response) => response.json())
    .then((data) => {
      let num = 0;
      const trips = data[0].items;
      const arr1 = [];
      for (const el of trips) {
        const from = data[0].from.name;
        const to = data[0].to.name;
        const departure = el.departure.timestamp;
        const arrival = el.arrival.timestamp;
        const duration = el.duration.hour + ":" + el.duration.minutes;
```

```
        const price = el.price_total_sum;
        num = num + 1;
        const id = num;
        arr1.push({
          id: id,
          date: tripDate,
          from: from,
          to: to,
          departure: departure,
          arrival: arrival,
          duration: duration,
          price: price,
        });
      }
      dispatch(tripsActions.uploadTrips({ trips: arr1 }));
      setFlixTripsError({ error: false });
      onShowTable(true);
    })
    .catch((err) => {
      console.error(err);
      dispatch(tripsActions.uploadTrips({ trips: [] }));
      setFlixTripsError({ error: true });
      onShowTable(false);
    });
};
```

Programski kod 15.: Dohvaćanje podataka pomoću Fetch API-ja
Izvor: autor

U ovom primjeru u programskom kodu br. 15 prikazana je primjena *fetch* metode pomoću koje se dohvaćaju putovanja s *Flixbus API*-ja čime je konkretno omogućen funkcionalni zahtjev F3. *Fetch* metoda prima dva argumenta: putanju do resursa i opcije u kojima su informacije o metodi i zaglavljima (engl. headers). Ti argumenti su dostupni u *API* dokumentaciji ako ste prijavljeni na *Rapid API* platformi. Nakon poslanog zahtjeva, vraćeni odgovor se pretvara u *JSON*, te se zatim dohvaćeni podaci mogu koristiti. Za svako dohvaćeno putovanje, u novi niz se sprema objekt s potrebnim podacima o putovanju te se na kraju cijeli niz objekata u *dispatch* metodi koristi u poslanoj akciji u *React store* gdje se stanje ažurira. Na kraju metode odobrava se prikaz tablica s ponudama putovanja na stranici. Ukoliko se dogodi nekakva pogreška prilikom dohvata podataka, u React store-u će se ažurirati prazni niz s putovanjima i tablica na stranici neće biti prikazana.

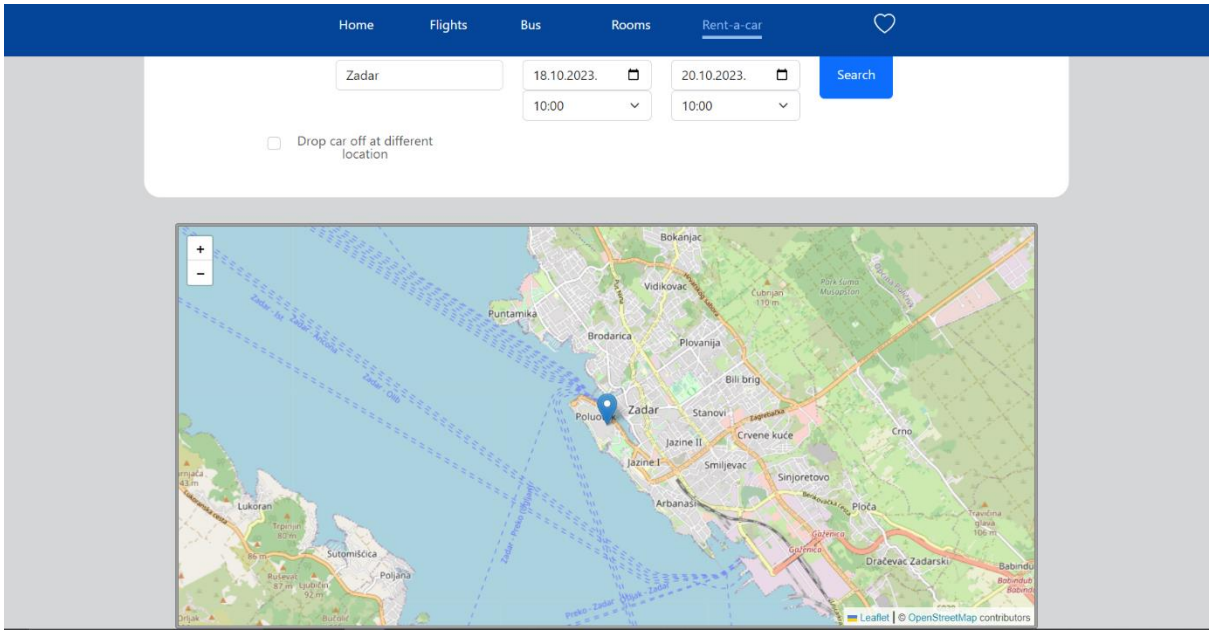
4.7. React Leaflet

React Leaflet je korišten u web aplikaciji za kreiranje interaktivnih karti. Smisao karti je prikazivanje lokacija na koje korisnik planira putovati i tako će se realizirati funkcionalni zahtjev F9. Na stranicama za pretraživanje smještaja i za pretraživanje automobila, na kartama se prikazuju lokacije na kojima se nalaze traženi smještaji i agencije za najam automobila

```
import React from "react";
import { Marker, TileLayer, MapContainer, useMapEvent } from "react-leaflet";
import { useSelector } from "react-redux";
import styles from "./Map.module.css";

const MapRental = () => {
  const position = useSelector((state) => state.rental.location);
  const isPositionChanged = useSelector(
    (state) => state.rental.isLocationChanged
  );

  const MapBounds = () => {
    const map = useMapEvent("click", () => {
      map.fitBounds([position, position]);
    });
  };

  return (
    <div>
      <MapContainer
        className={styles.map}
        center={[51.4381, 5.4752]}
        zoom={2}
        scrollWheelZoom={false}
      >
        <TileLayer
          attribution='&copy;        <a        href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors'
          url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
        />
        {isPositionChanged && <Marker position={position}></Marker>}
        {isPositionChanged && <MapBounds />}
      </MapContainer>
    </div>
  );
};

export default MapRental;.
```

Programski kod 16.: Primjer React komponente za Leaflet kartu
Izvor: autor

U programskom kodu br. 16 prikazana je *React* komponenta *MapRental* koja predstavlja *React Leaflet* kartu za stranicu koja služi za pretraživanje automobila za najam. Komponenta *MapContainer* odgovorna je za kreiranje instance *Leaflet* karte. *Marker* će biti prikazan na karti ovisno o stanjima koji se čitaju iz *React store*-a pomoću *useSelector Hook*-a, pa tako React Redux ima svoju ulogu u realizaciji funkcionalnog zahtjeva F9. Podatak *position* predstavlja koordinate lokacije, a *isPositionChanged* je pomoćni podatak tipa *boolean* o kojem ovisi hoće li se marker prikazivati na karti. *MapBounds* je funkcija koja koristi *useMapEvent Hook* koji omogućuje fokusiranje karte na označeni marker s klikom pokazivačem na kartu.



Slika 6.: Prikaz markera na karti na mjestu tražene lokacije
Izvor: autor

U prikazu br. 6 vidi se kako se lokacija markera na karti ažurira prilikom pretrage automobila na lokaciji koja korisnika zanima – u ovom primjeru tražena lokacija je grad Zadar.

4.8. Upravljanje pogreškama

Upravljanje pogreškama bitna je stavka u svakoj web aplikaciji. Svaka web aplikacija koja ima nekakvu komunikaciju s vanjskim poslužiteljem ili bazom podataka u vidu dohvata/slanja podataka ili slično, mora očekivati pogreške i njima upravljati. U ovoj web aplikaciji očekuje se da će se događati pogreške prilikom dohvaćanja podataka s *API*-ja ako se ne uputi dobar zahtjev, ali se ne smije dogoditi nefunkcionalnost aplikacije. U primjeru iz točke 4.6. prikazana je primjena *fetch* metode. Radi se o slanju *HTTP* zahtjeva što znači da postoji mogućnost mnogih mrežnih pogreški. Na kraju *fetch* metode nalazi se *catch* metoda koja hvata potencijalnu pogrešku. *Catch* metoda sadrži logiku koja sprječava prestanak funkcioniranja aplikacije. U navedenom slučaju pogreške u konzoli se ispisuje poruka o nastaloj pogrešci, šalje se akcija na *React store* za ažuriranje niza s putovanjima na prazan niz jer *fetch* nije uspio, unutar komponente *flixTripsError* stanje se ažurira na *true* i *props* metoda *onShowTable* naređuje ne prikazivanje *Table* komponente na stranici za pretraživanje autobusnih linija. Ako stanje *flixTripsError.error* ima vrijednost *true*, na dnu forme se prikazuje *React Bootstrap Alert* komponenta koja korisnika obavještava da nije pronađena nijedna autobusna linija kao što je prikazano u sljedećem isječku koda:

```
{flixTripsError.error && (
      <Alert key="warning" variant="warning">
        Couldn't find any bus line that matches your request.
      </Alert>
    )}
```

Programski kod 17.: Prikaz upozorenja ukoliko se dogodila pogreška
Izvor: autor

Sljedeći put kad cijela *fetch* metoda prođe bez pogreške, *error* stanje će se postaviti na *false* i maknut će se prikazana obavijest te će se na stranici prikazati tablica s ponudama za autobusne linije koje su dohvaćene s *API*-ja i spremljene u *store*. Na taj način je realiziran funkcionalni zahtjev F11.

4.9. Responzivnost

Već je ranije spomenuto da responzivnost znači prilagodba web stranice na različite zaslone. U ovom odjeljku bit će opisano kako je realiziran funkcionalni zahtjev F12. Prvo će biti prikazan jednostavniji primjer prikaza kartica s početne stranice aplikacije na širokom računalnom ekranu i na mobilnom uređaju. Na računalu su kartice poredane horizontalno (vidljivo ranije na prikazu br. 3 u ovoj cjelini rada), a prilikom posjeta iste stranice na mobilnom uređaju kartice se prikazane u vertikalnom rasporedu jedna ispod druge. To je omogućio sljedeći programski kod br. 18 u kojem su kartice predstavljene kao *Card* komponente omotane s blokom *div* čija *CSS* pravila mijenjaju raspored kartica ovisno o širini zaslona:

```
import Card from "../components/Card";
import styles from "./Home.module.css";

import airplaneImg from "../assets/airplane_img.jpg";
import busImg from "../assets/bus_img.jpg";
import carImg from "../assets/car_img.jpg";
import hotelImg from "../assets/hotel_img.jpg";

const Home = () => {
  return (
    <div>
      <h1 className={styles["heading-welcome"]}>Welcome to my travel app!</h1>
      <p className={styles["paragraph_intro"]}>
        Find information about bus routes, flights, rent-a-cars and
        accomodation!
      </p>
      <div className={styles.wrapper}>
        <Card
          img={airplaneImg}
          alt="Airplane"
          title="Flights"
          description="Find your perfect flight"
          route="/airlines"
        />
```

```
        <Card
          img={busImg}
          alt="Bus"
          title="Bus lines"
          description="Find your perfect bus line"
          route="/buses"
        />
        <Card
          img={hotelImg}
          alt="Hotel"
          title="Rooms"
          description="Find the best accomodation for your trip"
          route="/booking"
        />
        <Card
          img={carImg}
          alt="Car"
          title="Rent-a-car"
          description="Pick your favorite car for travel"
          route="/rentals"
        />
      </div>
    </div>
  );
};

export default Home;
```

Programski kod 18.: Početna stranica
Izvor: autor

CSS pravila iz *Home.module.css* datoteke vezana za omotač oko kartica prikazana su kroz programski kod br. 19.

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(12rem, 16rem));
  gap: 2rem;
  justify-content: center;
}
```

Programski kod 19.: CSS pravila za wrapper selektor
Izvor: autor

Iz koda br. 19 je vidljivo kako su kartice prvo prikazane mrežnim sustavom gdje su kartice prikazane horizontalno dok zaslon nije preuzak za to (tad bi se sužavanjem zaslona, kartica po kartica prebacivale u novi red). Prilikom širine od 74em, zadnja kartica prebacila bi se u drugi red, ali želimo to izbjeći i želimo da se u dva reda poslože po dvije kartice, pa je napisan medijski upit (engl. media query) kojim se kartice prikazuju u dva stupca, a to je prikazano u programskom kodu br. 20.

```
@media (max-width: 74em) {
  .wrapper {
    display: grid;
    grid-template-columns: 20rem 20rem;
    row-gap: 0;
    justify-content: center;
  }
}
```

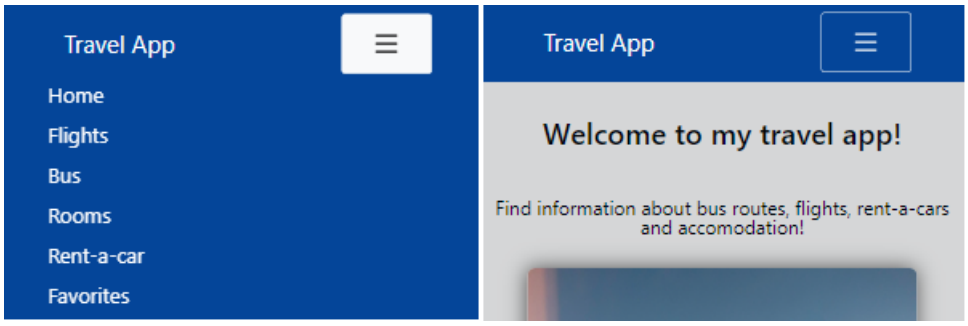
Programski kod 20.: Media upit za uži ekran
Izvor: autor

No, kad se zaslon suzi do 46em (mobilni uređaj), naš dizajn postaje nefunkcionalan jer je ekran preuzak za dva stupca, pa je definiran sljedeći medijski upit koji će ostaviti nešto prostora s lijeve i desne strane zaslona, a kartice će rasporediti pomoću *flexbox* vertikalnog raspoređivanja (raspored u jednom pravcu) sa 1rem razmaka između svake kartice što je prikazano u programskom kodu br. 21.

```
@media (max-width: 46em) {
  .wrapper {
    width: 80%;
    margin-left: auto;
    margin-right: auto;
    display: flex;
    flex-direction: column;
    gap: 1rem;
    justify-content: center;
    align-items: center;
  }
```

Programski kod 21.: Media upit za prikaz na mobitelu
Izvor: autor

No, neke komponente moraju se skroz promijeniti umjesto mijenjanja samo njihov rasporeda, pa je tako mobilna navigacijska traka skroz drukčija komponenta..



Slika 7.: Primjer izgleda mobilne navigacije iz aplikacije
Izvor: autor

U prikazu br. 7 nalazi se primjer izgleda navigacije na mobilnom zaslonu, a izgled navigacijske trake na širokim zaslonima vidljiv je ranije sa slika iz odlomka o korisničkom sučelju i

funkcionalnostima. Na mobilnoj navigacijskoj traci se umjesto poveznica nalazi *Menu* gumb koji klikom otvara i zatvara izbornik s vertikalno poredanim navigacijskim poveznicama.

Logika je ta da se na širokim zaslonima prikazuje desktop traka sve dok razmak između poveznica ne ostane postojan i dizajn postane nefunkcionalan, tada se s prikaza miče desktop traka i postavlja se mobilna traka s *Menu* gumbom. To je realizirano s medijskim upitom prikazanim u programskom kodu br. 22.

```
@media (max-width: 46em) {
 .desktop {
   display: none;
 }

 .mobile {
   display: flex;
   align-items: center;
   justify-content: space-between;
 }

 a {
   text-decoration: none;
 }

 .header2 {
   display: block;
   position: fixed;
   top: 5rem;
   width: 100%;

   background-color: #044599;
   padding: 0 10%;
   margin-bottom: 3rem;
   z-index: 100000;
 }

 .header2 a {
   color: white;
   font-size: 1.3rem;
   text-decoration: none;
 }

 .mobileOpen {
   height: 100%;
   list-style: none;
   display: flex;
   flex-direction: column;
   padding: 0;
   margin: 0;
   align-items: flex-start;
   justify-content: center;
  color: white;
   gap: 1.2rem;
 }

 .last2 {
   margin-bottom: 1.2rem;
```

```
    }
}
```

Programski kod 22.: Media upit za mobilnu navigacijsku trak
Izvor: autor

Selektor *desktop* se odnosi na desktop navigacijsku traku koja se sad više ne prikazuje. Selektor *header2* odnosi se na mobilnu traku koja na širim zaslonima nije prikazivana (display: none;), a sad je ona fiksno postavljena uvijek na vrhu zaslona i unutar nje je horizontalno posložena lista (čiji je selektor *mobile*) sa stavkama liste: tekstom „Travel app“ i gumbom za izbornik pomoću *flexbox*-a. Selektor *mobileOpen* predstavlja *CSS* pravila za izbornik koji se uvjetno otvara/zatvara (prilikom pritiska gumba na traci) i unutar kojeg su *NavLink*-ovi raspoređeni vertikalno.

5. MOGUĆE NADOGRADNJE

U ovoj cjelini navedene su moguće nadogradnje na web aplikaciji. Budući da je u radu opisana frontend aplikacija, logična mogućnost nadogradnje bila bi dodavanje backend dijela aplikacije. To znači da bi web aplikacija bila spojena s bazom podataka u koju bi se mogli spremati podaci. Značajan dodatak bio bi registriranje korisnika i postojanje korisničkih računa. Prilikom prijave u svoj korisnički račun, korisnici bi mogli pristupiti svojim spremljenim ponudama koje su označili kao *favorite*. U tom slučaju, podaci iz web aplikacije ne bi bili izgubljeni nakon ponovnog pokretanja, već bi bili spremljeni u bazi podataka.

Još jedna moguća nadogradnja je dodavanje dodatnih *API*-ja za dohvat podataka, pa bi se tako pokrio morski i željeznički promet. Onda bi aplikacija pokrivala sve moguće načine putovanja koja bi korisnik mogao pretraživati i sve informacije bi imao na jednom web mjestu.

Web aplikacija bi bila pristupačnija kad bi korisnik mogao birati među više ponuđenih govornih jezika, pa bi se sadržaj aplikacije prikazivao na odabranom jeziku. Tada bi pregled sadržaja aplikacije bio olakšan ljudima koji posve ne razumiju engleski jezik.

Korisna nadogradnja mnogim korisnicima bi bila dodavanje bloga unutar web aplikacije gdje bi korisnici mogli s drugima dijeliti svoje priče ili savjete s putovanja.

Još jedan korak dalje u vidu moguće nadogradnje bio bi kad bi se preko web aplikacije mogle kupovati putne karte putničkih agencija te rezervirati smještaj i automobili. Tad bi to postala poslovna aplikacija.

6. ZAKLJUČAK

U ovom radu opisan je kompletan proces izrade frontend web aplikacije za pomoć pri planiranju putovanja. Prvo je identificiran problem koji je potrebno riješiti. Dan je pregled i opis tehnoloških rješenja pomoću kojih će se naša web aplikacija izraditi. Za identificirani problem definirano je 12 funkcionalnih zahtjeva. Funkcionalni zahtjevi su realizirani primjenom opisanih tehnoloških rješenja. Izrađena je web aplikacija koja ima grafičko korisničko sučelje, forme za unos podataka o putovanju (autobus, avion, smještaj i najam automobila). Omogućeno je povezivanje web aplikacije s vanjskim izvorima podataka tj. omogućen je dohvat podataka putem API-ja. Podaci se prikazuju na korisničkom sučelju putem lista s ponudama i interaktivnih karti. Web aplikacija omogućava spremanje ponuda o putovanjima u favorite, upravljanje pogreškama te omogućava pristup različitim digitalnim uređajima (osobna računala, mobiteli, tableti itd.). Za kraj dan je pregled mogućih nadogradnji web aplikacije.

Web aplikacija je napravljena pomoću razvojnog okvira *React* koristeći *JavaScript* programski jezik. Uz ReactJS korišteno je nekoliko dodatnih biblioteka pomoću kojih su uspješnije realizirani funkcionalni zahtjevi. Za moderno stiliziranje i raspored komponenti korištena je *CSS* biblioteka *React Bootstrap*. Za izradu navigacije koristio se alat *React Router* pomoću kojeg je implementirano usmjeravanje na klijentskoj strani i time izbjegnuto slanje ponovnih zahtjeva za *HTML* dokumentima što bi usporavalo rad aplikacije. Interaktivne karte na kojima se dinamički prikazuju željene lokacije integrirane su pomoću *React Leaflet* biblioteke. Svi podaci vezani za putovanja dohvaćaju se s *REST API*-ja pomoću *Fetch* metode te se spremaju u *React Redux Store* čime je omogućeno fleksibilnije upravljanje istim podacima bilo gdje u aplikaciji.

7. LITERATURA I KORIŠTENI IZVORI

Agafonkin, Volodymyr. „Documentation - Leaflet - a JavaScript library for interactive maps“. Pristupljeno 21. rujan 2023. https://leafletjs.com/reference.html.

Arancio, Stephen. „What is JSX?. A look at what JSX is, what it is used… | Medium“, 06. listopad 2021. https://medium.com/@sjarancio/what-is-jsx-e3dda0af3490.

Banks, Alex, i Eve Porcello. „Learning React, 2nd Edition [Book]“, lipanj 2020. https://sd.blackball.lv/library/Learning_React_(2020).pdf.

Bugl, Daniel. *Learn React Hooks: Build and Refactor Modern React.Js Applications Using Hooks*. Packt Publishing Ltd, 2019.

Dhruti-Shah. *Node .Js*. BPB Publications, 2018.

Fedosejev, Artemij. *React.Js Essentials*. Packt Publishing Ltd, 2015.

Kaluža, Marin, i Bernard Vukelic. „Comparison of front-end frameworks for web applications development“. *Zbornik Veleučilišta u Rijeci* 6 (01. siječanj 2018.): 261–82. https://doi.org/10.31784/zvr.6.1.19.

Marcotte, Ethan. „Responsive Web Design“. A List Apart, 25. svibanj 2010. https://alistapart.com/article/responsive-web-design/.

Massé, Mark H., i Mark Massé. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. Beijing Köln: O’Reilly, 2012.

Meta Open Source. „Reusing Logic with Custom Hooks – React“. Pristupljeno 21. rujan 2023. https://react.dev/learn/reusing-logic-with-custom-hooks.

Mozilla. „JavaScript | MDN“. Pristupljeno 08. rujan 2023. https://developer.mozilla.org/en-US/docs/Web/JavaScript.

Powell, Thomas A., i Thomas A. Powell. *HTML & CSS: The Complete Reference*. 5th ed. New York: McGraw-Hill, 2010.

Rauschmayer, Axel. *JavaScript for Impatient Programmers*. s.l.: s.n., 2019.

React Bootstrap. „React Bootstrap“. Pristupljeno 11. rujan 2023. https://react-bootstrap.github.io/.

Škalac, Lorena. „ReactJS“. Diplomski rad, Sveučilište Josipa Jurja Strossmayera, Odjel za matematiku, 2017. https://urn.nsk.hr/urn:nbn:hr:126:915147.

Young, Devon, Craig Buckler, i Optimalworks Ltd. „Praise for the First Edition of The Book of CSS3“, 2015.

DEVELOPMENT OF WEB APPLICATION FOR HELP IN TRAVEL PLANNING

Summary

This paper shows the creation of web application for help in travel planning which will give information to consumer about few domains of travelling and will show locations on interactive maps. The initial problem which arises from lack of functionalities in other travel applications is identified. The overview of technology solutions which could help in problem resolving is also given. Requirements for the application are identified and described. Development of the application with help from described technologies and solutions is described and possible upgrades of the application are identified.

Keywords: web application for travel planning, ReactJS, Application Programming Interface, interactive maps

8. PRILOZI

Prilog br. 1 – POPIS SLIKA

Prilog br. 2 - POPIS PROGRAMSKOG KODA

Prilog br. 3 - POPIS TABLICA