UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

ZAKLJUČNA NALOGA

ZAKLJUČNA NALOGA

ZAPOLNI MOJ AVTO: RAZVOJ SPLETNE APLIKACIJE ZA SKUPNO VOŽNJO NEZNANIH POTNIKOV

ROK SAMSA

2021

SAMSA

UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga

Zapolni moj avto: Razvoj spletne aplikacije za skupno vožnjo neznanih potnikov

(Fill my car: Developing a web application for sharing drive
with unknown passengers)

Ime in priimek: Rok Samsa
Študijski program: Računalništvo in informatika
Mentor: doc. dr. Matjaž Kljun
Somentor: doc. dr. Klen Čopič Pucihar

Koper, avgust 2021

Ključna dokumentacijska informacija

Ime in PRIIMEK: Rok SAMSA

Naslov zaključne naloge: Zapolni moj avto: Razvoj spletne aplikacije za skupno vožnjo neznanih potnikov

Kraj: Koper

Leto: 2021

Število listov: 67 Število slik: 21 Število tabel: 4

Število prilog: 3 Št. strani prilog: 6

Število referenc: 34

Mentor: doc. dr. Matjaž Kljun

Somentor: doc. dr. Klen Čopič Pucihar

Ključne besede: spletna aplikacija, Angular, deljenje lastnega vozila, potovanja, skupna vožnja neznanih potnikov

Izvleček:

V zaključni nalogi je predstavljen razvoj spletne aplikacije »Zapolni moj avto«, preko katere lahko uporabniki delijo svoje prevoze z neznanimi potniki. Cilj aplikacije je izboljšava oziroma nadgradnja že obstoječe spletne aplikacije »Prevoz.org« - predvsem izbire prevoza in izbire potnikov. Pri obstoječi aplikaciji »Prevoz.org« lahko namreč ponudnik, ki želi deliti svoj prevoz, le-tega da na seznam vključno s svojo telefonsko številko. Uporabniki, ki iščejo prevoz, pa lahko ponudnike le pokličejo. Na koncu imata obe strani veliko neznanih telefonskih številk, saj vsak, ki pokliče ponudnika, ne izbere njegov prevoz in vsak, ki išče prevoz, ne pokliče le enega ponudnika. To težavo smo izboljšali z uvedbo prijave na samo pot in tako olajšali uporabo aplikacije. V zaključni nalogi je predstavljen celoten opis poteka razvoja omenjene rešitve v Angular okolju.

Key words documentation

Name and SURNAME: Rok SAMSA

Title of the final project paper: Fill my car: Developing a web application for sharing drive with unknown passengers

Place: Koper

Year: 2021

Number of pages: 67 Number of figures: 21 Number of tables: 4

Number of appendix: 3 Number of appendix pages: 6

Number of references: 34

Mentor: Assoc. Prof. Matjaž Kljun, PhD

Co-Mentor: Assoc. Prof. Klen Čopič Pucihar, PhD

Keywords: web application, Angular, sharing own vehicle, travel, sharing drive with unknown passengers

Abstract:

In this thesis are presented development steps of the online Angular application »Fill my car«, through which users will be able to share their vehicles with unknown passengers. The goal of the application is to improve and upgrade an already existing web application »Prevoz.org« - especially the choice of transport and the choice of passengers. With the existing application "Prevoz.org", a provider who wants to share his transport can only add it to the trips list, including his phone number. Users searching for transportation can only call the providers. In the end, both sides have many unknown telephone numbers, as not everyone who calls the provider also chooses his transport and everyone who seeks transport does not call only one provider. We solved this problem by introducing a »check in« system to the route itself, making it easier to use the application. In the thesis is presented a complete description of the development steps on the Angular platform.

ZAHVALA

Po vseh poskusih sem končno na cilju. Hvala vsem za podporo, spodbudo in razumevanje na tej dolgi poti.

Še posebej bi se rad iskreno zahvalil obema mentorjema doc. dr. Matjažu Kljunu in doc. dr. Klenu Čopiču Puciharju za mentorstvo, strokovno pomoč in svetovanje, ki sta mi jih nudila pri izdelavi zaključne naloge.

Zahvalil bi se tudi ostalim pedagoškim delavcem na UP FAMNIT, ki so mi nudili kakovostno izobraževanje.

Na koncu bi se rad zahvalil tudi svoji mami in sestri, ki sta mi vsa leta študija stali ob strani in verjeli vame, ter punci za vso pomoč in podporo.

Za Tato.

Hvala vsem.

KAZALO VSEBINE

KAZALO TABEL

KAZALO SLIK IN GRAFIKONOV

KAZALO PRILOG

SEZNAM KRATIC

Angular	TypeScript based open source web application framework (Odprtokodno okolje spletne za aplikacije, ki temelji na TypeScript skriptnem jeziku)
UML	Unified Modeling Language (Poenoteni jezik modeliranja)
GIT	Distributed version control system (Sistem za nadzor revizij)
HTML	HyperText Markup Language (Označevalni jezik za oblikovanje večpredstavnostnih vsebin)
CSS	Cascading Style Sheets (Kaskadne stilske podloge)
SASS	Syntactically Awesome Style Sheets (Sintaksično lepe stilske podloge)
SCSS	Sassy Cascading Style Sheets (Drzne kaskadne stilske podloge)
JavaScript	Object script programming language (Objektni skriptni programski jezik)
TypeScript	Open source object script programming language which compiles to JavaScript (Odprtokodni objektni skriptni programski jezik, ki se prevede v JavaScript)
PostgreSQL	Type of database (Vrsta podatkovne baze)
Node.js	Open source, cross platform runtime environment for developing server-side and networking applications (Odprtokodna izvajalna knjižnica za razvoj strežniških aplikacij)
Express.js	Web application framework for Node.js, designed for building web applications and APIs (Odprtokodna knjižnica za Node.js, zasnovana za izdelavo spletnih aplikacij in API-jev)
Sequelize.js	Object/relational mapper, which provides easy access to PostgreSQL database by mapping database entries to objects and vice versa (Objektno relacijski preslikator, ki omogoča enostaven dostop do baze podatkov PostgreSQL s preslikavo vnosov v bazo podatkov na objekte in obratno)
I/O	Input/output processes (Vhodno/izhodni procesi)
NoSQL	Non relational database (Nerelacijska podatkovna baza)
REST	Representational State Transfer (Reprezentativen prenos stanja)
API	Application Programming Interface (Aplikacijski programski vmesnik)
CORS	Cross-Origin Resource Sharing (Skupna raba virov navzkrižnega porekla)
JSON	JavaScript Object Notation (Objektna notacija za JavaScript)
NPM	Node Package Manager (Node upravitelj paketov)

1 UVOD

Živimo v času, ko se z večjo osredotočenostjo usmerjamo v zmanjševanje emisij motorjev na notranje izgorevanje. Eno od področij, ki lahko pripomore k temu, je zagotovo deljenje lastnega vozila z ljudmi, ki želijo potovati v isto smer.

1.1 Opis izhodiščnega problema

V Sloveniji že imamo spletno storitev Prevoz.org[1], preko katere lahko delimo svoje vozilo z drugimi potniki na poti do našega končnega cilja. S polnjenjem prostih mest v našem vozilu ravnamo okolju prijazno, si popestrimo vožnjo in delimo stroške goriva. Hkrati tudi ostali potniki na tovrsten način dobijo cenovno ugoden in časovno prilagodljiv prevoz do želenega cilja ter si lahko razširijo mrežo poznanstev. Tako voznik kot potniki pripomorejo k zmanjševanju prometa na cestah.

Tovrsten način potovanja postaja iz leta v leto bolj priljubljen, kar se med drugim kaže v vedno večjem obisku spletne strani Prevoz.org (več kot 2 milijona ogledov strani na mesec in več deset tisoč uporabnikov na dan) [14].

Spletna storitev Prevoz.org omogoča uporabnikom objavo poti, ki bi jo radi delili z drugimi. Komunikacija med voznikom in ostalimi sopotniki je mogoča le s klicem ali sistemom kratkih sporočil (SMS)[2] preko mobilnega telefona. V tem delu celotnega procesa deljenja vozila lahko komunikacijo in prijavo/odjavo na skupno vožnjo močno izboljšamo in s tem olajšamo uporabo aplikacije. Obstoječa aplikacija se v trenutni obliki tudi srečuje s problemom varnosti, saj ne beleži poti in potnikov, hkrati pa ne omogoča uporabnikom, da označijo neprimerne voznike in potnike.

1.2 Namen in cilji naloge

V okviru zaključne naloge bomo natančneje preučili obstoječe storitve za deljenje vozila in zasnovali ter izdelali spletno aplikacijo, pri kateri se bomo osredotočili na izboljšanje uporabniške izkušnje predvsem iz vidika komunikacije med voznikom in morebitnimi sopotniki ter zagotavljanju čim večje varnosti tovrstne vožnje. Ena od idej zajema avtomatizacijo postopkov prijave/odjave na skupno vožnjo in preproste komunikacije med voznikom in potniki. Druga se navezuje na povečevanje varnosti potnikov in voznikov preko mehanizmov ocenjevanja kakovosti potovanja, v katero bodo vključeni tako vozniki kot tudi

[1] https://prevoz.org/

[2] https://sl.wikipedia.org/wiki/Sistem_kratkih_sporo%C4%8Dil

potniki. Funkcionalnost ocenjevanja bomo pustili za drugo fazo razvoja aplikacije in se tako osredotočili na glavno funkcionalnost prijave/odjave.

1.3 Struktura zaključne naloge

V nadaljevanju zaključne naloge bomo definirano problematiko skušali rešiti z razvojem spletne aplikacije. Celotni proces bomo začeli z definicijo problema in preverili obstoječe rešitve ter kako jih lahko izboljšamo. V nadaljevanju bomo v poglavju Analize in definicije zahtev predstavili teoretično zasnovo projekta. Praktičen del izvedbe je opisan v poglavjih Načrtovanje in Izvedba. Zadnje poglavje je Testiranje, kjer je predstavljeno reševanje nastalih težav pri testiranju. V zaključek smo podali svoje mnenje o morebitnem nadaljnjem razvoju in vzdrževanju aplikacije.

2 PREDHODNO DELO

V tem poglavju bomo najprej opisali pomen deljenja prevozov v Sloveniji in drugje, pregledali dve rešitvi, nekaj raziskav in povzeli težave portala Prevoz.org.

2.1 Pomen deljenja prevozov

Z deljenjem praznih sedežev v vozilu imajo potniki na razpolago dodatno obliko prevozov, ki dopolnjuje druge oblike, kot so na primer javni prevozi. Deljenje vozila na skupnih poteh lahko nudi tudi časovno fleksibilen in udoben prevoz ter ponuja še eno ključno korist. Z združevanjem ljudi v vozilih občutno pripomoremo k zmanjšanju prometa na cestah in posledično zmanjšanju količine izpušnih plinov. Po oceni podjetja BlaBlaCar [17], ki ponuja svojim uporabnikom deljenje prevozov, se samo z njihovo spletno storitvijo izognemo 1,6 milijona CO_2 izpustov letno.

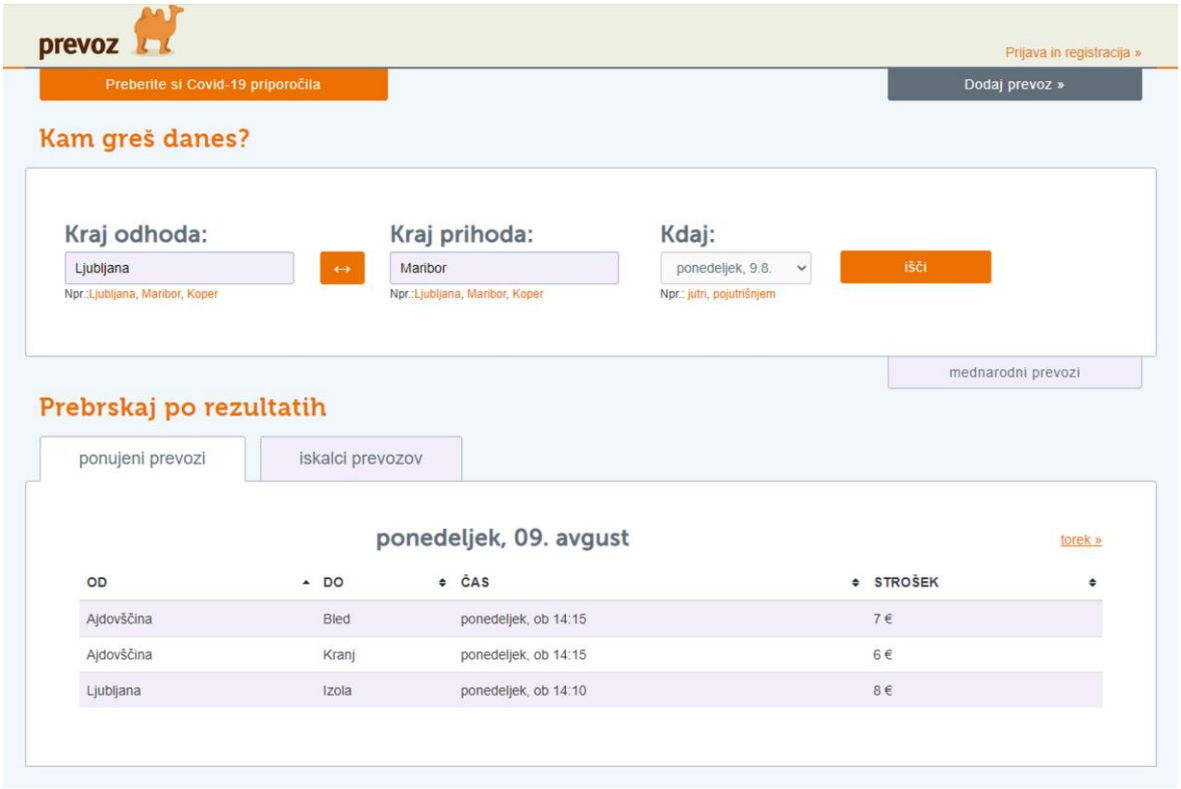
Med večje uporabnike teh storitev sodijo sodelavci, ki si delijo skupno pot v službo in tako privarčujejo nekaj denarja pri potnih stroških. Druga skupina so študenti, ki potrebujejo prevoz iz univerzitetnih mest v domače kraje in obratno. Ostanejo še manjše skupine, kot so posamezniki, ki morajo opraviti določene opravke v oddaljenih krajih. Med predloge za povečanje števila uporabnikov tovrstnih storitev sodi še ureditev parkirišč ob avtocestnih priključkih in vzpostavitev popustov pri plačilu parkirnin ter plačilu cestnin [11].

2.2 Obstoječe rešitve

V tem poglavju bomo najprej predstavili delovanje storitve Prevoz.org[3] in si pogledali nekaj raziskav o njej. Nato bomo predstavili delovanje najbolj znane svetovne platforme BlaBlaCar[4].

2.2.1 Prevoz.org

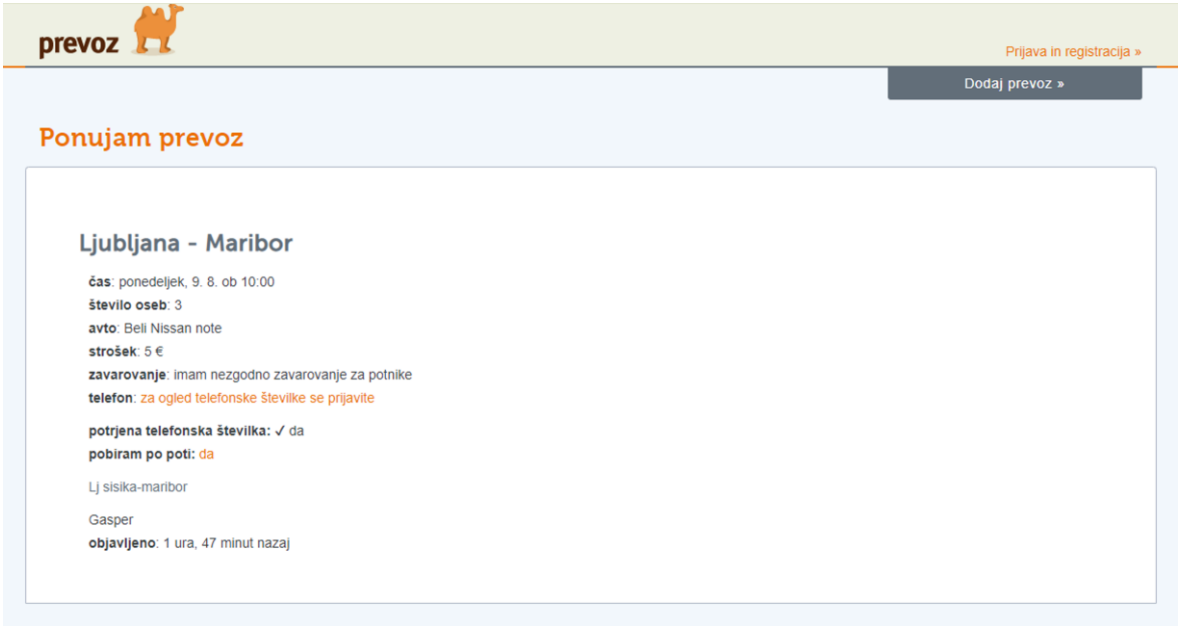
Spletna storitev Prevoz.org omogoča dodajanje in iskanje prevozov. Oseba, ki dodaja prevoz, se registrira in izpolni različna vnosna polja pri vnosu poti: začetno in končno točko potovanja, datum in čas odhoda, število oseb, ki se lahko prijavijo na pot, strošek poti na osebo, model avtomobila in svojo telefonsko številko. Iskalec prevoza pa na začetni strani spletnega mesta (Slika 1) vnese kraj odhoda in prihoda ter datum. Na izpisanem seznam rezultatov iskanja si uporabnik izbere ustrezen prevoz (Slika 2) in kliče oziroma piše SMS vozniku.



Slika 1: Začetna stran spletnega mesta Prevoz.org z možnostmi iskanja prevozov

[3] https://prevoz.org/

[4] https://www.blablacar.com/



Slika 2: Stran izbranega prevoza s podatki

2.2.2 Uporabniška izkušnja spletnega mesta Prevoz.org

Pri pregledu obstoječe literature smo našli nekaj raziskav o Prevozu.org. Ena od teh se je osredotočila na uporabniško izkušnjo uporabnikov storitve [13].

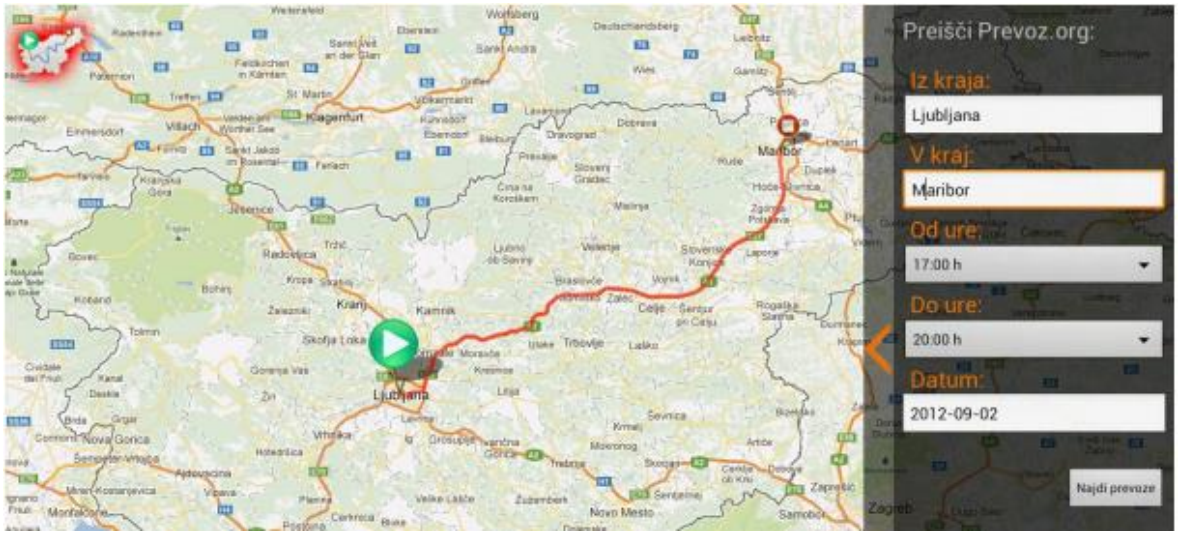
V tem delu avtor ugotavlja, da na temo uporabniške izkušnje spletnega mesta Prevoz.org še ni bilo raziskav in zato analizira odgovore uporabnikov spletne aplikacije preko ankete, v kateri ugotovi, da je spletna storitev uporabna, vsebuje koristne informacije, uporabniki zaupajo upravljavcem, so zadovoljni s spletnim mestom kot celoto in z uporabniško izkušnjo spletnega mesta. Anketiranci so tudi izpostavili grafično obliko spletnega mesta, pri kateri so ocenili, da spletna aplikacija oblikovno ni najbolj sodobna in privlačna. Aplikacija tudi ne nudi dovolj podrobnih informacij, ki bi dodatno pripomogle pri celotnem procesu vnosa in iskanja poti ter uporabnikom ne daje občutka pripadnosti skupnosti. Anketirance pa je najbolj zmotilo slabo informiranje in obveščanje upravljavcev spletnega mesta o spremembah na spletnem mestu in o aktualnih dogodkih in aktivnostih [13].

Avtor predlaga upravljavcem spletnega mesta izboljšave pri uporabniški izkušnji, grafični podobi in predvsem pri komunikaciji upravljavcev z uporabniki.

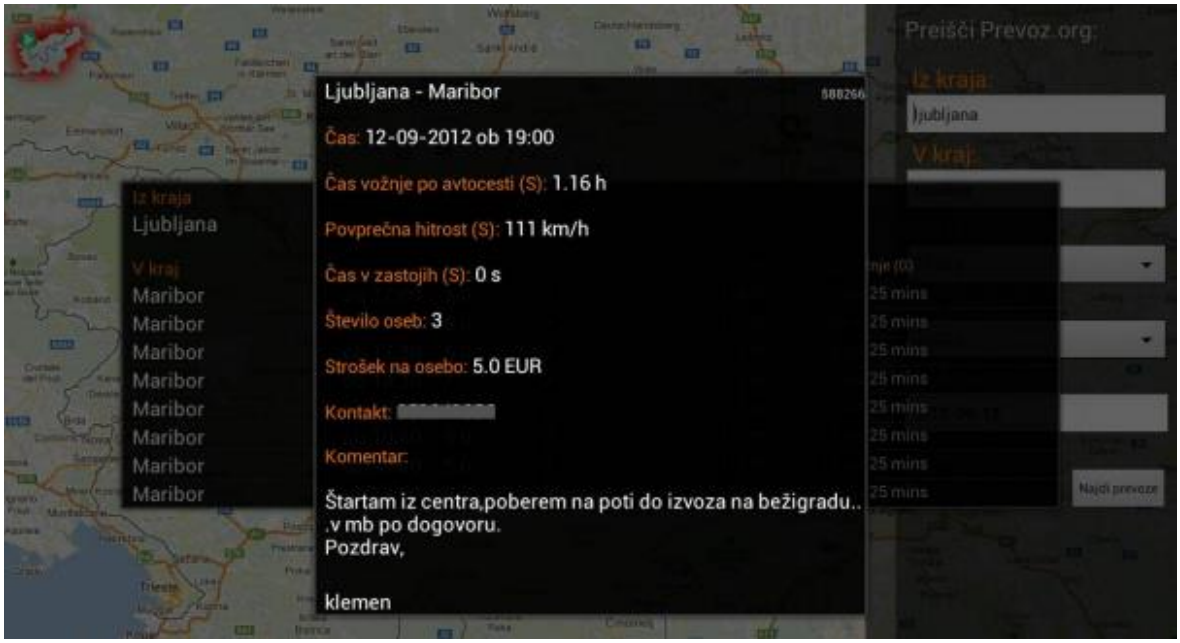
2.2.3 Obstoječa izboljšava spletne aplikacije za vozače

V diplomskem delu z naslovom “Spletna aplikacija za vozače” avtor poskuša izboljšati spletno storitev Prevoz.org z dodanim zemljevidom, na katerem uporabnik vidi začetno

točko, končno točko in celotno pot. Na zemljevidu si lahko ogleda tudi posebnosti na cestah v Sloveniji in izris trenutne lokacije uporabnika, kot je vidno na Sliki 3 [12].



Slika 3: Izrisana pot z začetno in končno točko v aplikaciji za vozače [12]



Slika 4: Okno izbranega prevoza s podatki v aplikaciji za vozače [12]

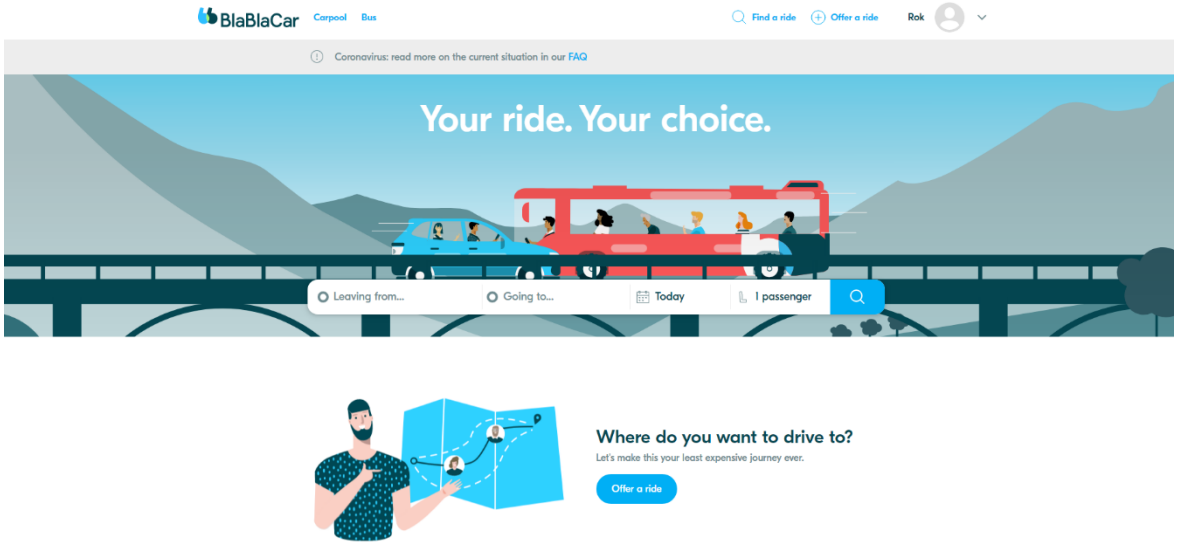
Interakcija uporabnika s spletnim mestom poteka v treh korakih: uporabnik najprej na desni strani aplikacije išče po vseh prevozih od začetne do končne točke z izbranim datumom in uro (na Sliki 4 med Ljubljano in Mariborom). Če se vsaj en prevoz ujema z vnesenimi podatki, se pot iz začetne točke do končne točke izriše na zemljevidu. Uporabnik nato klikne na zeleni gumb z ikono “Predvajaj” s čimer se mu odpre novo okno, kjer ima na voljo seznam vseh ustreznih prevozov. Po pregledu le-teh si izbere prevoz, ki ima začetno točko najbližje željeni in ima najugodnejšo ceno.

Ob pritisku na opcijo “Posebnosti na cestah”, storitev preko RSS protokola[5] dobi podatke o trenutnem stanju na cestah iz spletne storitve Promet.si[6]. Opcija “Prikaz trenutne lokacije” pa nam izriše našo trenutno lokacijo na zemljevidu.

Rešitev avtorja le deloma izboljša spletno mesto Prevoz.org z vizualnim izrisom poti in z opisanima dodatnima funkcijama: “Posebnosti na cestah” in “Prikaz trenutne lokacije”. Ne izboljša pa prijave in varnosti.

2.2.4 BlaBlaCar

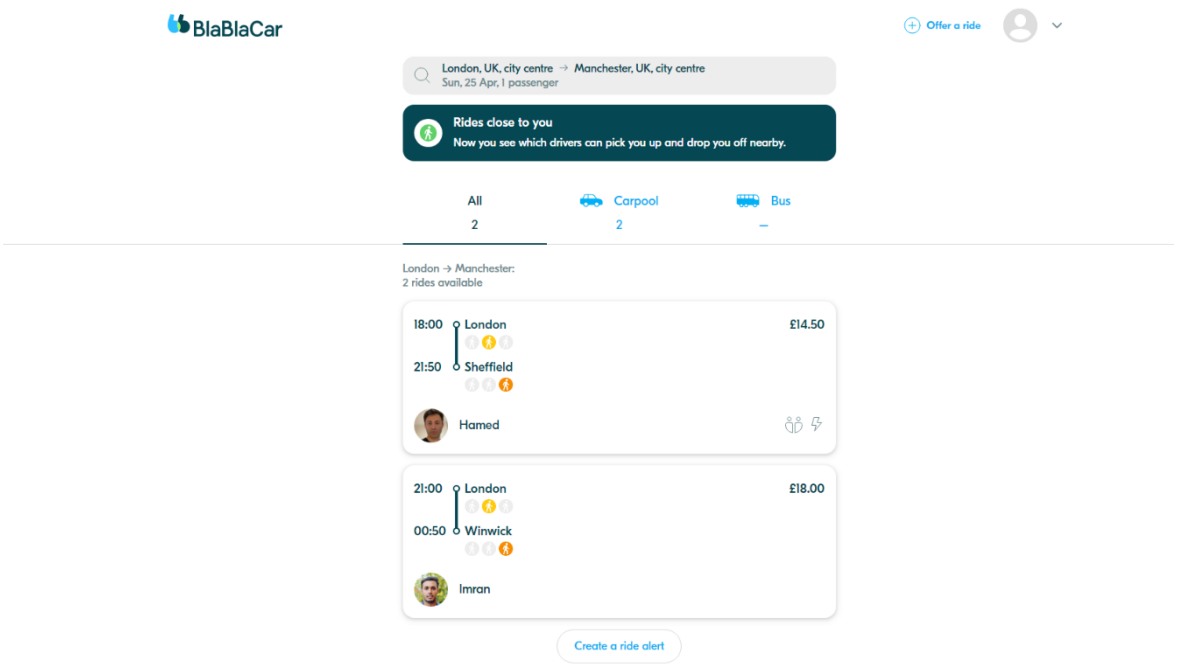
Poleg slovenske storitve smo si ogledali še storitev BlaBlaCar, ki deluje na širšem področju 22 držav sveta z 90 milijoni uporabnikov [15]. Grafični izgled spletne storitve je viden na Sliki 2. Kljub razširjenosti storitev še ni na voljo na slovenskem trgu [16].



Slika 5: Začetna stran spletne storitve BlaBlaCar[7]

Podoben koncept kot storitev Prevoz.org uporablja tudi tuja spletna storitev BlaBlaCar, s to razliko, da je precej bolj dodelana (Slika 5). Uporabnik lahko pri prijavi na prevoz kontaktira voznika še preden se prijavi na potovanje in kasneje opravi prijavo s plačilom preko PayPal storitve ali s plačilno kartico. Ponudnik prevoza ima na voljo enaka vnosna polja kot pri Prevozu.org z dodatnimi možnostmi natančne izbire točke odhoda in prihoda, z izbiro poti med točkama in z izbiro vmesnih postaj za sprejem novih sopotnikov. Primer najdenih ponujenih prevozov storitve je viden na Sliki 6.

[5] https://en.wikipedia.org/wiki/RSS
[6] https://promet.si/portal/sl/razmere.aspx
[7] https://www.blablacar.co.uk/



Slika 6: Stran z rezultati iskanja prevozov spletne storitve BlaBlaCar[8]

2.3 Težave obstoječe rešitve Prevoza.org in predlagane izboljšave

Pri pregledu interakcije s storitvama Prevoz.org in BlaBlaCar, raziskav o storitvi Prevoz.org in opisanega pregleda smo ugotovili, da spletna storitev Prevoz.org ne omogoča uporabnikom prijave na pot preko spletnega mesta oziroma preko mobilne aplikacije in s tem beleženja rezervacij. Uporabniki lahko komunicirajo le preko telefonskega klica ali SMS sporočila, kar lahko privede do zmede, ko se ponudnik prevoza znajde z veliko količini neznanih kličočih številk in SMS sporočil. Tudi iskalci prevozov lahko kličejo veliko število ponudnikov, preden se odločijo za primernega ali dobijo primernega. Storitev Prevoz.org hkrati ne omogoča uporabnikom, da voznike in potnike ocenijo. Tako ne morejo pomagati drugim uporabnikom bolje izbrati prevoz ali sopotnike.

Z razvojem naše spletne aplikacije »Zapolni moj avto« želimo omenjene težave odpraviti in hkrati nadgraditi storitev Prevoz.org z naslednjimi izboljšavami:

- Tako ponudnikom vožnje kot vsem potnikom želimo olajšati prijavo in odjavo na določeno pot. Izhodišče problema je omejenost komunikacije preko SMS sporočil in telefonskih klicev, kar privede do veliko neznanih telefonskih številk in težji pregled pomembnih informacij.

- Potnikom želimo ponuditi točkovalni sistem za voznike, s katerim bi ocenjevali voznike na podlagi načina vožnje, komunikacije, časovne točnosti in udobja v vozilu.

[8] https://www.blablacar.co.uk/

Potnikom bi s tem olajšali izbiro prevoza. To točko postavimo v načrt izdelave, vendar je predvidena implementacija v drugi fazi razvoja.

- Z implementacijo modernejše grafične podobe želimo uporabnikom olajšati razumevanje vseh podatkov v vseh korakih dodajanja poti ali prijave na pot.

3 ŠTUDIJA IZVEDLJIVOSTI

Za izvedljivost samega projekta moramo upoštevati več dejavnikov. Ker bo aplikacijo na začetku razvijal le en razvijalec, lahko pričakujemo dolg razvoj. A to ne predstavlja težavo, saj nimamo časovnih omejitev. Kasnejši cilj je, da bi k projektu pritegnili večje število ljudi, ki bi vanj prispevali.

Za projekt bomo potrebovali denarna sredstva. Ker je to le projekt za zaključno nalogo, na začetku ne potrebujemo velike investicije. Potrebovali bomo spletno gostovanje naše aplikacije in tudi domeno, preko katere nas bodo uporabniki našli. Strežnik že uporabljamo za druge projekte, tako da tu ne bo dodatnega stroška. Ostane le še domena, ki jo bomo letno obnavljali. V fazah, ko bo aplikacija že dobro testirana in uporabljena, se bomo morda odločili tudi za njeno oglaševanje. Pri izdelavi projekta bomo uporabili izključno odprtokodno in/ali brezplačno programsko opremo.

Z razvojem naše spletne aplikacije ne kršimo nobenih zakonov, prav tako naša aplikacija ni moralno sporna.

4 ANALIZA IN DEFINICIJA ZAHTEV

Pred samim začetkom načrtovanja in izdelave naše aplikacije je zelo pomembno, da že v začetnih fazah preučimo vse možne okoliščine in ovire ter se jim skušamo izogniti. Ta del običajno izvedeta naročnik in izvajalec projekta (v našem primeru je to ista oseba) s čimer definirata vse uporabnikove zahteve in pričakovanja. Za predstavo le-teh bomo uporabili UML diagrame [1].

UML ("Unified Modeling Language") ali poenoteni jezik modeliranja je splošni jezik vizualnega modeliranja, ki se uporablja za določanje, vizualizacijo, konstrukcijo in dokumentiranje vseh elementov programskega sistema. Zajema odločitve in razumevanje sistemov, ki jih je potrebno zgraditi. Uporablja se za razumevanje, načrtovanje, raziskovanje, konfiguriranje, vzdrževanje in nadzor informacij o takih sistemih [1].

V UML jeziku diagrame delimo glede na njihov namen v dve skupini [2]:

- strukturni (razredni, komponentni, objektni in paketni diagram) in
- vedenjski (aktivnostni, sekvenčni, komunikacijski, časovni diagram ter diagram stanj in diagram primera uporabe).

V naslednjih podpoglavjih bomo predstavili in obdelali sistemske, funkcijske in nefunkcijske zahteve.

4.1 Sistemske zahteve

Za razvoj aplikacije bomo potrebovali primerno razvojno okolje, ki nam bo omogočilo hiter razvoj. Po temeljitem pregledu trenutnih spletnih tehnologij za razvoj strežniškega dela storitve, podatkovnih baz in razvoj uporabniških vmesnikov, smo se v fazi "načrtovanja" odločili in izbrali primerno razvojno okolje.

Izbira strežniške tehnologije
Na izbiro strežniške tehnologije vplivajo različni dejavniki:

- poznavanje tehnologije in jezika,
- omogočanje izmenjave ogromne količine podatkov v realnem času,
- razvitost in količina knjižnic, specializiranih za avtorizacijo, komunikacijo z bazami podatkov in drugih knjižnic, ki pomagajo k optimizaciji dela,

- tehnologija, v kateri se podatki in funkcionalnosti obnašajo kot viri ter se večina storitev izvaja z osnovnimi operacijami za obdelavo podatkov – naredi, beri, popravi in briši.

Izbira podatkovne baze
Kot pri izbiri prave strežniške tehnologije tudi tu želimo izbrati kar se da najboljšo podatkovno bazo za naš projekt. Baza mora biti hitra pri branju in pisanju, prilagodljiva v smislu, da hitro kakšen segment baze spremenimo in ponujati nam mora bogat nabor funkcij.

Izbira platforme za hiter razvoj uporabniškega vmesnika
Za hitro, učinkovito in trajnostno aplikacijo, moramo temeljito razmisliti, katero okolje za izdelavo spletnih aplikacij bomo izbrali. Na izbiro imamo kar nekaj priljubljenih spletnih okolij, kot so Googlov Angular, Facebookov React in novejši Vue.js, ki ustrezajo zgoraj omenjenim zahtevam oziroma dejavnikom.

4.2 Funkcijske zahteve

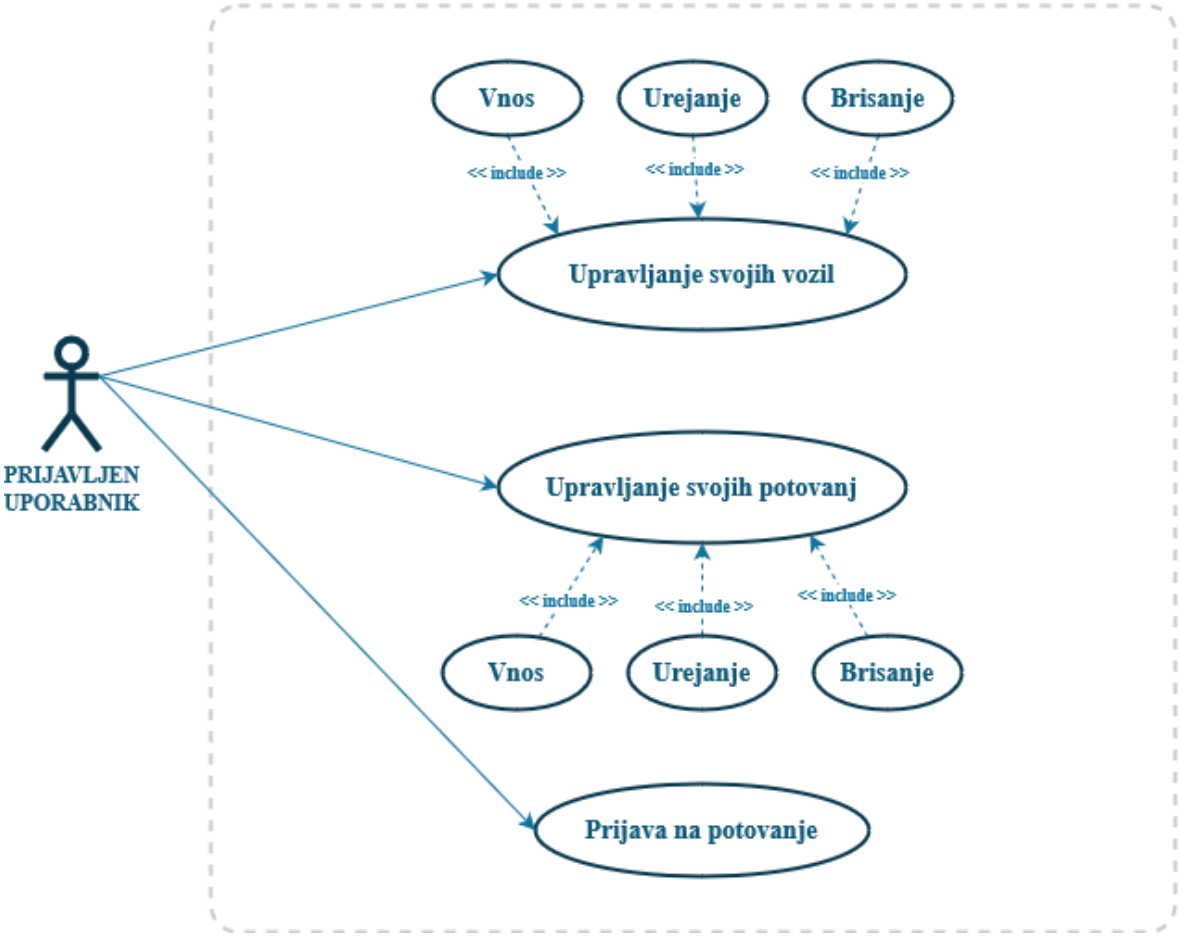
Funkcijske zahteve so pomembna kategorija dejanskih zahtev. Opisujejo, kaj mora sistem ali programska oprema omogočati. Včasih jih imenujemo vedenjske ali operativne zahteve, ker določajo vhodne procese v sistem, izhodne procese iz sistema in vedenjske odnose med njimi [3].

Pri začetni zasnovi smo se odločili za osnovne funkcionalne zahteve, ki predstavljajo razširitev funkcionalnosti trenutne storitve Prevoz.org:

- registracija v sistem,
- prijava v sistem,
- dodajanje vozila,
- dodajanje prevoza (vožnje),
- iskanje prevozov,
- izbira/prijava na prevoz,
- odjava od prevoza,
- ocenjevanje prevoza in komentar (druga faza izdelave projekta),
- ocenjevanje sopotnika in komentar (druga faza izdelave projekta) in
- ocenjevanje voznika in komentar (druga faza izdelave projekta).

Nekaj teh zahtev bomo prikazali skozi različne UML diagrame v naslednjih podrazdelkih.

4.2.1 Diagram primera uporabe



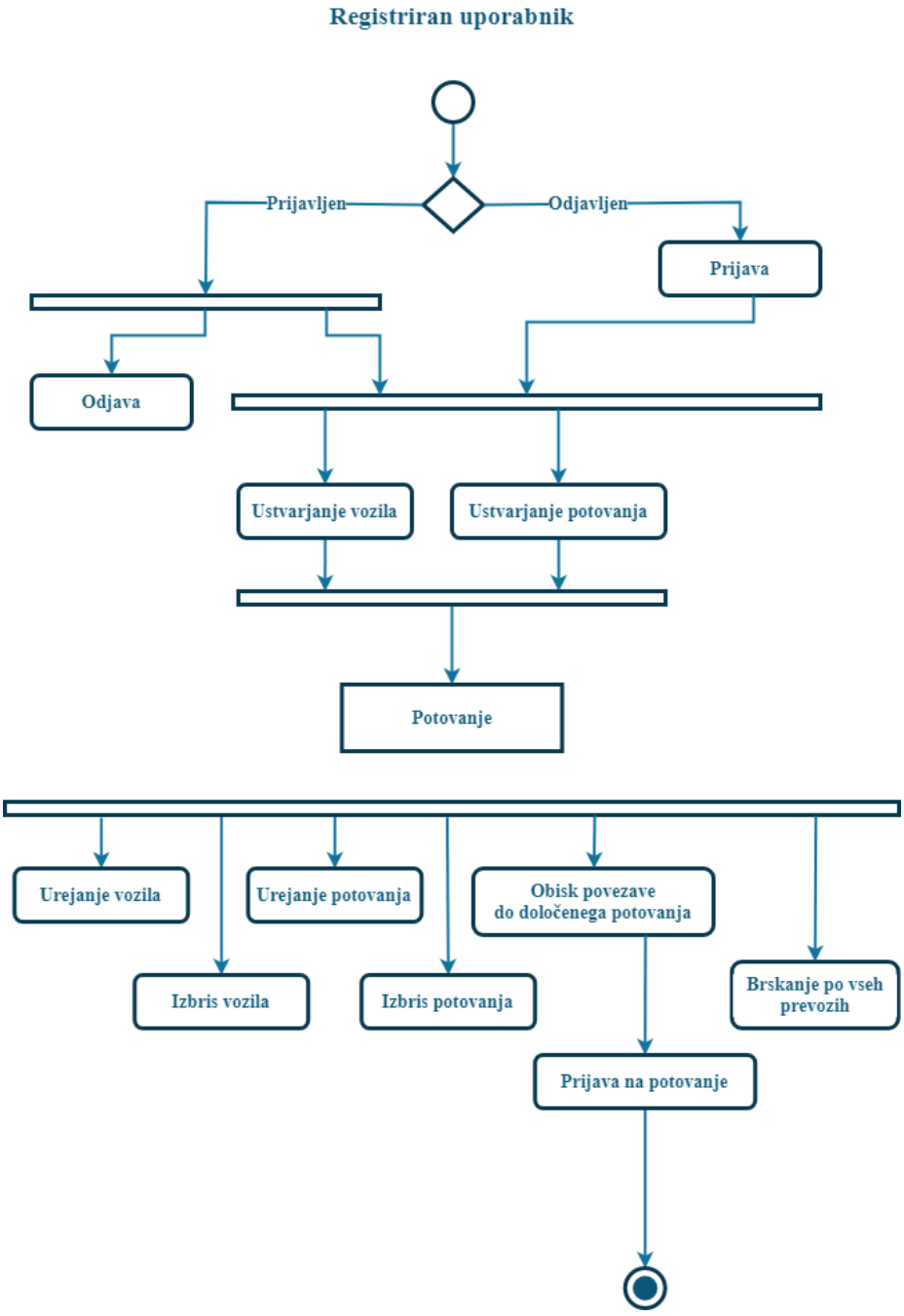
Slika 7: Diagram primera uporabe za nudenje prevoza in prijave na prevoz

Na Sliki 7 diagrama primera uporabe za nudenje prevoza in prijave na prevoz vidimo predstavo interakcije uporabnika s sistemom. Prijavljen uporabnik lahko prijavi na potovanje in ustvari svoje lastno potovanje z možnostjo dodajanja vozila in poti. Kot je razvidno iz diagrama je ustvarjanje poti neodvisno od ustvarjenja vozila. Seveda potovanje brez vozila nima smisla, tako za ustvarjalca potovanja, kot za potnike.

Opis poteka:

Prijavljen uporabnik ima na začetku možnost dodajanja vozila in kasneje poti. Brez ustvarjenega vozila uporabnik lahko ustvari potovanje in tako lahko kasneje doda vozilo. Seveda potovanje brez vozila nima smisla, tako za ustvarjalca potovanja kot za potnike. Na drugi strani pa potniki obiščejo generirano povezavo določenega potovanja in imajo možnost prijave. Potnika, ki se prijavlja na potovanje, vodijo enostavni koraki.

4.2.2 Aktivnostni diagram za registriranega uporabnika



Slika 8: Diagram aktivnosti za registriranega uporabnika

Aktivnostni diagram (Slika 8) nazorno prikazuje potek aktivnosti delovanja aplikacije. Koraki si sledijo:

1. Če je registrirani uporabnik neprijavljen, se lahko prijavi. Prijavljen uporabnik pa se lahko odjavi oz. uporablja aplikacijo.

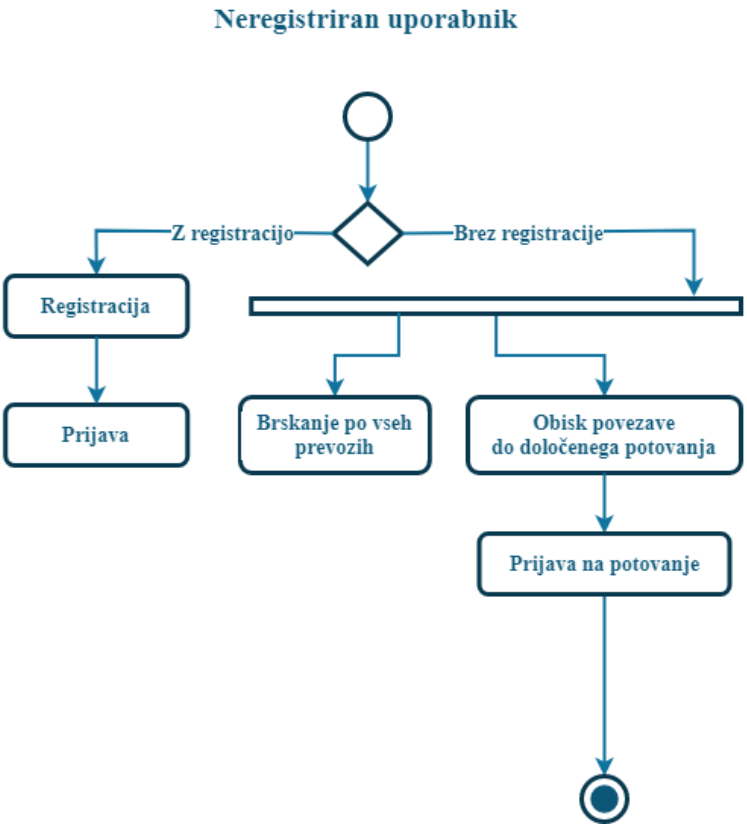
2. Prijavljen uporabnik lahko dodaja vozila in potovanja.

3. Do ustvarjenega potovanja lahko dostopajo tako prijavljeni kot neprijavljeni uporabniki.

4. Uporabnik lahko potovanje ureja, ga izbriše ali odstrani vozilo.

5. Uporabnik ima možnost iskanja drugih prevozov in tudi prijave na le-te.

4.2.3 Aktivnostni diagram za neregistriranega uporabnika



Slika 9: Diagram aktivnosti za neregistriranega uporabnika

Aktivnostni diagram (Slika 9) prikazuje potek aktivnosti za neregistriranega uporabnika. Koraki si sledijo:

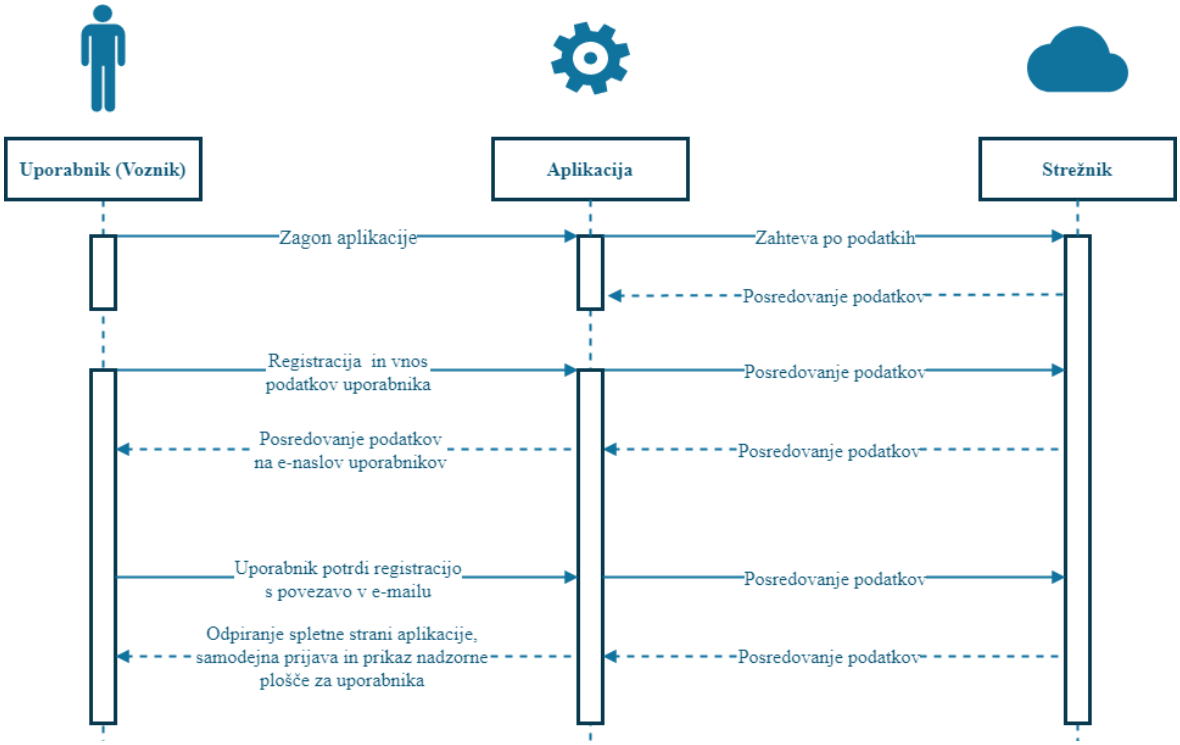
1. Če obiskovalec še nima uporabniškega profila, se lahko registrira. Po potrditvi e-naslova v elektronskem sporočilu aplikacija uporabnika samodejno prijavi.

2. Če pa se uporabnik ne želi registrirati, lahko še vedno deloma uporablja aplikacijo.

3. Uporabnik ima dostop do pregleda vseh prevozov/potovanj.

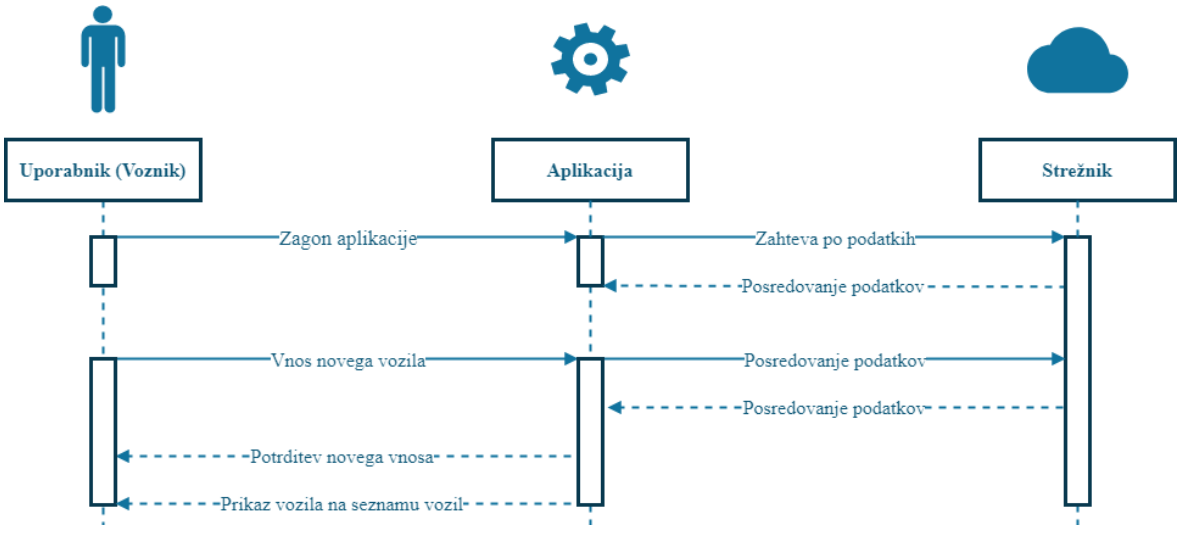
4. Lahko izbere določeno potovanje iz seznama in se nanj tudi prijavi.

4.2.4 Sekvenčni diagram



Slika 10: Sekvenčni diagram prikazuje registracijo in nato prijavo uporabnika

Uporabnik najprej zažene aplikacijo preko vnosa spletnega naslova aplikacije. Uporabnik ima na voljo registracijo in prijavo. Pred prvo prijavo se mora uporabnik registrirati. Po vnosu vseh zahtevanih podatkov in kliku na gumb “registracija”, se podatki uporabnika pošljejo strežniku. Po obdelavi podatkov uporabnik dobi samodejno elektronsko sporočilo, v katerem s klikom na potrditveni gumb zaključi proces registracije. Uporabnika nato aplikacija samodejno prijavi in ga preusmeri na nadzorno ploščo. Potek je viden na Sliki 10.



Slika 11: Sekvenčni diagram prikazuje vnos vozila

Na Sliki 11 vidimo korake med objekti pri vnosu vozila. Uporabnik ima na izbiro več možnosti, med katerimi sta tudi dodajanje vozil in potovanj. Pri obeh možnostih imamo enake korake. Pri vnosu vozila ali potovanja podatke pošljemo do strežnika. Strežnik se odzove s pozitivnim oziroma negativnim odgovorom in uporabniku to obvestilo prikažemo. Po uspešnem vnosu se dodano vozilo prikaže na seznamu.

4.3 Nefunkcijske zahteve

Nefunkcijske zahteve določajo lastnosti sistema in izvajanja funkcij. V sistemskem inženirstvu jih pogosto označujemo kot "nepravilnosti" [3].

Glede na karakteristike lahko nefunkcijske zahteve razdelimo v dve skupini [3]:

- obratovalne lastnosti (varnost, uporabnost, zahtevana prepustnost sistema),
- evolucijske lastnosti (možnost testiranja, možnost vzdrževanja, razširljivost, nadgradljivost).

4.3.1 Obratovalne lastnosti

Uporabnost
Vodilo razvoja aplikacije je visoka stopnja uporabnosti (angl. *usability*). Uporabnost je merilo, kako dobro lahko določen uporabnik v določenem kontekstu z izdelkom/zasnovo učinkovito, uspešno in zadovoljivo doseže določen cilj. Oblikovalci običajno merijo uporabnost zasnove v celotnem razvojnem procesu – od prototipov do končnega izdelka, da zagotovijo karseda visoko stopnjo uporabnosti [10]. Tudi pri razvoju naše aplikacije smo se odločili izvajati teste uporabnosti, a v manjšem obsegu in le z razvijalcem. Izvajanje uporabnosti presega okvirje te zaključne in je del nadaljnjih nalog.

Varnost
Živimo v časih, ko sta varnost in zasebnost zelo pomembni. Kraja osebnih podatkov s strani zlonamernih vdiralcev v sisteme [19] je dandanes stvarnost [20]. Tako je potrebno že med zasnovo in načrtovanjem projekta skrbno zasnovati varnost aplikacije. Pomisliti o varnosti sredi ali na koncu projekta lahko predstavlja prevelik finančni in časovni zalogaj, saj se lahko zgodi, da je potrebno na novo zasnovati aplikacijo in jo na novo tudi implementirati [18]. V našem primeru smo varnosti posvetili pozornost pri vseh korakih implementacije in sproti reševali težave med testiranji.

Zahtevana prepustnost sistema
Našo aplikacijo bo sočasno uporabljalo veliko ljudi preko različnih naprav (prenosni računalniki, namizni računalniki, mobilni telefoni, tablični računalniki …), zato moramo poskrbeti, da bo strežnik dovolj zmogljiv za hranjenje in procesiranje velike količine

podatkov. Da ne bi imeli prevelikih začetnih stroškov, bomo glede na potrebe kasneje opravili različne nadgradnje in optimizacijo strežnika.

4.3.2 Evolucijske lastnosti

Možnost testiranja

Testiranje aplikacije pride v ospredje po končani prvi različici, a vendar moramo tudi pred tem in v kasnejših korakih vedno imeti vse pripravljeno za sprotno testiranje. Testiramo tako funkcionalnosti aplikacije kot tudi različne interakcije z grafičnim vmesnikom iz vidika uporabniške izkušnje. Testiranju naše aplikacije se bomo posvetili v zaključkih te zaključne naloge.

Zmožnost vzdrževanja

Poskrbeti moramo, da bomo imeli po predaji aplikacije v uporabo na voljo ljudi, ki jo bodo vzdrževali. V našem primeru bo za urejanje in nadzorovanje zadolžen kar izdelovalec aplikacije. Z vzdrževanjem predvsem poskrbimo, da je aplikacija vseskozi posodobljena tako grafično kot sistemsko in da je tako v "stiku s časom". Potrebe uporabnikov, tehnologije, načini uporabe aplikacije in trg se namreč vseskozi spreminjajo.

Nadgradljivost

Sama struktura in zasnova modernih razvojnih okolij (kot so Angular, React ali pa Vue.js) je več kot primerna za vse nadgradnje. Okolja so zgrajena iz več komponent in tako lahko posodobimo le eno komponento, ne da bi s tem vplivali na druge komponente.

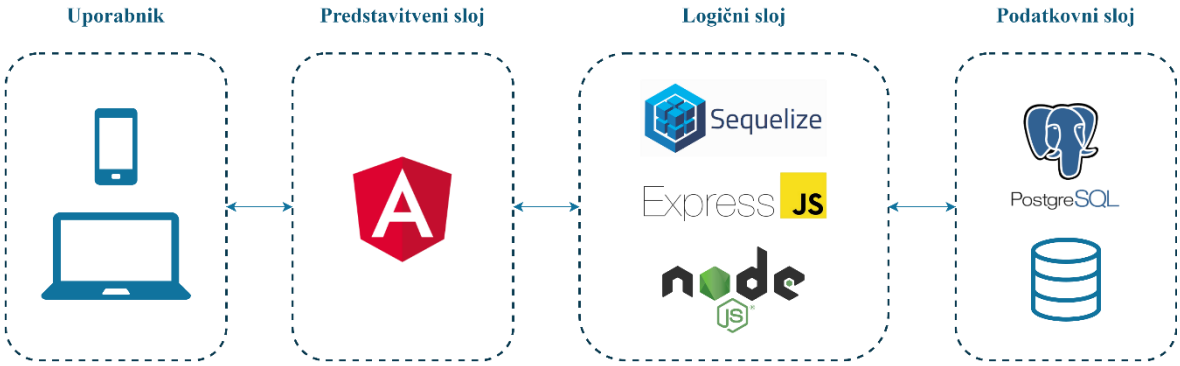
Razširljivost

Z našimi podatki o uporabnikih (katera vozila vozijo, katere poti so največkrat prevožene in ostale statistike) in z veliko bazo uporabnikov imamo vedno veliko možnosti za dodatne funkcionalnosti aplikacije. V prvih fazah razvoja se bomo osredotočili na ključne zahteve in funkcionalnosti, v naslednjih pa dodajali izboljšave. Kot je že zgoraj omenjeno, so moderna okolja ideala tudi za razširitve, saj lahko v aplikacijo le dodamo novo komponento. Uporabnikom lahko vse podatke kasneje prikažemo v različnih pogledih in s tem izboljšamo uporabniško izkušnjo. Naše poglede o prihodnosti bomo opisali v zaključku.

5 NAČRTOVANJE

Glede na izbrane funkcionalne in sistemske zahteve smo se lotili faze načrtovanja. Dobra priprava načrta za izdelavo aplikacije pomeni skoraj zagotovo manj težav pri sami izdelavi. Naredili bomo načrt izdelka, ki nam bo pomagal pri izvedbi, pogledali si bomo arhitekturo aplikacije, predstavili bomo vse uporabljene tehnologije, programsko opremo in knjižnice ter na koncu vse združili z načrtom podatkovne baze in zasnovo grafičnega vmesnika.

5.1 Pregled arhitekture aplikacije



Slika 12: Zasnova arhitekture spletne aplikacije

Izbrali smo si danes moderno, učinkovito in odzivno arhitekturo za izdelavo naše aplikacije, kot je vidno na Sliki 12. Sestavljajo jo trije sloji in na koncu uporabnik, ki preko interakcije sprejema podatke iz vseh treh slojev. Na predstavitvenem sloju bomo uporabili klasične tehnologije, kot so HTML, CSS in Typescript združene v Angular okolju. Logični sloj smo definirali s tremi različnimi knjižnicami Node.js, Express.js in Sequelize.js, ki pridobivajo podatke iz podatkovnega sloja. Za podatkovni sloj pa bomo uporabili brezplačno in odprtokodno PostgreSQL podatkovno bazo.

5.2 Izbrana in uporabljena programska oprema in tehnologije

Z odprtokodno in brezplačno uporabljeno programsko opremo ohranimo začetne nizke stroške, kar ne pomeni, da projekta ni mogoče kakovostno izvesti. Prej omenjene tehnologije uporabljajo tudi največja podjetja iz sveta informacijske tehnologije [21][22]. Po pregledu tehnologij, smo se odločali predvsem na podlagi izkušenj in poznavanja posamezne tehnologije oziroma okolja. V naslednjih podrazdelkih jih bomo na kratko predstavili.

5.2.1 Adobe XD

Adobe XD [24] je celovita rešitev za oblikovanje uporabniške izkušnje za mobilne aplikacije, spletna mesta ali aplikacije in ostale uporabniške vmesnike. Z istim orodjem

lahko oblikujemo, ustvarjamo prototipe in jih delimo z drugimi uporabniki. Z ustvarjenimi prototipi lahko tudi drugi deležniki dobijo občutek, kako bo aplikacija izgledala [4].

Program smo izbrali, ker je enostaven za uporabo, ima možnost delitve izdelkov z ostalimi člani razvojne ekipe, in ker je njegova uporaba trenutno brezplačna [23]. Zasnovan uporabniški vmesnik v tej fazi omogoča testiranje in hitrejši kasnejši razvoj.

5.2.2 Node.js, Express.js in Sequelize.js

Kombinacija JavaScript programskega jezika in vseh treh JavaScript knjižnic (Node.js, Express.js in Sequelize.js) je idealna izbira za razvojne ekipe, ki želijo močan, hitro uvajalni sveženj tehnologij, ki ga uporablja velik del razvojne skupnosti in veliko podjetij [5]. Cilj vseh omenjenih knjižnic je, da opravimo vse potrebno na strani strežnika z istim programskim jezikom.

Node.js je bil predstavljen leta 2009 in omogoča izdelavo dogodkovnega omrežnega strežnika, ki omogoča asinhrone vhodno/izhodne operacije. Node.js tako omogoča delovanje JavaScripta tudi v vlogi strežnika [6].
Express.js je minimalistična in prilagodljiva Node.js knjižnica, s katero lahko izdelamo naš aplikacijski programski vmesnik (API)[9] in uporabimo nekatere dodatne funkcije, ki nam olajšajo delo. Tako bomo poskrbeli, da dobi naš strežnik podatke iz podatkovne baze preko API klicev. Express.js je trenutno najbolj priljubljena Node.js knjižnica [6][25].

Zadnja knjižnica iz logičnega sloja je Sequelize.js. Knjižnico smo uporabili za definiranje JavaScript objektov in za izvajanje CRUD[10] operacij (izdelaj, preberi, popravi, izbriši) na naši PostgreSQL podatkovni bazi. Sequelize.js nam omogoča povezovanje med vnosi v podatkovni bazi in JavaScript objekti. Na ta način si delo precej olajšamo, saj se ukvarjamo le z JavaScript objekti, ki smo jih vajeni iz običajnega objektno orientiranega programiranja.

5.2.3 PostgreSQL

PostgreSQL je odprtokodni sistem za upravljanje relacijskih podatkovnih baz, ki ga razvijajo od leta 1982. PostgreSQL smo izbrali, ker je precej hitra [7] in jo naš gostitelj spletne strani tudi podpira.

[9] https://sl.wikipedia.org/wiki/Vmesnik_za_namensko_programiranje
[10] https://en.wikipedia.org/wiki/Create,_read,_update_and_delete

5.2.4 Postman

Za testiranje REST[11] (reprezentativen prenos stanja) API-ja smo uporabili orodje Postman. REST API je arhitekturni slog za API, ki za dostop do podatkov in njihovo uporabo uporablja zahteve glavnega protokola za prenos podatkov na spletu HTTP[12]. Protokol omogoča operacije GET, PUT, POST in DELETE (branje, posodabljanje, ustvarjanje in brisanje) podatkov.

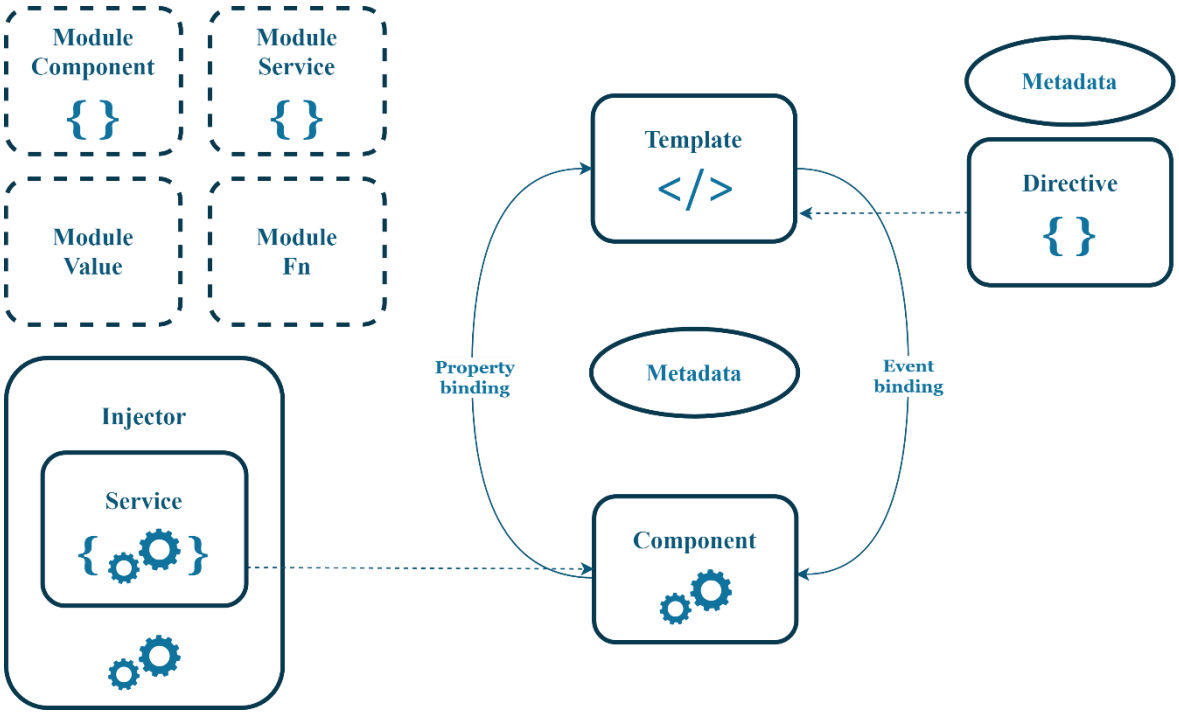
Program Postman je nastal leta 2012 kot stranski projekt za poenostavitev delovnega procesa API pri testiranju in razvoju. Ponuja eleganten uporabniški vmesnik, s katerim lahko na enostaven način pošiljamo zahtevke z JSON[13] (objektna notacija za JavaScript) vsebino za preizkus funkcionalnosti API-ja.

5.2.5 Angular in TypeScript

Obstaja več okolij, s katerimi si lahko olajšamo razvoj, kot na primer trenutno priljubljene React in Vue.js. Vendar smo si za predstavitveni sloj izbrali Angular okolje, ker smo z njim najbolj seznanjeni in imamo z njim tudi največ izkušenj.

Angular je razvilo podjetje Google leta 2016 kot odprtokodno okolje. Trenutna različica temelji na TypeScriptu jeziku. TypeScript je relativno nov programski jezik. Prvič je bil predstavljen leta 2012 s strani podjetja Microsoft. Je brezplačen in odprtokoden jezik, nadgradnja programskega jezika JavaScript, ki omogoča dodatne možnosti in nam tako olajša delo. Jezik omogoča objektno usmerjeno programiranje na osnovi razredov in modulov ter neobvezno statično tipkanje (angl. static typing), s katerim prevajalnik točno ve, za kakšen tip spremenljivke gre. Na koncu prevajalnik vso kodo prevede v JavaScript, tako da je le ta berljiva v vseh brskalnikih [28].

[11] https://en.wikipedia.org/wiki/Representational_state_transfer
[12] https://sl.wikipedia.org/wiki/HTTP
[13] https://en.wikipedia.org/wiki/JSON



Slika 13: Arhitektura Angular okolja [31]

Angular 2+ nam nudi zelo prilagodljivo arhitekturo (Slika 13) iz komponent (Components), modulov (Modules), storitev (Services), direktiv (Directives) in cevi (Pipes). Skoraj vse v Angularju je komponenta. Tudi sama aplikacija je v korenu pravzaprav komponenta. Komponenta je vedno sestavljena iz štirih datotek s končnicami `.html`, `.css` (v našem primeru smo si izbrali `.scss` vrsto datoteke za stil aplikacije; več v podpoglavju HTML in SCSS), `.ts` in zadnja datoteka ima končnico `.spec.ts`. V `.html` datoteki uredimo strukturo določene komponente. V stilski datoteki `.css` definiramo celotno obliko komponente. Datoteka `.ts` je namenjena vsem funkcijam, ki jih lahko kličemo v `.html` datoteki ali pa v različnih funkcijah življenjskega cikla komponente [8]. Na primer tri najpogostejše funkcije:

- **ngOnInit()** funkcija se izvede takoj ob inicializaciji[14] Angular komponente,
- **ngAfterViewInit()** funkcija se izvede po inicializaciji Angular komponente, ko je komponenta že grafično izrisana in
- **ngOnDestroy()** funkcija se izvede, ko se Angular komponenta odstrani iz HTML strukture.

5.2.6 Angular Material knjižnica

Angular Material je uradna knjižnica podjetja Google. Služi kot pomoč pri hitri izdelavi aplikacij in nudi veliko predefiniranih komponent (več kot 36), ki jih lahko uporabimo na

[14] https://en.wikipedia.org/wiki/Initialization_(programming)

različnih projektih. Med uporabne komponente na primer sodijo komponente »Progress spinner«, »Select«, »Stepper«, »Button toggle«, »Datepicker«, »Dialog« in »Tooltop«. Vsaka komponenta nam rešuje specifičen primer s pohitritvijo implementacije elementov spletne aplikacije:

- »Progress spinner« je grafičen element, ki je predstavljen z vrtečim modrim krogom in pomeni nalaganje določenega dela. Lahko mu določimo različne stile in način izvajanja – determiniran in nedeterminiran. Pri determiniranem določimo tudi vrednost.

- »Select« komponenta nadomesti običajno HTML komponento za izbiro možnosti. Podobno kot pri select html imamo na voljo predefinirane možnosti za različna stanja (onemogočeno, napaka ipd.).

- »Stepper« komponenta nam razdeli obrazec z vnosnimi polji na različne korake in tako dobimo obrazec v načinu čarovnika za obrazce[15] (pomočnik pri vnosu je vrsta uporabniškega vmesnika z zaporedjem pogovornih oken, ki vodijo uporabnika do končnega vnosa podatkov).

- »Button toggle« komponenta nadomesti izbirni gumb (ang. radio button) html komponento, s katero uporabnik določa, ali je možnost izbrana ali ne.

- »Datepicker« komponenta razširi navadno vnosno polje, pri katerem s klikom na polje dobimo dodatno grafično okno z možnostjo izbire datuma.

- »Dialog« komponenta nam olajša delo s pojavnimi okni. Komponenti podamo vsebino in stil ter funkcije, ki so že definirane znotraj komponente. Na voljo imamo funkcije: prikaz okna, odstrani okno, naslov okna, vsebina okna in akcije okna.

- »Tooltip« komponento lahko obesimo na vsak element v html strukturi in ji nato definiramo besedilo oz. html vsebino, ki se prikaže ob premiku miške na ta element.

5.2.7 HERE Maps knjižnica

Ker smo želeli omogočiti tudi prikaz poti na zemljevidu, podobno kot v diplomskem delu »Spletna aplikacija za vozače« [12], smo se odločili za uporabo HERE Maps knjižnice. Knjižnica nam bo predvsem prišla prav v začetnih fazah projekta, ko ne bomo imeli ogromno obiskovalcev, saj bomo s povečanjem obiska morali plačati uporabo knjižnice [27]. Obstaja tudi možnost kasnejše zamenjave z odprtokodno OpenStreetMap knjižnico. HERE Maps API nam nudi iskanje krajev po zemljevidu in postavljanje grafičnih elementov nanj.

[15] https://en.wikipedia.org/wiki/Wizard_(software)

5.2.8 HTML in SCSS (CSS)

Označevalna jezika HTML in CSS se uporabljata za izdelavo spletnih strani. Jezika nista omejena le na splet in se ju uporablja pri izdelavi uporabniških vmesnikov v avtomobilih, SpaceX Dragon 2 [32] vesoljskega plovila itd.

V HTML datoteki postavimo celotno strukturo spletne strani/aplikacije z različnimi HTML gradniki. V peti različici HTML jezika so nabor gradnikov, ki definirajo različne dele spletne strani, povečali. Med te gradnike med drugimi sodijo na primer:
`article`, `aside`, `audio`, `canvas`, `footer`, `header`, `nav`, `section`, `video`.

CSS pa je slogovni jezik, s katerim oblikujemo HTML gradnike. V našem primeru smo uporabili napredno obliko CSS jezika, imenovanega SASS. SASS je pred-procesorski skriptni jezik, ki se na koncu prevede v CSS. Podobno kot TypeScript nam SASS omogoči bogat nabor dodatkov k CSS, s katerimi si olajšamo delo. Na primer, najbolj uporabljena je zagotovo uporaba spremenljivk znotraj SCSS datoteke.

5.2.9 Git

Danes je Git najbolj razširjen [33] sodoben sistem nadzora različic. Je aktivno vzdrževan odprtokodni projekt, ki ga je leta 2005 prvotno razvil Linus Torvalds, začetnik jedra operacijskega sistema Linux. Git ima vse funkcionalnosti, zmogljivosti, varnost in prilagodljivost, ki jih potrebuje večina razvijalskih ekip in posameznih razvijalcev za nadzor različic [26].
Za naš projekt smo uporabili spletno storitev GitHub (**https://github.com/roksamsa/fill-my-car**). Danes si razvoja programske opreme brez sistema za nadzor različic ne moremo predstavljati, saj nam omogoča:

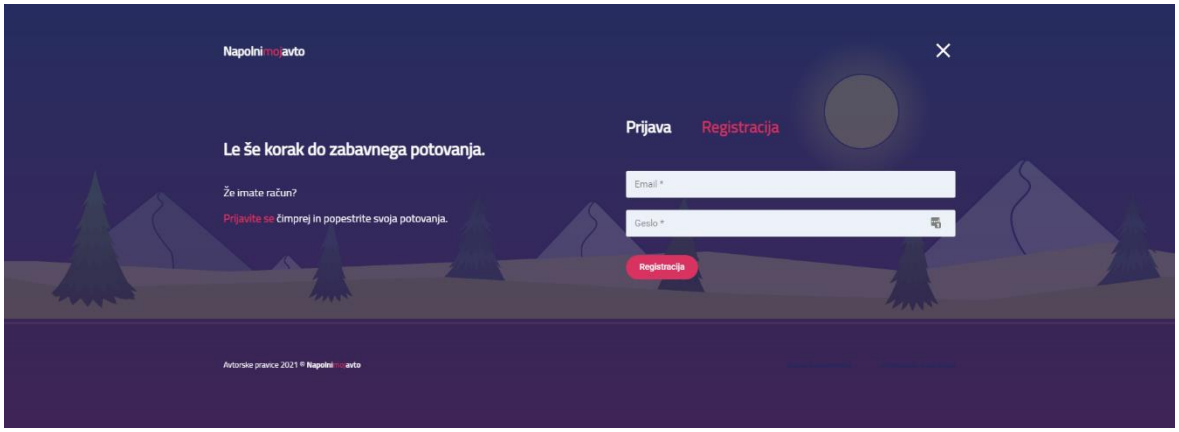
- hranjenje in varnostno različico vseh podatkov in datotek na projektu,
- z več različicami naše aplikacije vrnitev na katerokoli stanje v zgodovini razvoja in
- možnost sodelovanja več razvijalcev hkrati na istem projektu.

5.3 Grafični vmesnik

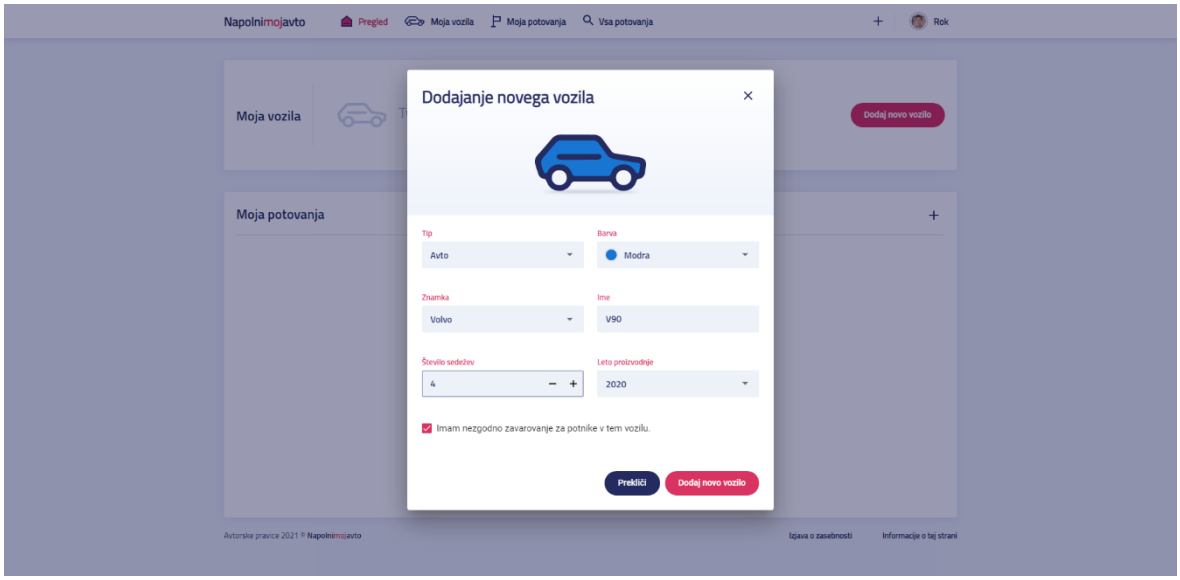
Grafični vmesnik je eden pomembnejših delov našega projekta, saj preko njega uporabnik opravi želeno nalogo in je tako neposredno povezan z uporabniško izkušnjo. Temu delu smo posvetili veliko pozornosti, saj smo ga želeli izboljšati v primerjavi z obstoječimi storitvami. Znano pravilo treh sekund pravi, da uporabnik potrebuje tri sekunde, da ugotovi, ali mu je vmesnik všeč in ali razume njegovo delovanje [29]. Kot smo že zgoraj omenili, smo vmesnik oblikovali z Adobe XD orodjem. Izgled vstopne strani, prijavne stran, strani za vnos novega

vozila, strani za vnos novega potovanja in strani ustvarjenega potovanja je viden na posameznih slikah od 14–18.

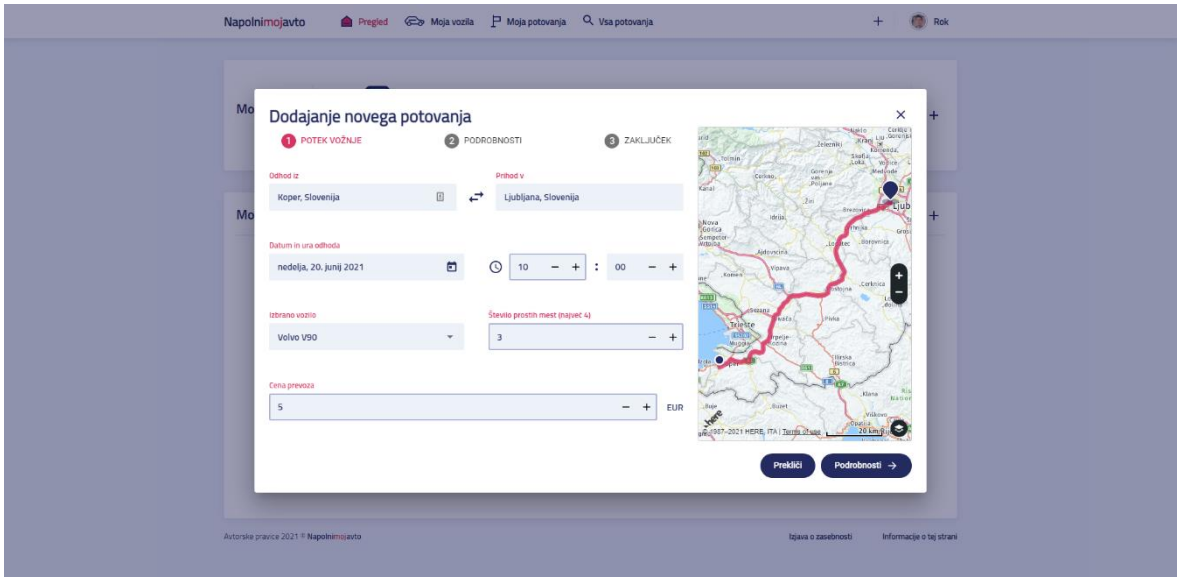
Slika 14: Izgled vstopne strani



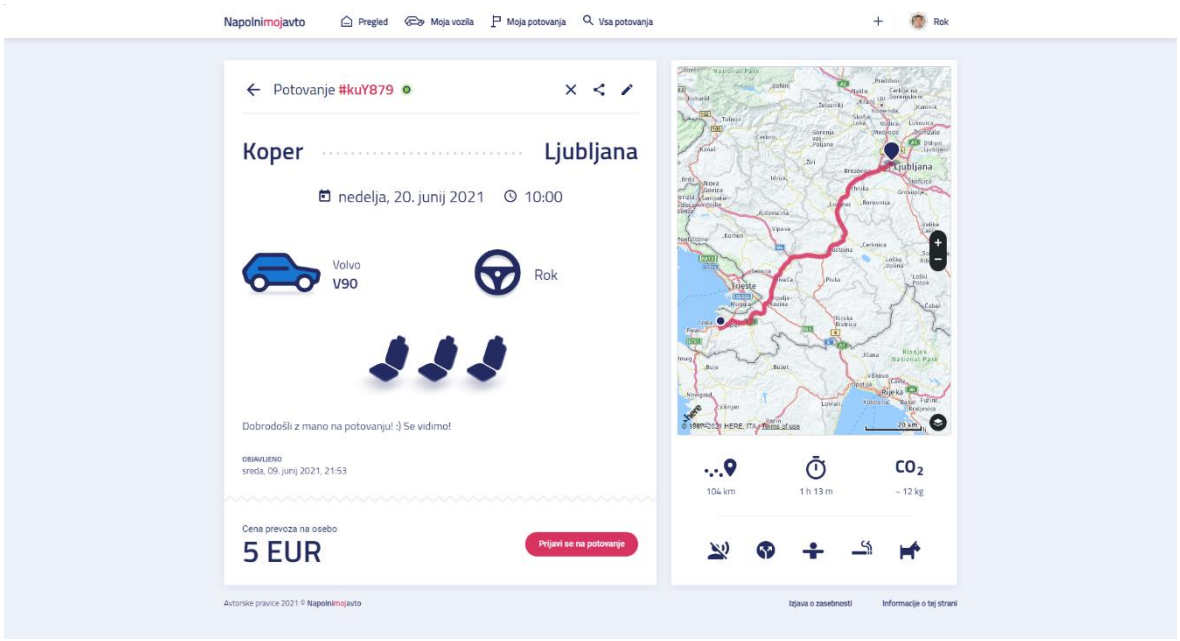
Slika 15: Izgled prijavne strani



Slika 16: Izgled strani za vnos novega vozila



Slika 17: Izgled strani za vnos novega potovanja



Slika 18: Izgled strani ustvarjenega potovanja

5.4 Primer uporabe aplikacije

Slika 19: Prikaz grafičnega vmesnika z vsemi koraki dodajanja vozila in poti

Na Sliki 19 je prikaz vseh korakov dodajanja vozila in poti. Diagram predstavlja grafičen potek (preko zaslonskih slik uporabniškega vmesnika) nekaterih korakov diagramov primerov uporabe iz poglavja “Funkcijske zahteve”. Koraki si sledijo:

1. Klik na gumb “Dodaj novo vozilo” in odpiranje pojavnega okna za dodajanje novega vozila.

2. Po vnosu v pojavnem oknu kliknemo na gumb “Dodaj novo vozilo”; aplikacija nas vrne na “Pregled”.

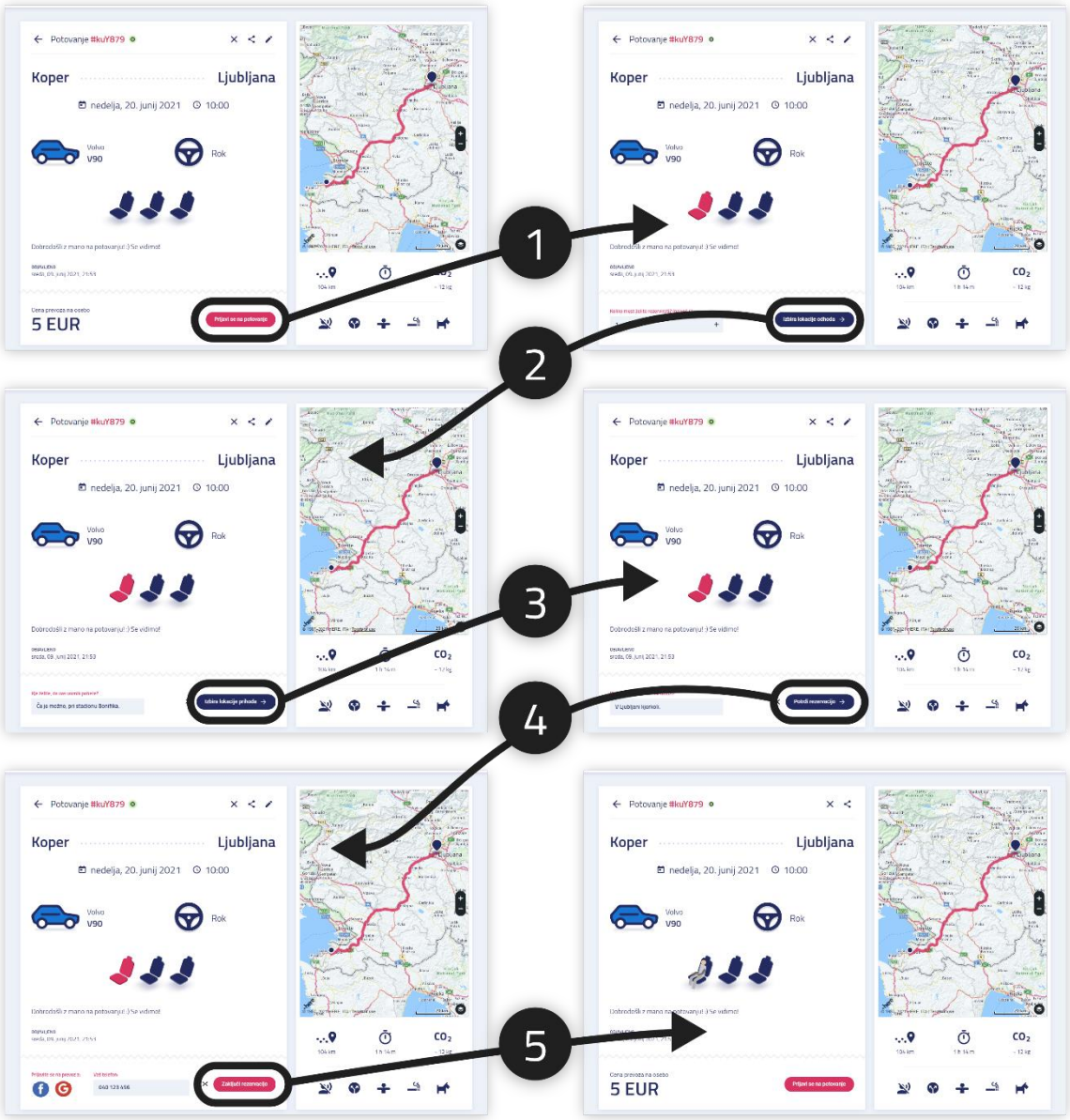
3. Ko imamo dodano vozilo, lahko dodamo pot z gumbom “Dodaj novo potovanje”.

4. Odpre se nam pojavno okno, kjer imamo 3 podkorake – “Potek vožnje”, “Podrobnosti” in “Zaključek”.

5. Pridemo do zadnjega podkoraka in potovanje shranimo z gumbom “Dodaj novo potovanje”.

6. Aplikacija nas ponovno vrne na stran pregleda, kjer lahko vidimo dodano vozilo in dodano potovanje.

7. S klikom na vrstico potovanja se nam odpre pregledna stran za naše potovanje. Stran je vidna vsem, ne samo tistemu, ki jo je ustvaril.



Slika 20: Prikaz grafičnega vmesnika z vsemi koraki prijave na potovanje

Na Sliki 20 je prikaz vseh korakov prijave na potovanje. V tem primeru uporabe aplikacije je predstavljena prijava na potovanje s koraki:

1. Klik na gumb “Prijavi se na potovanje” in odpre se nov vsebinski del, kjer mora uporabnik navesti, koliko mest želi rezervirati.

2. S klikom na gumb “Izbira lokacije odhoda” pridemo na nov korak, kjer napišemo našo željo o lokaciji odhoda.

3. S klikom na gumb “Izbira lokacije prihoda” podobno vnesemo željo o lokaciji prihoda.

4. S klikom na gumb “Potrdi rezervacijo” pridemo na zadnji korak, kjer vnesemo našo telefonsko številko in se po želji prijavimo s Facebookovim oziroma Googlovim računom.

5. Zadnji korak pa nas pripelje do končane rezervacije sedežev. Ob tem tudi dobimo elektronsko sporočilo o opravljeni registraciji.

5.5 Načrtovanje podatkovne baze in API-ja

Vse podatke bomo zapisovali in brali iz PostgreSQL podatkovne baze preko REST API metode. Edini del, ki ne bo na naši strani in za katerega bomo uporabljali zunanjo storitev (zgoraj omenjena Facebook in Google), je registracija in prijava/odjava uporabnika. S tem smo se hoteli izogniti skrbi za varnost uporabnikov v začetni fazi projekta.

Shemo podatkovne baze smo definirali s tabelami: Vozila, Potovanja, Potniki, Uporabnik – Potovanje in Uporabnik – Vozilo. Na strani logičnega sloja smo za definicijo uporabili JavaScript knjižnico Sequelize.js. Entitetno relacijski diagram podatkovne baze je viden na Sliki 21.

Podatke iz baze najprej prebere API, ki smo ga definirali na poddomeni api.napolnimojavto.si. Zaradi varnosti smo vpeljali CORS (Cross-Origin Resource Sharing) pravilo, da lahko samo naša domena napolnimojavto.si dostopa do te poddomene in posledično do API-ja. To je mehanizem, ki uporablja dodatne lastnosti v HTTP glavi, da brskalniku sporoči, naj spletna aplikacija, ki deluje pri enem izvoru (domeni), dovoli dostop do izbranih virov s strežnika z različnim izvorom. Spletna aplikacija izvrši zahtevo HTTP navzkrižnega izvora, kadar zahteva vir, ki ima drugačen izvor (domena, protokol in vrata) od svojega lastnega izvora [30].

5.5.1 Podatkovne tabele

Tabela »Vozila«

```
module.exports = (sequelize, Sequelize) => {
  const Vehicle = sequelize.define('Vehicle', {
    belongsToUser: {
      type: Sequelize.STRING
    },
    vehicleType: {
      type: VehicleType
    },
    vehicleBrand: {
      type: VehicleBrand
    },
    vehicleName: {
      type: Sequelize.STRING
    },
    vehicleModelYear: {
      type: Sequelize.INTEGER
    },
    vehicleColor: {
      type: VehicleColor
    },
    vehicleSeats: {
      type: Sequelize.INTEGER
    },
    vehicleInsurance: {
      type: Sequelize.BOOLEAN
    }
  });
  return Vehicle;
};
```

Tabela »Potniki«

```
module.exports = (sequelize, Sequelize) => {
  const TripPassenger = sequelize.define('TripPassenger', {
    belongsToUser: {
      type: Sequelize.STRING
    },
    belongsToVehicle: {
      type: Sequelize.STRING
    },
    belongsToTrip: {
      type: Sequelize.STRING
    },
    tripPassengerSeatsReservation: {
      type: Sequelize.INTEGER
    },
    tripPassengerStartLocation: {
      type: Sequelize.STRING
    },
    tripPassengerEndLocation: {
      type: Sequelize.STRING
    },
    tripPassengerName: {
      type: Sequelize.STRING
    },
    tripPassengerEmail: {
      type: Sequelize.STRING
    },
```

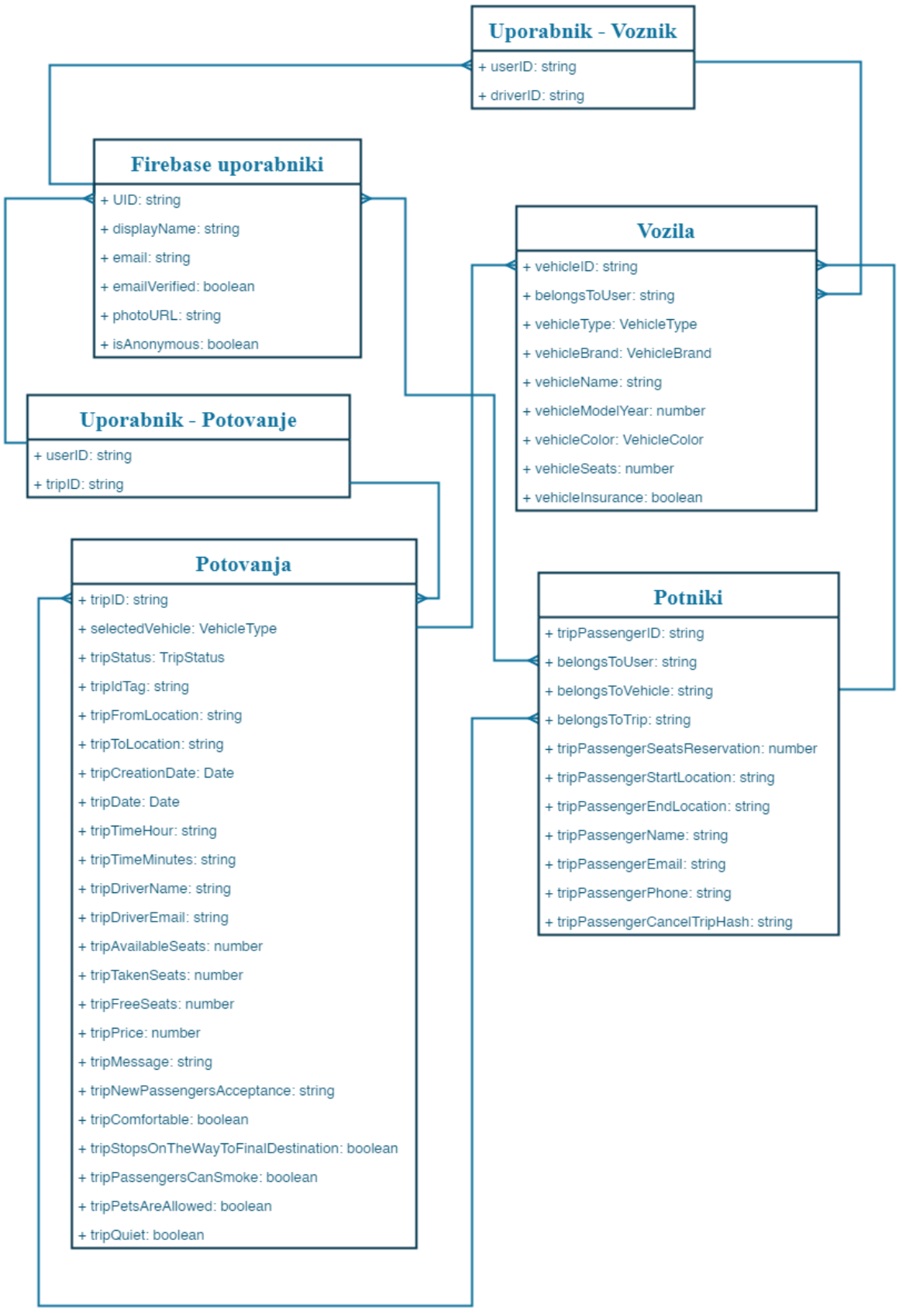
```
    tripPassengerPhone: {
      type: Sequelize.STRING
    },
    tripPassengerCancelTripHash: {
      type: Sequelize.STRING
    }
  });
  return TripPassenger;
};
```

Tabela »Potovanja«

```
module.exports = (sequelize, Sequelize) => {
  const Trip = sequelize.define('Trip', {
    belongsToUser: {
      type: Sequelize.STRING
    },
    selectedVehicle: {
      type: Sequelize.STRING
    },
    tripStatus: {
      type: Sequelize.STRING
    },
    tripIdTag: {
      type: Sequelize.STRING
    },
    tripFromLocation: {
      type: Sequelize.STRING
    },
    tripToLocation: {
      type: Sequelize.STRING
    },
    tripCreationDate: {
      type: Sequelize.DATE
    },
    tripEditedDate: {
      type: Sequelize.DATE
    },
    tripDate: {
      type: Sequelize.DATE
    },
    tripTimeHour: {
      type: Sequelize.STRING
    },
    tripTimeMinutes: {
      type: Sequelize.STRING
    },
    tripDriverName: {
      type: Sequelize.STRING
    },
    tripDriverEmail: {
      type: Sequelize.STRING
    },
    tripAvailableSeats: {
      type: Sequelize.INTEGER
    },
    tripTakenSeats: {
      type: Sequelize.INTEGER
    },
    tripFreeSeats: {
      type: Sequelize.INTEGER
    },
```

```
    tripPrice: {
      type: Sequelize.INTEGER
    },
    tripMessage: {
      type: Sequelize.TEXT,
      unique: false
    },
    tripNewPassengersAcceptance: {
      type: Sequelize.STRING
    },
    tripComfortable: {
      type: Sequelize.BOOLEAN
    },
    tripStopsOnTheWayToFinalDestination: {
      type: Sequelize.BOOLEAN
    },
    tripPassengersCanSmoke: {
      type: Sequelize.BOOLEAN
    },
    tripPetsAreAllowed: {
      type: Sequelize.BOOLEAN
    },
    tripQuiet: {
      type: Sequelize.BOOLEAN
    }
  });
  return Trip;
};
```

5.5.2 Entitetno relacijski diagram



Slika 21: Entitetno relacijski diagram podatkovne baze

5.5.3 Google Firebase registracija/prijava

Za avtentikacijo uporabnikov bomo uporabili Firebase storitev podjetja Google, ki nam omogoča klicanje njihovega API-ja. S tem pohitrimo proces razvoja, poskrbimo za varnost uporabnikov, dobimo več možnosti prijave v našo aplikacijo (Google, Facebook, Twitter in GitHub) in dobimo samodejno pošiljanje e-sporočil ob različnih prijavnih dogodkih.

6 IZVEDBA

V tej fazi bomo na podlagi izdelanega načrta in zbranih zahtev predstavili vse korake implementacije. Najprej bomo opisali postavitev razvojnega okolja, potem bomo predstavili potek implementacije grafičnega vmesnika in nazadnje še kako so vse druge funkcionalnosti povezane v celoto. Skozi razvoj smo sproti odpravljali najdene napake in pomanjkljivosti.

6.1 Postavitev razvojnega okolja

Naš razvoj je v začetni fazi potekal lokalno na lastnem računalniku. Potek postavitve razvojnega okolja je potekal v tem vrstnem redu:

1. Najprej smo namestili in pripravili potrebna orodja za delo: GIT (git-scm.com/download/win), Node.js (nodejs.org/en), program Postman (postman.com/downloads) podatkovno bazo s celotnim sistemom PostgreSQL (postgresql.org/download/windows) in Microsoft Visual Studio Code (code.visualstudio.com) za pisanje kode.

   Z namestitvijo Node.js paketa, smo dobili tudi npm paketnega upravitelja (angl. package manager), s katerim kasneje tudi namestimo vse ostale potrebne knjižnice (@angular/animations, @angular/fire, @angular/material, rxjs, firebase, angularx-social-login, angular-moment, node-sass ...). Knjižnice smo namestili z ukazom:
   `npm install -g ime-knjižnice`

   V tem primeru `-g` pomeni, da namestimo vsako knjižnico globalno in ne samo lokalno za naš projekt. Torej lahko vse ukaze te knjižnice uporabljamo v vseh direktorijih in pod-direktorijih našega operacijskega sistema.

2. Ko smo namestili vse potrebno, smo izpeljali naslednji korak nameščanja. Našemu Git Bash terminalu smo spremenili položaj direktorija v »server« direktorij, in sicer z ukazom `$cd server`. To pa zato, ker bomo vanj shranili določene knjižnice, kot so knjižnice, od katerih je naš projekt odvisen (angl. dependency) in jih bomo kasneje vedno namestili z ukazom `npm install`. Namestili smo knjižnice Angular, Express.js in Sequelize.js:

   ```
   npm install -g @angular/cli
   npm install express --save
   npm install sequelize -save
   ```

3. V naslednjem koraku smo ustvarili projekt na GitHub novega repozitorija (angl. repository) z imenom »fill-my-car«. Tako smo poskrbeli, da bo projekt vedno

shranjen varno v oblaku in da bomo imeli vse korake razvoja kasneje na voljo, če pride do razvojnih problemov.

4. Ustvarjen GitHub projekt smo potem prenesli na lokalni računalnik z ukazom
   `git clone https://github.com/roksamsa/fill-my-car.git`
   in ustvarili strukturo našega projekta:

Projekt smo ločili na dva glavna direktorija, in sicer na »server« direktorij, ki je namenjen vsem datotekam strežnika in direkotrij »public«, namenjen Angular projektu. To tudi zato, ker bodo vse datoteke »server« direktorija tudi na drugi lokaciji strežnika in drugi domeni. V našem primeru na poddomeni api.napolnimojavto.si. V naslednjem koraku se bomo lotili priprave kode za zagon lokalnega strežnika.

6.2 Priprava Node.js in Express.js strežnika

V že ustvarjeni datoteki `server.js` definiramo vse potrebne ukaze za zagon strežnika. Ukaz za zagon strežnika smo definirali kot zadnjo vrstico:
`app.listen(portForServer, () => console.log(`App listening at http://localhost:${portForServer}`));`
Pred tem smo definirali še druge funkcionalnosti:

- dodali smo nastavitve za povezavo s podatkovno bazo: `database/database-setup`,
- definirali smo spremenljivko, s katero enostavno preidemo iz nastavitev produkcije na nastavitve razvoja/testiranja,
- dodali smo »Cookie parser« knjižnico, s katero hranimo vse piškotke v spremenljivki, da jih lahko kasneje uporabimo,
- uporabili smo »Helmet« knjižnico, ki nam izboljša varnost Node.js in Express.js strežnika z dodajanjem različnih nastavitev v HTTP glavo (angl. HTTP header),
- dodali smo URL poti (angl. routes) REST API strežnika, preko katerih bomo dobili JSON podatke iz podatkovne baze,
- dodali smo URL poti za pošiljanje e-sporočil uporabnikom ob določenih dogodkih in
- CORS.

Za delovanje vseh knjižnic smo morali tudi vsako še prej namestiti, in sicer z ukazom: `npm i ime-knjiznice`. Na koncu imamo v vsakem takem projektu datoteko `package.json`, v kateri so definirane vse zahtevane knjižnice. Tako lahko projekt namestimo samo z enim ukazom `npm install` oziroma s krajšim ukazom `npm i`.

Strežnik smo zagnali z ukazom `babel-watch server.js`. Ukaz prihaja iz istoimenske knjižnice in skrbi, da se strežnik ponovno naloži, če smo spremenili katerokoli datoteko na strežniku.

6.3 Angular spletna aplikacija

Naslednji korak je razvoj Angular aplikacije. Z ukazom `ng new zapolni-moj-avto --style=scss` smo ustvarili aplikacijo znotraj direktorija »public«. Začetnem ukazu smo

dodali še dodatno možnost `--style=scss` za globalno uporabo `.scss` datotek, s katerimi bomo oblikovali vse komponente Angular aplikacije. Aplikacijo smo pognali z ukazom `ng serve --open --port 4444`. S tem ukazom smo definirali, katera vrata v brskalniku naj naša aplikacija uporablja in zahtevali, da se aplikacija po končanem nalaganju samodejno odpre v privzetem brskalniku.

V naši aplikaciji smo definirali 50 Angular komponent (osnovni gradniki Angular okolja, s katero razbijemo aplikacijo na več komponent, ki jih kasneje uporabljamo eno v drugi ali ločeno) s svojimi `.html`, `.scss` in `.ts` datotekami, od tega:

- 6 komponent za vsebinske dele strani, kot je domača stran,
- 4 komponente za stil (na primer uporabili smo enak stil za vse bele okvirje s senco, ki smo jih napolnili z vsebino),
- 6 komponent za strani z vsebino iz podatkovne baze,
- 2 komponenti za grafično postavitev celotne strani (lahko smo izbrali med tipom strani z glavo in nogo ali med tipom strani brez glave in noge),
- 6 komponent za vsa pojavna okna (angl. popup dialog),
- in vse ostale komponente kot dele drugih komponent, ki jih je bilo lažje uporabljati zaradi pregleda in urejenosti kode.

7 TESTIRANJE

Preden aplikacijo damo v uporabo, jo je potrebno testirati. Pri izvedbi smo sproti testirali, vendar je potrebno narediti še končni celoviti test. Poznamo dva načina testiranja. Testiranje lahko opravi razvijalec aplikacije ali neodvisna oseba (ali več oseb), ki ne pozna projekta. Slednji način je boljši, ker imajo neodvisni/zunanji uporabniki drugačen pogled na razvito aplikacijo. Najdene napake je potrebno rešiti [9].

Načrtovanje testiranja in samo testiranje bo v našem primeru izvedel razvijalec aplikacije zaradi zdravstvenega epidemiološkega stanja v državi v času izdelave aplikacije. Za testiranje take aplikacije, ki smo jo razvili v sklopu tega dela, je potrebnih od 5 do 15 uporabnikov [34]. V vsaki seji uporabimo 5 uporabnikov in po vsaki seji odpravimo najdene pomanjkljivosti in napake sistema. V treh sejah uporabniki dokazano ugotovijo vse uporabniške pomanjkljivosti in napake. Tega žal ni bilo možno izvesti. V naslednjem podpoglavju bomo načrtovali testne primere, ki morajo temeljiti na specifikaciji zahtev.

7.1 Načrtovanje testiranja

V tem podpoglavju smo definirali komponente testiranja in pričakovane rezultate.

Testiranje bomo izvedli na Windows 10 operacijskem sistemu v vseh danes priljubljenih brskalnikih (Google Chrome 83.0.4103.116 – 64-bitni, Mozilla Firefox 78.0b9 – 64-bitni in Microsoft Edge 83.0.478.58 64-bitni).

Sistemsko testiranje

Tabela 1: Sistemsko testiranje in njihovi pričakovani rezultati

TESTIRANE KOMPONENTE	PRIČAKOVANI REZULTAT
Prenos podatkov iz in v podatkovno bazo	podatki prenešeni
Varnost	varna aplikacija
Izgled aplikacije v vseh napravah	urejenost izgleda na telefonih, računalnikih in ostalih podprtih napravah
Enotno obnašanje v različnih brskalnikih	enak izgled in obnašanje
Prepustnost	do 50 uporabnikov
Pomnilnik	primerna zasedenost

Testiranje uporabnikov

Tabela 2: Testiranje uporabnikov in njihovi pričakovani rezultati

TESTIRANE KOMPONENTE	PRIČAKOVANI REZULTAT
Registracija/prijava v aplikacijo	uporabnik registriran in prijavljen v aplikacijo
Vnos novega avtomobila	dodano vozilo
Vnos novega potovanja	dodano potovanje
Dodajanje vozila k potovanju	dodano vozilo k potovanju
Izbira prevoza	izbran prevoz
Odjava iz aplikacije	odjava

7.2 Izvedba in rezultati testiranja

Sistemsko testiranje

Tabela 3: Rezultati sistemskega testiranja

TESTIRANE KOMPONENTE	PRIČAKOVANI REZULTAT	REZULTATI
Prenos podatkov iz in v podatkovno bazo	podatki prenešeni	uspešen prenos
Varnost	varna aplikacija	zagotovljena
Izgled aplikacije v vseh napravah	urejenost izgleda na telefonih, računalnikih in ostalih podprtih napravah	deloma zagotovljeno
Enotno obnašanje v različnih brskalnikih	enak izgled in obnašanje	zagotovljena
Prepustnost	do 50 uporabnikov	v redu
Pomnilnik	primerna zasedenost	zasedenost pomnilnika optimizirana

Testiranje uporabnikov

Tabela 4: Rezultati testiranja uporabnikov

TESTIRANE KOMPONENTE	PRIČAKOVANI REZULTAT	REZULTATI
Registracija/prijava v aplikacijo	uporabnik registriran in prijavljen v aplikacijo	uspešna registracija/prijava uporabnika v aplikacijo
Vnos novega avtomobila	dodano vozilo	uspešno dodano vozilo
Vnos novega potovanja	dodano potovanje	uspešno dodano potovanje
Dodajanje vozila k potovanju	dodano vozilo k potovanju	uspešno dodano potovanje
Izbira prevoza	izbran prevoz	uspešno izbran prevoz
Odjava iz aplikacije	odjava	uspešna odjava

1. Testiranje smo pričeli na strežniškem nivoju. Testirali smo osnovne funkcije za branje in pisanje v lokalno podatkovno bazo, in sicer prenos podatkov iz podatkovne baze preko REST API-ja do prikaza podatkov na spletni strani. Pridobili smo vse željene podatke z vsemi funkcijami. Na primer: `getAllVehicles()`, `getVehicleById(vehicleId: string)`, `getVehicleByUser(belongsToUser: string)`, `addVehicle()`, `updateVehicle()` in `deleteVehicle(id: any)`.

2. Registracijo in kasnejšo prijavo smo v prvi fazi razvili samo za socialna omrežja. Tako se uporabnik lahko prijavi z: Google, Facebook, Twitter ali GitHub računom. Prijava je bila uspešna.

3. Vozilo smo brez težav dodali.

4. Potovanje smo brez težav ustvarili.

5. Med ustvarjanjem potovanja smo uspešno dodali vozilo, s katerim bomo peljali potnike.

6. Odjava je bila uspešna. Po odjavi nas je aplikacija preusmerila na začetno stran, kar je bil tudi namen.

7. Varnost aplikacije smo testirali na različne načine. Obiskali smo poddomeno api.napolnimojavto.si, na kateri bi morali dobiti napako. Do podatkovne baze tudi nismo imeli dostopa. Preverili smo, kaj vse nam konzole brskalnikov pokažejo in

skušali predvideti, kje bi lahko imeli varnostne probleme. V nobenem primeru nismo zasledili možnosti vdora.

8. Edino težavo, na katero smo naleteli, je bil izgled aplikacije na vseh napravah. Elementi niso bili pravilno poravnani, ponekod je bilo besedilo premajhno in zemljevid ni bil viden. Vse te napake so nam kasneje vzele še nekaj dodatnega časa in smo jih odpravili.

9. Aplikacijo smo testirali tudi na različnih brskalnikih na našem prenosniku z ločljivostjo 1920 * 1080 pik. Vse se je obnašalo enako in izgled je bil enak.

Testiranje smo uspešno zaključili in naša aplikacija je pripravljena na objavo.

8 ZAKLJUČEK

V zaključni nalogi smo predstavili primer načrtovanja programskega izdelka »Zapolni moj avto«, od same ugotovitve problema do izvedbe in testiranja. Z razvojem aplikacije smo hoteli izboljšati in olajšati uporabo aplikacije Prevoz.org. Skozi celotni razvojni proces smo aplikacijo pripeljali do osnovne faze uporabe. Za nadaljnji razvoj imamo v mislih veliko idej. Prva je testiranje uporabnosti z neodvisnimi uporabniki, ki bi lahko našli dodatne uporabniške težave. Druga je razvoj mobilne aplikacije za operacijska sistema Android in iOS. Aplikaciji bi dodali lokalizacijo in s tem možnost prevoda v druge jezike in uporabo na širšem geografskem področju. Uporabnikom aplikacije bomo dodali možnost statistike in zgodovine, ocenjevanje voznikov, potnikov in vozil, česar v trenutni različici nismo naredili zaradi velikega obsega celotne aplikacije. Želeli bi si tudi razširiti ekipo razvijalcev in tako pospešiti razvoj.

9 LITERATURA IN VIRI

[1] JAMES RUMBAUGH, IVAR JACOBSON, GRADY BOOCH, *The Unified Modeling Language Reference Manual, Second Edition,* Addison-Wesley Professional, 2004. (Citirano na strani 6).

[2] MARTIN FOWLER, *UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition,* Addison-Wesley Professional, 2003. (Citirano na strani 6).

[3] CHARLES S. WASSON, *System Analysis, Design, and Development Concepts, Principles, and Practices, First Edition,* Wiley-Interscience, 2005. (Citirano na straneh 6 in 10).

[4] BRIAN WOOD, *Adobe XD CC Classroom in a Book, First Edition,* Adobe Press, 2019. (Citirano na strani 13).

[5] ETHAN BROWN, *Web Development with Node and Express: Leveraging the JavaScript Stack First Edition,* O'Reilly Media, 2014. (Citirano na strani 13).

[6] GUILLERMO RAUCH, *Smashing Node.js: JavaScript Everywhere, Second Edition,* Wiley, 2012. (Citirano na strani 13).

[7] REGINA OBE, LEO HSU, *PostgreSQL: Up and Running, First Edition,* O'Reilly Media, 2012. (Citirano na strani 14).

[8] ADAM FREEMAN, *Pro Angular 9, First Edition,* Apress, 2020. (Citirano na strani 15).

[9] STEVE MCCONNELL, *Rapid Development: Taming Wild Software Schedules First Edition,* Microsoft Press, 1996.

[10] CHRISTOPHER REID BECKER, *Learn Human-Computer Interaction: Solve human problems and focus on rapid prototyping and validating solutions through user testing,* Packt Publishing, 2020

[11] TINE TROBEC, *Diplomsko delo: Analiza skupnih voženj dnevnih migrantov z osebnimi vozili na slovenskih avtocestah,* 2011.

[12] NEJC GAŠPERIN, *Diplomsko delo: Spletna aplikacija za vozače,* 2012.

[13] ANŽE MIHEVC, *Diplomsko delo: Uporabniška izkušnja spletnega mesta www.prevoz.org,* 2014.

[14] Statistika za Prevoz.org na osebni spletni strani ustanovitelja in razvojnika storitve Jureta Čuhaleva. Dostopno 7. 6. 2020 na internetu: http://www.jurecuhalev.com/blog/projects

[15] Statistika za BlaBlaCar spletno storitev iz uradne spletne podstrani »O nas«. Dostopno 11. 6. 2020 na internetu: https://blog.blablacar.com/about-us

[16] Seznam držav prisotnosti BlaBlaCar spletne storitve. Dostopno 12. 6. 2020 na internetu: https://en.wikipedia.org/wiki/BlaBlaCar

[17] Poročilo podjetja BlaBlaCar o približnih privarčevanih količinah CO_2 izpustov pri vožnjah na letni ravni. Dostopno 12. 6. 2020 na internetu: https://blog.blablacar.com/newsroom/news-list/zeroemptyseats

[18] Prvi slovenski etični heker Milan Gabor pojasni zakaj je dobro zasnovati dobro varnost aplikacije že takoj na začetku razvoja. Dostopno 28. 6. 2020 na internetu (od 05.22 naprej): https://www.youtube.com/watch?v=rloemh2U_z0

[19] Članek »Razlike med vsemi poimenovanji hekerjev« na uradni strani znanega podjetja za varnostno programsko opremo Norton. Dostopno 28. 6. 2020 na internetu: https://us.norton.com/internetsecurity-emerging-threats-what-is-the-difference-between-black-white-and-grey-hat-hackers.html

[20] Članek o najbolj znanih primerih vdorov v 21. stoletju. Dostopno 28. 6. 2020 na internetu: https://www.csoonline.com/article/2130877/the-biggest-data-breaches-of-the-21st-century.html

[21] Članek o najbolj znanih podjetjih, ki uporabljajo Node.js za svoje produkte. Dostopno 1. 7. 2020 na: https://softwarebrothers.co/blog/companies-that-use-node-js/

[22] Spletna stran z zbranimi izdelki, ki so narejeni v Angular okolju. Dostopno 1. 7. 2020 na: https://www.madewithangular.com/

[23] Uradna spletna stran Adobe.com s cenikom za Adobe XD program. Dostopno 1. 7. 2020 na: https://www.adobe.com/products/xd/compare-plans.html

[24] Kratka zgodovina programa Adobe XD. Dostopno 1. 7. 2020 na: https://www.getcloudapp.com/blog/adobexd-cc-experience-design

[25] Uradna GitHub stran za Express.js knjižnico. Dostopno 1. 7. 2020 na internetu: https://github.com/expressjs/express

[26] Članek »Kaj je Git?« Dostopno 1. 7. 2020 na internetu: https://www.atlassian.com/git/tutorials/what-is-git

[27] Uradna HERE Maps stran o planih plačevanja. Dostopno 2. 7. 2020 na internetu: https://developer.here.com/pricing

[28] Kaj je TypeScript? Dostopno 2. 7. 2020 na internetu: https://www.cleverism.com/skills-and-tools/typescript/

[29] Kako pritegniti pozornost uporabnika, pravilo treh sekund. Dostopno 2. 7. 2020 na internetu: http://vsellis.com/how-to-get-customers-to-take-action-the-3-second-rule/

[30] Cross-Origin Resource Sharing (CORS). Dostopno 2. 7. 2020 na internetu: https://developer.mozilla.org/nl/docs/Web/HTTP/CORS

[31] Uvod v koncepte Angular okolja. Dostopno 27. 4. 2021 na internetu: https://angular.io/guide/architecture

[32] Članek o uporabi HTML, CSS in Javascript jezikov v vesoljnem plovilu SpaceX Crew Dragon. Dostopno 27. 4. 2021 na internetu: https://hackaday.com/2020/06/08/displaying-html-interfaces-and-managing-network-nodes-in-space/

[33] Članek o različnih sistemih za nadzor revizij programske opreme in kako izbrati najboljšo. Dostopno 27. 4. 2021 na internetu: https://mentormate.medium.com/svn-git-out-of-here-how-to-choose-your-version-control-system-10b44750a75c

[34] Zakaj ne potrebujemo več kot pet ljudi za preizkus uporabnosti. Dostopno 9. 8. 2021 na internetu: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/

PRILOGE

PRILOGA A *Node.js koda za zagon strežnika*

```
const express = require('express');
const session = require('express-session');
const cors = require('cors');
const bodyParser = require('body-parser');
const cookieParser = require('cookie-parser');
const path = require('path');
const http = require('http');
const helmet = require('helmet');
const morgan = require('morgan');

// Import database settings
const database = require('./database/database-setup');

// Is in production?
const isProduction = false;

// Other variables
const portForServer = 4000;
const localDomainName = 'https://localhost:4444';
const webDomainName = 'https://api.napolnimojavto.si/';
let defaultDomainName = '';

// All console messages
const connectionMessageRunning =
'Connection from PostgreSQL database has been established successfully and it is running.';
const connectionMessageNotRunning ='Unable to connect to the PostgreSQL database. Error:';
const consoleAppRunningTypeMessagePROD = 'Application is in PRODUCTION mode.';
const consoleAppRunningTypeMessageDEV = 'Application is in DEVELOPMENT mode.';

const router = express.Router();
const app = express();
const corsOptions = {
  origin: [localDomainName, webDomainName, 'http://localhost:4000'],
  optionsSuccessStatus: 200,
  credentials: true
};
app.use(cors(corsOptions));
app.use(cookieParser());
app.use(helmet());

// Parse application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({
  extended: true
}));

// Parse application/json
app.use(bodyParser.json());
app.use(require('morgan')('dev'));
app.use(express.static(path.join(__dirname, '')));
app.use(session({
  secret: '',
  cookie: { maxAge: 60000 },
  resave: true,
  saveUninitialized: true
}));
app.use(morgan('combined'));

```

```
// Database setup
database.sequelize.sync().then(() => {
  console.log(connectionMessageRunning);
}).catch((error) => {
  console.error(connectionMessageNotRunning, error);
});

// Import data
require('./controller/auth');

// App
app.use('/', router);
app.use(require('./controller/index'));
app.use(require('./controller/vehicles/vehicles.route'));
app.use(require('./controller/trips/trips.route'));
app.use(require('./controller/trip-passengers/trip-passengers.route'));

// Emails
app.use(require('./controller/emails/email/email-join-trip-driver'));
app.use(require('./controller/emails/email/email-join-trip-passenger'));
app.use(require('./controller/emails/email/email-cancels-trip-driver'));

// HTTP Server
const serverHTTP = http.createServer(app);

if (isProduction === true) {
  defaultDomainName = webDomainName;
  defaultServerType = serverHTTP;
  consoleAppRunningTypeMessage = consoleAppRunningTypeMessagePROD;
} else {
  defaultDomainName = localDomainName;
  defaultServerType = serverHTTP;
  consoleAppRunningTypeMessage = consoleAppRunningTypeMessageDEV;
}

// Add CORS middleware headers
app.use(function (req, res, next) {
  res.header('Access-Control-Allow-Origin', webDomainName);
  res.header('Access-Control-Allow-Origin', defaultDomainName);
  res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, PATCH, DELETE');
  res.setHeader('Access-Control-Allow-Headers', 'X-Requested-With, Content-
Type, Accept, Authorization, Origin');
  res.setHeader('Access-Control-Allow-Credentials', true);

  next();
});

app.listen(portForServer, () => console.log(`
  App listening at http://localhost:${portForServer}`
));
```

PRILOGA B *REST API poti za primer vozila*

```
const express = require('express');
const database = require('../../database/database-setup');
const router = express.Router();
const Vehicle = database.vehicles;
const vehiclesURI = '/vehicles';

// Get all vehicles
router.get(vehiclesURI, function (req, res) {
  Vehicle.findAll().then(vehicles => {
    // Send All Vehicles to Client
    res.json(vehicles);
  }).catch(error => {
    console.log(error);
    res.status(500).json({msg: "error", details: error});
  });
});

// Get a single vehicle by id
router.get(vehiclesURI + '/:id', function (req, res) {
  const id = req.params.id;
  Vehicle.findOne({where: {id: id}})
    .then(vehicle => {
      res.json(vehicle);
    })
    .catch(error => {
      res.status(500).json({msg: "error", details: error});
    });
});

// Get all vehicles for specific user
router.get(vehiclesURI + '/user/:belongsToUser', function (req, res) {
  Vehicle.findAll({where: {belongsToUser: req.params.belongsToUser},
    order: [
      ['id', 'DESC']
    ]})
    .then(vehicles => {
      res.json(vehicles);
    }).catch(error => {
      console.log(error);
      res.render('error', { error: err });
      res.status(500).json({msg: "Error!", details: error});
    });
});

// Create a new vehicle
router.post(vehiclesURI + '/add', function (req, res) {
  const vehicle = new Vehicle(req.body);
  vehicle.save()
    .then(vehicle => {
      res.status(200).json({'Vehicle': vehicle + ' added successfully'});
    })
    .catch(error => {
      res.status(400).send('Failed to create new vehicle, with error: ' + error);
    });
});

// Delete a specific vehicle with id
router.delete(vehiclesURI + '/delete/:id', function (req, res) {
  const id = req.params.id;
  Vehicle.destroy({where: {id: id}})
```

```
    .then(vehicle => {
      res.status(200).json({msg: 'Vehicle with id: ' + vehicle + ' deleted successfully.'});
    })
    .catch(error => {
      res.status(500).json({msg: "error", details: error});
    });
});

// Update a vehicle with id
router.patch(vehiclesURI + '/update/:id', function (req, res) {
  const id = req.params.id;
  const vehicleUpdatedData = req.body;

  Vehicle.update(vehicleUpdatedData, {where: {id: id}})
    .then(vehicle => {
      if (!vehicle) {
        return res.status(404).json({
          message: 'Vehicle with id: ' + id + ' can not be found! Sorry :/'
        });
      } else {
        res.json(vehicle);
      }
    })
    .catch(error => {
      console.log(error);
      res.status(500).json({msg: "error", details: error});
    });
});

module.exports = router;
```

PRILOGA C *HERE Maps komponenta*

```
hereMapInitialSetup() {
  const defaultLayers = this.hereMap.platform.createDefaultLayers();
  defaultLayers.normal.map.setMin(2);

  this.map = new H.Map(
    this.mapElement.nativeElement,
    defaultLayers.normal.map, {
      zoom: 10,
      center: {
        lat: this.hereMapCenterLAT,
        lng: this.hereMapCenterLNG
      },
      pixelRatio: window.devicePixelRatio || 1
    }
  );

  const mapEvents = new H.mapevents.MapEvents(this.map);
  const behavior = new H.mapevents.Behavior(mapEvents);
  this.hereMapUI = H.ui.UI.createDefault(this.map, defaultLayers);
  this.map.setCenter({ lat: 46.119944, lng: 14.815333 }); // Center is GEOSS Slovenije
  this.map.setZoom(7.2);
  this.hereMapsRoute(this.hereMapStart, this.hereMapFinish);

  // Add a resize listener to make sure that the map occupies the whole container
  window.addEventListener('resize', () => this.map.getViewPort().resize());
}

hereMapsRoute(start: string, finish: string) {
  if (start !== '' || finish !== '') {
    this.hereMap.getCoordinates(start).then(geocoderResult1 => {
      this.hereMapRouteStartLat = geocoderResult1[0].Location.DisplayPosition.Latitude;
      this.hereMapRouteStartLng = geocoderResult1[0].Location.DisplayPosition.Longitude;

      if (this.map != null && this.map.getObjects() != null) {
        this.map.removeObjects(this.map.getObjects());
      }

      // Start marker
      const startMarker = new H.map.Marker({
        lat: this.hereMapRouteStartLat,
        lng: this.hereMapRouteStartLng
      }, { icon: this.hereMapStartMarkerIcon });

      this.map.addObject(startMarker);
      this.map.setCenter({ lat: this.hereMapRouteStartLat, lng: this.hereMapRouteStartLng });

      this.hereMap.getCoordinates(finish).then(geocoderResult2 => {
        this.hereMapRouteFinishLat = geocoderResult2[0].Location.DisplayPosition.Latitude;
        this.hereMapRouteFinishLng = geocoderResult2[0].Location.DisplayPosition.Longitude;

        // Finish marker
        const finishMarker = new H.map.Marker({
          lat: this.hereMapRouteFinishLat,
          lng: this.hereMapRouteFinishLng
        }, { icon: this.hereMapFinishMarkerIcon });

        this.map.addObject(finishMarker);

        const routeParameters = {
          'mode': 'fastest;car;traffic:enabled',
```

```
          'waypoint0': 'geo!' + this.hereMapRouteStartLat + ',' + this.hereMapRouteStartLng,
          'waypoint1': 'geo!' + this.hereMapRouteFinishLat + ',' + this.hereMapRouteFinishLng,
          'representation': 'display',
          'routeattributes': 'summary',
          'departure': 'Now',
          'language ': 'sl-sl',
          'country': 'SVN',
          'metricSystem': 'metric'
        };

        this.hereMap.router.calculateRoute(routeParameters, data => {
          const route = data.response.route[0];

          if (route) {
            this.preloadingSpinnerVisibility = false;
            this.hereMap.directions = route.leg[0].maneuver;
            data = route;

            const lineString = new H.geo.LineString();
            data.shape.forEach(point => {
              const parts = point.split(',');
              lineString.pushLatLngAlt(parts[0], parts[1]);
            });

            const routeLine = new H.map.Polyline(lineString, {
              style: {
                strokeColor: 'rgba(217, 51, 98, .75)',
                lineWidth: 8,
                lineCap: 'round'
              }
            });

            this.map.addObject(routeLine);
            this.map.setViewBounds(routeLine.getBounds());
            this.getRouteSummaryData(route.summary);
          } else {
            this.preloadingSpinnerVisibility = true;
          }
        }, error => {
          console.log(error);
        });
      })
    });
  }
}

public getRouteSummaryData(summary: any): void {
  // 1000 is number for km, as 1000m is 1 km
  this.hereMapRouteDistance = this.constant.roundNumber(summary.distance / 1000, 0, 'up');
  this.hereMapRouteTravelTime = this.constant.roundNumber(summary.travelTime / 60, 0, 'up')
  this.hereMapRouteTravelHours = this.constant.roundNumber(this.hereMapRouteTravelTime / 60, 0, 'down');
  this.hereMapRouteTravelMinutes = this.hereMapRouteTravelTime % 60;
  // Route distance divided by average vehicle CO2 emissions and
  in the end multiplied with 1000, so we get kg from g
  this.hereMapRouteVehicleCO2emissions =
    this.constant.roundNumber(this.hereMapRouteDistance * this.averageVehicleCO2emissions, 0, 'down');
}
```