Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nejc Žerjal

Deljenje prevoza v realnem času

diplomsko delo
na univerzitetnem študiju

Mentor: doc. dr. Dejan Lavbič

Ljubljana, 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01983 / 2014
Datum: 11.2.2014

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **NEJC ŽERJAL**

Naslov: **DELJENJE PREVOZA V REALNEM ČASU**
REAL-TIME FLEXIBLE CARPOOLING

Vrsta naloge: DIPLOMSKO DELO UNIVERZITETNEGA ŠTUDIJA

Tematika naloge:

Število avtomobilov na prebivalca se v svetu ves čas povečuje in eden izmed ključnih razlogov je uporaba avtomobila za prevoz v službo. Po svetu, in še posebej v Sloveniji, je velika težava majhna zasedenost avtomobila pri njegovi uporabi, kar je finančno neučinkovito in ima tudi zelo negativne posledice na okolje. Študent naj v okviru diplomske naloge predlaga rešitev omenjenega problema v obliki deljenja prevoza v realnem času z implementacijo prototipne informacijske rešitve za povečanje števila potnikov v vozilu. Pri predlogu se naj predpostavi postavitev zbirnih točk na bolj prometnih mestih, kjer se lahko potnik in voznik srečata. Informacijska rešitev naj na strani odjemalca uporablja mobilne naprave vseh sodelujočih akterjev in učinkovito uporablja senzorje, ki jih nudi (npr. GPS, potisna obvestila idr.) ter zahteva čim manj uporabniške interakcije. Strežniški del rešitve naj bo razvit v obliki storitve v oblaku in naj ima dobro podporo skalabilnosti ob povečani uporabi. Podprto naj bo tudi mikro obračunavanje storitve prevoza s pomočjo digitalne valute Bitcoin. Celoten sistem naj podpira tudi zagotavljanje kvalitete storitve z možnostjo ocenjevanja voznikov in tudi potnikov.

Mentor:
doc. dr. Dejan Lavbič

Dekan:
prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Nejc Žerjal, z vpisno številko **63060320**, sem avtor diplomskega dela z naslovom:

Deljenje prevoza v realnem času

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Dejana Lavbiča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 5. marca 2014 Podpis avtorja:

Zahvaljujem se mentorju, doc. dr. Dejanu Lavbiču, za vse predloge in strokovno pomoč pri izdelavi diplomskega dela.

Posebna zahvala gre staršem za podporo in spodbudo med študijem.

Kazalo

Seznam uporabljanih kratic in simbolov

ACID (ang. Atomicity, Consistency, Isolation, Durability) – Atomarnost, konsistentnost, izolacija, trajnost je množica lastnosti transakcij v podatkovnih bazah, ki omogoča zanesljivo delovanje transakcij.

AJAX (ang. Asynchronous JavaScript and XML) – Skupina razvojnih tehnik, ki omogočajo asinhrono izmenjavo podatkov s strežnikom.

API (ang. Application Programming Interface) – Programski vmesnik, ki omogoča dostop do funkcij aplikacije iz ostalih aplikacije.

BTC Simbol valute bitcoin. V enakem pomenu, kot je EUR simbol za valuto evro.

CSS (ang. Cascading Style Sheet) – Prekrivni slogi, s katerimi je definirana oblika spletne strani.

GAE (ang. Google App Engine) – Platforma za razvoj in izvajanje spletnih aplikacij podjetja Google Inc.

GCM (ang. Google Cloud Messaging) – Storitev, ki omogoča pošiljanje podatkov s strežnika na naprave z operacijskim sistemom Android.

GPS (ang. Global positioning system) – Globalni sistem za določanja položaja je satelitski navigacijski sistem, ki ga uporabljamo za določanje položaja kjerkoli na Zemlji.

HTML (ang. HyperText Markup Language) – Označevalni jezik za oblikovanje večpredstavnostnih dokumentov.

HTTP (ang. HyperText Transfer Protocol) – Internetni komunikacijski protokol, namenjen za izmenjavo besedila in grafičnih, zvočnih ter drugih večpredstavnostnih vsebin na spletu.

IDE (ang. Integrated development enviroment) – Integrirano razvojno okolje je orodje, ki razvijalcu pomaga pri izdelavi aplikacij.

JSON (ang. JavaScript Object Notation) – Standard za izmenjavo podatkov, kjer so podatki človeku berljivi in shranjeni v obliki atribut-vrednost.

REST (ang. Representational State Transfer) – Arhitekturni stil, ki se uporablja pri razvoju spletnih storitev.

TCP (ang. Transmission Control Protocol) – Povezavni protokol transportnega sloja v protokolnem skladu TCP/IP.

URL (ang. Uniform Resource Locator) – Nabor znakov, ki predstavlja enoličen naslov vira.

XML (ang. Extensible Markup Language) – Označevalni jezik, ki omogoča izdelovanje človeku berljivih strukturiranih dokumentov.

XMPP (ang. Extensible Messaging and Presence Protocol) – Odprti komunikacijski protokol, ki temelji na XML-sporočilih in se uporablja za pošiljanje sporočil med odjemalci v realnem času.

Povzetek

Diplomsko delo zajema celoten postopek razvoja rešitve za deljenje prevoza v realnem času skupaj s podrobnim opisom implementacije prototipa.

Rešitev predvideva vzpostavitev zbirnih mest ob pomembnih cestnih vozliščih, na katerih bi se vozniki in potniki lahko srečevali ter se dogovarjali za prevoze. Vozniki bi morali med vožnjo imeti pri sebi mobilno napravo, ki je sposobna oddajati trenutno lokacijo voznika, saj s tem nudimo potnikom informacije glede oddaljenosti in predvidenega časa prihoda na zbirno mesto v realnem času. Implementiran je tudi proces prijave na vožnjo in elektronsko plačevanje stroškov vožnje preko plačilnega sistema Bitcoin.

Prototip je sestavljen iz spletne in mobilne aplikacije. Spletna aplikacija se izvaja na platformi Google App Engine, njen uporabniški vmesnik pa je bil napisan z uporabo programskega ogrodja Twitter Bootstrap in z upoštevanjem standardov HTML5. Za pridobitev in manipulacijo z lokacijskimi podatki uporabljamo programski vmesnik Google Maps. Vsak uporabnik mora imeti na svoji mobilni napravi nameščeno tudi našo mobilno aplikacijo, ki se uporablja v procesu prijave na vožnjo in nam omogoča spremljanje lokacije voznika.

V diplomskem delu so opisane tudi težave pri razvoju, možne nadgradnje in izboljšave, obstoječe rešitve ter pogled v prihodnost osebnega prevoza.

Ključne besede: deljenje prevoza v realnem času, Google App Engine, Android, Bitcoin, Google Maps

Abstract

The thesis covers the entire development process of a flexible carpooling in real time solution with a detailed description of the prototype implementation.

Solution proposes the creation of meeting places at important road junctions where drivers and passengers can meet and form carpools. During the transfer drivers must carry a mobile device capable of transmitting current driver's location, as this offers information about the distance and estimated time of arrival at the meeting place to passengers in real time. Transfer application process and electronic payment of transfer costs through the bitcoin payment system are implemented as well.

The prototype consists of web and mobile application. The web application is implemented for the Google App Engine platform; its user interface was developed with Twitter Bootstrap framework and supports the HTML5 standard. Google Maps API is used for retrieval and manipulation of location data. Users must have a mobile device with our mobile application installed, which is used during transfer application process and enables us to monitor the driver's location.

Thesis also contains the description of the difficulties encountered in development, potential upgrades and improvements to our solution and a look into the future of personal transportation.

Keywords: flexible carpooling, Google App Engine, Android, bitcoin, Google Maps

Poglavje 1

Uvod

V zadnjih desetletjih se je zaradi padca cen avtomobilov, goriva in dviga življenjskega standarda močno povečalo lastništvo osebnih avtomobilov. Iz podatkov Statističnega urada Republike Slovenije za leto 2012 je razvidno, da ima na 1000 prebivalcev kar 518 oseb svoj osebni avtomobil [1]. Kombinacija velikega lastništva avtomobilov in slabega omrežja javnega prevoza je pripeljala do tega, da se veliko ljudi v službo vozi z osebnimi avtomobili. Podatki Evropskega statističnega urada za Slovenijo kažejo, da je bilo v letu 2010 kar 86,5 odstotkov vseh prevoženih potniških kilometrov opravljenih v osebnem avtomobilu [2]. Ta odstotek nas uvršča na 3. mesto v Evropski uniji, samo za Litvo in Poljsko. Tako je bilo v Sloveniji leta 2010 z osebnimi avtomobili narejenih 25,6 tisoč milijonov potniških kilometrov. Potniški kilometer je enota za merjenje prevoza enega potnika na en kilometer. To pomeni, da če imamo v avtomobilu 2 osebi in naredimo 10 kilometrov, smo skupaj opravili 20 potniških kilometrov.

Posledica vsega je izjemno nizka povprečna zasedenost slovenskega avtomobila, ki je po podatkih za leto 2009 približno 1,4 potnika na avtomobil [3]. Če vemo, da imajo avtomobili 4 ali 5 prostih sedežev, lahko ugotovimo, da je to izredno neučinkovito. Zaradi tega je na cestah veliko več vozil, kot je potrebno, kar povzroča zastoje in večjo porabo goriva, s tem pa tudi večjo onesnaženost zraka. V nekaterih državah ta problem rešujejo s posebnimi

voznimi pasovi na avtocestah (ang. high-occupancy vehicle line). Na takih pasovih lahko v urah največjega prometa vozijo le avtomobili z več kot 2 potnikoma. V zadnjem času lahko predvsem zaradi naraščajočih cen goriva opazimo večjo pripravljenost ljudi za deljenje prevoza. Tako lahko ob vhodih na avtoceste in ob križiščih pomembnih cest v času službe opazimo parkirane avtomobile. Na teh lokacijah potniki puščajo svoje avtomobile in iščejo prevoz do svojega cilja.

Cilj diplomskega dela je predstavitev ene izmed možnih rešitev za povezovanje voznikov in potnikov z namenom povečanja števila oseb, ki se hkrati vozijo v avtomobilu. V ta namen bomo razvili prototip rešitve, ki bo z uporabo spletnih in lokacijskih tehnologij omogočala povezovanje voznikov in potnikov v realnem času.

V drugem poglavju bomo predstavili konceptualno rešitev problema in osnovne tehnologije, ki jih bomo uporabili pri implementaciji prototipa. Prav tako pa bomo pregledali trenutno obstoječe rešitve na trgu in jih primerjali s svojim predlogom rešitve. V tretjem poglavju opisujemo tehnologije, platforme in orodja, ki so bili uporabljeni pri razvoju prototipa. Sledi četrto poglavje, v katerem je predstavljen razvoj prototipa. Poglavje je dodatno razdeljeno na spletni in mobilni del aplikacije, v katerih podrobneje opisujemo zanimivejše funkcionalnosti skupaj s podrobnosti implementacije. Poglavje zaključujemo z možnostmi za nadgradnjo in izboljšavami prototipa. V zadnjem, petem poglavju bomo analizirali rešitev in jo primerjali s cilji, ki smo si jih zadali pred začetkom. Diplomsko delo pa bomo sklenili s pogledom v prihodnost osebnega prevoza.

Poglavje 2

Predlog rešitve deljenja prevoza v realnem času

Naš predlog rešitve problema je izdelava storitve, ki informacijsko podpira enkratno delitev prevoza v realnem času. Tak tip delitve prevoza se v angleščini imenuje flexible carpooling in o njem je bilo narejenih že več raziskav [4]. Pri enkratnem deljenju prevoza v realnem času gre za prevoz, ki ni urejen predčasno, ampak se potniki zbirajo na posebej določenih zbirnih mestih ter na njih iščejo prevoz do cilja.

Taka storitev mora zagotavljati:

- **Zaupanje in zanesljivost:** vzpostaviti je treba visoko stopnjo zaupanja med potniki in vozniki.
- **Korist za obe strani:** čim hitreje je treba povezati potnika z voznikom in tako minimizirati čakanje potnika.
- **Plačilo:** zagotoviti varen in hiter sistem plačila.

Rešitev predvideva postavitev zbirnih mest na pomembnih cestnih vozliščih, na katerih bi potniki iskali prevoz, vozniki pa bi se na njih ustavljali in pobirali potnike. Zbirna mesta bi morala imeti veliko parkirišče, na katerem bi bilo mogoče pustiti svoj avtomobil in uporabiti možnost deljenja

prevoza. Priporočljivo bi bilo, da so zbirna mesta povezana tudi z javnim prevozom. Idealen primer zbirnega mesta bi bilo parkirišče na Dolgem mostu v Ljubljani, saj ima na voljo veliko parkirišče, odlično lokacijo pri uvozu na avtocesto in povezavo z avtobusnim javnim prevozom. Možna pa so tudi manjša zbirna mesta, kot so nakupovalna središča, križišča pomembnih cest, velike stanovanjske soseske ... Na vsakem od večjih zbirnih mest bi bila postavljena tudi interaktivna tabla s podatki o voznikih, ki se približujejo zbirnemu mestu. Na tabli bi se prikazovali podatki, kot so končna lokacija voznika, število prostih mest, cena na kilometer in ocena voznikovega profila. Potnik se lahko na prevoz prijavi preko katerekoli naprave, ki vsebuje brskalnik s podporo standarda HTML5. Plačilo poti se izvede preko elektronskega načina plačevanja.

2.1 Uporaba storitve

Tako voznik kot potnik morata imeti pri sebi prenosno napravo, na katero je mogoče namestiti brskalnik, ki podpira standard HTML5 in ima povezljivost z internetom. Voznikova prenosna naprava mora imeti nameščeno tudi posebno aplikacijo, ki bo omogočala spremljanje lokacije voznika. Potniki take aplikacije ne potrebujejo, saj lahko potnikovo lokacijo po potrebi pridobimo z uporabo programskega vmesnika HTML5 Geolocation.

Primer uporabe: Voznik

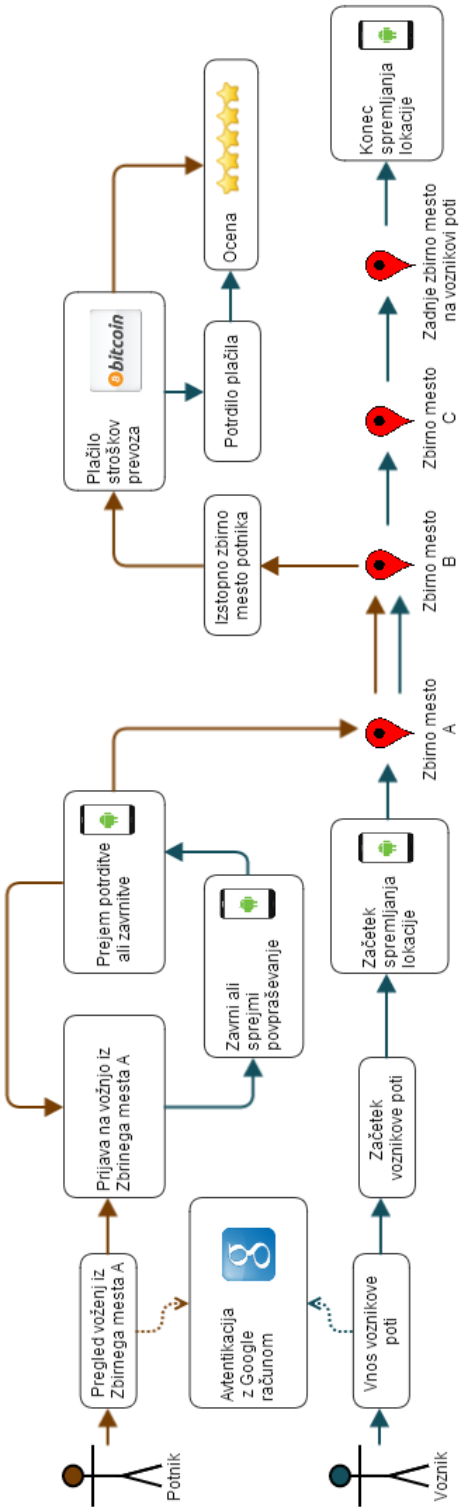
1. Voznik se mora v sistem prijaviti s svojim računom Google.
2. Ob prvi prijavi mora v sistem vnesti model, barvo in registrsko številko vozila, s katerim bo opravljal prevoz, saj ga bo le tako potnik lahko našel.
3. Voznik v sistem vnese svojo pot, za katero mora podati začetno in končno točko, vrsto računanja stroškov prevoza ter število prostih mest v avtomobilu. Vrste računanja stroškov prevoza:

- zastonj prevoz,
- strošek na kilometer za osebo,
- fiksen strošek za prevoz.

4. Pred odhodom od doma pritisne gumb odhod in vklopi spremljanje lokacije preko posebne aplikacije. Od tega trenutka dalje je mogoča rezervacija sedeža, saj se bo vožnja prikazala na interaktivnih tablah.
5. Ob prijavi potnika lahko voznik pregleda anonimiziran profil potnika in prijavo potrdi ali zavrne (na profilu vidi le oceno potnika in število voženj v sistemu).
6. Ob koncu prevoza voznik prejme plačilo potnika.
7. Potnika oceni z oceno od 1 do 5.

Primer uporabe: Potnik

1. Potniku aplikacija glede na njegovo trenutno lokacijo izpiše najbližja zbirna mesta.
2. Aplikacija mu na podlagi ocene in časovnega okvira predlaga relevantne voznike. (Lahko tudi pogleda na interaktivno tablo, če je na zbirališču.)
3. Potnik lahko pregleda profil voznika in lahko preko aplikacije pošlje zahtevo vozniku (pred tem se mora prijaviti).
4. Potnik prejme potrdilo ali zavrnitev voznika. V potrdilu so tudi podatki o vozilu, s katerimi bo potnik lahko našel vozilo, s katerim voznik opravlja prevoz.
5. Ob koncu vožnje potniki plačajo vozniku preko sistema elektronskega plačevanja, ki je del aplikacije. Strošek se samodejno izračuna na podlagi izbrane vrste računanja stroškov prevoza.
6. Ocenijo voznika z oceno od 1 do 5. Če potnik oceni voznika z oceno 1 ju sistem ne bo nikoli več povezal.



Slika 2.1: Konceptualni diagram rešitve.

2.2 Tehnične podrobnosti

Rešitev bo napisana kot spletna aplikacija z uporabo standarda **HTML5**, pri izdelavi uporabniškega vmesnika si bomo pomagali z ogrodjem za razvoj uporabniških vmesnikov **Twitter Bootstrap**, ki nam omogoča izdelavo grafičnega vmesnika, ki se samodejno prilagaja (ang. Responsive design) velikosti zaslona. Strežniški del rešitve bo gostoval na platformi **Google App Engine** in bo napisan v programskem jeziku **Python**. Za pridobivanje lokacijskih podatkov voznikov bomo razvili svojo aplikacijo, ki deluje na napravah z operacijskim sistemom Android. Pri pridobivanju podatkov o lokacijah in poteh pa si bomo pomagali s programskim vmesnikom **Google Maps**. Elektronsko plačevanje voženj bo omogočeno preko plačilnega sistema **Bitcoin**.

2.3 Razlike med začetnim predlogom rešitve in prototipom

Na začetku implementacije prototipa smo predvidevali, da bomo lokacijske podatke pridobivali preko aplikacije Google Latitude ter tako pokrili platformi Android in iOS. Toda Google se je 9. avgusta 2013 odločil ukiniti storitev Google Latitude. Zamenjava za Latitude naj bi bil Google+ Location, ampak v času implementacije prototipa storitev Google+ Location še ni popolnoma delovala. Prav tako nismo našli ustrezne zamenjave, ki bi delovala na obeh največjih mobilnih platformah in omogočala programski dostop do zbranih podatkov.

Ugotovili smo tudi, da samo s standardom HTML5 ni mogoče s strežnika pošiljati obvestil uporabniku, če ta v brskalniku nima odprte naše spletne aplikacije. Taka obvestila potrebujemo za potrditev prevoza in pošiljanje podatkov o vozilu. V specifikaciji standarda HTML5 je sicer definirana tudi funkcionalnost za pošiljanje potisnih obvestil (ang. push notifications), ampak je zelo slabo podprta v pomembnejših brskalnikih za namizne računalnike, v brskalnikih za mobilne naprave pa v času izdelave prototipa

sploh ni bila podprta. Zaradi tega smo se odločili za uporabo potisnih obvestil, ki jih ponuja operacijski sistem. Uporaba teh potisnih obvestil je trenutno omejena le na aplikacije, ki so posebej napisane za operacijski sistem (ang. native applications), in tako ne podpira pošiljanja potisnih obvestil spletnih aplikacijam. Možna bi bila uporaba posebnih aplikacij, ki jih namestimo na napravo, in preko njih omogočajo prejemanje potisnih obvestil. Druga možnost rešitve tega problema bi bilo pošiljanje obvestil preko SMS-sporočil.

Ker bi moral uporabnik za polno uporabo naše storitve na svoj telefon namestiti še dve dodatni aplikaciji (prvo za spremljanje lokacije, drugo za sprejemanje potisnih obvestil), smo se odločili, da se osredotočimo samo na operacijski sistem Android in implementiramo svojo mobilno aplikacijo, ki bo omogočala prejemanje potisnih obvestil ter spremljanje lokacije uporabnika. S to odločitvijo smo pridobili tudi možnost, da vozniku samodejno vklopimo/izklopimo spremljanje lokacije. To nam je omogočilo znebiti se dodatnega koraka ob začetku in koncu vožnje, saj se bo ob kreiranju vožnje spremljanje lokacije voznika samodejno vklopilo, ob odhodu z zadnje točke pa samodejno izklopilo.

2.4 Obstoječe rešitve

V zadnjih letih sta se zanimanje in potreba za deljenje prevoza močno povečala. Na to kaže veliko število podjetij, ustanovljenih v letih 2012 in 2013, ki nastajajo predvsem v San Franciscu in okolici. Da so taki načini prevoza zelo priljubljeni, kažejo številne pritožbe že obstoječih prevoznikov, kot so taksisti, saj nova podjetja niso licencirana za opravljanje prevozov in tako lahko ponujajo prevoze z manjšimi stroški. Po več tožbah je zvezna država Kalifornija v ta namen dovolila nov tip organizacije, tj. podjetje s transportnim omrežjem (ang. Transportation network company), ki dovoljuje ponujanje prevozov preko spletne platforme z uporabo osebnih avtomobilov. Zaradi tega nas še vedno večina ponudnikov prevoza z osebnim avtomobilom ob koncu prevoza

ne prosi za plačilo računa, ampak za donacijo ali plačilo stroškov. Skupne lastnosti novih družb so personalizacija storitve, možnost izbire, s katerim voznikom bo vožnja opravljena, ocenjevanje voznikov, spremljanje oddaljenosti voznika in avtomatsko plačevanje. Najbolj znani ponudniki storitev so Lyft, Uber [5] in Sidecar [6], v slovenskem okolju pa delno funkcionalnost omogočata storitvi Prevoz.org in GoOpti.

2.4.1 Lyft

Lyft [7] je podjetje iz San Francisca, ki ponuja prevoze v realnem času v večjih mestih v Združenih državah Amerike in je trenutno najbolj priljubljen ponudnik takih prevozov. Tako se na vožnjo naročimo preko mobilne aplikacije, ki nam pokaže najbližje voznike. Aplikacija avtomatsko predlaga primernega voznika, lahko pa preko aplikacije sami izberemo voznika. Vsi vozniki, ki sodelujejo v omrežju Lyft, morajo izpolnjevati določene pogoje in jih podjetje pred prvo vožnjo preveri. Po končani vožnji potnik plača voznika preko mobilne aplikacije. Potnik mora voznika tudi obvezno oceniti, saj v nasprotnem primeru ne more zahtevati naslednjega prevoza. Avtomobili, ki opravljajo prevoze Lyft, so prepoznavni po tem, da imajo na vidnem mestu postavljene rožnate brke.

Med storitvijo, ki jo ponuja Lyft, in našim predlogom obstajata dve večji razliki. Prva razlika je v tem, kdo lahko opravlja prevoze. Pri Lyftu mora voznik izpolnjevati posebne pogoje in mora prestati proces verifikacije, v našem predlogu pa lahko vožnjo ponuja vsakdo. Razlika je tudi med lokacijo začetka prevoza. Tako uporabniki Lyfta vozniku sporočijo, kje želijo, da se srečata. Naša rešitev pa predvideva vnaprej določena zbirna mesta. Tako Lyft deluje bolj kot nadomestek prevozov s taksijem, s svojo rešitvijo pa hočemo predvsem pokriti potrebo prevoza na delo iz bolj oddaljenih krajev.

2.4.2 Prevoz.org

V Sloveniji deluje več spletnih strani, ki delujejo kot oglasne deske za opravljanje prevozov. Najbolj poznana je spletna stran Prevoz.org [8], ki voznikom omogoča objavo vožnje in fiksne cene za prevoz. Voznik mora ob objavi vožnje pustiti telefonsko številko, na kateri je dosegljiv. Vsa nadaljnja komunikacija med voznikom in potencialnimi potniki poteka po telefonu.

Predlagana rešitev je nadgradnja storitve, ki jih ponuja Prevoz.org s samodejnim procesom prijave na prevoz, samodejnim izračunom stroškov ter možnostjo elektronskega plačevanja. Poleg tega lahko potnik spremlja oddaljenost voznika in predviden čas prihoda. Razlika pa je tudi med lokacijo, kjer se srečata voznik in potnik. Tako se trenutno preko telefona voznik dogovori s potnikom za lokacijo, v našem predlogu pa so možne lokacije vnaprej definirane.

2.4.3 GoOpti

GoOpti [9] je slovensko podjetje, ki se ukvarja z nizkocenovnimi cestnimi prevozi. Ponujajo prevoze s svojimi vozili do bližnjih letališč in mest. Delujejo podobno kot nizkocenovni letalski prevozniki, saj je mogoče preko interneta vnaprej rezervirati prevoze. Cene teh prevozov pa se spreminjajo glede na povpraševanje in čas do prevoza.

Prevozi, ki jih ponuja GoOpti, niso popolnoma primerljivi z našimi, saj večinoma ponujajo prevoze na vnaprej določenih relacijah s profesionalnimi vozniki. Naša rešitev pa je namenjena temu, da lahko vsak voznik ponudi prevoz.

Poglavje 3

Uporabljene tehnologije in orodja

3.1 Google App Engine

Google App Engine [10] (ali na kratko GAE) je platforma za razvoj in izvajanje spletnih aplikacij, ki deluje na infrastrukturi podjetja Google, in je steber Googlove ponudbe storitev računalništva v oblaku [11].

Vrste storitvenega modela računalništva v oblaku delimo na tri kategorije.

- **Infrastruktura kot storitev** (ang. Infrastructure as a service ali IaaS) je osnovni model računalništva v oblaku, kjer ponudniki v najem dajejo fizične ali virtualne računalnike, na katere lahko uporabnik namesti in zaganja poljubno programsko opremo.

- **Računalniško okolje kot storitev** (ang. Platform as a service ali PaaS) je model, kjer ponudniki ponujajo računalniško okolje, ki običajno vsebuje operacijski sistem, spletni strežnik, podatkovno bazo ter programsko izvajalno okolje. Razvijalci nato na taki platformi razvijajo in izvajajo svoje aplikacije. Primera računalniškega okolja kot storitve sta Google App Engine in Microsoft Azure.

- **Programska oprema kot storitev** (ang. Software as a service ali SaaS), kjer je na spletu celotna storitev in njeni podatki, do storitve pa dostopamo preko lahkega odjemalca. Tipična predstavnika te kategorije sta Gmail in Google Docs.

Platforma Google App Engine kot tipičen ponudnik računalniškega okolja kot storitve omogoča najem platforme, na kateri lahko izvajamo aplikacije. Glavne prednosti platforme so enostaven razvoj in vzdrževanje ter skalabilnost virov, ki jih omogoča Googlova skalabilna infrastruktura. Ob uporabi storitve GAE tako ne potrebujemo prisotnosti administratorja, ki bi vzdrževal računalniško infrastrukturo in skrbel za nemoteno delovanje sistema. Skalabilna infrastruktura nam omogoča, da se aplikacija izvaja na takem številu strežnikov, kot je to optimalno, glede na trenutno rabo aplikacije. Tako se ob povečanju uporabe aplikacije število strežnikov samodejno poveča, ob zmanjšanju pa zmanjša, kar nam omogoča optimizacijo stroškov.

Začetni razvoj na platformi GAE je brezplačen, saj si mora razvijalec le ustvariti brezplačni račun Google, s katerim pridobi možnost kreiranja 10 brezplačnih aplikacij. Vsaka aplikacija se izvaja v svojem peskovniku (ang. sandbox), kar zagotavlja izolacijo od operacijskega sistema in ostalih aplikacij. S tem omogoči zaščito aplikacije, neodvisnost od strojne opreme, operacijskega sistema in fizične lokacije spletnega strežnika. Razvijalec ob registraciji nove aplikacije brezplačno pridobi domeno oblike **imeaplikacije.appspot.com**, preko katere lahko nato uporabniki dostopajo do aplikacije. Ob registraciji aplikacije dobi razvijalec dostop do administracijskega portala, na katerem lahko spremlja delovanje in porabo virov.

V nadaljevanju bomo opisali nekaj lastnosti, najpomembnejših funkcionalnosti in dodatnih storitev, med katerimi lahko izbirate pri razvoju na platformi GAE.

3.1.1 Programski jeziki in programska izvajalna okolja

Platforma trenutno omogoča razvoj aplikacij v naslednjih programskih jezikih:

- Java in ostali programski jeziki, ki se izvajajo na navideznem javanskem stroju (JVM),
- Python,
- Go, ki je v fazi eksperimentiranja (ang. experimental), in
- PHP, ki je v fazi predogleda (ang. preview).

Vsak programski jezik ima svoje izvajalno okolje in svoj paket razvojnih knjižnic ter orodij. Ta orodja med drugim omogočajo vzpostavitev lokalnega razvojnega okolja, v katerem je mogoče testirati aplikacije za platformo GAE, in avtomatsko postavitev aplikacije na produkcijsko okolje.

Glavna programska jezika sta Java in Python, velika večina novih funkcionalnosti platforme je najprej podprta v teh jezikih in šele kasneje v ostalih. Jezika Go in PHP sta bila šele nedavno podprta in sta še v fazi razvoja. V Googlu pa so že večkrat poudarili, da nameravajo v prihodnosti podpreti še več programskih jezikov.

Pri implementaciji prototipa rešitve smo se odločili za programski jezik **Python**.

3.1.2 Shranjevanje podatkov

Platforma Google App Engine nam ponuja več načinov shranjevanja podatkov. Tako lahko uporabimo storitev **Cloud SQL**, ki nam omogoča shranjevanje podatkov v relacijski podatkovni bazi MySQL, ki je najbolj uporabljana odprtokodna podatkovna baza. Pri tem Google skrbi za upravljanje z bazo in replikacijo podatkov med različnimi strežniki. Če potrebujemo podatkovno

bazo, namenjeno za shranjevanje objektov, s katero želimo komunicirati preko programskega vmesnika REST, lahko uporabimo storitev **Cloud Storage**.

Pri razvoju prototipa pa smo se odločili za uporabo podatkovne baze **Cloud Datastore**, ki je nerelacijska podatkovna baza brez vnaprej definirane sheme. Ob tem pa z uporabo optimističnega nadzora nad sočasnim izvajanjem transakcij omogoča transakcije z ACID-lastnostmi. Optimizirana je za skalabilnost in hitrost ter zaradi zagotavljanja redundance za replikacijo podatkov med več podatkovnimi skladišči. Podpira opravljanje poizvedb s posebnim programskim jezikom GQL (Google Query Language), ki je po sintaksi zelo podoben SQL-u.

3.1.3 Dodatne storitve platforme

Platforma omogoča uporabo vrste storitev, ki jih lahko razvijalec po potrebi vključuje v svoje rešitve. Primeri podprtih storitev so Blobstore, Images, Logs, Mail, Memcache, OAuth, Prospective search, XMPP ...

V nadaljevanju bomo podrobneje opisali storitve, ki so bile uporabljene ob implementaciji prototipa.

Search

Storitev Search [12] ponuja model za indeksiranje dokumentov, ki vsebujejo strukturirane podatke. Dokumenti in indeksi so shranjeni v posebni podatkovni bazi, ki je optimizirana za izvedbo operacij iskanja. Pred uporabo storitve moramo uporabniške podatke organizirati v dokumente. Vsak uporabniški podatek mora biti shranjen v ločenem podatkovnem polju in mora imeti določen podatkovni tip. Pred začetkom iskanja moramo dokument dodati v indeks, ki je optimizirana zbirka dokumentov, na kateri lahko izvajamo poizvedbe. Največji prednosti storitve Search sta optimizacija iskanja po besedilih (ang. full-text search) in podpora kompleksnih poizvedb z lokacijskimi podatki.

Scheduled tasks

Storitev App Engine Cron Service [13] omogoča nastavljanje rednih načrtovanih opravil, ki jih lahko zaženemo ob točno določenem času ali v rednih intervalih. Ob zagonu opravila bo storitev izdala zahtevo HTTP GET na vnaprej določen URL. Naloga storitve, ki je na tem spletnem naslovu, je, da izvede izbrano opravilo. Opravilo lahko tako določimo samo z 2 parametroma. To sta:

- **urnik opravila**, ki določi, kdaj se bo opravilo izvedlo, in
- **spletni naslov opravila**, ki nam pove, kateri naslov se bo poklical ob izvedbi.

Obdelave HTTP-zahteve, ki jo izdamo ob izvedbi, lahko traja največ 10 minut.

URL Fetch

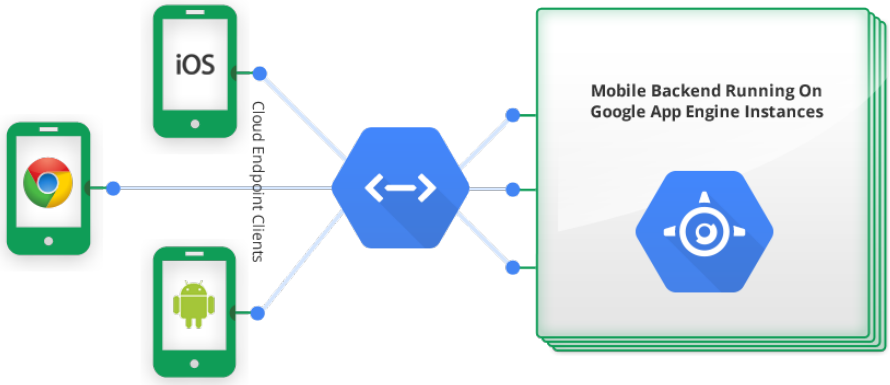
URL Fetch [14] omogoča izdajanje HTTP in HTTPS-zahtev ter prejem odgovorov na izdane zahteve. Z uporabo te storitve lahko aplikacija komunicira z drugimi aplikacijami ali pa dostopa do drugih zunanjih virov preko URL-naslova. Storitev se pogosto uporablja za komunikacijo preko programskih vmesnikov REST.

Users

Google App Engine ima vgrajene 3 načine avtentikacije uporabnika [15]. Tako se lahko avtenticiramo z Googlovimi uporabniškimi računi, uporabniškimi računi v domenah, ki uporabljajo Google Apps, in preko standarda OpenID. Ob uporabi teh načinov avtentikacije lahko aplikacija zazna, ali je uporabnik prijavljen. V primeru, ko uporabnik ni prijavljen, ga samodejno preusmeri na stran za prijavo in nato nazaj ob uspešni prijavi. Ko je uporabnik v aplikacijo prijavljen, imamo dostop do njegovih osnovnih podatkov (npr. elektronska pošta). Podatki so odvisni od izbranega načina prijave.

Google Cloud Endpoints

Google Cloud Endpoints [16] je skupek orodij in programskih knjižnic, ki omogočajo delovanje programskih vmesnikov REST, gostujočih na platformi GAE, in generiranje programskih knjižnic za odjemalce. Generirane programske knjižnice so ovojnice okoli programskih vmesnikov REST in tako olajšajo uporabo odjemalcu. Podprto je samodejno generiranje programskih knjižnic za naprave z operacijskim sistemom Android in iOS ter za odjemalce JavaScript. Zelo pomembna funkcionalnost je tudi uporaba protokola OAuth 2.0 za avtorizacijo dostopa.



Slika 3.1: Osnovna arhitektura Google Cloud Endpoints.

Postopek razvoja programskega vmesnika.

1. **Implementacija zalednega sistema**
   S pomočjo knjižnic, ki so na voljo za vse programske jezike, ki jih podpira platforma GAE, implementiramo programsko logiko zalednega dela sistema. Vsaki metodi in celotnemu vmesniku dodamo označbe, ki nam pomagajo pri generiranju programskih knjižnic za odjemalce.

2. **Testiranje programskih vmesnikov**
   S pomočjo lokalnega razvojnega okolje in Google raziskovalca program-

skih vmesnikov (ang. Google Api Explorer) lahko pregledujemo in testiramo delovanje vmesnikov.

3. **Generiranje programskih knjižnic in uporaba v odjemalcu**
   Z uporabo orodij Google Cloud Endpoints generiramo programske knjižnice za izbrano platformo. Knjižnice nato vključimo v razvoj odjemalca.

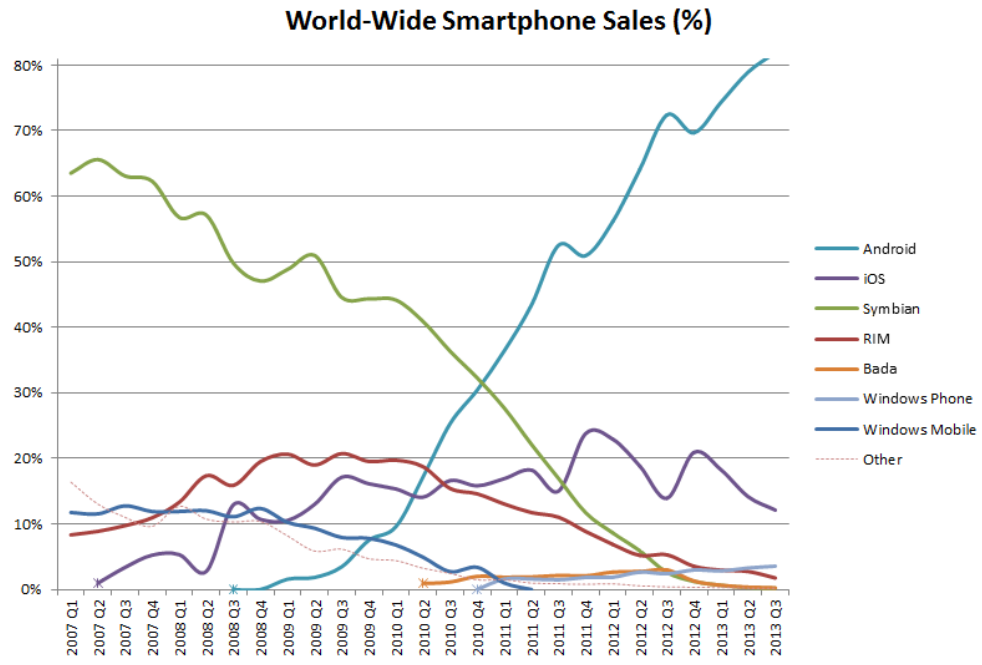
4. **Omejevanje dostopa do programskih vmesnikov**
   Če želimo omejiti dostop do programskih vmesnikov, moramo v Google konzoli generirati posebne ključe za vsakega odjemalca. Te ključe je treba dodati kot označbe v zalednem sistemu in v vsaki metodi, ki jo želimo zavarovati, preverjati, ali je klicatelj avtoriziran za dostop. Nato moramo ponovno generirati programske knjižnice za odjemalce in ob vsakem klicu programskega vmesnika dodati ključ odjemalca in klicatelja.

3.1.4 Model obračunavanja

Uporaba platforme Google App Engine se obračuna glede na porabo virov v obdobju 24 ur. Viri so razdeljeni v različne kategorije, kot so količina shranjenih podatkov, število operacij branja, velikost porabljene izhodne pasovne širine, število ur delovanja instanc spletnega strežnika ... Vsaka izmed kategorij ima podano določeno količino vira, ki je v obdobju 24 ur na voljo brezplačno. Če je bila brezplačna količina presežena, plačujemo samo porabljene vire nad brezplačno količino. Tak način obračunavanja je značilen za večino ponudnikov računalniških okolij kot storitev, kjer cena storitve raste sorazmerno s porabo. Posledica tega je za razvijalce zelo nizek vstopni prag, saj med razvojem in v primeru majhnega obiska aplikacije ni treba plačevati stroškov izvajanja aplikacije.

3.2 Android

Android [20] je operacijski sistem, zasnovan na Linuxovem jedru. Namenjen je uporabi na napravah z zasloni na dotik, kot so mobilni telefoni in tablični računalniki. Razvoj operacijskega sistema je začelo podjetje Android Inc., ki pa ga je leta 2005 kupil Google. Prva verzija operacijskega sistema je bila izdana 23. septembra 2008. Kot lahko razberemo s slike 3.2, je Android postal najbolj popularen mobilni operacijski sistem. Po podatkih, ki so bili objavljeni na razvojni konferenci Google I/O 2013 [21], je na svetu več kot 900 milijonov registriranih naprav Android in vsak dan je registriran 1 milijon novih naprav. Posebnost Androida je, da je brezplačen in odprtokoden, kar pomeni, da ga lahko vsak brezplačno uporabi ter tudi naredi svojo verzijo.



Slika 3.2: Deleži prodanih pametnih mobilnih naprav po četrtletjih [22].

Zelo pomembna lastnost Androida je možnost nalaganja aplikacij. Zato je veliko pozornosti posvečene težavnosti razvoja aplikacij, ki je zelo preprost. Večina aplikacij je razvitih v programskem jeziku Java z uporabo razvoj-

nega okolja Android SDK [23]. Dele aplikacij pa je mogoče razviti tudi v programskih jezikih C in C++.

3.3 Google Maps API

Programski vmesnik Google Maps [17] je skupek knjižnic, orodij in spletnih storitev REST za dostop do funkcij, ki jih ponujajo Google zemljevidi. Uporaba vmesnika je brezplačna z omejitvijo 25000 zahtev na dan. Osnovna verzija vmesnika je bila izdana že leta 2005 z namenom integracije zemljevidov v spletne strani. Od takrat pa se vmesnik precej razvil in povečal število funkcionalnosti. Po nekaterih raziskavah je najbolj uporabljan [18] spletni razvojni programski vmesnik. V času pisanja diplomskega dela je bil sestavljen iz 3 knjižnic programske opreme, ki so ločene glede na vrsto odjemalca.

- Google Maps Javascript, ki je trenutno v verziji 3, je namenjen uporabi v spletnih aplikacijah.
- Google Maps for Android je namenjen za uporabo na mobilnih telefonih z operacijskim sistemom Android.
- Google Maps SDK for iOS je skupek knjižnic, namenjenih za uporabo na mobilnih telefonih z operacijskim sistemom iOS.

Knjižnice programske opreme ponujajo možnost integracije in interakcije z zemljevidi. Omogočajo dodajanje različnih označevalcev lokacij (ang. marker), risanje poti in geometrijskih oblik, različnih slojev, personalizacijo zemljevidov ... Knjižnice vsebujejo tudi ovojnice spletnih storitev REST in nam tako olajšajo dostop.

Spletne storitve lahko delimo na 5 osnovnih storitev:

- **Navodila za pot (ang. directions)**
  Vmesnik je namenjen računanju poti med lokacijami. Podamo mu lahko podobne parametre kot na uporabniškem vmesniku Google zemljevidov.

- **Nadmorska višina (ang. elevation)**
  Zagotavlja podatke o nadmorski višini, tako za površino kot za morsko gladino. Če lokacije ni v podatkovni bazi, se podatki interpolirajo iz bližnjih točk.

- **Geokodiranje (ang. geocoding)**
  Geokodiranje je proces pretvorbe naslova v geografske koordinate. Podprto je tudi povratno geokodiranje, kar pomeni pretvorbo geografskih koordinat v naslov.

- **Časovni pas (ang. time zone)**
  Omogoča dostop do podatkov o časovnem pasu za določeno lokacijo.

- **Matrika razdalj (ang. distance matrix)**
  Storitev nam vrača podatke o razdaljah in potovalnih časih med podanimi lokacijami.

Ker je osrednji del naše rešitve spletna aplikacija, smo se odločili za kombinacijo uporabe knjižic Google Maps Javascript za klice iz odjemalca in programskega vmesnika REST za klice s strežnika.

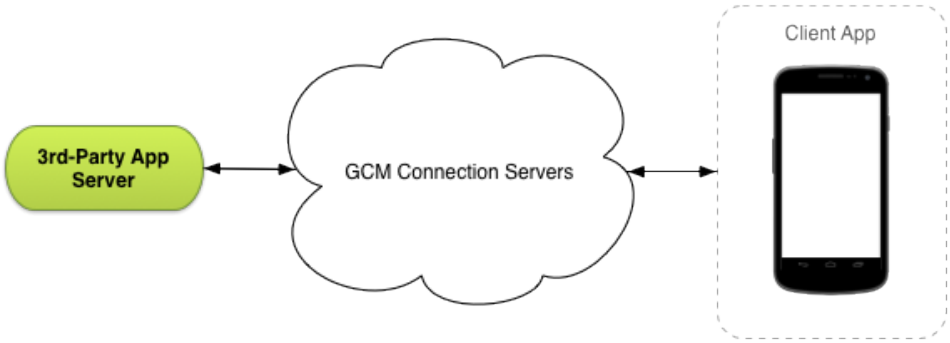
3.4 Google Cloud Messaging za Android

Google Cloud Messaging za Android [19] ali na kratko GCM je storitev, ki omogoča pošiljanje podatkov s strežnika na naprave z operacijskim sistemom Android in obratno. Storitev je bila predstavljena junija 2012 na konferenci Google I/O, in sicer kot nadomestilo za storitev Android Cloud to Device Messaging Service. Leto kasneje je bila predstavljena tudi različica storitve, ki dovoljuje pošiljanje sporočil tudi aplikacijam in razširitvam brskalnika Chrome. Za razliko od večine Googlovih storitev je GCM popolnoma brezplačna. Tako lahko brez kakršnihkoli stroškov pošljemo neomejeno število sporočil, ne glede na njihovo velikost. Trenutno se storitev večinoma uporablja za pošiljanje potisnih obvestil (ang. push notification) in za delovanje

aplikacij za klepetanje (ang. chat applications).

Lastnosti in omejitve storitve Google Cloud Messaging:

- Omogoča pošiljanje sporočil s strežnika na napravo Android.
- Z uporabo posebnega strežnika GCM Cloud Connection Server, ki temelji na XMPP-protokolu, je mogoče sporočila pošiljati tudi z naprave Android na strežnik.
- Ni treba, da ima uporabnik ob dostavi sporočila zagnano aplikacijo Android, saj jo v primeru, ko ima aplikacija vsa predpisana dovoljenja, v ozadju zažene operacijski sistem.
- Za delovanje je zahtevan operacijski sistem Android z verzijo, večjo ali enako 2.2. Naprava mora imeti nameščeno tudi aplikacijo Google Trgovina (ang. Google Play Store).



Slika 3.3: Arhitektura storitve Google Cloud Messaging.

Kot lahko vidimo na sliki 3.3, je arhitektura storitve sestavljena iz treh komponent:

1. **GCM-strežniki (GCM Connection Servers)**
   Strežniki, ki jih zagotavlja Google. Njihovi glavni nalogi sta prevzem sporočil od aplikacijskih strežnikov in njihova dostava do aplikacije An-

droid ter obratno. GCM-strežniki skrbijo tudi za čakalno vrsto sporočil in v primeru, ko je telefon izklopljen, poskrbijo, da se poročilo dostavi, ko telefon spet vklopimo. Trenutno obstajata 2 vrsti GCM-strežnikov. Najbolj preprosta je uporaba strežnikov GCM HTTP, saj v takem primeru sporočilo pošljemo kot POST-zahtevo s podatki v JSON-formatu. Največja omejitev tega pristopa je, da strežniki GCM HTTP omogočajo pošiljanje podatkov le v smeri s strežnika na napravo. Če hočemo pošiljati podatke v obe smeri, moramo uporabljati strežnike GCM CCS (Cloud Connection Server), ki med CCS-strežnikom in napravo vzpostavijo obstojno TCP-povezavo z uporabo protokola XMPP.

2. **Aplikacijski strežnik (3rd-Party App Server)**
   Komponenta, ki jo morajo implementirati razvijalci za delovanje z izbrano vrsto GCM-strežnikov. Glavni nalogi aplikacijskega strežnika sta pripravljanje sporočil in komunikacija z GCM-strežnikom.

3. **Aplikacija (Client App)**
   Aplikacija za operacijski sistem Android, ki ima vsa dovoljenja za uporabo GCM-storitve. Ob zagonu take aplikacije se moramo najprej registrirati za uporabo GCM-storitve. Ob registraciji dobimo identifikator naprave, ki je edinstven znotraj sistema GCM. Ta identifikator potem ob pošiljanju sporočil uporabimo kot naslov.

3.5 Bitcoin

Bitcoin [24] je virtualna valuta in omrežje, ki omogoča delovanje plačilnega sistema za valuto bitcoin. Je prvo plačilno omrežje, ki deluje na principu vsak z vsakim (ang. peer to peer) in tako ne vsebuje nobene centralne avtoritete ali posrednika, ki bi nadzoroval transakcije. Bitcoin je ena izmed prvih implementacij koncepta, imenovanega kripto-valuta (ang. cyrpto-currency), ki predvideva, da se za izdelavo denarja in potrjevanje transakcij uporablja kriptografija. Število uporabnikov valute močno narašča, kar je precej pove-

zano z velikim skokom v menjalnem tečaju, saj je bil januarja 2013 tečaj [25] za 1 bitcoin v bližini 14 ameriških dolarjev, januarja 2014 pa se je gibal med 800 in 900 ameriškimi dolarji. Najvišji menjalni tečaj v zgodovini obstoja valute pa je bil 29. novembra 2013, in sicer pri 1242 ameriških dolarjih za 1 bitcoin. V istem obdobju se je povprečno število transakcij na dan [26] skoraj podvojilo, saj je povečalo z 31000 na začetku leta 2013 do 60000 na začetku leta 2014.

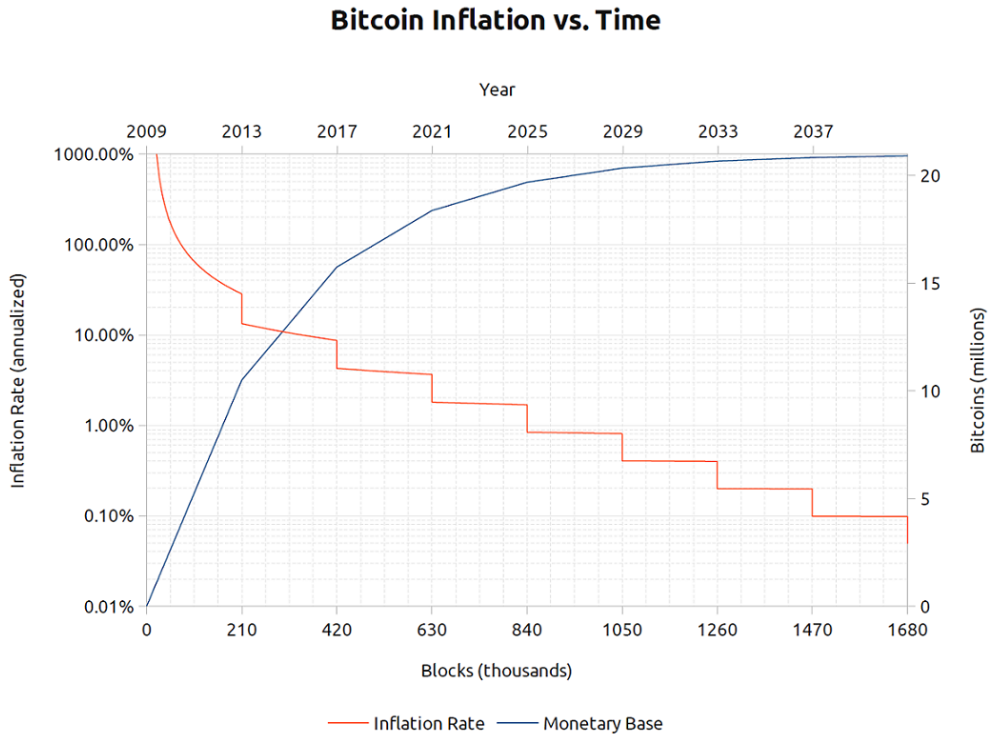
Prvo specifikacijo Bitcoin omrežja in valute ter dokaz koncepta je objavil uporabnik z imenom Satoshi Nakamoto leta 2009 na forumu za kriptografijo. Od takrat naprej število razvijalcev, ki delajo na bitcoinu, neprestano raste. Zanimivo je tudi, da je Satoshi konec leta 2010 zapustil projekt, ne da bi razkril svojo pravo identiteto. Celotna programska oprema, na kateri teče omrežje Bitcoin, je odprta, kar omogoča, da lahko vsak pregleda izvorno kodo in tudi izdela svojo verzijo protokola.

3.5.1 Plačevanje z bitcoinom

Pred plačevanjem z valuto bitcoin si mora vsak uporabnik ustvariti svojo denarnico (ang. wallet) in nanjo naložiti denar. Najlažje si denarnico ustvarimo z uporabo enega izmed številnih spletnih ponudnikov denarnic. Ob nastanku denarnice se generira Bitcoin naslov, ki je edinstven v celotnem omrežju Bitcoin in predstavlja identifikator denarnice. V trenutnem bančnem sistemu ima enako funkcijo številka transakcijskega računa. Tako ob nakazilu na drugo denarnico kot cilj vpišemo Bitcoin naslov ciljne denarnice in znesek nakazila. Ko imamo denarnico ustvarjeno, moramo nanjo naložiti denar. To lahko naredimo preko specializiranih menjalnic, ki valute, kot sta evro in ameriški dolar, menjujejo za valuto bitcoin. Ena največjih takih menjalnic je Bitstamp [27], ki je osnovana v Sloveniji in omogoča zamenjavo ameriških dolarjev v bitcoine in obratno. Ko imamo na denarnici pozitivno stanje, lahko z njo plačujemo.

3.5.2 Delovanje omrežja

V ozadju delovanja omrežje je glavna knjiga vseh transakcij z imenom veriga blokov (ang. block chain), ki si jo delijo vsa vozlišča znotraj omrežja. Glavna knjiga je sestavljena iz blokov, ki vsebujejo vse potrjene transakcije, in tako omogoča vsakemu uporabniku, da preveri veljavnost vsake od transakcij. Blok je enota, ki jo omrežje potrjuje v procesu, imenovanem rudarjenje (ang. mining), in je sestavljena iz transakcij. Rudarjenje je zelo delovno intenziven proces, ki je bil načrtovan z namenom, da bi se število potrjenih blokov na dan čim manj spreminjalo. Vsak blok mora vsebovati dokaz o delu (ang. proof of work), ki je bilo opravljano, za potrditev bloka. Uporablja se sistem hashcash [28], ki potrdilo o delu računa iz zgoščevalne funkcije.



Slika 3.4: Število izdanih bitcoinov in inflacija bitcoina skozi leta [29].

3.5.3 Izdajanje novih bitcoinov

Nove bitcoine omrežje razdeli med posameznike za opravljene storitve v omrežju. Tako bitcoin rudarji obdelujejo in potrjujejo transakcije ter zagotavljajo varnost omrežju, v zameno pa jih omrežje plačuje z novo izdanimi bitcoini.

Algoritem za izdajanje je načrtovan tako, da se novi bitcoini izdajajo v vnaprej določenem fiksnem številu. To število se vsako leto manjša in se bo popolnoma ustavilo, ko bo število vseh bitcoinov doseglo 21 milijonov (slika 3.4). Zaradi tega je postalo rudarjenje zelo konkurenčno, saj morajo rudarji za dosego dobička močno optimizirati svoje delovanje. Ob začetku delovanja valute se je izdajalo veliko bitcoinov in je bilo rentabilno tudi rudarjenje na osebnih računalnikih, sedaj pa se je to spremenilo v pravi posel, s podjetji, ki se selijo v države z najnižjo ceno električne energije in izdelujejo specializirano strojno opremo.

Ker je število izdanih bitcoinov omejeno navzgor, je v transakcije vključena tudi možnost provizije, ki bo kot plačilo za rudarjenje nadomestila izdane bitcoine.

3.6 Twitter Bootstrap

Twitter Bootstrap [30] je programersko ogrodje, ki nam olajša razvoj uporabniškega vmesnika. Vsebuje standardne predloge CSS in JavaScript, ki delujejo v brskalnikih različnih ponudnikov. V predloge pa je privzeto vključen način odzivne (ang. responsive) spletne strani, ki nam omogoča uporabo iste spletne strani na napravah z različno velikimi zasloni. Odzivnost spletne strani dosežemo s sistemom mreže (ang. grid system), kjer širino spletne strani razdelimo v 12 stolpcev. Ogrodje nato ugotovi, koliko stolpcev lahko prikaže glede na širino zaslona. Če je zaslon dovolj širok, prikaže vseh 12 stolpcev v eni vrstici, če je zaslon dovolj velik za samo 6 stolpcev, razdeli vsebino prvih 6 stolpcev v prvo vrstico, zadnjih 6 pa v drugo vrstico, itd.

3.7 Razvojna orodja

V procesu implementacije prototipa smo uporabljali orodja, opisana v nadaljevanju.

3.7.1 Android Studio

URL: `http://developer.android.com/sdk/installing/studio.html`

Android Studio je integrirano razvojno okolje (IDE), namenjeno za izdelavo aplikacij za operacijski sistem Android. IDE je osnovan na znanem integriranem razvojnem okolju za programski jezik Java IntelliJ IDEA. Razvija ga Google in je na voljo za operacijske sisteme Windows, Mac OS X in Linux. Trenutno je še v razvojni fazi (early access preview) in je na voljo brezplačno.

3.7.2 PyCharm

URL: `http://developer.android.com/sdk/installing/studio.html`

PyCharm je integrirano razvojno okolje, namenjeno za izdelavo aplikacij v programskem jeziku Python. Razvija ga češko podjetje JetBrains in je na voljo za operacijske sisteme Windows, Mac OS X in Linux. Vključuje pa podporo in avtomatsko integracijo z različnimi platformami ter programskimi ogrodji, kot sta Google App Engine in Django.

3.7.3 Postman

URL:`http://www.getpostman.com/`

Postman je HTTP-klient, ki nam omogoča pošiljanje različnih HTTP-zahtev (ang. request) in pregled odgovorov (ang. response). Izjemno olajša testiranje programskih vmesnikov REST.

3.7.4 Google App Engine SDK

URL: `https://developers.google.com/appengine/downloads`

App Engine SDK je razvojno okolje, ki pomaga pri razvoju in testiranju aplikacij, namenjenih za platformo App Engine. V SDK je vključena lokalna implementacija strežnika, skupaj s podatkovno bazo Datastore in večino dodatnih funkcionalnosti platforme App Engine. Tako je mogoče razviti aplikacijo popolnoma lokalno, brez vmesnih nalaganj na produkcijski strežnik.

3.7.5 Git

URL: `http://git-scm.com/`

Git je brezplačen in odprt porazdeljen sistem za nadzor različic (ang. distributed version control system) dokumentov. Sistem je posebno optimiziran za nadzor različic izvorne kode. Njegovi začetki segajo v leto 2005, ko so se razvijalci jedra operacijskega sistema Linux odločili, da napišejo svojo rešitev za vodenje različic kode. Tak sistem nam omogoča, da ob vsaki spremembi dokumenta različico tega dokumenta shranimo v sistem, ki nam potem omogoča spremljanje sprememb med različicami. Zelo je uporaben, če delamo v ekipi, saj lahko več ljudi dela znotraj istega dokumenta, ne da bi se med seboj motili. Na trgu obstaja veliko sistemov za nadzor različic. Najbolj znani med njimi so Git, Mercurial in Subversion.

3.7.6 BitBucket

URL: `https://bitbucket.org/`

BitBucket je storitev, ki ponuja gostovanje za projekte, ki kot sistem za nadzor različic uporabljajo Git ali Mercurial. Lastnik storitve je podjetje Atlassian, ki ponuja rešitve za projektno vodenje in podporo procesom razvoja programske opreme. BitBucket tako vključuje integracijo z več Atlassiano-

vimi rešitvami. Poleg tega je v storitev vključen tudi sistem za opravljanje pregleda kode (ang. Code review). Glavni tekmec BitBucketa na tem področju je podjetje GitHub.

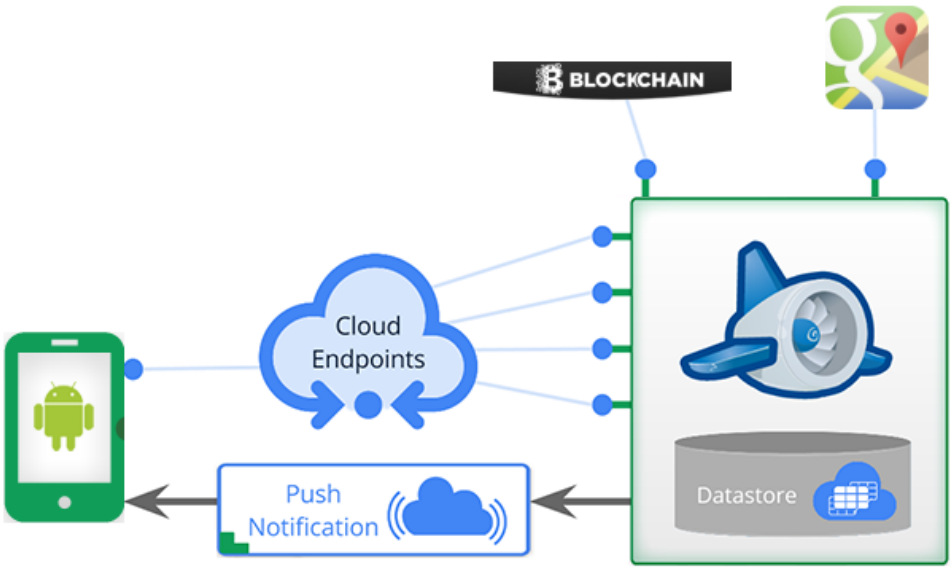
Dostop do izvorne kode prototipa je mogoč na naslovu `https://bitbucket.org/Nejchy/p2ptransfers`.

Poglavje 4

Implementacija prototipa

4.1 Arhitektura rešitve

Rešitev je sestavljena iz spletne aplikacije, napisane za platformo Google App Engine, in aplikacije, namenjene za naprave z operacijskim sistemom Android (slika 4.1).



Slika 4.1: Arhitekturni diagram rešitve.

Spletna aplikacija pri svojem delovanju uporablja 2 zunanja programska vmesnika (ang. API):

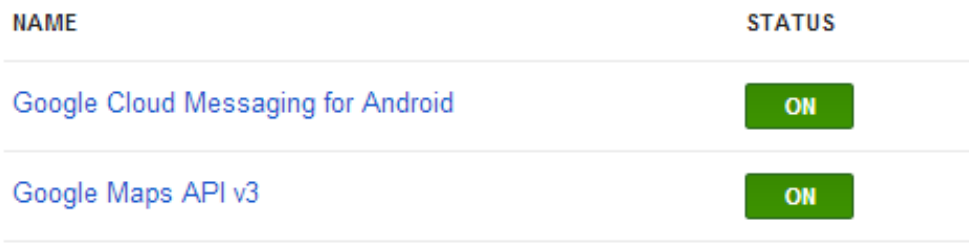
- Programski vmesnik **Google Maps**, verzija 3, s katerim si pomagamo pri prikazu zemljevidov in pri izračunu razdalj med geografskimi točkami.
- Programski vmesnik **Blockchain Recieve Payments**, ki omogoča prejemanje plačil v omrežju Bitcoin.

Komunikacija med mobilno in spletno aplikacijo poteka na 2 načina:

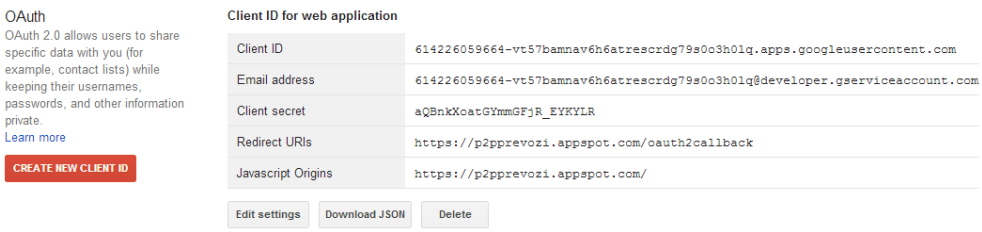
- Preko programskega vmesnika REST, ki smo ga razvili s pomočjo orodij **Google Cloud Endpoints**.
- Preko sistema **Google Cloud Messaging** je implementirano pošiljanje potisnih obvestil (ang. push notification).

4.2 Google razvojna konzola

Google razvojna konzola (ang. Google Developer Console) je osrednje mesto, na katerem lahko nadziramo dostop do vseh Googlovih programskih vmesnikov. Tukaj lahko na enem mestu vidimo število klicev do določenega vmesnika, ki jih je opravila aplikacija. Ker je za večino vmesnikov število klicev na dnevnem ravni omejeno, je tak pregled izjemno pomemben. Razvojna konzola omogoča tudi generiranje ključev in različnih identifikatorjev aplikacije, ki v Googlovem ekosistemu enolično določajo našo aplikacijo. Ti ključi se nato uporabljajo za merjenje dostopa do različnih programskih vmesnikov in za avtorizacijo. Ker smo v prototipu uporabljali več Googlovih tehnologij in programskih vmesnikov (slika 4.2), smo pred začetkom razvoja v razvojni konzoli generirali nov projekt, na katerega smo vezali aplikacijo na platformi App Engine in vse dostope do Googlovih programskih vmesnikov. To naredimo tako, da dovolimo dostop do programskih vmesnikov, ki nas zanimajo. Nato generiramo ključe (slika 4.3), potrebne za identifikacijo naše aplikacije, ki jih ob vsakem klicu vmesnika podamo kot dodatni parameter.



Slika 4.2: Googlovi programski vmesniki, ki jih uporablja aplikacija.



Slika 4.3: Primer ključa, ki je generiran z namenom avtentikacijo preko protokola OAuth 2.0 za dostop do programskega vmesnika REST.

4.3 Spletna aplikacija

Spletna aplikacija je osrednji del naše rešitve, saj vsebuje podatkovno bazo, celotno poslovno logiko, ki skrbi za upravljanje z lokacijami in zbirnimi mesti ter generiranje spletnega uporabniškega vmesnika. Napisana je za platformo Google App Engine (v nadaljevanju GAE) v programskem jeziku Python, verzije 2.7. Pri implementaciji smo se držali programerskega arhitekturnega vzorca (ang. software design pattern) model – pogled – kontroler (ang. model – view – controller), ki je priporočen pri pisanju spletnih aplikacij za platformo GAE. Spletna aplikacija je dostopna na spletnem naslovu `http://p2pprevozi.appspot.com/`.

Shranjevanje podatkov

Za shranjevanje uporabljamo podatkovno bazo Datastore, ki je del GAE-platforme in je objektna podatkovna baza brez vnaprej definirane sheme. Podatki o zbirnih mestih pa so shranjeni ločeno v posebni dokumentni podatkovni bazi, ki je optimizirana za hitro iskanje preko programskega vmesnika Search API. Za ta način shranjevanja smo se odločili, ker je Search API optimiziran za iskanje po lokacijskih podatkih. Podpira tudi poizvedbe tipa - Želim vsa zbirna mesta, ki so v radiju 1000 metrov od podane geografske koordinate.

Uporabniški vmesnik

Uporabniški vmesnik je bil razvit z uporabo standardov HTML5 in CSS3. Pomagali smo si tudi s programskim ogrodjem za razvoj uporabniških vmesnikov Twitter Bootstrap, kar nam omogoča odziven uporabniški vmesnik, ki se samodejno prilagaja velikost zaslona naprave. Uporabljali smo tudi programski jezik Jinja2 [31], ki se v predlogah za generiranje spletne strani uporablja za večjo berljivost in lažje generiranje HTML-ja.

Prijava v aplikacijo

Preden želi uporabnik dostopati do naprednejših funkcionalnosti spletne aplikacije, se mora prijaviti. Brez prijave je mogoč dostop le do prikaza najbližjih zbirnih mest in do seznama voženj na vsakem zbirnem mestu. Ker nismo želeli, da bi morali uporabniki kreirati nov račun, ki se uporablja samo za našo aplikacijo, smo omogočili prijavo z vsemi vrstami računov Google. Tako aplikacija ne shranjuje nobenih gesel, o uporabniku pa imamo shranjeno samo naslov njegove elektronske pošte, ki ga potrebujemo za komunikacijo z mobilno aplikacijo. Tako mora uporabnik, ki je že prijavljen v svojo elektronsko pošto Google ob prijavi v spletno aplikacijo samo dovoliti dostop do podatkov svojega računa Google.

V nadaljevanju bomo opisali nekaj najbolj zanimivih funkcionalnosti spletne aplikacije.

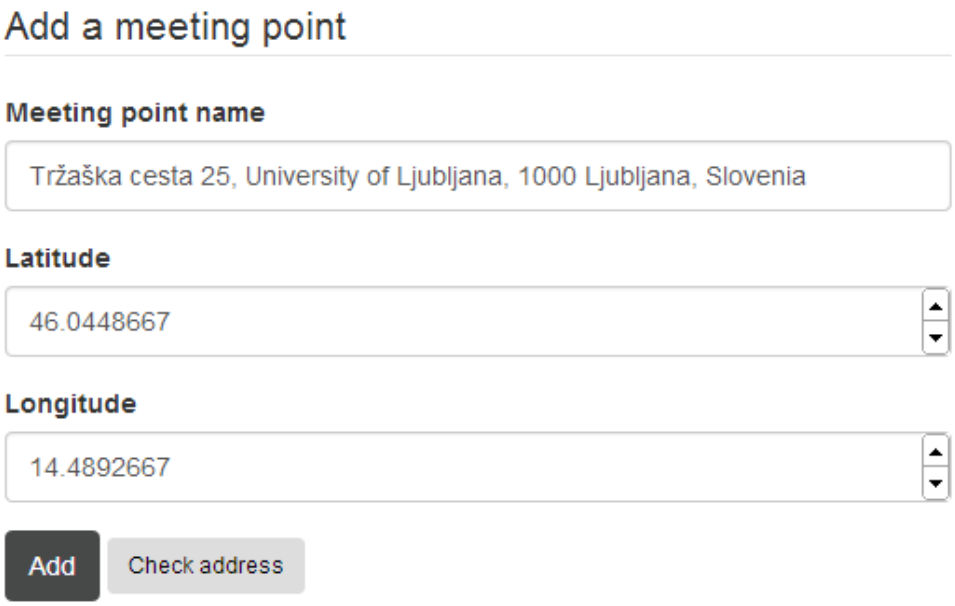
4.3.1 Dodajanje zbirnih mest

Prijavljeni uporabniki imajo možnost dodajanja nove lokacije, za katero želijo, da postane zbirno mesto. Za lažje dodajanje najprej poskušamo pridobiti koordinate trenutne lokacije uporabnika in ime/naslov trenutne lokacije. Podatke o koordinatah poskušamo pridobiti s pomočjo programskega vmesnika HTML5 Geolocation, ime/naslov lokacije pa s programskim vmesnikom Google Geocoding.

HTML5 Geolocation

Programski vmesnik HTML5 Geolocation je eden izmed najbolj podprtih delov HTML5-specifikacije, saj ga podpirajo vsi večji ponudniki brskalnikov, tako na namiznih računalnikih kot na mobilnih napravah.

Od vsakega brskalnika in naprave, na kateri deluje, je odvisno, kako bo pridobil podatke o lokaciji in kako natančni so ti podatki. Tako se podatki o lokaciji z brskalnika na namiznem računalniku običajno pridobivajo



Slika 4.4: Primer dodajanja zbirnega mesta.

```
function initialize(){
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(success);
  } else {
    alert("Geolocation is not supported by this browser.");
  }
}

function success(position){
  alert("Latitude: " + position.coords.latitude +  "
     Longitude: " + position.coords.longitude);
}
```

Izvorna koda 4.1: Primer pridobitve lokacije s programskim vmesnikom HTML5 Geolocation.

s pomočjo IP-naslova in so lahko precej nenatančni. Podatki o lokaciji z mobilnih naprav pa so navadno precej bolj natančni, saj si lahko pri pridobivanju lokacije pomagajo z omrežnimi stolpi in z GPS-om. Ker specifikacija ne predpisuje, kako naj odjemalci pridobijo lokacijo, se lahko zgodi, da se trenutna lokacija na različnih brskalnikih na isti napravi močno razlikuje.

Specifikacija tudi predpisuje, da spletna aplikacija ne sme brez eksplicitnega dovoljenja uporabnika pridobiti trenutne lokacije, zato je ob prvem obisku treba dovoliti aplikaciji dostop do lokacije. Če uporabnik aplikaciji ne dovoli dostopa do trenutne lokacije, vse funkcionalnosti, ki so vezane na trenutno lokacijo uporabnika, ne bodo delovale.

Google Geocoding

Spletna storitev Google Geocoding je del storitev, ki jih Google ponuja s programskim vmesnikom Google Maps in omogoča pretvarjanje med geografskimi koordinatami ter imeni/naslovi lokacij in obratno.

Do storitve dostopamo z zahtevo HTTP GET na spodnji spletni naslov s podanimi ustreznimi parametri. Kot rezultat klica dobimo odgovor v obliki JSON ali XML. Google nudi tudi JavaScript knjižnico, ki olajša klicanje spletne storitve z brskalnika.

```
https://maps.googleapis.com/maps/api/geocode/output?parameters
```

Primer uporabe spletne storitve lahko vidimo v izseku izvorne kode 4.2 in 4.3.

4.3.2 Dodajanje voznikove poti

Voznik mora pred začetkom vožnje izbrati pot, po kateri bo prevažal, in zbirna mesta, na katerih je pripravljen pobirati potnike.

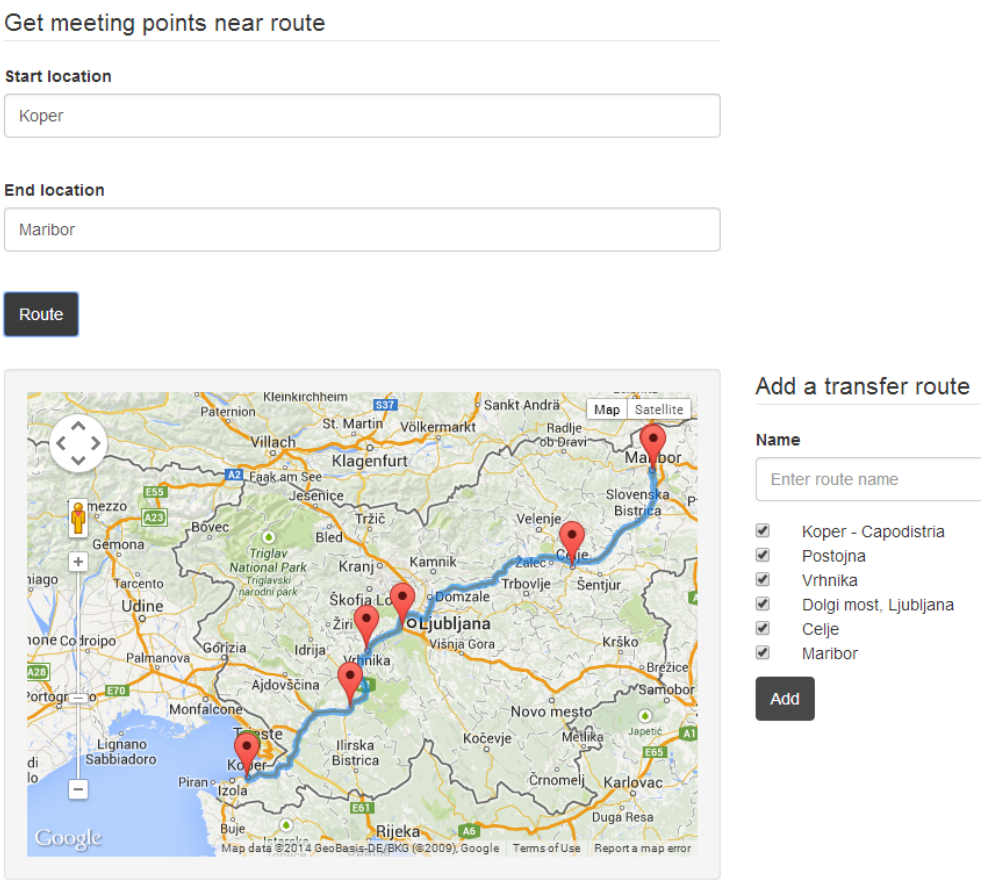
Uporabnik najprej izbere začetno in končno lokacijo svoje vožnje. Aplikacija mu na podlagi teh podatkov predlaga vsa vmesna zbirna mesta, ki so

```
function getAddress() {
 var geocoder = new google.maps.Geocoder();
 var latlng = new google.maps.LatLng(46.0448667, 14.489266);
 geocoder.geocode({'latLng': latlng}, function(results,
     status) {;
  if (status == google.maps.GeocoderStatus.OK) {
   if (results[0]) {
    alert(results[0].formatted_address);
    }
   }
  }
 });
}
```

Izvorna koda 4.2: Primer klica storitve Google Geolocation za lokacijo Fakultete za računalništvo in informatiko v Ljubljani.

```
"formatted_address" : "Trzaska cesta 25, University of
   Ljubljana, 1000 Ljubljana, Slovenia",
 "geometry" : {
  "location" : {
   "lat" : 46.0448994,
   "lng" : 14.4892307
         },
          "location\_type" : "ROOFTOP",
```

Izvorna koda 4.3: Izsek iz odgovora v formatu JSON.



Slika 4.5: Prikaz uporabniškega vmesika za dodajanje voznikove poti.

v radiju 2 km od najhitrejše poti med začetno in končno lokacijo. Zatem uporabnik izbere zbirna mesta, na katerih se je pripravljen ustaviti.

Google Directions

Spletna storitev Google Directions je del programskega vmesnika Google Maps in nam omogoča pridobitev podatkov o prometnih povezavah med podanimi točkami. Parametri, ki jih določamo, so enaki tistim, ki jih lahko izbiramo preko grafičnega vmesnika Googlovih zemljevidov. Mogoče je izbirati med tipi prevoza (npr. avtomobil, javni promet, kolo ...), različnimi omejitvami poti (npr. brez plačljivih cest, brez avtocest) ... V sklopu pro-

totipa so za te parametre nastavljene privzete vrednosti. Za tip vozila je izbran avtomobil, vse ostale omejitve pa so odstranjene. Tako bo rezultat vedno najhitrejša pot z avtomobilom med začetno in končno lokacijo.

Do storitve dostopamo z zahtevo HTTP GET na spodnji spletni naslov s podanimi ustreznimi parametri. Kot rezultat dobimo odgovor v obliki JSON ali XML. Google nudi tudi JavaScript knjižnico, ki olajša klicanje spletne storitve z brskalnika.

```
https://maps.googleapis.com/maps/api/directions/output?parameters
```

Primer rezultata ob klicu spletne storitve Google Directions za pot med Mariborom in Koprom lahko vidimo v izseku izvorne kode 4.4.

Določitev zbirnih mest v bližini poti

Za določitev zbirnih mest v bližini poti si pomagamo z JavaScript razredom RouteBoxer, ki je del knjižnice Google Maps Utility [32]. Namen razreda je razdelitev poti na območja v obliki pravokotnika, ki so od poti oddaljena za izbrano razdaljo. Razredu kot parameter podamo odseke poti in zahtevan odmik od poti (v našem primeru 2 km), kot rezultat pa vrne seznam območij, ki se čim bolj natančno prilegajo poti.

Postopek določanja območij, ki ga implementira RouteBoxer:

1. Na zemljevid položi mrežo, ki pokriva celotno območje med začetno in končno lokacijo. Vsaka celica v mreži je široka za podan odmik od poti.

2. Označijo se vse celice mreže, preko katerih vodi pot.

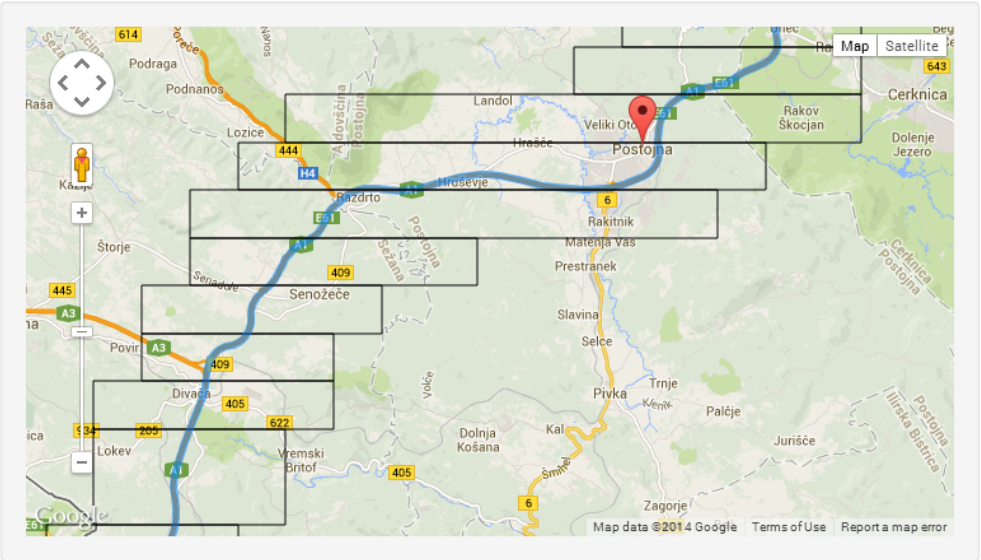
3. Za vsako označeno celico se označi vseh 8 sosednjih celic, kar zagotavlja, da so označena vsa območja znotraj odmika od poti.

4. Vse označene celice poskuša združiti v minimalno število pravokotnih območij. To poskuša doseči na 2 načina. Pri prvem celice združimo

```
"legs" : [
     {
        "distance" : {
           "text" : "233 km",
           "value" : 232657
        },
        "duration" : {
           "text" : "2 hours 16 mins",
           "value" : 8182
        },
        "end_address" : "Maribor, Slovenia",
        "end_location" : {
           "lat" : 46.554659,
           "lng" : 15.6458481
        },
        "start_address" : "6000 Koper, Slovenia",
        "start_location" : {
           "lat" : 45.54806079999999,
           "lng" : 13.7301879
        },
```

Izvorna koda 4.4: Izsek iz rezultata v formatu JSON za pot med Mariborom in Koprom.

najprej vertikalno, nato pa še horizontalno, v drugem pa najprej horizontalno, nato pa vertikalno. Izberemo postopek, ki kreira najmanj pravokotnih območij.



Slika 4.6: Primeri območij, ki nam jih vrne RouteBoxer, s podanim odmikom 2 km.

Območja nato z AJAX-klicem pošljemo na strežnik, kjer za vsako območje preverimo, ali obstajajo zbirna mesta znotraj območja (izsek izvorne kode 4.5). Za izvajanje poizvedb po zbirnih mestih uporabljamo programski vmesnik Search API, ki je del platforme Google App Engine. Zbirna mesta nato vrnemo klientu in jih narišemo na zemljevidu, kot je to razvidno na izseku izvorne kode 4.6.

4.3.3 Dodajanje voženj

Voznik ponudi svojo storitev prevoza tako, da ustvari novo vožnjo. Pri tem mora izbrati vozilo, s katerim bo opravljal prevoz, pot, na kateri je pripravljen pobirati potnike, ter ceno prevoza v valuti bitcoin. Na izbiro ima 3

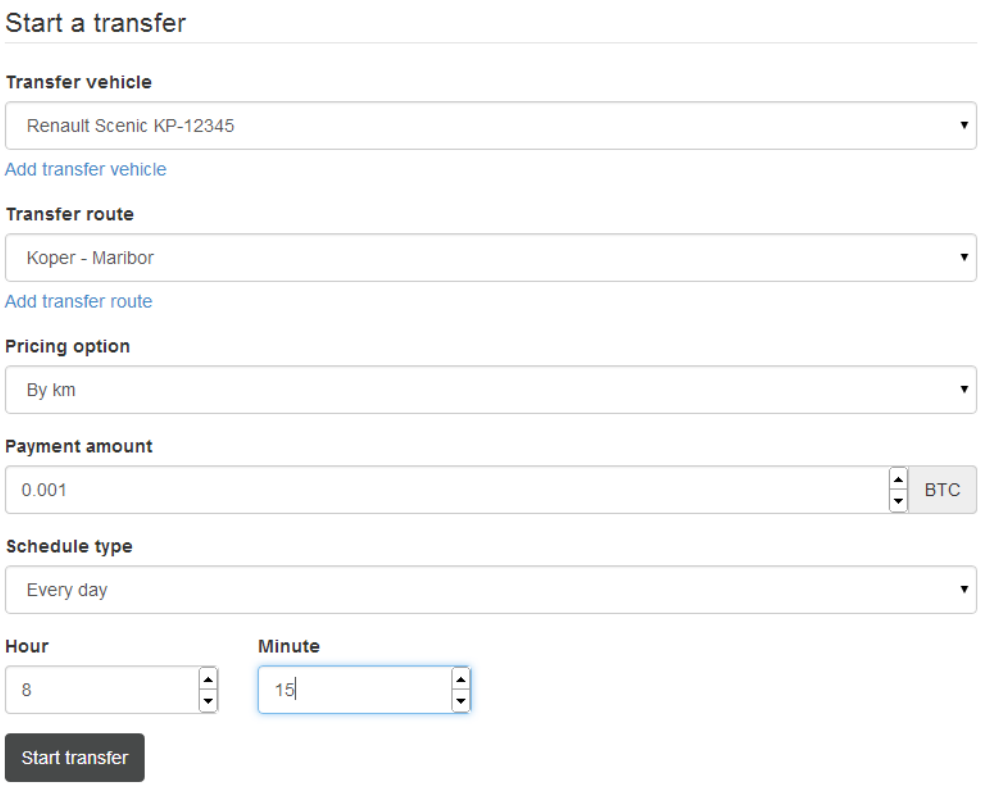
```
def GetDocumentInsideBounds(upper_left_point,
    down_right_point):
     try:
          query = _FIELD_LATITUDE + ' <= ' + str(
              upper_left_point.latitude) + ' AND ' + \
                   _FIELD_LATITUDE + ' >= ' + str(
                       down_right_point.latitude) + ' AND ' + \
                   _FIELD_LONGITUDE + ' <= ' + str(
                       upper_left_point.longitude) + ' AND ' + \
                   _FIELD_LONGITUDE + ' >= ' + str(
                       down_right_point.longitude)
          query_obj = search.Query(query_string=query)
          return search.Index(name=_INDEX_NAME).search(query=
              query_obj)
     except:
          logging.exception("search exception")
          return [CreateDocument('test1', 45.775864, 14.213661)
              ]
```

Izvorna koda 4.5: Primer klica, ki nam vrne zbirna mesta znotraj območja podanega zgornjo levo geografsko koordinato in desno spodnjo geografsko koordinato.

```
function addPointsToMap(center_latitude, center_longitude,
    points){
 var options = {
  zoom: 8,
  center: new google.maps.LatLng(center_latitude,
      center_longitude),
  mapTypeControl: false,
  navigationControlOptions:
  {style: google.maps.NavigationControlStyle.SMALL},
  mapTypeId: google.maps.MapTypeId.ROADMAP
 };

 var map = new google.maps.Map(document.getElementById("map-
     canvas"), options);

 for(var i=0; i < points.length; i++){
  var marker = new google.maps.Marker({
   position: new google.maps.LatLng(points[i].lat, points[i].
       lon),
   map: map,
   title: points[i].title
  });
 }
}
```

Izvorna koda 4.6: Označevanje lokacij zbirnih točk na Google zemljevidu.

načine izračuna stroškov, in sicer zastonj prevoz, strošek za prevožen kilometer in fiksni strošek za celoten prevoz. V trenutku, ko je prevoz ustvarjen, se šteje kot začet in potencialni potniki se lahko nanj prijavijo. Vozniki, ki pogosto opravljajo isto pot (npr. vsak delovni dan v službo), lahko nastavljajo ponavljajoče se prevoze. Izbirajo lahko med ponavljanjem prevoza vsak dan, vsak delovni dan in vsak dan med koncem tedna. Poleg tega morajo vpisati tudi uro začetka prevoza. Če želijo vozniki zaračunavati za svoje prevoze, morajo v profil vpisati svoj Bitcoin naslov.



Slika 4.7: Prikaz uporabniškega vmesnika za dodajanje novega prevoza.

Načrtovana opravila na platformi Google App Engine

Platforma GAE za izvajanje načrtovanih opravil (ang. Scheduled tasks) ponuja posebno funkcionalnost App Engine Cron Service. To funkcionalnost

smo tudi uporabili pri implementaciji ponavljajočih se prevozov.

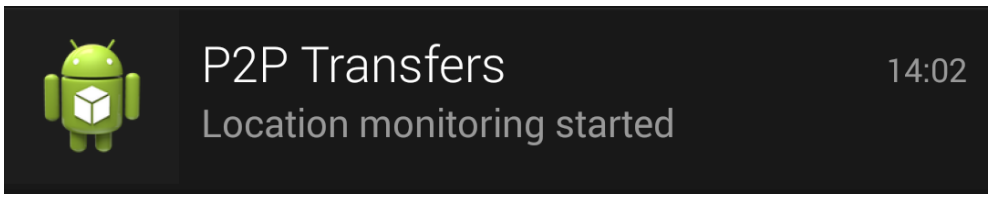
```
cron:
- description: Start scheduled transfers
  url: /cron/start_transfers
  schedule: every 10 minutes
```

Izvorna koda 4.7: Nastavitve načrtovanega opravila za kreiranje ponavljajočih prevozov.

App Engine Cron Service zagotavlja, da bo ob vsakem načrtovanem opravilu (v našem primeru vsakih 10 minut) izvedel zahtevo HTTP GET na podani spletni naslov. Ob klicu na naslov ugotovimo, katere prevoze je treba začeti in jih tudi začnemo.

Začetek sledenja lokacije vozila

Vsak voznik, ki nudi prevoz, mora imeti med vožnjo s seboj napravo z operacijskim sistemom Android s pripadajočo mobilno aplikacijo, saj nam mora periodično pošiljati svojo trenutno lokacijo. Voznik ima v aplikaciji Android možnost ročno vklapljati in izklapljati spremljanje lokacije, ampak ker mu nismo želeli povzročati dodatnega dela, se vozniku ob kreiranju vožnje (ne glede ali je vožnja kreirana ročno ali samodejno) preko sistema Google Cloud Messaging (v nadaljevanju GCM) pošlje potisno obvestilo (slika 4.8), ki samodejno sproži sledenje lokacije. Vozniku pošljemo potisno obvestilo tudi, ko zapusti zadnje zbirno mesto, in tako avtomatsko izklopimo sledenje lokaciji.



Slika 4.8: Potisno obvestilo, ki ga na napravi Android prejme voznik.

```
def SendStartLocationMonitoringToCloudMessaging(
    application_user):
    json_dict = {'registration_ids': [application_user.
        device_registration_id],
                 'data': {'type': 'START_MONITORING'}}
    result = urlfetch.fetch(url=GetGcmUrl(),
                            payload=json.dumps(json_dict),
                            method=urlfetch.POST,
                            headers={'Content-Type': '
                                application/json', '
                                Authorization': 'key=' +
                                GetApiKey()})

    if result.status_code == 200:
        return True
    return False
```

Izvorna koda 4.8: Metoda s katero pošljemo potisno obvestilo preko sistema GCM.

4.3.4 Pregled trenutnih voženj na zbirnem mestu in prijava na vožnjo

Za vsako zbirno mesto lahko pogledamo seznam vseh voženj oz. voznikov, ki se nameravajo na izbranem zbirnem mestu ustaviti in pobirati potnike.

Arriving at meeting point **Vrhnika**

#	Final destination	Rating	Pricing method	Price in BTC	Available seats	Distance	ETA	Request a transfer
1	Maribor	★★★☆☆	By km	0.01	3	55.4 km	37 mins	Request
2	Ljubljana	☆☆☆☆☆	Free	0.0	2	87.6 km	53 mins	Request

Slika 4.9: Primer seznama trenutnih voženj na zbirnem mestu Vrhnika.

S seznama lahko razberemo:

- **zadnje zbirno mesto**, na katerem se bo voznik ustavil,
- **oceno** voznika, ki je izračunana kot povprečje prejetih ocen,
- **način izračuna stroška** in **strošek v valuti bitcoin** glede na način izračuna; na podlagi teh podatkov lahko potnik izračuna strošek prevoza,
- **število prostih sedežev** v vozilu, s katerim opravljamo prevoz,
- **oddaljenost** in **predviden čas prihoda** na izbrano zbirno mesto.

Vožnje so urejene po oddaljenosti od zbirnega mesta, saj lahko tako uporabniki hitro ugotovijo, kateri voznik bo naslednji prispel na zbirno mesto. Ko je uporabnik prijavljen v spletno aplikacijo, ima možnost prijave na vožnjo. Ob prijavi na vožnjo se vozniku preko sistema GCM pošlje potisno obvestilo, na katerega mora odgovoriti in sprejeti ali zavrniti vožnjo. Da bi izboljšali zaupanje med potniki in vozniki, lahko prijavljenemu potniku

priporočimo, katerega voznika naj izbere. Zaradi tega prijavljenemu potniku ne omogočamo prijave na vožnje voznika, ki ga je v preteklosti ocenil z 1.

Izračun razdalje in predvidenega čas prihoda

Za izračun podatkov za vsako vožnjo oz. voznika, ki se namerava ustaviti na izbranem zbirnem mestu, uporabljamo programski vmesnik Google Directions, kjer kot parametre podamo zadnjo lokacijo voznika, ki jo pridobimo iz mobilne aplikacije, in lokacijo zbirnega mesta. Iz rezultata nato preberemo dolžino najkrajše poti z avtomobilom in predviden čas vožnje v idealnih razmerah med obema lokacijama.

4.3.5 Plačevanje s sistemom Bitcoin

Ob zaključku vožnje potnik pred izstopom iz vozila začne postopek plačevanja. Takrat se pridobi trenutna lokacija potnika in s programskim vmesnikom Google Directions izračunamo razdaljo med vstopnim zbirnim mestom potnika in izstopno lokacijo. Če je izbrani način izračuna stroškov na kilometer, se ob tem izračuna tudi končni strošek prevoza.

V sistem je integrirano tudi elektronsko plačevanje stroškov prevoza preko plačilnega sistema Bitcoin. Plačevanje je implementirano s pomočjo programskega vmesnika za sprejemanje plačil ponudnika **Blockchain.info** [33]. Programski vmesnik je brezplačen in zelo preprost, saj sta za sprejetje plačila dovolj dve zahtevi HTTP GET. Poleg tega pa omogoča neposredno plačevanje voznikom, ne da bi ostalim uporabnikom razkrili voznikov Bitcoin naslov. Slabost uporabe tega programskega vmesnika pa je, da je znesek minimalne transakcije 0.0005, kar bo zaradi naraščajoče vrednosti bitcoina v primerjavi z ostalimi valutami mogoče predstavljalo težave v prihodnosti.

1. **Generiranje naslova za sprejem plačila**
   Z GET-klicem na spodnji URL generiramo edinstven Bitcoin naslov, ki ga nato prikažemo potniku. Ko potnik izvede nakazilo na naslov, se celoten znesek prenese na **$ciljni_naslov** in Blockchain.info poskrbi, da

nas bo o transakciji obvestil s klicem na **$povratni_url**. Kot odgovor dobimo podatke o generiranem naslovu in proviziji v JSON-formatu.

```
https://blockchain.info/api/receive?method=create&address=$
ciljni_naslov&callback=$povratni_url
```

- $ciljni_naslov – Bitcoin naslov, na katerega želimo, da se znesek nakaže. V našem primeru Bitcoin naslov voznika.
- $povratni_url – URL-naslov, na katerega želimo, da se izvede povratni klic v trenutku, ko je bilo plačilo prejeto. Naslovu je priporočljivo dodati tudi dodatne parametre, s katerimi lahko identificiramo transakcijo.

2. **Potrditev plačila**
   Ko je potnik izvedel nakazilo, Blockchain.info izvede klic na povratni URL-naslov. Pomembnejši parametri, ki jih pri klicu dobimo, so predstavljeni spodaj.

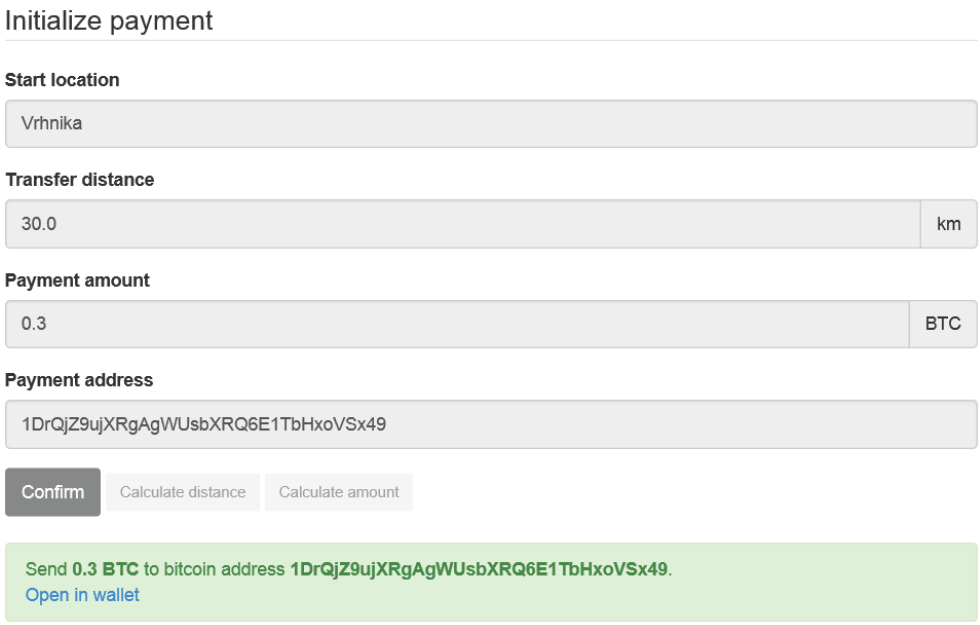
   - **Value** – Vrednost plačila v satoshijih. Satoshi je najmanjši znesek, s katerim lahko plačujemo v omrežju Bitcoin (pri plačilu z evri je najmanjši znesek, ki ga lahko plačamo 1 cent). Trenutno velja 1 satoshi = 0.00000001 BTC.
   - **Confirmations** – Število ločenih potrditev transakcije v omrežju. Da se izognemo prevaram, trenutno velja, da je transakcija potrjena in varna, če ima vsaj 6 potrditev.
   - **Dodatni parametri** – Dodatni parametri, ki smo jih podali ob generiranju naslova in se uporabljajo za identifikacijo transakcij.
   - **Transaction_hash** in **Input_transaction_hash** – Zgoščeni vrednosti obeh transakcij.

```
driver_user = ApplicationUser.GetApplicationByUserID(transfer
    .application_user_id)
callback_url = urllib.quote_plus(
  self.request.host_url + webapp2.uri_for('PaymentCallback',
     payment=entity_key.id()))
address = driver_user.bitcoin_address

url = "https://blockchain.info/api/receive?method=create&
    address={0}&callback={1}"
url = url.format(address, callback_url)

result = urlfetch.fetch(url, method=urlfetch.GET)

if result.status_code == 200:
  payment_data = json.loads(result.content)
  payment = Payment.GetPaymentByID(long(entity_key.id()))
  payment.receive_address = payment_data['input_address']
  payment.put()
```

Izvorna koda 4.9: Primer generiranja naslova za sprejem plačila s programskim vmesnikom Blockchain.

- **Input_address** in **Destination_address** – Oba Bitcoin naslova, ki sta vključena v transakcijo.

Obvestila o izvedenih transakcijah dobivamo, dokler na obvestilo ne odgovorimo z besedilom *ok*.



Slika 4.10: Primer plačila preko plačilnega sistema Bitcoin.

Bitcoin URI-shema

V omrežju Bitcoin ima vsak uporabnik svojo Bitcoin denarnico. Ker sta glavni značilnosti Bitcoina njegova porazdeljenost in odsotnost centralnega upravljanja, je nastalo veliko različnih ponudnikov denarnic, ki med seboj niso povezane. Trenutno ne obstaja sistem, po katerem bi lahko pri plačilu samo vpisal uporabniško ime in geslo ter plačal iz katerekoli denarnice. Ponudniki plačevanja z valuto bitcoin imajo običajno implementirano integracijo z nekaj denarnicami, vsi ostali pa se morajo ločeno prijaviti v svojo denarnico in nakazati denar na podani naslov. Da bi vsaj delno obšli te omejitve, ki so

do uporabnika zelo neprijazne, je bila uvedena Bitcoin URI-shema, na podlagi katere lahko s klikom zaženemo lokalno nameščenega Bitcoin odjemalca. Ob zagonu imamo že izpolnjene potrebne podatke za transakcijo.

Sintaksa za Bitcoin URI-shemo

```
bitcoin:<address>[?amount=<amount>][?label=<label>][?message
   =<message>]
```

Tako, če želimo na naslov 175tWpb8K1S7NmH4Zx6rewF9WQrcZv245W nakazati 20.3 BTC

```
bitcoin:175tWpb8K1S7NmH4Zx6rewF9WQrcZv245W?amount=20.3
```

4.3.6 Medsebojno ocenjevanje uporabnikov

Da bi sistem naredili čim bolj uporabniku prijazen in povečali zaupanje med vozniki ter potniki, imajo vsi uporabniki možnost ocenjevati ostale uporabnike, s katerimi so bili v stiku. Tako lahko vozniki ocenijo potnika in potniki ocenijo voznike z oceno med 1 in 5. Te ocene so nato prikazane pri vseh korakih izbire prevoza in uporabnikom pomagajo pri izbiri.

4.3.7 Programski vmesnik REST

Za potrebe pošiljanja podatkov z mobilnih naprav na strežnik smo razvili programski vmesnik REST, ki za avtentikacijo uporablja protokol OAuth 2.0. Vmesnik je bil razvit s pomočjo orodij Google Cloud Endpoints. Vanj so vključene 4 metode:

- **ping** – uporablja se za preverjanje delovanja vmesnika in povezave med klientom ter strežnikom,
- **save_location_data** – uporablja se za shranjevanje lokacijskih podatkov voznika,

- **save_android_device_registration_id** – uporablja se za shranjevanje identifikatorja naprave Android, s katerim si pomagamo pri pošiljanju potisnih obvestil,
- **respond_to_user_request** – metoda se uporablja pri odgovoru voznika na prijavo potnika na vožnjo.

Pri definiranju programskega vmesnika za uporabo v spletnih in mobilnih aplikacijah moramo v Google konzoli generirate ključe, ki se bodo uporabljali pri avtorizaciji, ime in verzijo programskega vmesnika ter kakšne podatke pričakujemo od uporabnika. V našem primeru je to EMAIL_SCOPE, kar pomeni, da dobimo dostop do naslova uporabnikove elektronske pošte.

```
WEB_CLIENT_ID = '<web_client_id>.apps.googleusercontent.com'
ANDROID_CLIENT_ID = '<android_client_id>.apps.
    googleusercontent.com'
ANDROID_AUDIENCE = WEB_CLIENT_ID

package = 'P2PTransfers'
@endpoints.api(name='p2ptransfers', version='v1', description
    ="P2PTransfers API",
                allowed_client_ids=[WEB_CLIENT_ID,
                    ANDROID_CLIENT_ID,
                                        endpoints.
                                            API_EXPLORER_CLIENT_ID
                                            ],
                audiences=[WEB_CLIENT_ID],
                scopes=[endpoints.EMAIL_SCOPE])
```

Izvorna koda 4.10: Primer definicije REST programskega vmesnika razvitega s pomočjo Google Cloud Endpoints.

Za vsako metodo v programskem vmesniku moramo definirati, kakšnega tipa so njeni vhodni in izhodni parametri, preko katere HTTP-metode kličemo in na katerem spletnem naslovu metoda je.

Nato z orodji, ki so del razvojnega paketa Google Cloud Endpoints, generiramo knjižnice v Javi, ki so namenjene klicanju vmesnika iz aplikacije

```
@endpoints.method(RegistrationIDMessage, VoidMessage,
    path='API/SaveAndroidID', http_method='POST',
                    name='p2ptransfers.SaveAndroidID')
def save_android_device_registration_id(self, request):
  user = self.get_signed_user()
  if user is not None:
 user.device_registration_id = request.registration_id
    user.put()
   else:
     raise endpoints.UnauthorizedException('Invalid token.')
  return VoidMessage()
```

Izvorna koda 4.11: Primer metode save_android_device_registration_id, ki za vhod potrebuje ID za registracijo vrne pa prazno sporočilo.Metoda tudi uporablja trenutno prijavljenega uporabnika, zato zahteva avtentikacijo.

Android. Ko so knjižnice vključene v projekt Android, lahko zgornjo metodo preprosto kličemo kot navadno metodo. Vsa koda, ki je potrebna za pridobitev žetona OAuth, in HTTP klic sta skrita v generiranih knjižnicah.

```
GoogleAccountCredential credential = GoogleAccountCredential.
   usingAudience(MainActivity.this,"server:client_id:" +
   Utilities.WEB_CLIENT_ID);
P2ptransfers service = new P2ptransfers(AndroidHttp.
   newCompatibleTransport(), new GsonFactory(), credential);
APIAPIMessagesRegistrationIDMessage message = new
   APIAPIMessagesRegistrationIDMessage();
message.setRegistrationId("test");

service.p2ptransfers().saveAndroidID(message).execute();
```

Izvorna koda 4.12: Primer klica metode save_android_registration_id preko knjižnic, ki so bile generirane z orodji Google Cloud Endpoints.

4.4 Mobilna aplikacija

Mobilna aplikacija je bila napisana, ko je postalo jasno, da zaradi nekaterih neimplementiranih funkcionalnostih HTML5-specifikacije in z ukinitvijo storitve Google Latitude ne bo mogoče implementirati prototipa samo s spletno aplikacijo. Glavne funkcionalnosti aplikacije so spremljanje trenutne lokacije uporabnika in prejemanje potisnih sporočil ter odgovori nanje. Da uporabniku ne bo treba preklapljati med aplikacijo in brskalnikom, je bil v aplikacijo vgrajen tudi brskalnik, s katerim je mogoče brskati po spletni aplikaciji.

Aplikacija je napisana v programskem jeziku Java in je ciljana na naprave z operacijskim sistemom Android, verzije 4.3 (verzija API 18), deluje pa na vseh napravah z verzijo operacijskega sistema večjo ali enako 4. Dodatna zahteva je, da morajo naprave imeti nameščeno in posodobljeno aplikacijo Google trgovina (ang. Google Play Store), saj ta aplikacija omogoča dodatne programske vmesnike, ki so ločeni od samega operacijskega sistema in se uporabljajo predvsem pri avtentikaciji. Vsi vozniki morajo imeti napravo Android z GPS-sprejemnikom, ki ga uporabljamo za spremljanje lokacije voznika, saj je to edina izbira, ki zagotavlja doseganje zahtevane natančnosti.

Mobilna aplikacija je bila testirana na mobilni napravi Sony Xperia Z, ki je imela nameščen operacijski sistem Android, verzije 4.3.

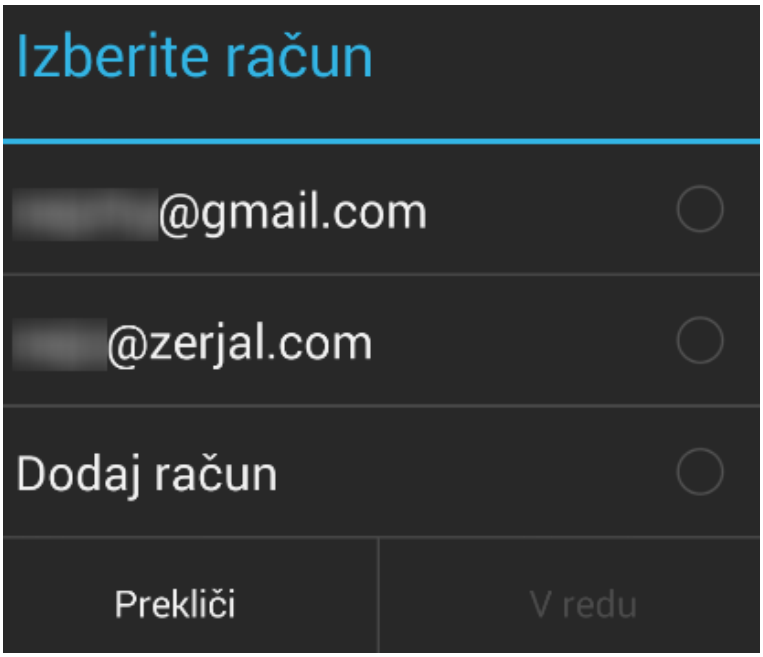
4.4.1 Prijava v aplikacijo

Pred uporabo aplikacije je nujna prijava z Googlovim računom, s katerim je uporabnik že prijavljen v spletno aplikacijo. Isti Googlov račun mora biti predtem dodan kot račun za sinhronizacijo na mobilni napravi. Prijava je nujna, ker za pošiljanje podatkov med spletno in mobilno aplikacijo uporabljamo storitev Google Cloud Endpoints s standardom OAuth 2.0 za avtentikacijo uporabnika. Avtentikacija poteka preko računa, s katerim se prijavimo v aplikacijo. Ker za prijavo uporabljamo sistemske račune, mora aplikacija zahtevati naslednje pravice, ki omogočajo dostop do sistemskih računov in za njihovo uporabo pri avtentikaciji:

```
<uses-permission android:name="android.permission.
    GET_ACCOUNTS" />
<uses-permission android:name="android.permission.
    USE_CREDENTIALS" />
```

Izvorna koda 4.13: Pravice, ki jih moramo od uporabnika zahtevati za avtentikacijo.

Za tak način avtentikacije moramo na napravi imeti nameščene tudi storitve Google Play, ki jih namestimo skupaj z aplikacijo Google trgovina.



Slika 4.11: Izbira računa za prijavo v aplikacijo preko komponente izbiralec računov.

Prijavo izvedemo z uporabo komponente izbiralec računov (ang. Account Picker), ki nam ob klicu odpre okno, v katerem so našteti vsi trenutni sistemski računi. Z izbiro računa povzročimo, da bodo vsi naslednji klici na strežnik avtenticirani z izbranim računom.

```
private static final int ACCOUNT_CHOOSER_INTENT = 5;
private GoogleAccountCredential _Credential;

_Credential = GoogleAccountCredential.usingAudience(
    MainActivity.this,"server:client_id: " + Utilities.
    WEB_CLIENT_ID);
if (!SignedIn){
  startActivityForResult(_Credential.newChooseAccountIntent()
      ,  ACCOUNT_CHOOSER_INTENT);
}

@Override
protected void onActivityResult(int requestCode, int
    resultCode, Intent data) {
 super.onActivityResult(requestCode, resultCode, data);
     switch (requestCode) {
      case ACCOUNT_CHOOSER_INTENT:
          if (data != null && data.getExtras() != null) {
              String accountName = data.getExtras().getString(
                  AccountManager.KEY_ACCOUNT_NAME);
                  if (accountName != null) {
            _Credential.setSelectedAccountName(accountName);
           }
          }
         break;
 }
}
```

Izvorna koda 4.14: Primer implementacije prijave v aplikacijo z uporabo komponente izbiralec računov.

4.4.2 Prejemanje potisnih obvestil

Potisno obvestilo (ang. push notification) dovoljuje aplikaciji, da uporabnika obvesti o dogodku, ki se je zgodil v ozadju, ne da bi uporabnik odprl aplikacijo. V našem primeru se potisna obvestila uporabljajo za obveščanje uporabnikov o naslednjih dogodkih:

- začetek spremljanja lokacije,
- konec spremljanja lokacije,
- obvestilo vozniku, da se je potencialni potnik prijavil za prevoz,
- obvestilo potniku, ali ga je voznik sprejel za prevoz.

V prototipu je ta funkcionalnost implementirana s pomočjo storitve Google Cloud Messaging (v nadaljevanju GCM). GCM ponuja 2 načina implementacije strežnika, in sicer preko standarda XMPP, ki omogoča dvostransko komunikacija med strežnikom in odjemalcem, ter preko standarda HTTP, ki omogoča komunikacijo samo v smeri s strežnika do odjemalca. Ker za delovanja potisnih obvestil potrebujemo samo komunikacijo s strežnika do odjemalca, smo se odločili za uporabo HTTP-strežnika. Za uporabo te storitve moramo imeti na napravi nameščeno in posodobljeno Google trgovino, uporabnik pa nam mora dovoliti uporabo pravic iz izseka izvorne kode 4.15.

Te pravice nam omogočajo pošiljanje in prejemanje obvestil preko interneta. Poleg tega pa parameter <application_package>preprečuje ostalim aplikacijam, da bi brale obvestila naše aplikacije.

Preden lahko aplikacija sprejeme prva potisna obvestila preko storitve GCM, se mora registrirati pri GCM-strežnikih in registrsko številko poslati na naš strežnik (izsek izvorne kode 4.16). Ta številka predstavlja identifikator naprave. Ko želimo s strežniškega dela poslati obvestilo na napravo, to številko uporabimo kot naslov naprave.

Za sprejemanje GCM-sporočil mora biti na odjemalcu implementiran poseben GCM-sprejemnik sporočil. Ta ob sprejetju sporočila zažene storitev v

```
<uses-permission android:name="android.permission.INTERNET" /
    >
<uses-permission android:name="android.permission.
    GET_ACCOUNTS" />
<uses-permission android:name="android.permission.WAKE_LOCK"
    />
<uses-permission android:name="com.google.android.c2dm.
    permission.RECEIVE" />
<permission android:name="<application_package>.C2D_MESSAGE"
        android:protectionLevel="signature" />
<uses-permission android:name="<application_package>.
    C2D_MESSAGE" />
```

Izvorna koda 4.15: Pravice, ki jih moramo od uporabnika zahtevati za implementacijo storitve Google Cloud Messaging.

```
private String registerGCM() {
  String registrationID = "";
  try {
    GoogleCloudMessaging gcm = GoogleCloudMessaging.
       getInstance(context);
    registrationID = gcm.register("<ID aplikacije iz Google
       Console>");
 sendRegistrationIdToBackend();
  } catch (IOException ex) {
    String error = Log.getStackTraceString(e);
    Log.e("registerGCM", error);
  }
  return registrationID;
}
```

Izvorna koda 4.16: Pridobivanje registrske številke za uporabo v storitvi GCM.

ozadju, ki poskrbi, da bo sporočilo pravilno obdelano. Ob tem pa skrbi tudi, da naprava ne bo prešla v mirovanje v času, ko se sporočilo obdeluje.

```
protected void onHandleIntent(Intent intent) {
  Bundle extras = intent.getExtras();
  GoogleCloudMessaging gcm = GoogleCloudMessaging.getInstance
     (this);
  String messageType = gcm.getMessageType(intent);

  if (GoogleCloudMessaging.MESSAGE_TYPE_MESSAGE.equals(
     messageType)) {
    if (extras.getString("type", "").equals("START_MONITORING
       ")){
      Intent startServiceIntent= new Intent(this,
         GPSLocationService.class);
      startServiceIntent.setAction(GPSLocationService.
         START_LOCATION_MONITORING);
      startService(startServiceIntent);
      sendNotification("Location monitoring started", null);
    }
    else if (extras.getString("type", "").equals("
       STOP_MONITORING")){
      Intent startServiceIntent = new Intent(this,
         GPSLocationService.class);
      startServiceIntent.setAction(GPSLocationService.
         STOP_LOCATION_MONITORING);
      startService(startServiceIntent);
      sendNotification("Location monitoring stopped", null);
    }
    ...
  }
}
```

Izvorna koda 4.17: Prikaz odziva na potisna obvestila za vklop in izklop spremljanja lokacije voznika.

4.4.3 Spremljanje lokacije uporabnika

Eden izmed glavnih ciljev diplomskega dela je tudi spremljanje oddaljenosti voznika od zbirnega mesta v realnem času. Zaradi tega mora voznik imeti pri sebi aplikacijo, ki neprestano oddaja trenutno lokacijo. Android trenutno omogoča dva načina pridobivanja lokacije.

- GPS-omrežje
  Pri pridobivanju lokacije preko GPS uporabljamo satelite, zato je ta način pridobivanje lokacije najbolj natančen. V idealnih razmerah je mogoča natančnost do 5 metrov. Uporaba GPS pa prinaša tudi nekaj omejitev. Naprava mora imeti poseben GPS-čip in precej starejših naprav takega čipa nima, zato na taki napravi ni mogoče pridobivati lokacije preko sistema GPS. Zahteva je tudi, da je naprava v vidnem polju s sateliti, zato včasih precej dolgo traja pred pridobitvijo prve lokacije. Velika slabost GPS je tudi zelo velika poraba energije.

- Omrežni stolpi, omrežja WiFi
  Za pridobivanje lokacije se uporabljajo omrežni stolpi mobilnega omrežja in brezžična internetna omrežja na katera je uporabnik trenutno prijavljen. Velika prednost tega načina je, da je uporabnik vedno prijavljen v mobilno omrežje in zaradi tega naprava ne porabi nič dodatne energije za pridobitev lokacije. Slabost tega načina pa je natančnost lokacije, saj običajno ne moremo pričakovati natančnosti, večje od 1 kilometra. Natančnost je zelo odvisna od gostote omrežnih stolpov, tako je v mestih lahko natančnost tudi večja, na podeželju, kjer je omrežnih stolpov manj, pa precej manjša.

Ker za spremljanje lokacije v realnem času potrebujemo čim bolj natančne podatke in ker večina novih naprav vsebuje GPS-čip, smo se odločili, da bomo lokacijo spremljali preko GPS-omrežja.

Za dostop do lokacijskih podatkov moramo od uporabnika zahtevati naslednjo pravico, ki nam dovoljuje dostop do natančne lokacije uporabnika:

```
<uses-permission android:name="android.permission.
    ACCESS_FINE_LOCATION" />
```

Izvorna koda 4.18: Pravica, ki jih moramo od uporabnika zahtevati za spremljanje lokacije.

V operacijskem sistemu Android do podatkov o lokaciji dostopamo preko sistemske storitve LOCATION_SERVICE. Lokacijo na strežnik pošiljamo vsakih 30 sekund ali po vsakih 1000 metrih spremembe lokacije. Klic na strežnik se izvede kot klic metode save_location_data iz programskega vmesnika spletne aplikacije preko storitve Google Cloud Endpoints.

```
private void startMonitoring() {
 LocationManager locationManager = (LocationManager)
    getSystemService(LOCATION_SERVICE);
    _locationListener = new SenderLocationListener();
    locationManager.requestLocationUpdates(LocationManager.
       GPS_PROVIDER, 30000, 1000, _locationListener, _looper)
       ;
}
```

Izvorna koda 4.19: Zagon spremljanja lokacacije voznika.

Vklop/izklop spremljanja lokacije je možen preko aplikacije, vendar ker to otežuje uporabo, smo se odločili, da dodamo samodejni vklop in izklop spremljanja lokacije. Tako v trenutku, ko kreiramo nov prevoz (ročno ali s samodejno preko urnika), s strežnika pošljemo potisno obvestilo, ki zažene spremljanje lokacije. Podobno se zgodi ob odhodu voznika z zadnjega zbirnega mesta na trenutni poti.

Ker ne želimo, da bi moral imeti voznik telefon prižgan skozi celotno pot, smo spremljanje lokacije implementirali kot storitev, ki se izvaja v ozadju in tako ne ovira uporabnika. Zaradi tega moramo od uporabnika zahtevati pravico iz izseka izvorne kode 4.20.

Dovoljenje za to pravico potrebujemo, ker moramo napravi preprečiti, da bi prešla v mirovanje. Tako ob zagonu spremljanja lokacije zahtevamo delno

```
<uses-permission android:name="android.permission.WAKE_LOCK"
    />
```

Izvorna koda 4.20: Pravica, ki jih moramo od uporabnika zahtevati za preprečevanje prehoda v mirovanje.

preprečitev mirovanja. To pomeni, da preprečimo procesorju naprave, da bi prešel v mirovanje, še vedno pa dovolimo zaslonu in tipkovnici, da se izklopita in tako privarčujemo precej energije.

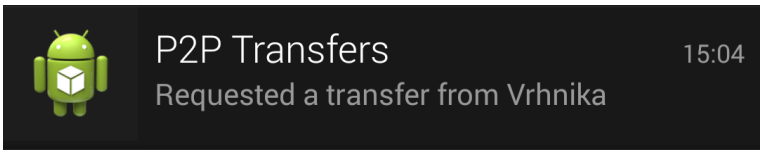
```
PowerManager.WakeLock _lock;
//Ob vklopu spremljanja lokacije
PowerManager pm = (PowerManager) getSystemService(Context.
    POWER_SERVICE);
_lock = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,
    LOG_TAG);
_lock.acquire();

//Ob izklopu spremljanja lokacije
if(_lock != null){
      _lock.release();
 }
```

Izvorna koda 4.21: Preprečitev procesorju da bi prešel v mirovanje.

4.4.4 Prijava na vožnjo

Voznik

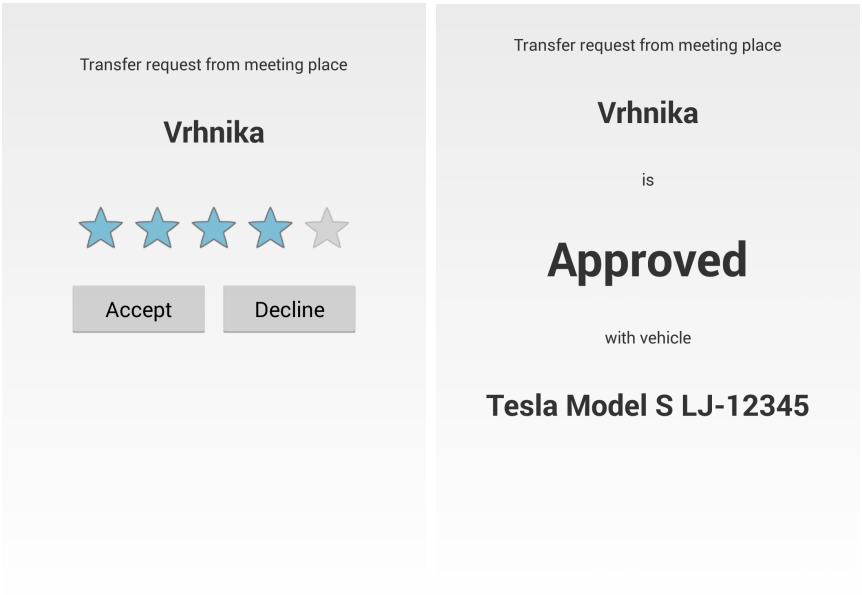
Ob prijavi potnika na vožnjo dobi voznik potisno obvestilo z imenom začetnega zbirnega mesta. Če voznik odpre potisno obvestilo, se mu prikaže zaslon s podatkom o začetnem zbirnem mestu in povprečno oceno potnika. Na podlagi teh podatkov se voznik odloči, ali bo sprejel ali zavrnil prijavo na vožnjo.



Slika 4.12: Potisno obvestilo ob prijavi potnika na prevoz z zbirne točke Vrhnika.

Potnik

Ko se voznik odloči, ali bo sprejel ali zavrnil prijavo na vožnjo, strežnik potniku pošlje potisno obvestilo z odgovorom. V primeru, ko je prijava sprejeta, dobi potnik obvestilo, s kakšnim vozilom se bo prevoz izvršil.



Slika 4.13: Zaslonski slika prijave potnika na vožnjo (levo) in slika s podatki o vozilu, s katerim bo prevoz opravljen (desno).

4.4.5 Možne izboljšave in nadgradnje

Med samo implementacijo prototipa se nam je porodilo še nekaj idej, s katerimi bi lahko dodatno nadgradili funkcionalnosti rešitve. Zanimivo bi bilo razširiti uporabo uporabniških ocen. Sedaj vsak uporabnik poda samo eno oceno med 1 in 5. V prihodnosti pa bilo lahko dodali več različnih kategorij ocenjevanja. Tako bi lahko pri vozniku ocenjevali primernost hitrosti vožnje, primernost vozila, s katerim se opravlja prevoz ... Na podlagi teh ocen bi bila mogoča dodatna personalizacija storitve, kjer bi si vsak potnik lahko nastavil minimalne ocene za vsako kategorijo. Dodati bi bilo treba možnost spreminjanja priporočene poti v sklopu dodajanja voznikove poti, saj je trenutno možno izbirati samo najhitrejšo pot med dvema lokacijama. Smiselno bi bilo dodati nove možnosti določanja cene prevoza. Zanimiv bi bil način, pri katerem bi se stroški poti delili glede na število potnikov v avtomobilu, ali način, kjer bi bili stroški odvisni od trenutne cene goriva in povprečne porabe avtomobila.

Pri preizkušanju prototipa smo večkrat prišli do dnevnega limita brezplačne uporabe kompleksnih poizvedb, ki so narejene s programskim vmesnikom Search API. V času razvoja prototipa je bilo mogoče brezplačno izvesti le 100 poizvedb na dan. Te kompleksne poizvedbe se uporabljajo pri določanju zbirnih mest v bližini poti. Zato bi bilo treba za resno uporabo najti drugačen način iskanja ali s plačilom povečati dnevni limit.

Poglavje 5

Sklepne ugotovitve

V sklopu izdelave diplomskega dela smo predstavili svoj predlog rešitve za povečanje povprečne zasedenosti avtomobila z uvedbo zbirnih mest, na katerih bi se vozniki in potniki lahko srečevali ter se dogovarjali za prevoze. Uspešno smo implementirali tudi prototip, ki predstavlja informacijsko podporo naši rešitvi in izpolnjuje cilje, ki smo si jih na začetku zadali. Tako z medsebojnim ocenjevanjem uporabnikov poskušamo vzpostaviti visoko stopnjo zaupanja med potniki in vozniki ter s tem tudi večjo varnost storitve in večje zadovoljstvo uporabnikov. Korist za voznike in potnike zagotavljamo z avtomatiziranim postopkom prijave na vožnjo in elektronskim načinom plačevanja stroškov. Na začetku je bil prototip zamišljen samo kot spletna aplikacija, napisana z upoštevanjem standarda HTML5, ki bi delovala na vseh napravah in bi za pridobitev lokacijskih podatkov uporabljala storitev Google Latitude. Ker je bila ta storitev med razvojem prototipa ukinjena in ker podpora standardu HTML5 še ni dovolj dobra za pošiljanje potisnih obvestil, smo se odločili, da se osredotočimo samo na izdelavo prototipa za operacijski sistem Android.

Ker je rešitev v fazi prototipa, seveda obstaja več možnosti za nadgradnjo in izboljšavo. Ena izmed pomembnejših nadgradenj je širitev podpore na preostale operacijske sisteme, kot sta Applov iOS in Microsoftov Windows Phone. Poleg tega pa bi bilo treba še večjo pozornost posvetiti personalizaciji

storitve tako za voznike kot za potnike. To lahko naredimo z razširitvijo kategorij ocenjevanja, večjo uporabo ocen pri priporočanju voženj ... Poskusili bi lahko glede na uro, dan in trenutno lokacijo predvideti kočno destinacijo potnika in mu glede na izbrane prioritete samodejno predlagati voznika.

Prihodnost osebnega prevoza je nedvomno v večji personalizaciji ponudnikov storitev, kar se trenutno že pospešeno dogaja v Združenih državah Amerike. Tako v večjih mestih že obstajajo ponudniki storitev, kjer je prevoz mogoče naročiti in plačati preko mobilnih naprav. V malo bolj oddaljeni prihodnosti pa si že lahko predstavljamo, kako bodo osebni prevoz prevzeli samostojno vozeči avtomobili [34]. Že sedaj ima več avtomobilskih podjetij in tudi Google delujoč prototip samostojno vozečega avtomobila. V svetu, kjer za osebni prevoz skrbijo samostojni vozeči avtomobili, si lahko predstavljamo, da preko mobilnih naprav naročimo prevoz na točno določeno destinacijo ob točno določeni uri. In ob izbrani uri nas bo avtomobil z vsemi podatki, potrebnimi za vožnjo, že čakal na dogovorjeni lokaciji. Samostojno vozeči avtomobili se bodo med seboj tudi usklajevali in tako bo mogoče optimizirati čas prevoza, saj se dolge kolone vozil ne bodo več pojavljale. Potniki bodo ogromno časa pridobili tudi, ker jim ne bo več treba iskati parkirnega mesta. Tega bo našel avtomobil sam ali pa se bo odpeljal na naslednjo vožnjo. Predvideva se tudi, da za veliko večino ljudi lastništvo avtomobila ne bo več potrebno, saj bodo prevoz lahko naročili s klikom na gumb.

Spletna aplikacija je na voljo na naslovu: `http://p2pprevozi.appspot.com/`.

Izvorna koda prototipa je na volja na naslovu: `https://bitbucket.org/Nejchy/p2ptransfers`.

Literatura

[1] (Feb. 2014) Statistični urad Republike Slovenije - Število osebnih avtomobilov na 1000 prebivalcev po letih
Dostopno na: http://pxweb.stat.si/pxweb/Dialog/varval.asp?ma=2221103s\&ti=\&path=../Database/Ekonomsko/22_transport/01_22211_transport_panoge/\&lang=2

[2] (Feb. 2014) EU transport in figures 2012, str. 47-48
Dostopno na: http://ec.europa.eu/transport/facts-fundings/statistics/doc/2012/pocketbook2012.pdf

[3] (Feb. 2014) Poland at a crossroads: The impact of CO2 and fuel economy regulation on Poland, str. 10
Dostopno na: http://www.transportenvironment.org/sites/te/files/publications/2013%20Poland_at_crossroads_final.pdf

[4] (Feb. 2014) Dorinson, Diana; Gay, Deanna; Minett, Paul; & Shaheen, Susan. (2009). Flexible Carpooling: Exploratory Study. UC Davis: Institute of Transportation Studies (UCD).
Dostopno na: http://escholarship.org/uc/item/5fk84617

[5] (Feb. 2014) Domača stran podjetja Uber
Dostopno na: https://www.uber.com/

[6] (Feb. 2014) Domača stran podjetja Sidecar
Dostopno na: https://www.side.cr/

[7] (Feb. 2014) Domača stran podjetja Lyft
Dostopno na: http://www.lyft.me/

[8] (Feb. 2014) Domača stran storitve Prevoz.org
Dostopno na:https://prevoz.org/

[9] (Feb. 2014) Domača stran podjetja GoOpti
Dostopno na: http://www.goopti.com/

[10] (Feb. 2014) Začetna stran dokumentacije za razvijalce na platformi Google App Engine
Dostopno na: https://developers.google.com/appengine/?csw=1

[11] (Feb. 2014) Domača stran Google ponudbe računalništva v oblaku
Dostopno na: https://cloud.google.com/

[12] (Feb. 2014) Search API
Dostopno na:
https://developers.google.com/appengine/docs/python/search/

[13] (Feb. 2014) Scheduled tasks
Dostopno na: https://developers.google.com/appengine/docs/python/config/cron

[14] (Feb. 2014) URL Fetch service
Dostopno na: https://developers.google.com/appengine/docs/python/urlfetch/

[15] (Feb. 2014) URL Users API
Dostopno na:
https://developers.google.com/appengine/docs/python/users/

[16] (Feb. 2014) Google Cloud Endpoints
Dostopno na:
https://cloud.google.com/products/cloud-endpoints/

[17] (Feb. 2014) Google Maps API
Dostopno na: `https://developers.google.com/maps/`

[18] (Feb. 2014) Pregled najbolj uporabljanih spletnih programskih vmesnikov
Dostopno na: `http://www.programmableweb.com/apis/directory/1?sort=mashups`

[19] (Feb. 2014) Google Cloud Messaging for Android
Dostopno na:
`http://developer.android.com/google/gcm/index.html`

[20] (Feb. 2014) Android
Dostopno na: `http://www.android.com/`

[21] (Feb. 2014) Google I/O 2012 Keynote
Dostopno na: `https://www.youtube.com/watch?v=9pmPa_KxsAM`

[22] (Feb. 2014) Mobilni operacijski sistemi
Dostopno na:
`http://en.wikipedia.org/wiki/Mobile_operating_system`

[23] (Feb. 2014) Domača stran Andoid SDK
Dostopno na: `http://developer.android.com/sdk/index.html`

[24] (Feb. 2014) Domača stran plačilnega omrežja Bitcoin
Dostopno na: `https://bitcoin.org/en/`

[25] (Feb. 2014) Prikaz gibanja tečaja med valuto Bitcoin in ameriškim dolarjem
Dostopno na: `http://blockchain.info/charts/market-price`

[26] (Feb. 2014) Prikaz števila transakcij na dan v plačilnem omrežju Bitcoin
Dostopno na: `http://blockchain.info/charts/n-transactions`

[27] (Feb. 2014) Domača stran Bitcoin menjalnice Bitstamp.
Dostopno na: https://www.bitstamp.net/

[28] (Feb. 2014) Opis algoritma hashcash za potrjevanje Bitcoin transakcij
Dostopno na: https://en.bitcoin.it/wiki/Hashcash

[29] (Feb. 2014) Prikaz izdanih bitcoinov in inflacije valute Bitcoin skozi leta
Dostopno na: http://www.mattwhitlock.com/Bitcoin%20Inflation%20logarithmic.pdf

[30] (Feb. 2014) Domača stran programskega ogrodja Twitter Bootstrap
Dostopno na: http://getbootstrap.com/

[31] (Feb. 2014) Domača stran programskega jezika Jinja2
Dostopno na: http://jinja.pocoo.org/docs/

[32] (Feb. 2014) Domača stran JavaScript knjižnice Google Maps Utility
Dostopno na:
https://code.google.com/p/google-maps-utility-library-v3/

[33] (Feb. 2014) Dokumentacija programskega vmesnika za prejemanje plačil Blockchain.info
Dostopno na: https://blockchain.info/api/api_receive

[34] (Feb. 2014) KPMG - Self driving cars: The next revolution
Dostopno na: https://www.kpmg.com/US/en/IssuesAndInsights/ArticlesPublications/Documents/self-driving-cars-next-revolution.pdf