UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Marko Daris

# Podatkovna platforma za hiter razvoj aplikacij na področju spremljanja vozil ter voznikovih navad

MAGISTRSKO DELO

Mentor: prof. dr. Marko Bajec

Ljubljana, 2016

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

*Večna pot 113*
*1000 Ljubljana, Slovenija*
*telefon: 01 47 98 100*
*www.fri.uni-lj.si*
*e-mail: dekanat@fri.uni-lj.si*

Številka: 151-MAG-RI/2016
Datum: 29. 02. 2016

**Marko DARIS**, univ. dipl. inž. rač. in inf.

**L j u b l j a n a**

Fakulteta za računalništvo in informatiko Univerze v Ljubljani izdaja naslednjo magistrsko nalogo

Naslov naloge: **Podatkovna platforma za hiter razvoj aplikacij na področju spremljanja vozil ter voznikovih navad**

**Data platform for rapid application development in the field of monitoring of vehicle and driver habits**

Tematika naloge:

Vsi avtomobili, proizvedeni po 1.1.1996, morajo biti opremljeni z vmesnikom OBD2 (On-board diagnostic), ki omogoča zunanjo diagnostiko vozila. Prek omenjenega vmesnika je možno pridobiti številne podatke, od zelo tehničnih (npr. pritisk v izgorevni komori) kot tudi takih, ki jih običajno spremljamo prek avtomobilske kontrolne plošče (hitrost, število obratov, temperatura vode, trenutna poraba itn.). V okviru magistrskega dela razvijte podatkovno platformo, ki bo združevala podatke iz OBD2 vmesnika s podatki, ki jih je možno zajeti prek pametnih mobilnih naprav (prek različnih senzorjev). Platforma naj omogoča povezovanje z različnimi oblačnimi storitvami ter oddaljenimi strežniki (npr. branje zemljevida, pridobivanje različnih geo informacij, pridobivanje podatkov o hitrostnih omejitvah itd). Kot primer uporabe platforme razvijte rešitev za analizo voznikovih voznih navad (npr. kako dosledno se voznik drži prometnih predpisov, kako pogosto pospešuje, kako pogosto zavira, kako vozi v ovinkih – radialni pospešek).

M e n t o r :

prof. dr. Marko Bajec

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

D e k a n :

prof. dr. Nikolaj Zimic

*Zahvaljujem se*

*mentorju prof. dr. Marku Bajcu za ideje in nasvete pri izdelavi magistrskega dela ter*

*partnerici Tinki Majaron za podporo in spodbujanje h končanju študija.*

*Posvečeno mami Giuseppini in očetu Luigiu,*

*moji dragi Tinki*

*in sinovoma Martinu in Damjanu.*

# Kazalo

# Seznam uporabljenih kratic in simbolov

| Kratica | Angleško | Slovensko |
|---|---|---|
| **API** | application programming interface | aplikacijski programski vmesnik |
| **CRUD** | create, read, update, delete | ustvari, beri, posodobi, briši |
| **ECU** | engine control unit | avtomobilska elektronska krmilna enota |
| **GPS** | global positioning system | globalni sistem pozicioniranja |
| **IKT** | information and communications technology (ICT) | informacijsko-komunikacijska tehnologija |
| **JAX-RS** | Java API for RESTful Web Services | javanski aplikacijski programski vmesnik za spletne storitve RESTful |
| **JPA** | Java Persistence API | aplikacijski programski vmesnik za javansko predmetno obstojnost |
| **MVC** | model-view-controller | model-pogled-kontrolor |
| **OBD** | on-board diagnostics | vgrajena avtomobilska diagnostika |
| **ORM** | object-relational mapping | predmetno relacijsko preslikovanje |
| **PaaS** | platform as a service | platforma kot storitev |
| **REST** | representational state transfer | predstavitveni prenos stanja |
| **SAE** | Society of automotive engineers | Združenje inženirjev avtomobilske industrije |

# Podatkovna platforma za hiter razvoj aplikacij na področju spremljanja vozil ter voznikovih navad

**Ključne besede:** avtomobilizem, OBD, podatkovna platforma, večplatformni razvoj mobilnih aplikacij, prostorska baza, analiza vozil, analiza voznikovih navad

**Povzetek:** Magistrsko delo opisuje podatkovno platformo za hiter razvoj aplikacij na področju avtomobilizma. Platformo smo razvili na primeru uporabe spremljanja vozil in voznikovih navad (upoštevanje prometnih predpisov, način vožnje).

Podatkovna platforma je sestavljena iz mobilne aplikacije in strežnika.

Naloga mobilne aplikacije je zbiranje podatkov iz vozila prek senzorjev mobilne naprave (GPS, pospeškomer) in vmesnika OBD2. Predstavljen je večplatformni razvoj mobilnih aplikacij, ki ustreza zahtevi po čim večji pokritosti mobilnih platform in čim večji hitrosti izvajanja.

Strežnik sprejema podatke iz mobilnih naprav in jih shranjuje v prostorsko podatkovno bazo za potrebe analize. Je hkrati aplikacijski strežnik REST in spletni strežnik za prikazovanje uporabniškega vmesnika. Za podporo analizi omogoča pridobivanje podatkov o hitrostnih omejitvah in prikazovanje rezultatov na zemljevidu s pomočjo storitev v oblaku.

Besedilo magistrskega dela raziskovalca uvede v področje spremljanja vozil in voznikovih navad, podatkovna platforma pa je osnova za raziskave in razvoj novih aplikacij in algoritmov. Premišljena izbira tehnologij in standardov omogoča hiter razvoj in enostavno uporabo brez izgube splošnosti in hitrosti izvajanja. Podatkovna platforma ostaja odprta za zamenjave tehnologij in prihodnje razširitve. Učenje uporabljenih tehnologij je olajšano z uporabo kode podatkovne platforme kot primera. Uporabniku je prihranjena kompleksnost izbire ustreznih tehnologij.

Predstavljeni so viri podatkov, arhitektura sistema, uporabljene tehnologije in težave, na katere smo naleteli. Podani so nasveti za izboljšave in dopolnitev platforme. Priloženi so izvorna koda in testni podatki, s katerimi je mogoče testirati obstoječe algoritme in razviti nove.

Kljub množici že obstoječih raziskav in projektov se na področju ponujajo vedno nove možnosti, še posebej, če upoštevamo vedno večje število podatkov, ki bodo v prihodnosti na voljo v vozilu preko OBD2, ter vedno večje zmogljivosti in zmožnosti pametnih mobilnih naprav. Kakršenkoli razvoj je nepredstavljiv brez podatkovne platforme, ki bo zmožna zajemanja in obdelovanja tako obsežnih in raznolikih podatkov. Predstavljena platforma je dober začetek na področju, izbrane tehnologije in standardi pa bodo za svoj namen uporabni še vrsto let.

# Data platform for rapid application development in the field of monitoring of vehicle and driver habits

**Keywords:** automotive, OBD, data platform, multiplatform development of mobile applications, spatial database, vehicle analysis, driver habit analysis

**Abstract:** This master's thesis describes a data platform for the fast development of applications in the automotive field. The platform is developed after the use case of monitoring of vehicle and driver habits (compliance with traffic regulations, way of driving).

The data platform is composed of a mobile application and a server.

The task of mobile applications is to collect data on the vehicle via the mobile device sensors (GPS, accelerometer) and OBD2 interface. Featured is a multi-platform mobile application development that meets the requirement for maximum coverage of mobile platforms and maximum speed of execution.

The server receives and stores data from mobile devices in a spatial database for the purpose of analysis. It servers both as a REST application server and web server to display a user interface. In support of analysis it allows obtaining information on the speed limit and displacing the results on a map by using cloud services.

The thesis introduces the researcher in the field of monitoring of vehicle and driver habits, the data platform serves as a base for research and development of new applications and algorithms. Careful selection of technologies and standards enable the rapid development and ease of use without loss of generality and speed of execution. The platform allows switching technologies and adding future extension. Learning the is facilitated by using the code as an example. The user is spared the complexity of the selection of appropriate technologies.

Data sources, system architecture, used technologies and encountered difficulties are presented. Tips for improvements and additions platform are suggested. The source code and test data to test existing and develop new algorithms is included.

Despite the multitude of existing research and projects, the field still offers new opportunities, especially considering the growing quantity of data, which will be available in the in future via OBD2 and the ever increasing capabilities of smart mobile devices. Without a data platform, which is capable of capturing and processing such quantity and variety of data, development is unimaginable. The presented platform is a good start in the field and the selected technologies and standards will serve its purpose for many years to come.

# 1 Uvod

Vedno večja razširjenost pametnih mobilnih naprav (v nadaljevanju skrajšano: mobilne naprave) in uvedba obveznega vmesnika OBD2 (ang. *On Board Diagnostics*) v vseh vozilih po letu 1996 odpirata nove možnosti na področju panog, povezanih z avtomobilizmom. Mobilne naprave so opremljene s številnimi senzorji (GPS, magnetometer, pospeškomer, žiroskop, senzor gravitacije, barometer, kamera, mikrofon ...), s katerimi zajemamo številne ambientalne podatke (kontekst uporabnika), obenem pa se uporabljajo kot enota za obdelavo ter kot povezava do spleta. Prek vmesnika OBD2 pa lahko po drugi strani pridobivamo podatke, ki predstavljajo kontekst vozila (npr. identifikacijska številka vozila, hitrost, število obratov, temperatura hladilne tekočine, pritisk vbrizganega goriva, trenutna vrednost emisij itd.).

Zbrani podatki so uporabni na različnih področjih: pri upravljanju in vzdrževanju voznega parka, zavarovalništvu (npr. popust na premijo za voznike, ki vozijo varno), spremljanju prometnih razmer na cesti, navigaciji, preprečevanju in beleženju trkov, upravljanju vozil, cestninjenju, raziskavi načina vožnje različnih demografskih kategorij (npr. mladi šoferji) ...

## 1.1 Uporaba IKT za spremljanje vozil in voznikovih navad

Cilj magistrskega dela je povezati podatke, ki jih je mogoče pridobiti iz pametnih naprav, vmesnika OBD2 (preko ključa OBD2) ter različnih storitev v oblaku ali z oddaljenih strežnikov (v nadaljevanju poenostavljeno: oblak), ter s tem ponuditi bogat podatkovni nabor (podatkovna platforma), ki bo omogočal številne možnosti uporabe v praksi. Poleg zgoraj omenjenih tudi:

- Simulator za razvoj novih mobilnih aplikacij in strojne opreme,
- Pomoč pri opravljanju vozniškega izpita,
- Napredno obračunavanje cestnine,
- Analiza stanja cestišč na slovenskih cestah,
- Črna skrinjica za hranjenje podatkov v primeru prometne nesreče,
- Spremljanje in evidentiranje prometne signalizacije.

Podatkovno platformo razvijemo na primeru uporabe spremljanja vozil in voznikovih navad.

Podatkovna platforma vključuje:

- zajemanje podatkov iz senzorjev mobilne naprave in vmesnika OBD2,
- pošiljanje podatkov na strežnik,
- shranjevanje podatkov v prostorsko podatkovno bazo na strežniku,
- analizo podatkov,
- vizualizacijo podatkov in rezultatov analiz.

Platforma je zasnovana tako, da ostaja odprta za spremembe in prihodnje dograditve. Na primer: v začetni fazi je uporabna kot orodje za zajemanje podatkov, njihovo analizo in raziskave, kasneje pa lahko isto kodo specializiramo za uporabo v konkretni aplikaciji (kot npr. Triglav Drajv [16], Pothole [3] itn.). Uporabljene tehnologije omogočajo uporabo mobilne aplikacije na vseh večjih mobilnih platformah (iOS, Android, Windows Phone), strežniški del pa je mogoče z manjšimi spremembami prenesti na bolj skalabilen Google App Engine PaaS.

## 1.2 Sorodne rešitve in njihove omejitve

Z uporabo podatkov, ki so dosegljivi prek OBD2 ali/in mobilnih naprav, so se ukvarjali mnogi strokovnjaki in raziskovalci. Razvitih je bilo več rešitev in pristopov, ki rešujejo različne s tem povezane scenarije. Vitruvius [4] je ogrodje za zajemanje, shranjevanje in hiter razvoj aplikacij za prikazovanje podatkov iz OBD2. Omogoča razvoj spletnih aplikacij brez znanja programiranja. Žal na račun splošnosti uporabe in možnosti optimizacije.

Razvoj črne skrinjice, ki poleg videa iz kamere zajema tudi podatke OBD2, je opisan v [1]. Podobna naprava s poudarkom na preprečevanju možnosti ponarejanja posnete vsebine s pomočjo oblaka je opisana v [7]. V [10] avtorji predlagajo format za izmenjavo podatkov OBD2 v okviru sistema za pregled in vzdrževanje vozil. Gre za format XML, ki podpira le podatke OBD2, ne pa tudi npr. podatke iz pospeškomera. Z optimizacijo prenašanja podatkov iz vozil s pomočjo mehke logike so se ukvarjali v [5]. Raziskava voznih navad mladih voznikov je opisana v [2].

Med komercialnimi aplikacijami s tega področja so najbolj znane TomTom [40] in Waze [44], ki s spremljanjem podatkov iz mobilnih naprav uporabnikov izračunavajo gostoto prometa in morebitne zastoje. Zanimiv je tudi projekt Nericell [8], ki senzorje na mobilni napravi uporablja za določanje različnih prometnih situacij. Z uporabo mikrofona zaznava hupanje, v kombinaciji s pospeškomerom pa luknje na vozišču. Z zaznavanjem kakovosti vozišča se ukvarja tudi projekt Pothole [3].

V Sloveniji sta na tem področju najbolj znana aplikacija Drajv [16] in priključek ULU [42]. Drajv je mobilna aplikacija (brez podpore za OBD2) za merjenje voznikovih navad in pridobivanje točk, ki jih je mogoče pretvoriti v popust pri avtomobilskem zavarovanju. Rešitev uporablja zavarovalnica Triglav. Žal pa aplikacija ne omogoča povezave OBD2, kar pomeni, da posnete poti ne moremo povezati z vozilom.

ULU je izdelek slovenskega zagonskega podjetja, ki je dobilo finančno investicijo in se kasneje preselilo na Nizozemsko. Njihov produkt je tako imenovani ključek ULU, ki je razširjen priključek OBD2 z dodanim modemom GPRS in sprejemnikom GPS. Prednost ključka ULU je, da ne potrebujemo mobilne naprave – slabost (za naše potrebe) pa je nezmožnost razširitve z dodatnimi senzorji.

Omeniti je treba tudi Torque [41], eno boljših mobilnih aplikacij za delo z OBD2. Aplikacijo odlikuje odlična podpora OBD2 – podpira tudi nekatere starejše avtomobile, ki odstopajo od specifikacije OBD2. Torque je torej odličen vir podatkov iz mobilne naprave, težave pa bi se

pojavile, če bi želeli napisati svojo mobilno aplikacijo. Ker koda ni na voljo, bi morali mobilno aplikacijo napisati od začetka.

Kljub omenjenim rešitvam, ki se ukvarjajo bodisi z uporabo ambientalnih podatkov, pridobljenih prek mobilnih naprav, bodisi z uporabo podatkov, pridobljenih prek vmesnika OBD2, nismo zasledili rešitev, ki bi bile odprte (dostop do programske kode, brezplačna uporaba ...) in bi jih lahko uporabili za podporo različnih scenarijev, povezanih z vožnjo, vozili, vozniki, cestami itn.

## 1.3 Struktura dela

Magistrsko delo se prične z opisom vseh uporabljenih podatkovnih virov v poglavju 2.1. Opisani so viri izvornih podatkov (GPS, pospeškomer, OBD2) in viri pomožnih podatkov (hitrostne omejitve, zemljevidi, nevarni cestni odseki).

Arhitektura sistema je opisana v poglavju 0. V poglavju 2.2.1 prikažemo arhitekturo mobilne aplikacije, v 2.2.2 pa arhitekturo strežnika. Opisane so naloge in lastnosti sistema. Arhitekturo razložimo na podlagi primera uporabe.

Tehnologije, ki smo jih uporabili v mobilni aplikaciji in na strežniku, so naštete in opisane v poglavju 2.3. Poglavje je razdeljeno na tehnologije, ki so uporabljene v mobilni aplikaciji (2.3.1), in tiste, ki so uporabljene na strežniku (2.3.2).

V poglavju 2.4 prikažemo tehnične izzive, s katerimi smo se srečali.

Prikaz rešitve z zaslonskimi maskami je v poglavju 2.5.

V poglavju 3 povzamemo rezultate dela in podamo sklepne misli.

# 2 Podatkovna platforma za razvoj aplikacij

## 2.1 Podatkovni viri

Za spremljanje vozil in voznikovih navad uporabljamo podatke iz različnih virov. Vire lahko v grobem razdelimo na dve skupini glede na njihov tip.

- Izvorni podatki: V to skupino spadajo podatki, ki jih zbiramo na samem vozilu med vožnjo. Vir podatkov na vozilu so senzorji mobilne naprave in podatki iz vmesnika OBD2.
- Pomožni podatki: V to skupino spadajo podatki, ki jih potrebujemo na strežniku za analizo podatkov iz vozila in prikazovanje rezultatov.

### 2.1.1 Izvorni podatki

Mobilne naprave s svojimi senzorji so bogat izvor podatkov, s katerimi lahko spremljamo obnašanje vozila in voznika. V sodobnih pametnih mobilnih napravah lahko najdemo naslednje senzorje [37]:

- GPS,
- pospeškomer,
- žiroskop,
- kompas,
- bližinski senzor,
- svetlobni senzor,
- mikrofon,
- kamera,
- barometer,
- termometer,
- vlagomer,
- merilec korakov,
- merilec srčnega utripa,
- čitalec prstnih odtisov ...

Za potrebe magistrskega dela smo se osredotočili na GPS in pospeškomer, kar pa ne pomeni, da drugi senzorji niso uporabni. Primeri uporabe teh senzorjev so: žiroskop lahko uporabimo za izboljšanje natančnosti podatkov o pospešku; z mikrofonom lahko spremljamo raven hrupa v potniški kabini; z barometrom, vlagomerom in svetlobnim senzorjem lahko sklepamo o vremenskih razmerah na cestišču; čitalec prstnih odtisov uporabimo za identifikacijo voznika itn.

Pametno mobilno napravo uporabimo tudi za pridobivanje podatkov iz zunanjih naprav preko brezžične povezave bluetooth oz. wi-fi. V našem primeru je to ključ ODB2, s katerim preko

vmesnika OBD2 dostopamo do podatkov na avtomobilski krmilni enoti (ang. *engine control unit*, skrajšano ECU).

Pametna mobilna naprava ima torej dvojno vlogo:

- izvor podatkov iz lastnih senzorjev in zunanjih naprav,
- zbiranje, shranjevanje in posredovanje podatkov strežniku.

V nadaljevanju si oglejmo uporabljene senzorje, njihove podatke in vlogo pri spremljanju vozil in voznikovih navad.

#### 2.1.1.1 GPS

Globalni sistem pozicioniranja – GPS (ang. *Global Positioning System*) uporabljamo za pridobivanje podatkov o legi vozila (zemljepisna širina, zemljepisna dolžina in nadmorska višina) in njegovi hitrosti. Sprejemnik GPS oz. njegov gonilnik nam sporoča tudi vertikalno in horizontalno natančnost (ang. *vertical/horizontal accuracy*) pridobljenih podatkov.

GPS je vgrajen v večino mobilnih naprav z izjemo cenejših tabličnih računalnikov, ki te funkcionalnosti ne potrebujejo. Razlike med napravami so predvsem v občutljivosti na signal, natančnosti meritve in podpori različnih sistemov GPS (ameriški Navstar, ruski GLONAS oz. kitajski BeiDou). Natančnost GPS v mobilnih napravah je tipično od enega do petih metrov.

Hitrost vzorčenja naprave GPS je omejena navzdol na približno eno sekundo, kar zadostuje za naše potrebe. Ker je GPS en večjih porabnikov energije v mobilni napravi, je v aplikaciji, ki je namenjena končnemu uporabniku, interval vzorčenja smiselno povečati oziroma GPS uporabljati čim redkeje.

Lego potrebujemo za prikazovanje opravljene poti in za določanje lastnosti ceste, kot so hitrostna omejitev, povprečna hitrost, gostota prometa, tip ceste itn.

GPS uporabimo tudi kot primarni izvor podatkov o hitrosti (in posledično pospešku) vozila. V večini primerov je hitrost, pridobljena preko GPS, natančnejša od hitrosti, pridobljene preko vmesnika OBD oz. avtomobilskega tahometra. Naprava GPS računa hitrost na podlagi prevožene razdalje in časa, v kateri je bila ta razdalja prevožena. Hitrost, pridobljena preko GPS, ni odvisna od velikosti avtomobilskih pnevmatik. Proizvajalci avtomobilov nastavijo tahometer tako, da je prikazana hitrost višja od resnične, saj se s tem izognejo morebitnim tožbam.

Slabost naprave GPS pa je njena neuporabnost v določenih situacijah: vožnja v predorih, vožnja po cesti, ki jo obdajajo visoke stavbe ali stene.

#### 2.1.1.2 Pospeškomer

Pospeškomer oziroma senzor za merjenje pospeška je prisoten v večini mobilnih naprav. Uporablja se predvsem kot alternativa za krmiljenje uporabniškega vmesnika. Z njim naprava ugotovi svoj položaj (naklon) in tako prilagodi prikaz uporabniškega vmesnika.

Pospeškomer meri pospeške v vseh treh prostorskih smereh – x, y, z. Vektorska vsota pospeškov naprave v mirovanju ustreza pospešku zemlje – g (9,8 $m/s^2$).

Frekvenca vzorčenja pospeškomera je tipično od 5 do 100 Hz. Večja frekvenca izboljša natančnost meritve, vendar poveča število podatkov, ki jih je treba obdelati, shraniti in poslati. V aplikaciji, ki je namenjena končnemu uporabniku, je smiselno podatke filtrirati na napravi sami in sporočati samo izjemne dogodke (prekoračitve določenih pragov).

Na področju spremljanja vozil ter voznikovih navad lahko z uporabo pospeškomera ugotovimo:

- način pospeševanja oz. zaviranja,
- način vožnje v ovinkih,
- stanje cestišča.

Voznik, ki se izogiba naglega pospeševanja in zaviranja in speljuje ovinke zmerno, bo ocenjen bolje. Hitra vožnja po cesti, ki povzroča veliko tresljajev, predstavlja večje tveganje kot vožnja po dobro vzdrževani cesti.

Pri interpretaciji podatkov s pospeškomera moramo upoštevati:

- Način pritrditve naprave: v idealnem primeru bi se morala mobilna naprava premikati skupaj z vozilom, vendar pa je pogosto pritrjena na nosilec, ki prispeva k povečanju oz. blaženju tresljajev.
- Položaj oz. orientacijo naprave: ker obstajajo različni nosilci, je naprava na enih v navpičnem položaju, na drugih v vodoravnem ali nekje vmes; naprava je lahko usmerjena v smer vožnje ali pod določenim kotom.
- Lastne vibracije vozila: motor vozila povzroča tresenje že, ko vozilo miruje; vozila s tršim podvozjem prenašajo na mobilno napravo več tresljajev kot vozila z boljšim vzmetenjem.

Zaradi omenjenega moramo pri uporabi podatkov iz pospeškomera za izračun ocene vožnje podatke dodatno obdelati:

- ugotoviti orientacijo naprave,
- filtrirati lastno tresenje vozila,
- izločiti kratkoročne sunke,
- ugotoviti pragove, ki predstavljajo prekomerno pospeševanje, zaviranje in vožnjo v ovinke.

Za razvoj algoritmov in obdelavo teh podatkov si lahko pomagamo z rezultati projektov Nericell [8] in Pothole [3].

#### 2.1.1.3 OBD2

OBD (ang. *On Board Diagnostics*) je termin v avtomobilski industriji, ki se nanaša na zmožnost samodiagnostike in sporočanja napak na vozilu. OBD2 (ali tudi OBD-II) je

standard, ki so ga razvili v SAE (ang. *Society of Automotive Engineers*, slo. Združenje inženirjev avtomobilske industrije) in predpisuje obliko diagnostičnega priključka, razporeditev signalnih nožic, razpoložljive elektronske signalne protokole in obliko pošiljanja sporočil. [48]

V ZDA mora biti po zakonu vsako vozilo, ki je bilo proizvedeno po letu 1996, opremljeno s priključkom po specifikaciji OBD2. V EU je priključek obvezen za vsa bencinska vozila po letu 2001 oziroma vsa dizelska vozila po letu 2003. Zakon je nastal kot posledica zahtev CARB (California Air Resource Board) iz leta 1991, da naj vozila vsebujejo vsaj osnovno zmožnost diagnostike za potrebe spremljanja emisij. Večina parametrov, ki jih preberemo iz priključka OBD2, se zato nanaša na količine, povezane z emisijami vozila.

Poskusi vgradnje računalniških diagnostičnih naprav v vozila segajo v leto 1968 (Volkswagen), vendar je do uvedbe standardov vsak proizvajalec imel svojo rešitev. Prvo standardizacijo je predlagal SAE leta 1988. Sledil je standard OBD1 leta 1991 in OBD2 leta 1996.

Standard OBD2 dovoljuje uporabo različnih signalnih protokolov. Ti so:

- SAE J1850 PWM,
- SAE J1850 VPW,
- ISO9141-2,
- ISO14230-4 (KWP2000),
- ISO15765-4/SAE J2480 (CAN-BUS).

Signalni protokoli se razlikujejo v razporeditvi signalnih nožic, dolžini sporočila in hitrosti prenosa sporočila.

Prek priključka OBD2 lahko dostopamo do vodila z eno ali več avtomobilskimi elektronskimi krmilnimi enotami (ang. *engine control unit*, skrajšano ECU).

Za branje podatkov iz vmesnika OBD2 smo včasih potrebovali posebne naprave. Te naprave so zamenjali ključi OBD2, ki vsebujejo mikrokontroler ELM327 podjetja ELM Electronics [19]. Mikrokontroler abstrahira nizkonivojske signale protokolov OBD2 v enostaven vmesnik, do katerega lahko dostopamo preko USB, RS232, bluetootha in wi-fija.

Izbira ključa OBD2 je pogojena z izbiro platforme mobilne naprave. Mobilne naprave Apple lahko uporabljajo le od Appla odobrene naprave z nizkoenergijskim bluetoothom (ang. *bluetooth Low Energy*). V času pisanja tega magistrskega dela ni obstajal noben tak ključ OBD2, zato je bil izbran ključ OBD2, ki za komunikacijo uporablja vmesnik wi-fi.

Z ELM327 se sporazumevamo s t. i. naborom ukazov AT, podobnim tistemu, ki ga poznamo iz časa telefonskih modemov. Prek priključka OBD2 lahko dostopamo do podatkov na avtomobilski elektronski krmilni enoti (ang. *engine control unit*). Poleg zgoraj omenjenih emisijskih parametrov pa lahko preberemo še druge, ki so vezani na delovanje vozila (hitrost, število vrtljajev itn). Prek OBD2 lahko preberemo in ponastavimo diagnostične kode težav (ang. *Diagnostic Trouble Codes*, skrajšano DTC).

Vsakemu od parametrov ustreza t. i. identifikacijska koda OBD2 PID (Parameter IDs). Kode PID so združene v t. i. načine delovanja (ang. *modes of operation*). Za vsak parameter moramo podati način (dvomestna šestnajstiška številka od 00 do 0A) in kodo (dvomestna šestnajstiška številka od 00 do C4). V nadaljevanju bomo zaradi enostavnosti skupek načina in kode PID imenovali kar PID.

ELM327 vrne za vsak niz PID bajtov, ki jih s formulo pretvorimo v končno vrednost. Vsakemu PID-u ustreza njegova formula.

Celoten seznam kod PID je na voljo na [47]. Za naše potrebe so najbolj zanimive:

- 0100 – podprte kode PID za način delovanja 01 (bitna maska: bit 1 = PID 01, bit 2 = PID 02 ...),
- 010C – število vrtljajev motorja,
- 010D – hitrost vozila,
- 0111 – pozicija stopalke za plin,
- 0902 – identifikacijska številka vozila (ang. *Vehicle Identification Number*, VIN).

Kljub standardu so nekatere kode PID različne od proizvajalca do proizvajalca, zato je treba pri njihovi interpretaciji upoštevati proizvajalca in model vozila.

Hitrost vzorčenja z uporabo ELM327 je okoli 2 Hz. Standard J1979 omejuje hitrost vzorčenja na največ 10 Hz. Omejitev je posledica tega, da so na istem vodilu še druge naprave, ki morajo za pravilno delovanje vozila medsebojno komunicirati. Zato je bolj smiselno kot vir hitrosti uporabiti GPS, iz OBD2 pa brati število vrtljajev, položaj stopalke za plin ali kateri drug parameter OBD2.

### 2.1.2 Pomožni podatki

Med pomožne podatke štejemo podatke, ki ne prihajajo neposredno iz vozila. Potrebujemo jih za:

- analizo podatkov, ki prihajajo iz vozila,
- prikazovanje prevožene poti.

Da ocenimo voznikov način vožnje oz. faktor tveganja, potrebujemo poleg podatkov o hitrosti in pospeških še:

- hitrostno omejitev na določenem odseku,
- nevarnost določenega cestnega odseka.

Za interpretacijo rezultatov je treba poti in izmerjene oz. izračunane količine prikazovati na zemljevidu. Tako lažje ugotovimo, ali uporabljeni algoritmi delujejo pravilno.

#### 2.1.2.1 Hitrostne omejitve

Podatki o hitrostnih omejitvah so na voljo pri vseh večjih ponudnikih spletnih zemljevidov, ki podpirajo navigacijo. Ovrednoteni so bili naslednji:

- Google Maps [22],
- OpenStreetMap (OSM) [35],
- Here WeGo [25].

Google Maps poleg plačljive ponuja tudi brezplačno uporabo svojih zemljevidov in povezanih storitev, vendar to ne vključuje podatkov o hitrostni omejitvi. Ti so na voljo samo v plačljivi različici. Plačljiva različica ponuja še eno posebno funkcionalnost, ki je drugi ponudniki nimajo – "prilepi na cesto" (ang. *snap to road*). "Prilepi na cesto" vrne najverjetnejšo pot po cesti, ki jo predstavlja niz koordinat GPS.

Podatki z OpenStreetMaps (OSM) so na voljo brezplačno in vključujejo tudi podatke o hitrosti. Težava pri OSM so strežniki. Ker ne gre za komercialni projekt, so sredstva za financiranje delovanja strežnikov omejena. Zato so strežniki pogosto zasedeni oziroma nedostopni. Za uporabo podatkov iz OSM bi bilo treba postaviti in vzdrževati svoj strežnik.

Here WeGo (bivši Nokia Maps) ponuja podatke o hitrostni omejitvi tudi v brezplačni različici, zato je bila storitev izbrana za potrebe magistrskega dela. Do storitev Here WeGo dostopamo preko vmesnika REST – podatki o hitrostni omejitvi so na voljo preko vmesnika Routing REST API, getLinkInfo. Za uporabo storitev se je treba prijaviti kot uporabnik/razvijalec. Prijaviti je treba tudi aplikacijo, ki bo storitve uporabljala. V brezplačni različici je na voljo 50.000 transakcij mesečno.

Kakovost podatkov o hitrostni omejitvi v Here WeGo je zadovoljiva. Podatek o hitrostni omejitvi je na voljo za večino cest in glavnih ulic, izjema so manjše ulice. Here WeGo ne ponuja storitve, primerljive s "prilepi na cesto" Google Maps, zato včasih dobimo hitrost vzporednice oz. nadvoza. Z upoštevanjem podatka, kateri cesti pripada hitrost, se je mogoče takim napakam izogniti.

#### 2.1.2.2 Zemljevidi

Zemljevide prikazujemo s pomočjo vmesnika API Here WeGo Javascript. Funkcionalnosti, ki jih potrebujemo:

- izrisovanje zemljevida,
- premikanje, povečevanje, pomanjševanje zemljevida,
- označevanje točk na zemljevidu,
- risanje poligonov,
- pridobivanje informacije o izbrani točki/elementu.

Ker gre za Javascript API, ki se izvaja na odjemalčevem brskalniku, in ker podatke o zemljevidu strežejo strežniki HERE, za strežnike namenjeni hrambi prevoženih poti in analizi podatkov ni dodatne obremenitve.

#### 2.1.2.3 Nevarni cestni odseki

V Sloveniji je na spletni povezavi Nesreče [32] na voljo zemljevid s statistko nesreč na slovenskih cestah. Strežniški del spletne aplikacije gosti Javna agencija Republike Slovenije za varnost prometa. Aplikacija uporablja bazo podatkov o prometnih nesrečah slovenske policije.

Aplikacija prikazuje zemljevid Slovenije, na katerem je mogoče prikazati, kje so se zgodile nesreče. Te je mogoče filtrirati po različnih kriterijih: časovno, po regiji, kraju, vzroku nesreče, tipu udeleženih vozil itn. Vsaka nesreča je prikazana kot točka na zemljevidu. S klikom na točko je mogoče pridobiti natančen opis dogodka.

Pri spremljanju vozil in voznikovih navad je treba te podatke dodatno obdelati. Namesto posameznih nesreč potrebujemo podatek o tem, koliko nesreč se je zgodilo na določenem odseku ceste – množico točk posameznih nesreč pretvorimo v poligone, ki predstavljajo odseke cest. Vsak tak odsek ima določeno oceno nevarnosti. S tako obdelanimi podatki lahko za določeno prevoženo pot izračunamo njeno nevarnost.

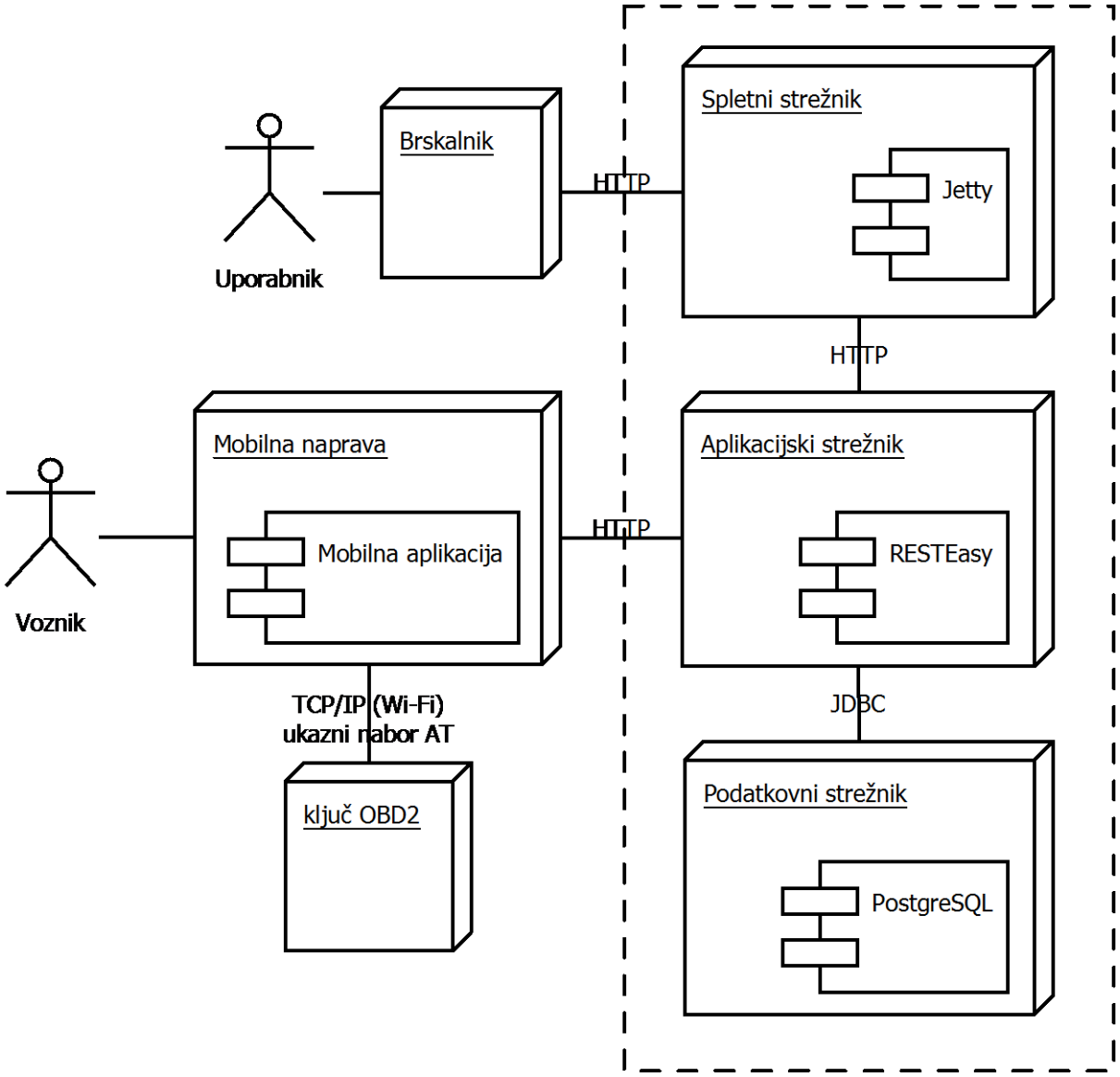
Zaradi zahtevnosti take analize je za razvrščanje poti glede na nevarnost dodana samo podpora za prihodnjo dograditev, in sicer strežniški del aplikacije hrani podatke o prevoženih poteh v prostorski podatkovni bazi (ang. *spatial database*). Prostorska podatkovna baza je optimizirana za hranjenje prostorskih informacij, kot so točke, premice in poligoni. Podpira tudi prostorska povpraševanja, npr. iskanje vseh poligonov, ki so vsebovani v določenem poligonu, iskanje vseh točk, ki so od določene točke oddaljene manj kot kilometer, itn.

S pomočjo prostorske baze lahko za vsako pot dobimo seznam nevarnih odsekov, ki jih določena pot prečka, in to upoštevamo pri izračunu nevarnosti prevožene poti.

## 2.2 Arhitektura sistema

Sistem sestavljajo naslednje komponente (Slika 1):

- ključ OBD2 (wi-fi),
- mobilna aplikacija za zajemanje podatkov,
- strežniški del (Slika 1, črtkani del) za hranjenje in analizo podatkov.




**Slika 1 – Postavitveni diagram sistema**

### 2.2.1 Mobilna aplikacija

Glavne naloge mobilne aplikacije (imenovani mobdlog, okrajšava za multi obd logger) so:

- zbiranje podatkov iz podatkovnih virov v vozilu (GPS, pospeškomer, OBD2),
- shranjevanje podatkov v spomin mobilne naprave,
- pošiljanje shranjenih podatkov na strežnik.

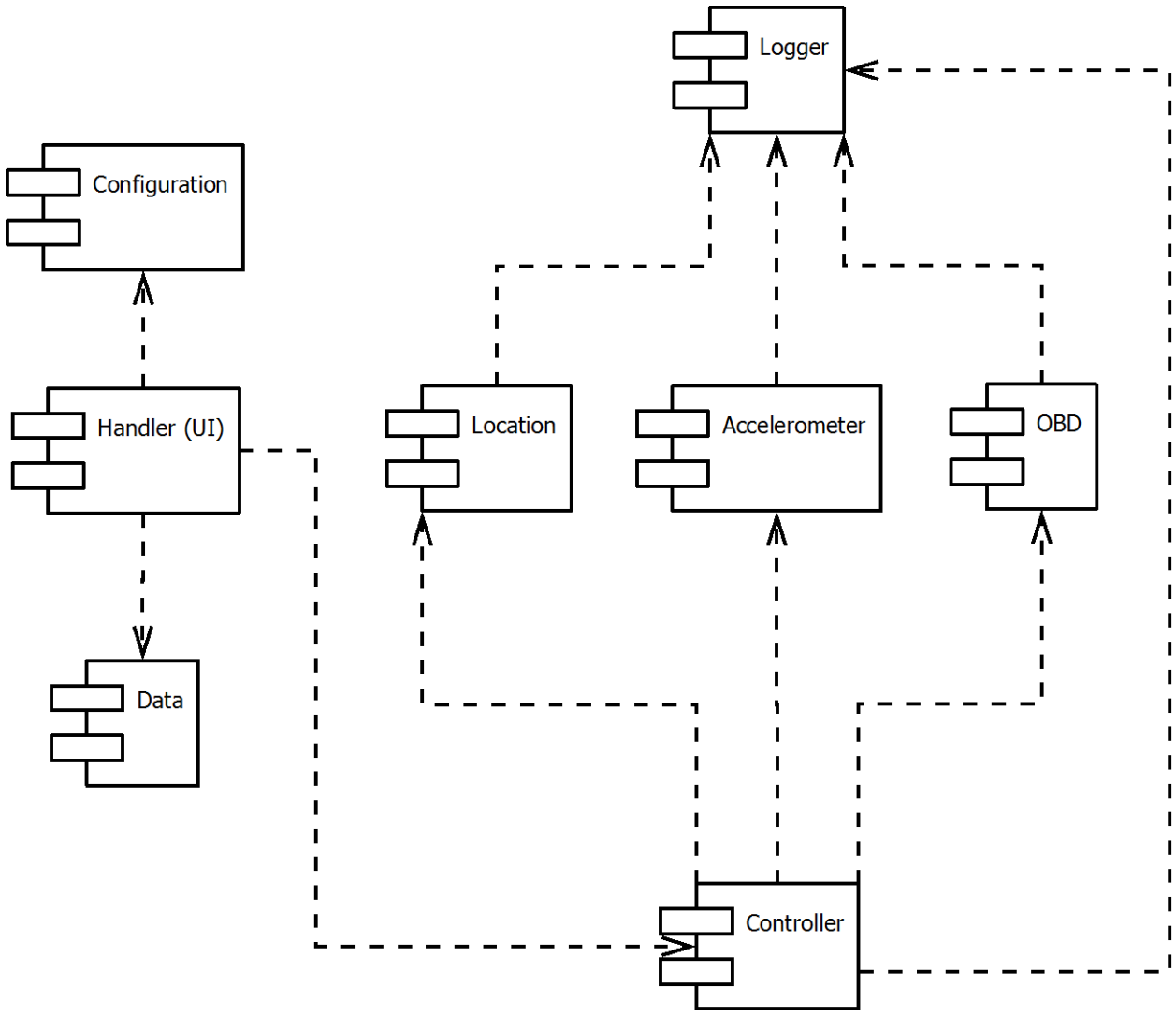
Lastnosti mobilne aplikacije:

- podpora glavnih mobilnih platform: Google Android, Apple iOS, Microsoft Windows Phone,
- enotna koda na vseh podprtih platformah: za hiter razvoj mobilnih aplikacij je potrebna ena kodna osnova (ang. *code base*) in enotno razvojno okolje, sicer je za vsako novo zmožnost (ang. *feature*) potreben (v našem primeru) trikratni trud,
- hitrost delovanja: mobilna aplikacija mora biti dovolj hitra, da uspe shraniti vse podatke, ki prihajajo iz GPS, pospeškomera in OBD2,
- smotrna poraba spomina za shranjevanje podatkov in pošiljanje: mobilna aplikacija zbrane podatke stisne, da zavzamejo čim manj prostora,
- enostavna možnost razširitve: mobilno aplikacijo je mogoče razširiti z vtičniki, ki zajemajo podatke iz drugih senzorjev mobilne naprave,
- uporabna kot okolje za testiranje algoritmov: mobilna aplikacija se lahko uporabi za testiranje algoritmov, ki so razviti na podlagi rezultatov analize na strežniku,
- ogrodje za razvoj mobilnih aplikacij, namenjenih končnemu uporabniku: kodo mobilne aplikacije je mogoče uporabiti za razvoj mobilnih aplikacij, ki so namenjene končnemu uporabniku (npr. Triglav Drajv [16]).

Mobilno aplikacijo sestavljajo:

- uporabniški vmesnik (*Handler*);
- modul za shranjevanje podatkov (*Logger*);
- modul za nastavitve (*Configuration*);
- modul za zbiranje podatkov (*Controller*) s podmoduli:
    - lega (*Location*),
    - pospeškomer (*Accelerometer*),
    - OBD2 (*OBD*);
- modul za pošiljanje podatkov po protokolu HTTP (*Data*).

Slika 2 prikazuje komponentni diagram mobilne aplikacije. Zaradi preglednosti so nekatere povezave izpuščene.




**Slika 2 – Komponentni diagram mobilne aplikacije**

Uporabniški vmesnik mobilne aplikacije ponuja minimalno potrebno funkcionalnost. Omogoča spreminjanje osnovnih nastavitev, kot so: frekvenca vzorčenja pospeškomera, hitrost beleženja lege, naslov IP in vrata (ang. *port*) ključa OBD2, identifikacijsko številko vozila, naslov (URL) za pošiljanje podatkov. Če je aplikacija povezana s ključem OBD2, pa omogoča tudi odkrivanje parametrov OBD2 (ang. *scanning*) oz. PID-ov, ki jih vozilo podpira. Za vsakega od najdenih parametrov lahko izberemo, ali naj se beleži ali ne.

Ko uporabnik (igralec voznik) zahteva začetek beleženja s pritiskom na Start, *Controller* zažene module *Logger*, *Location*, *Accelerometer* in *OBD*. Naloga modulov je pridobivanje podatkov. Da bi čim bolj zmanjšali vpliv podatkovnih modulov drug na drugega, se vsak izvaja v svoji niti.

Takoj ko je na voljo podatek, ki ga želimo beležiti, se ta posreduje modulu za shranjevanje podatkov (*Logger*). *Logger* podatek shrani v vmesni pomnilnik (ang. buffer) in nemudoma vrne kontrolo klicatelju. Ko se vmesni pomnilnik napolni, ga *Logger* zamenja s praznim. Vsebina prvega vmesnega pomnilnika se v ločeni niti stisne (z algoritmom gzip) in zapiše v datoteko v pomnilniku mobilne naprave. Tako mobilna aplikacija shranjuje podatke brez

prekinitev, ki bi jih povzročile zakasnitve zaradi pisanja v spomin mobilne naprave (masovni spomin mobilne naprave je relativno počasen, lahko je pa tudi zaseden s strani drugega procesa).

Podatkovni moduli so medsebojno neodvisni. Izjema je *Location*, ki skrbi za pridobivanje lege, saj so brez nje drugi podatki neuporabni.

Med vožnjo lahko uporabnik označi nek dogodek s pritiskom na Mark.

Po končani vožnji uporabnik ustavi beleženje s pritiskom na Stop.

Shranjene podatke pošljemo na strežnik preko zahteve HTTP POST. Po uspešnem pošiljanju se datoteke izbrišejo.

Podatki se shranjujejo v datoteko v obliki JSON, vsak dogodek v svoji vrstici. Format datoteke je opisan v 4.1.

### 2.2.2 Strežnik

Glavne naloge strežnika (imenovanega mobdsrv) so:

- sprejemanje podatkov iz mobilnih aplikacij,
- uvoz in hramba sprejetih podatkov v podatkovno bazo,
- streženje shranjenih podatkov,
- izvajanje analiz, hranjenje in streženje njihovih rezultatov,
- spletni strežnik za potrebe uporabniškega vmesnika (imenovan mobdapp).

Lastnosti strežnika:

- javanski strežnik;
- spletne storitve REST (ang. *representational state tranfer*) vse prednosti, ki jih prinaša arhitekturni stil REST;
- enostaven razvoj:
    - strežnik se izvaja v vgrajenem (ang. *embedded*) strežniku Jetty (glej 2.3.2.1); za testiranje ni treba izvesti postavitve (ang. *deployment*) kot pri uporabi drugih aplikacijski strežnikov,
    - strežnik Jetty deluje hkrati kot aplikacijski strežnik in spletni strežnik,
    - uporaba označb JAX-RS (glej 2.3.2.2) za definicijo vmesnika REST pohitri razvoj;
- uporaba industrijski standardov:
    - spletne storitve RESTful – JAX-RS, JBoss RESTEasy,
    - predmetna obstojnost – JPA, Hibernate ORM (glej 2.3.2.3),
    - podpora prostorskih podatkov – JTS, Hibernate ORM Spatial (glej 2.3.2.3);
- neodvisnost od izbire podatkovne baze: zaradi uporabe JPA lahko podatkovno bazo PostreSQL/PostGIS (glej 2.3.2.4) enostavno zamenjamo z drugo, ki ima enake (prostorske) zmožnosti (recimo MySQL, Oracle, SQLServer, GeoDB ...);

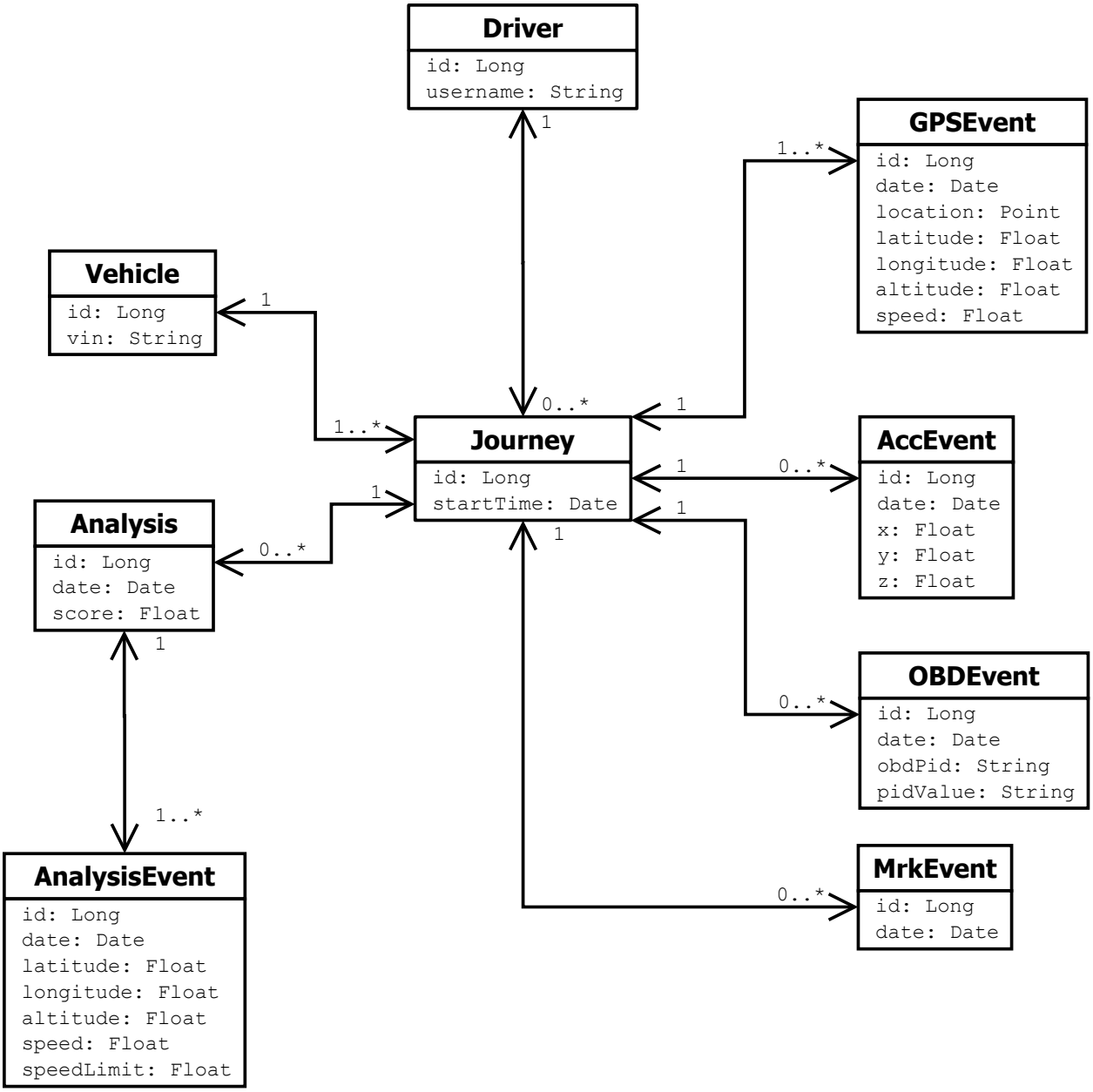
- možnost prenosa (ang. *porting*) na Google App Engine (skrajšano GAE) PaaS: večino uporabljenih tehnologij (razen prostorske podpore, ki je trenutno v GAE še v stanju alfa) lahko uporabimo na GAE;
- enostavnost razvoja spletnega uporabniškega vmesnika: spletno ogrodje AngularJS temelji na oblikovalnem vzorcu (ang. *design pattern*) MVC (*Model-View-Controller*). Zaradi ločitve interesov (ang. *separation of concerns*) koda MVC ostaja obvladljiva tudi pri večjih projektih.

Strežniški del je implementiran kot javanski strežnik, ki se izvaja v vgrajenem spletnem strežniku Jetty. Jetty služi kot servletni vsebnik za potrebe vmesnika REST in hkrati spletni strežnik za streženje uporabniških strani. Možna je tudi uporaba drugega servletnega vsebnika in spletnega strežnika.

Vmesniki REST so implementiran v fasadah (ang. *facade*):

- fasada za sprejemanje in uvoz podatkov iz mobilne aplikacije (*UploadFacade*):
  - POST /api/upload: prenos datoteke iz mobilne aplikacije na strežnik,
  - GET /api/files: spisek prenesenih datotek,
  - GET /api/import: uvoz vseh prenesenih datotek,
  - GET /api/import/{file}: uvoz ene datoteke;
- fasada za delo z vozniki in vožnjami (*DataFacade*):
  - GET /api/drivers: spisek voznikov,
  - GET /api/journeys/{driverId}: spisek voženj določenega voznika,
  - GET /api/journey/{journeyId}: vsi podatki o določeni vožnji;
- fasada za analizo podatkov (*AnalysisFacade*):
  - GET /api/analysis/{journeyId}: analiza vožnje.

Za lažje razumevanje spodnjega primera uporabe si oglejmo podatkovno shemo oziroma, ker gre za predmetno obstojnost, relacije med entitetami (Slika 3).




**Slika 3 – Relacije med entitetami**

Tipičen primer uporabe se prične z voznikovo zahtevo po hranjenju podatkov vožnje. Mobilna aplikacija pošlje zahtevo POST na URI /api/upload. Telo zahteve (ang. *request body*) vsebuje podatke vožnje. Strežnik sprejme zahtevo in shrani podatke v datoteko.

Uporabnik obišče spletno stran strežnika in zahteva spisek vseh naloženih datotek (GET /api/files). Za želeno datoteko zahteva uvoz (GET /api/import/{file}). Strežnik prebere datoteko, ustvari novo vožnjo (*Journey*) v podatkovni bazi in vanjo uvozi vse podatke (*Vehicle*, *Driver*, *GPSEvent*, *AccEvent*, *OBDEvent*, *MrkEvent*).

V uporabniškem vmesniku se pojavi nova vožnja. Pot vožnje je mogoče prikazati na zemljevidu.

Uporabnik zahteva analizo vožnje (GET /api/analysis/{journeyId}). Strežnik ustvari novo analizo (*Analysis*). Za vsako lego v poti vožnje pridobi podatke o hitrostni omejitvi (Here WeGo REST) in jih zapiše kot del analize (*AnalysisEvent*). Uporabnik lahko rezultate analize prikaže na zemljevidu.

V *GPSEvent* se poleg zemljepisne širine in zemljepisne dolžine shrani še iz njiju izpeljan *location*. Ta je tipa *Point* (*JTS Point*) in se uporablja pri prostorskih poizvedbah.

## 2.3 Uporabljene tehnologije

### 2.3.1 Mobilna aplikacija

#### 2.3.1.1 Večplatformno razvojno okolje: Marmalade SDK

Marmalade SDK [31] je večplatformno razvojno okolje in pogon za igre (ang. *game engine*) [46]. Gre za skupek knjižnic in orodij, ki olajšajo razvoj večplatformnih aplikacij. Filozofija Marmalade SDK je "napiši enkrat, izvajaj povsod" – ista kodna osnova (ang. *codebase*) se prevaja in izvaja na vseh podprtih platformah: Android, iOS, Windows Phone, Windows Desktop, MacOSX, Tizen in druge. Podporo vseh teh platform doseže z uporabo abstrakcijskega sloja (ang. *abstraction layer*) C/C++, ki skrije implementacijske podrobnosti, ki so specifične za vsako platformo (aplikacijski programski vmesniki za abstrakcijski sloj je predznačeni s predpono s3e). Če funkcionalnosti, ki jo potrebujemo, še ni, je mogoče napisati svojo z uporabo EDK (Extension Development Kit).

Razvoj je mogoč v programskih jezikih C/C++, Lua, Objective-C in HTML5/Javascript. Za C/C++ so podprta razvojna okolja Visual Studio (tudi Community Edition), Xcode in Scons.

Mobilno aplikacijo je mogoče v celoti razviti brez uporabe mobilne naprave, tako da za tarčo (ang. *target*) izberemo Windows Desktop Simulator. V tem primeru se aplikacija prevede z uporabo nabora ukazov (ang. *instruction set*) x86 in se izvaja v simulatorju mobilne naprave, ki se požene na razvojnem računalniku (v tem primeru Windows). Testiranje na pravi mobilni napravi se opravi šele na koncu oziroma ko je treba.

Marmalade SDK ponuja več tipov licenc, med njimi tudi t. i. *community*, ki je brezplačna. Omejitve brezplačne verzije so: nezmožnost izklopa prikazovanja uvodnega zaslona (ang. *splash screen*), nezmožnost prevajanja novih razširitev EDK (Marmalade Extension Development Kit) in nezmožnost neposrednega razhroščevanja na mobilni napravi.

Mobilna aplikacija (mobdlog) je v celoti napisana z uporabo Marmalade SDK. Uporabljajo se naslednji Marmalade moduli:

- S3E Accelerometer – za delo s pospeškomerom,
- S3E Device – za pridobivanje informacij o mobilni napravi,
- S3E File – za delo z datotekami,
- S3E Location – za ugotavljanje lege s pomočjo GPS,
- S3E Socket – za mrežne vtičnice (ang. *socket*), ki so potrebne za komunikacijo z OBD2,
- S3E Thread – za delo z nitmi (mogoče zamenjati s pthread API),
- IwHTTP – za komunikacijo HTTP,
- IwNUI – za uporabniški vmesnik,
- zlib – za stiskanje podatkov.

Performance aplikacije, napisane z Marmalade SDK, so podobne rodnim (ang. *native*), če ne še boljše, saj se prevedena C/C++ koda izvaja neposredno na procesni enoti brez vmesnega tolmačenja (ang. *interpreting*).

#### 2.3.1.2 Večplatformno razvojno okolje: Apache Cordova

Apache Cordova [14] je večplatformno razvojno okolje, ki omogoča razvoj mobilnih aplikacij z uporabo HTML5, CSS3 in programskega jezika Javascript.

Aplikacija, napisana s pomočjo Apache Cordove, se izvaja v WebView – minimalnem brskalniku mobilne naprave. Gre za zbirko strani HTML, stilov CSS in kode JavaScript, ki ima s pomočjo vtičnikov Apache Cordova dostop do funkcionalnosti mobilne naprave. S pomočjo vtičnikov lahko dostopamo do datotek, mrežnih vtičnic, GPS-a, pospeškomera itn.

Začetni poskus implementacije mobilne aplikacije (mobdlog) s pomočjo Apache Cordove je bil opuščen. Za razumevanje vzroka moramo poznati nekaj osnov programskega jezika Javascript.

Javascript se izvaja v eni niti v brskalniku, ki bi v primeru blokiranja (ang. *blocking*) kode blokirala celotni brskalnik. Da do tega ne bi prišlo, je jezik po naravi asinhron. Namesto, da klic funkcije blokira, se kontrola nemudoma vrne klicatelju. Ko je rezultat klicane funkcije na voljo, je sporočen s pomočjo povratnega klica (ang. *callback*).

Asinhrona narava jezika zaplete najosnovnejše operacije, kot so npr. branje iz datoteke, čakanje na podatke iz mrežne vtičnice, stiskanje bloka podatkov itd. Koda je posejana s povratnimi klici in kmalu postane težka za razumevanje in vzdrževanje.

Poglavitni razlog za opustitev Apache Cordove pa je tudi nezmožnost uporabe več kot ene niti hkrati oz. pomanjkanje nadzora nad časom izvajanja kode. Zaradi prikazovanja uporabniškega vmesnika, hkratnega branja podatkov iz različnih virov, stiskanja podatkov in pisanja v datoteko bi se hitro zgodilo, da bi določen podatek dobili prepozno oziroma da bi uporabniški vmesnik postal neodziven. Novejše različice komponente WebView sicer

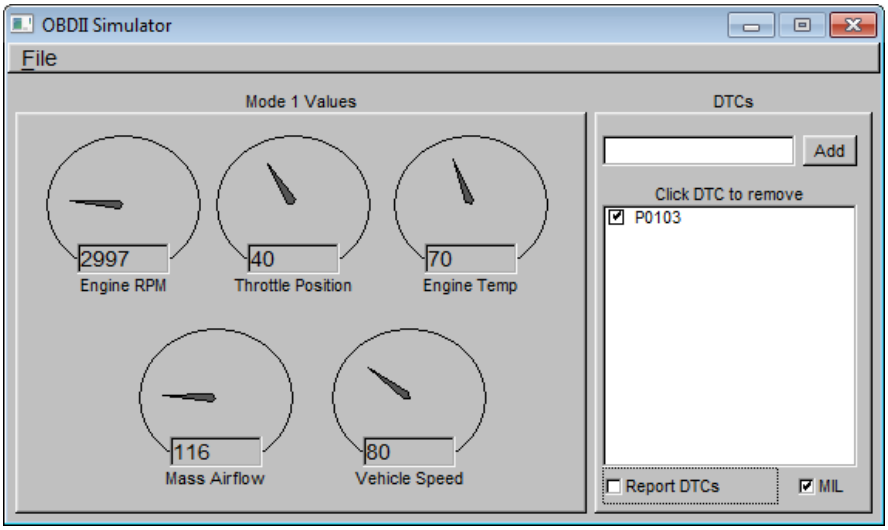
podpirajo vzporedno izvajanje kode s t. i. *webworkerji*, vendar pa v njih ni mogoče uporabljati vtičnikov Apache Cordova.

Ena od prednosti Apache Cordove je v tem, da je enostavno napisati privlačen uporabniški vmesnik. Če bi za uporabniški vmesnik želeli še naprej uporabljati Apache Cordovo, bi bilo treba premakniti osnovno funkcionalnost (branje, pisanje, stiskanje podatkov) v lastni vtičnik Apache Cordova – to pa seveda pomeni, da bi bilo zanj treba uporabiti rodni (ang. *native*) razvoj, s čimer bi izgubili vse prednosti večplatformnega razvoja.

#### 2.3.1.3 Simulator OBD2: OBDSim

OBDSim [33] je programski simulator protokola OBD2. Njegove glavne lastnosti so:

- možnost upravljanj iz ukazne vrstice,
- podpira ELM327 nabor ukazov AT,
- večplatformen – deluje na operacijskih sistemih Linux, Windows in MacOSX,
- podpira simuliranje serijskega vmesnika in vmesnika bluetooth,
- podpira vtičnike – uporabili smo vtičnik uporabniškega vmesnika (Slika 4), s premikanjem kazalcev spreminjamo vrednosti parametrov.




**Slika 4 – Vtičnik OBDSim uporabniškega vmesnika**

Za uporabo OBDSim je treba namestiti še com0com in com2tcp [39]. Prvi omogoča ustvarjanje navideznih vrat COM, ki jih povežemo z OBDSim. Drugi pa ta vrata predstavi preko mrežne vtičnice.

Uporaba simulatorja OBD2 močno pohitri razvoj, saj bi bilo sicer testiranje treba opraviti v bližini prižganega vozila. Kljub uporabi simulatorja je bilo v končni fazi potrebno testiranje na pravem ključu OBD2 v pravem vozilu. Za delovanje komunikacije so bile potrebne manjše spremembe v kodi za inicializacijo ključa OBD2.

### 2.3.2 Strežnik

Strežniški del je v celoti napisan v programskem jeziku Java. Kot razvojno okolje se uporablja JetBrains IntelliJ IDEA [29], programski projekt se upravlja s pomočjo Apache Maven [15].

Sledi opis glavnih tehnologij. Nekatere povezane tehnologije in standardi so omenjeni le na kratko – njihova natančna razlaga presega okvir magistrskega dela.

#### 2.3.2.1 Spletni strežnik in servletni vsebnik: Jetty

Eclipse Jetty [17] je spletni strežnik in vsebnik javax.servlet. Poglavitne lastnosti, zaradi katerih ga uporabljamo, so:

- majhna poraba sredstev,
- je vgradnen (ang. *embeddable*),
- je skalabilen,
- uporablja se na Google App Engine.

V vlogi spletnega strežnika se uporablja za streženje uporabniških strani, kot servletni vsebnik pa služi ogrodju RESTful za spletne storitve RESTEasy.

Možnost vgraditve pomeni, da (vsaj med razvojem) ne potrebujemo ločenega aplikacijskega strežnika. To pohitri postavitev razvojnega okolja in hkrati skrajša razvojne cikle, saj aplikacije ni treba vedno znova postavljati (ang. *deploy*) na aplikacijski strežnik.

Ena od slabosti strežnika Jetty je morda le ta, da je Jetty samo vsebnik in ne ponuja zmožnosti, ki jih imajo pravi aplikacijski strežniki, kot so npr. JNDI, JMX, označbe in podpora za Java EE. V začetni fazi implementacije teh zmožnosti nismo potrebovali.

#### 2.3.2.2 Spletne storitve RESTful: RESTEasy

JBoss RESTEasy [28] je skupek ogrodij, ki olajšajo grajenje spletnih storitev RESTful in javanskih aplikacij RESTful. RESTEasy je implementacija specifikacije JAX-RS 2.0 (Java API for RESTful Web Services). Več o JAX-RS in javanskih strežniških storitvah RESTful lahko preberemo v [9].

Prednosti RESTEasy so:

- s pomočjo označb (ang. *annotations*) JAX-RS se zmanjša količina ponavljajoče se kode (ang. *boilerplate code*), ki je potrebna za implementacijo spletnih virov (ang. *resources*) REST,
- avtomatično preslikovanje objektov v/iz XML, JSON, Atom in drugih formatov,
- avtomatično stiskanje (GZIP) odgovorov (ang. *response*),
- uporaben za implementacijo HTTP odjemalcev (ang. *clients*).

Primer vira RESTful:

```
@GET
@Path("/journeys/{driverId}")
public List<JourneyDetails> listDriversJourneys( @PathParam("driverId") long driverId ) {
    JourneyService journeyService = new JourneyService( entityManager );
    return journeyService.getJourneys( driverId );
}
```

in odgovora JSON, ki ga vir vrne na zahtevo HTTP GET /api/journeys/123:

```
[ { "id": 123, "name": "20160712_081502", "startTime": "2016-06-23T09:40:53.656",
    "vehicleId": 567 }, { "id": 123, ... } ]
```

#### 2.3.2.3 Predmetna obstojnost: Hibernate ORM (Spatial)

Hibernate ORM (Object Relational Mapping) [26] je ogrodje za predmetno relacijsko preslikovanje v programskem jeziku Java. Naloga ORM je preslikovanje predmetno usmerjenega (ang. *object oriented)* razrednega modela v relacijsko bazo. Hibernate preslikuje javanske predmete v podatke v tabelah relacijske baze. Omogoča obstojnost predmetov, ne da bi morali ročno pretvarjati rezultate poizvedb SQL v predmete in obratno.

Hibernate ima svoj rodni (ang. *native*) aplikacijski vmesnik, je pa tudi implementacija specifikacije JPA (Java Persistence API) [36]. Več o JPA si lahko preberemo v knjigi [6].

JPA deluje nad entitetami (ang. *entity*), med katerimi so definirane relacije. Možne relacije so: ena na ena, ena na mnogo in mnogo na mnogo. Preslikovanje in relacije lahko definiramo s pomočjo konfiguracije datoteke XML ali, še bolje, z uporabo javanskih označb v kodi sami.

Za poizvedovanje se uporablja poizvedbeni jezik JP QL (Java Persistence Query Language). JP QL sintaktično spominja na SQL, vendar je drugačen. Je namreč jezik za poizvedovanje nad entitetami in predmeti in ne nad tabelami in vrsticami. Ena od prednosti JP QL je njegova prenosljivost – poizvedbe se prevedejo v večino znanih narečij SQL. Druga prednost pa je, da je treba za njegovo uporabo poznati samo domenski model in entitete, ne pa tega, v kateri tabeli je shranjena katera entiteta.

Prednosti uporabe JPA so:

- neodvisnost od implementacije ORM,
- deluje v Java SE, EE ali vsebnikih OSGi,
- uporaba označb za definicijo relacij med entitetami,
- uporaba JP QL za povpraševanja.

Primer entitete JPA za dogodek GPS (*GPSEvent*):

```
@Entity
public class GPSEvent {
    @Id
    @GeneratedValue
    private Long id;

    @OneToOne
    @JoinColumn(name = "journey_id")
    private Journey journey;

    private Date date;

    private Point location; // Point is a JTS type
    ...
}
```

Hibernate je bil izbran tudi zato, ker ponuja podporo za prostorske podatke z razširitvijo (ang. *extension*) Hibernate Spatial. Hibernate Spatial [27] omogoča delo s prostorskimi podatki na standardiziran način. Podpira več prostorskih baz in skrije posebnosti posamezne baze.

Hibernate Spatial za svoj geometrijski model uporablja JTS (Java Topology Suite) API [30]. JTS je implementacija OpenGIS Simple Features Implementation Specification for SQL v1.1 (skrajšano SFS) [34]. Večina relacijskih baz s prostorsko podporo implementira OpenGIS SFS.

SFS podpira geometrične tipe, kot so npr. točke (Points), zbirke točk (MultiPoints), nize daljic (LineStrings), poligone (Polygons) in druge. Podpira operacije, kot so računanje ploščine in obsega, računanje razdalje med različnimi geometrijskimi predmeti (geometrijami, ang. *geometry*), ugotavljanje, ali je geometrija znotraj neke razdalje, predikate *vsebuje*, *pokriva*, *seka*, *prečka*, *prekriva*, *se dotika* in *je enak* in drugo. Za podroben opis glej [30].

#### 2.3.2.4 Podatkovna baza: PostgreSQL/PostGIS

PostGIS [38] je prostorska razširitev za relacijsko bazo PostgreSQL – dodaja podporo za prostorske/geometrijske predmete. PostGIS implementira specifikacijo SFS.

Prednosti PostGIS:

- uporablja PostgreSQL: odprta koda, deluje na različnih operacijskih sistemih,
- PostGIS je dobro podprt v Hibernate Spatial.

Ker uporabljamo Hibernate JPA, je mogoče podatkovno bazo enostavno zamenjati z drugo, ki ponuja podobne zmožnosti.

#### 2.3.2.5 Spletno aplikacijsko ogrodje: AngularJS

Googlov AngularJS [12] je ogrodje Javascript za grajenje dinamičnih enostranskih spletnih aplikacij (ang. *single page application*, skrajšano SPA). Temelji na oblikovalnem vzorcu (ang. *design pattern*) MVC (ang. *Model-View-Controller*, slo. Model-Pogled-Kontrolor).

AngularJS prebere spletno stran HTML, ki uporablja posebne zaznamke (ang. *tags*) in jo prevede v model. Model je dostopen neposredno preko spremenljivk Javascript. Prirejanje vrednosti spremenljivki v modelu povzroči spremembo pogleda in obratno (npr. sprememba vnosnega polja spremeni vrednost spremenljivke, ki je s tem poljem povezana) – temu pravimo dvosmerno povezovanje podatkov (ang. *data-binding*) [45].

Za razliko od večine spletnih aplikacij, kjer se HTML generira na strežniku in prikaže v brskalniku, se pri uporabi AngularJS stran generira v odjemalčevem brskalniku. Strežnik služi le kot strežnik podatkov, ki jih taka stran zahteva.

AngularJS omogoča:

- strukturirano grajenje spletnih aplikacij – koda je organizirana v modulih,
- uporabo pogledov in kalupov – pogled je projekcija modela na kalup,
- inicializacijo modela s pomočjo kontrolorjev,
- uporabo dosega (ang. *scope*) kot povezavo med modelom, pogledom in kontrolorjem,
- definiranje svojih zaznamkov HTML s pomočjo direktiv AngularJS,
- vrivanje odvisnosti (ang. *dependency injection*),
- oblikovanje prikazanih podatkov s pomočjo filtrov,
- prikazovanje različnih pogledov s pomočjo usmerjanja (ang. *routing*),
- definicijo lastnih storitev (ang. *services*) npr. za dostop do vmesnika REST.

Na spletu bomo pogosto zasledili ogrodja Javascript ali knjižnice, ki podpirajo AngularJS. Različice AngularJS prepoznamo po tem, da imajo v imenu niz `ng`. Npr. Google Maps s podporo AngularJS se imenuje maps-ng.

Uporabniški vmesnik spletne aplikacije (mobdapp) je v celoti napisan z uporabo AngularJS. Za osnovo smo uporabili projekt vadnice na [11].

#### 2.3.2.6 Vizualizacija, zemljevidi: Here WeGo Javascript API

Here WeGo [25] (podjetje HERE Global B.V. je v lasti konzorcija Audi, BMW, Daimler) je aplikacija za zemljevide in navigacijo za mobilne naprave in splet.

Here WeGo ponuja Javascript [23] knjižnico, s katero lahko:

- prikazujemo zemljevide (navadni in satelitski prikaz),
- navigiramo po zemljevidu (približevanje, oddaljevanje, premikanje),
- prikazujemo zaznamke na zemljevidu,
- rišemo črte in poligone,
- sprašujemo po informaciji o neki legi (ang. *geocoding*),
- načrtujemo poti.

Uporabniški vmesnik uporablja zemljevide Here WeGo za vizualizacijo podatkov. Here WeGo uporabljamo posredno preko knjižnice angularjs-heremaps [20], ki doda podporo vmesniku AngularJS API Here WeGo Javascript.

Razlog za uporabo Here WeGo je predvsem ta, da Here WeGo že uporabljamo za pridobitev podatkov o hitrosti. Po potrebi lahko za prikazovanje uporabimo Google Maps oziroma njegovo različico AngularJS maps-ng.

Za uporabo Here WeGo se je treba prijaviti. Obstajajo različni plani, tudi brezplačni, ki omogoča 50.000 transakcij na mesec.

#### 2.3.2.7 Podatki o hitrostnih omejitvah: Here WeGo REST API

Here WeGo poleg vmesnika Javascript API ponuja tudi REST API [24]. S pomočjo REST API lahko dobimo podatke o hitrostni omejitvi na določenem odseku. Za to uporabimo vmesnik Routing REST API, getLinkInfo.

Strežnik uporablja podatke o hitrostni omejitvi za potrebe analize voženj. Da bi zmanjšali število povpraševanj, so rezultati zahtev shranjeni v posebni entiteti (*HERECache*), ki se uporablja kot predpomnilnik (ang. *cache*). Ob ponovnem povpraševanju po isti legi se vrne vrednost iz predpomnilnika.

## 2.4 Tehnični izzivi

### 2.4.1 Večplatformni razvoj mobilnih aplikacij

Razvoj mobilnih aplikacij je otežen, ker ima vsaka platforma svoj jezik in razvojno okolje. Za razvoj aplikacij iOS se uporablja razvojno okolje Xcode s programskim jezikom Objective-C. Na platformi Android se uporabljata Android Studio in programski jezik Java. Na Windows Phone pa Visual Studio in programski jezik C#.

Vse tri platforme sicer omogočajo uporabo C/C++ vendar se aplikacijski programski vmesniki za dostop do funkcionalnosti naprave med seboj razlikujejo. To pomeni, da moramo del kode (npr. uporabniški vmesnik, dostop do naprave GPS, dostop do pospeškomera itd.) še vedno napisati (in testirati) za vsako platformo posebej.

Rodni razvoj na treh platformah najmanj potroji stroške razvoja. Potrebujemo tri razvijalce (težko najdemo nekoga, ki dobro pozna vse tri platforme), vsako novo zmožnost je treba implementirati in testirati trikrat, vzdrževanje in podpora sta dražji, verjetnost, da ima aplikacija hrošča, je večja itn.

Zaradi opisanega je bolj smiselno uporabljati katero od ogrodij za večplatformni razvoj mobilnih aplikacij. Ta ogrodja poenotijo razvoj večplatformnih mobilnih aplikacij – eno razvojno okolje, ena kodna osnova (ang. *codebase*).

Obstaja več večplatformnih razvojnih okolij. Razlikujejo se glede na namen uporabe (igre ali aplikacije), po tem, kako rešujejo problem prenosljivosti, in po programskem jeziku (Javascript, C/C++, Lua, itn), ki ga uporabljajo. Za potrebe magistrskega dela smo si podrobneje ogledali dve: Marmalade SDK in Apache Cordova. Opisani sta v 2.3.1.1 in 2.3.1.2

Omenimo še, da obstajajo tudi primeri, v katerih je rodni razvoj neizbežen. To je v primeru, ko želimo na neki platformi imeti rodni videz (ang. *native look and feel*). V tem primeru je treba vsaj uporabniški vmesnik implementirati z rodnimi orodji.

### 2.4.2 OBD2

Pri implementaciji komunikacije z ELM327 smo si pomagali z dokumentacijo na [18]. Zelo pomembno je postavitveno zaporedje ukazov. Uporabili smo:

1) ATZ (reset all, ponastavitev v začetno stanje),
2) ATE0 (echo off, izključi odmev),
3) ATL0 (linefeeds off, izključi uporabo znakov za novo vrstico),
4) ATS0 (spaces off, izključi uporabo presledkov),
5) ATH0 (headers off, izključi izpisovanje glav),
6) ATAT1 (adaptive timinig, časovno prilagajanje),
7) ATSP0 (set auto protocol, avtomatična izbira protokola).

Kot smo omenili že v 2.1.1.3, je za izračun vrednosti iz šestnajstiških kod treba poznati ustrezno formulo, ta pa se lahko razlikuje od proizvajalca do proizvajalca. Mobilna aplikacija zato shranjuje (surovo) šestnajstiško vrednost. Odgovornost interpretacije in izračuna vrednosti prenese na strežnik. Strežnik lahko hrani "znanje" o različnih vozilih in ga uporabi za pravilen izračun vrednosti.

Dodaten problem pri izračunu končne vrednosti je, da moramo poznati proizvajalca in tip vozila. Nobeden od standardnih PID-ov ne vrne neposredno teh podatkov, dobimo pa jih lahko z dekodiranjem kode VIN (ang. *okrajšava za Vehicle Identification Number*). Iz kode VIN lahko ugotovimo: model vozila, proizvajalca, leto proizvodnje, državo proizvodnje, zaporedno številko vozila, obliko šasije, tip zavor, pogon (prednji, zadnji), tip motorja, velikost rezervoarja za gorivo, obračalni radij. VIN lahko dekodiramo s pomočjo spletnih strani, kot je npr. Vindecoder [43].

VIN je na voljo na PID-u 0902, vendar ni nujno, da ta PID obstaja. Predvsem v starejših vozilih tega podatka ni mogoče prebrati. V tem primeru moramo podatek o tipu vozila vnesti ročno ali uporabiti katero drugo metodo za prepoznavanje vozila (mogoče prepoznavanje po najdenih OBD2 PID-ih in vrednostih – neke vrste prstni odtis OBD2).

Zaradi pomanjkanja testnih vozil je mobilna aplikacija uspešno testirana samo na enem vozilu (Fiat Punto 2006). Branje OBD2 smo poskusili še na dveh starejših vozilih s priključkom OBD2 (MB A160 2001/bencin, Peugeot 706 2001/dizel), vendar brez uspeha. Tudi mobilna aplikacija Torque, ki ima najboljšo podporo za branje OBD2, na teh dveh vozilih ni delovala. Vozili sta očitno izdelani pred uvedbo standarda.

### 2.4.3 Analiza podatkov

#### 2.4.3.1 Hitrost

Pri analizi vožnje na podlagi hitrostnih omejitev naletimo na naslednje težave:

- pomanjkanje podatkov o hitrostni omejitvi na določenih cestnih odsekih,
- nenatančnost podatkov GPS,
- nezadostnost podatkov GPS,
- pomanjkanje podatkov GPS (predori).

Kot je omenjeno v 2.1.2.1, Here WeGo nima podatkov o hitrostnih omejitvah na določenih cestnih odsekih, npr. v manjših ulicah v mestu. V tem primeru moramo te cestne odseke izključiti iz izračuna ocene ali pa manjkajoče podatke poiskati drugod (npr. v OSM).

V nekaterih primerih se zgodi, da Here WeGo vrne hitrostno omejitev za vzporedno cesto oziroma nadvoz. Prvi razlog je (ne)natančnost lege GPS, nad katero nimamo vpliva, drugi pa ta, da Here WeGo nima oziroma ne upošteva podatka o nadmorski višini ceste. V primeru nadvozov in podvozov vrne omejitev višje ceste. Napake lahko zmanjšamo tako, da upoštevamo še identifikacijsko številko ceste, po kateri se vozilo premika. Če se ta spremeni le za trenutek, lahko sklepamo, da gre za napako.

Idealna rešitev bi bila, če bi lahko iz koordinat GPS ugotovili, po kateri cesti se vozilo vozi in v katero smer. Za to bi morali cesto obravnavati kot poligon. Tega ne omogočata niti Here WeGo niti Google Maps. Slednji sicer ponuja funkcionalnost "prilepi na cesto", vendar rezultata algoritma ne moremo preveriti. Edini (neplačljiv) zemljevid s temi podatki je OSM. Za uporabo bi morali razviti svoje algoritme, ki pa niso enostavni.

Zadnja težava je pomanjkanje podatkov o hitrosti, ko GPS ne sprejema signala – npr. v predorih. V tem primeru lahko:

- uporabimo hitrost, ki jo preberemo preko vmesnika OBD2 (podporo temu bi morali dodati v mobilno aplikacijo),
- hitrost izpeljemo s pomočjo pospeška (preveriti bi morali natančnost takega načina računanja hitrosti v odvisnosti od prevožene razdalje).

#### 2.4.3.2 Pospešek

Zaradi obsežnosti magistrskega dela je glede pospeškov implementirano samo zbiranje podatkov na mobilni aplikaciji in njihovo shranjevanje v bazo. Analize pospeškov bi se lotili na naslednji način:

1) Interpolacije lege: če želimo dobiti podatek o legi za vsak vzorec (ang. *sample*) pospeškomera, moramo lego izračunati z interpolacijo. Najenostavneje je uporabiti linearno interpolacijo med dvema legama. Večjo natančnost dobimo z upoštevanjem hitrosti ali celo pospeška.

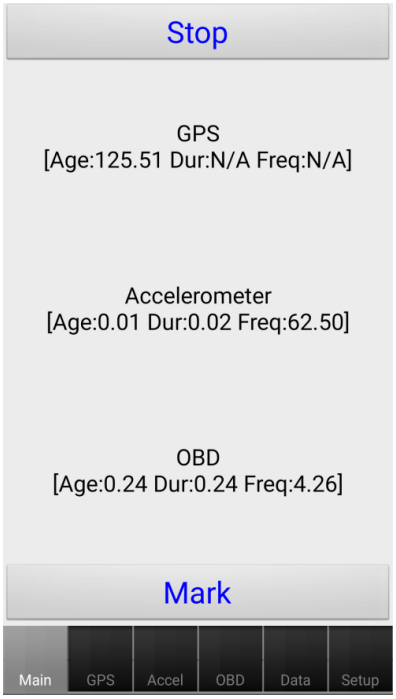
2) Ugotavljanje orientacije naprave; za pravilno interpretacijo pospeška moramo vedeti, kako je mobilna naprava obrnjena glede na smer vožnje.
3) Ugotavljanje pragov, ki predstavljajo prekomerne pospeške.
4) Ugotavljanje stanja vozišča. Prekomerne vertikalne pospeške shranjujemo v podatkovno bazo. Tako lahko ugotovimo, na katerih odsekih je največ tresljajev. S to informacijo lahko dodatno izboljšamo oceno vožnje. Hitra vožnja po takih odsekih poslabša oceno voznika.

## 2.5 Prikaz rešitve

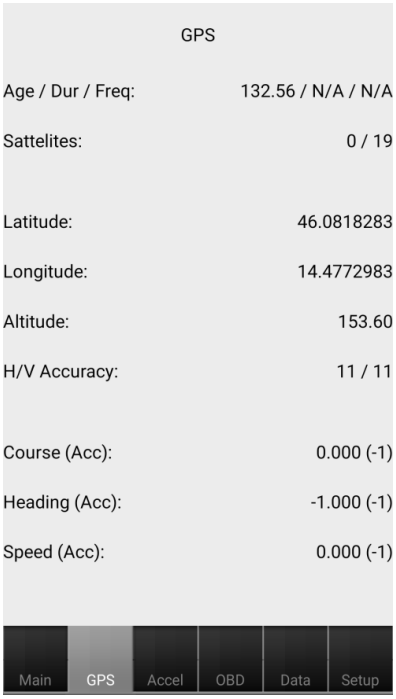
### 2.5.1 Mobilna aplikacija

Spodnje slike prikazujejo mobilno aplikacijo med delovanjem:

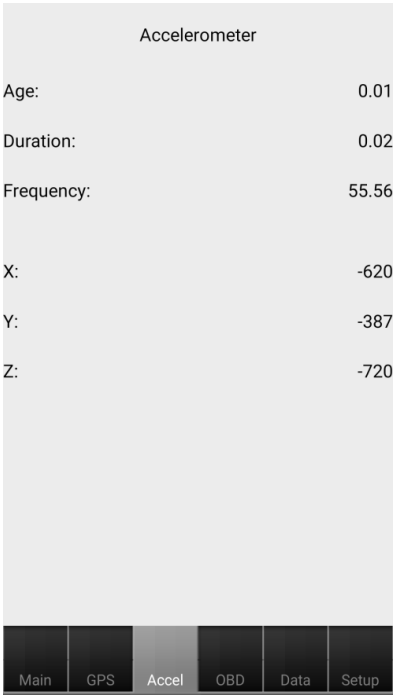
- Slika 5: Glavni pogled, ki ga uporabljamo za pričetek (*Start* – ni prikazan na sliki) in zaključek snemanja (*Stop* na sliki) podatkov. Z uporabo ukaza *Mark* lahko zabeležimo nek dogodek. Sredinski del prikazuje stanje modulov *GPS*, *Accelerometer* in *OBD*: *Age* predstavlja čas, odkar smo prebrali zadnji podatek; *Dur* (duration) pomeni čas prejšnjega branja; *Freq* predstavlja frekvenco branja. Med časom snemanja ostane zaslon naprave prižgan.
- Slika 6: Podrobnosti o delovanju modula GPS. Prikazani so število uporabljenih/vidnih satelitov (*Sattelites*), zemljepisna širina (*Latitude*), zemljepisna dolžina (*Longitude*), nadmorska višina (*Altitude*), vodoravna in navpična natančnost (*H/V Accuracy*), kurz (*Course*), smer (*Heading*) in hitrost (*Speed*). *Acc* predstavlja natančnost podatka.
- Slika 7: Podrobnost o delovanju pospeškomera. *X*, *Y* in *Z* predstavljajo pospešek po oseh. Pri napravi v mirovanju je vektorski seštevek vseh treh vrednosti enak 1000, kar predstavlja gravitacijski pospešek na površju Zemlje (1,0 $m/s^2$) .
- Slika 8: Podrobnost modula OBD2. S pritiskom na *Scan* ugotovimo, katere PID-e podpira vozilo. Vrednosti označenih parametrov se bodo shranjevale. Polega vsakega parametra je še njegov opis in vrednost v surovi šestnajstiški obliki. *Save* shrani nastavitve.
- Slika 9: Podrobnosti o shranjenih podatkih. *Status* nam pove, ali se podatki snemajo ali ne. *Data size* predstavlja shranjeno količino podatkov (v bajtih). *Data files* predstavlja število shranjenih datotek. *Upload speed* in *Upload size* predstavljata hitrost prenosa in količino prenesenih podatkov. Podatke je mogoče pošiljati, ko snemanje ni aktivno.
- Slika 10: Nastavitve mobilne aplikacije. Mogoče je nastaviti frekvenco vzorčenja pospeškomera (*Accelerometer Sampling Rate*, Hz), pogostost beleženja lege (*GPS refresh interval*, sekunde), naslov IP in vrata *(OBD IP, OBD Port*) ključa OBD2, identifikacijsko številko vozila (*VIN*) in naslov (*URL*) za pošiljanje podatkov (ni vidno na sliki).



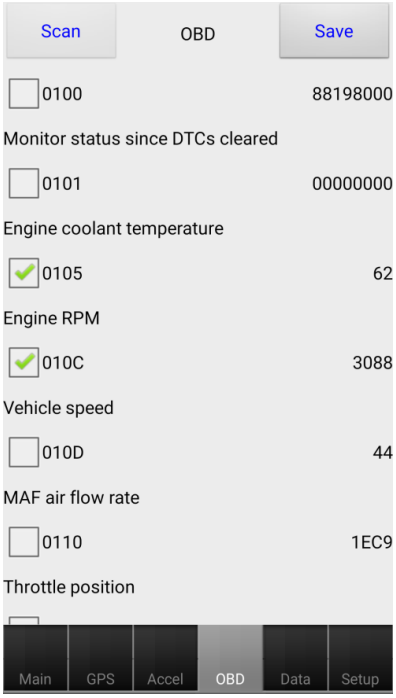

Slika 5 – Glavni pogled



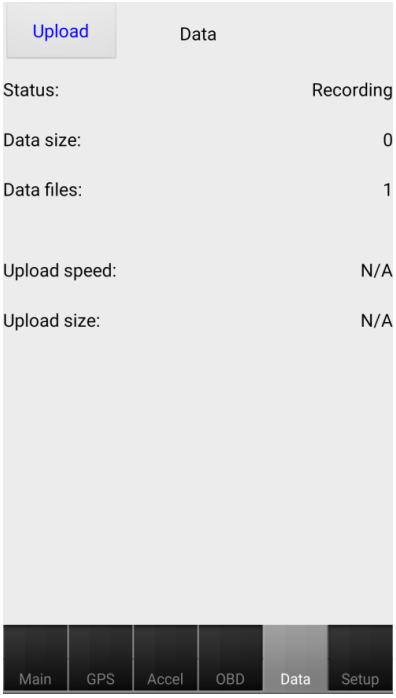

Slika 6 – GPS



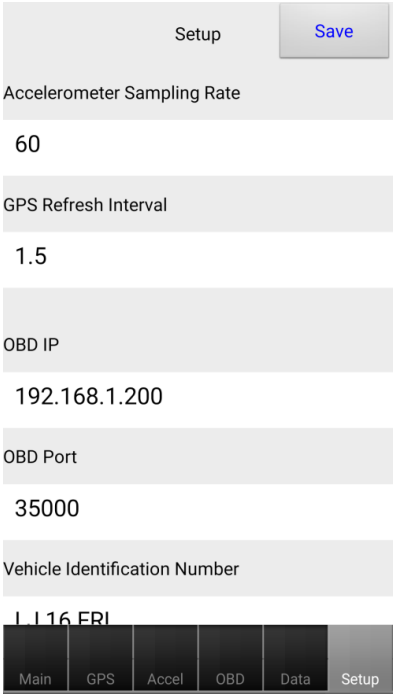

Slika 7 – Pospeškomer




Slika 8 – OBD2




Slika 9 – Pošiljanje podatkov

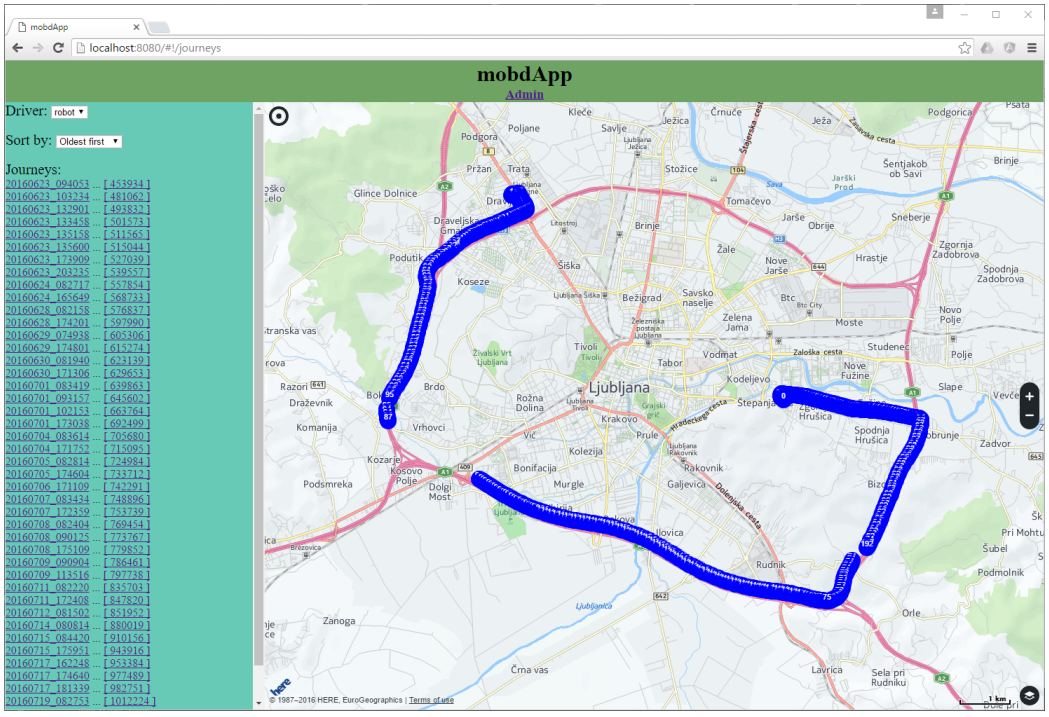



Slika 10 – Nastavitve

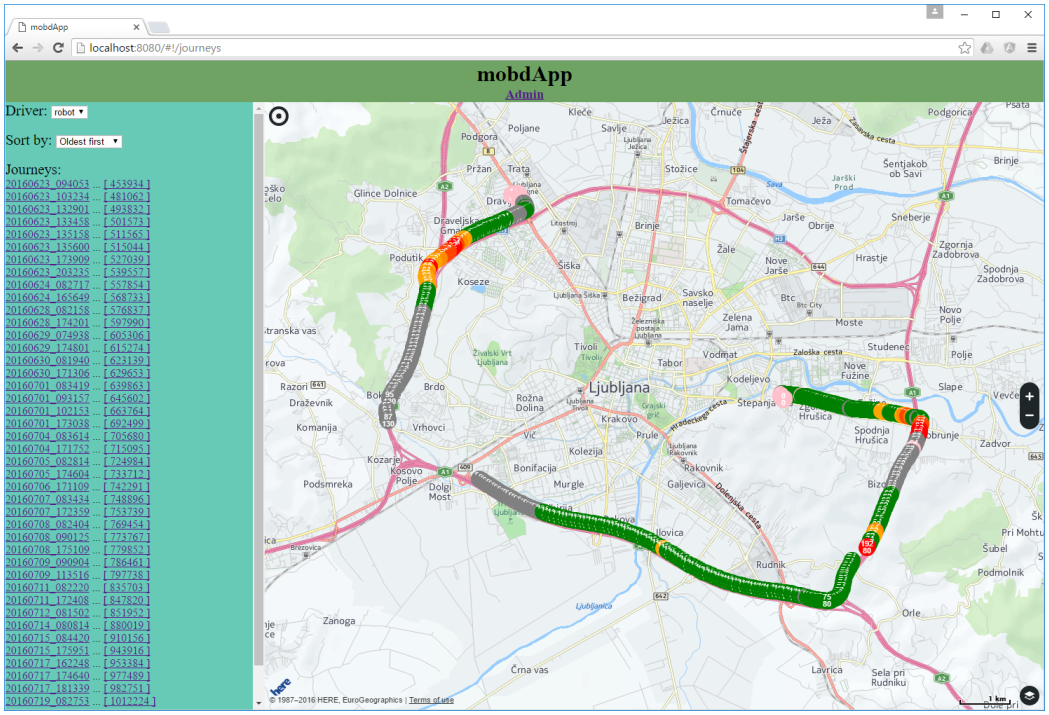
### 2.5.2 Strežnik

Spodnje tri slike prikazujejo prototip uporabniškega vmesnika:

- Slika 11: Z modro barvo je prikazana vožnja. V modrem zaznamku je prikazana samo hitrost GPS. Na vrhu (temno zeleno) vidimo ime aplikacije in povezavo *Admin*, kjer je mogoče izbrati datoteke za uvoz (glej Slika 13). Na levi strani lahko izberemo voznika (*Driver*), vrstni red prikazovanja voženj (*Sort by*), vožnjo samo (npr. 20160623_094053) in njeno analizo (v oglatih oklepajih, npr. [ 453934 ]).
- Slika 12: Analiza hitrosti za vožnjo s prejšnje slike. Pomen barv:
  - siva: hitrost je 30 km/h pod omejitvijo,
  - zelena: hitrost je med 30 km/h pod omejitvijo in 5 km/h nad omejitvijo,
  - oranžna: omejitev je prekoračena za več kot 5 km/h,
  - temno oranžna: omejitev je prekoračena za več kot 10 km/h,
  - rdeča: omejitev je prekoračena za več kot 20 km/h,
  - rožnata: podatka o hitrostni omejitvi ni (vrnjena omejitev je 0).
- Slika 13: Z izbiro datoteke (*Files*) zahtevamo njen uvoz.

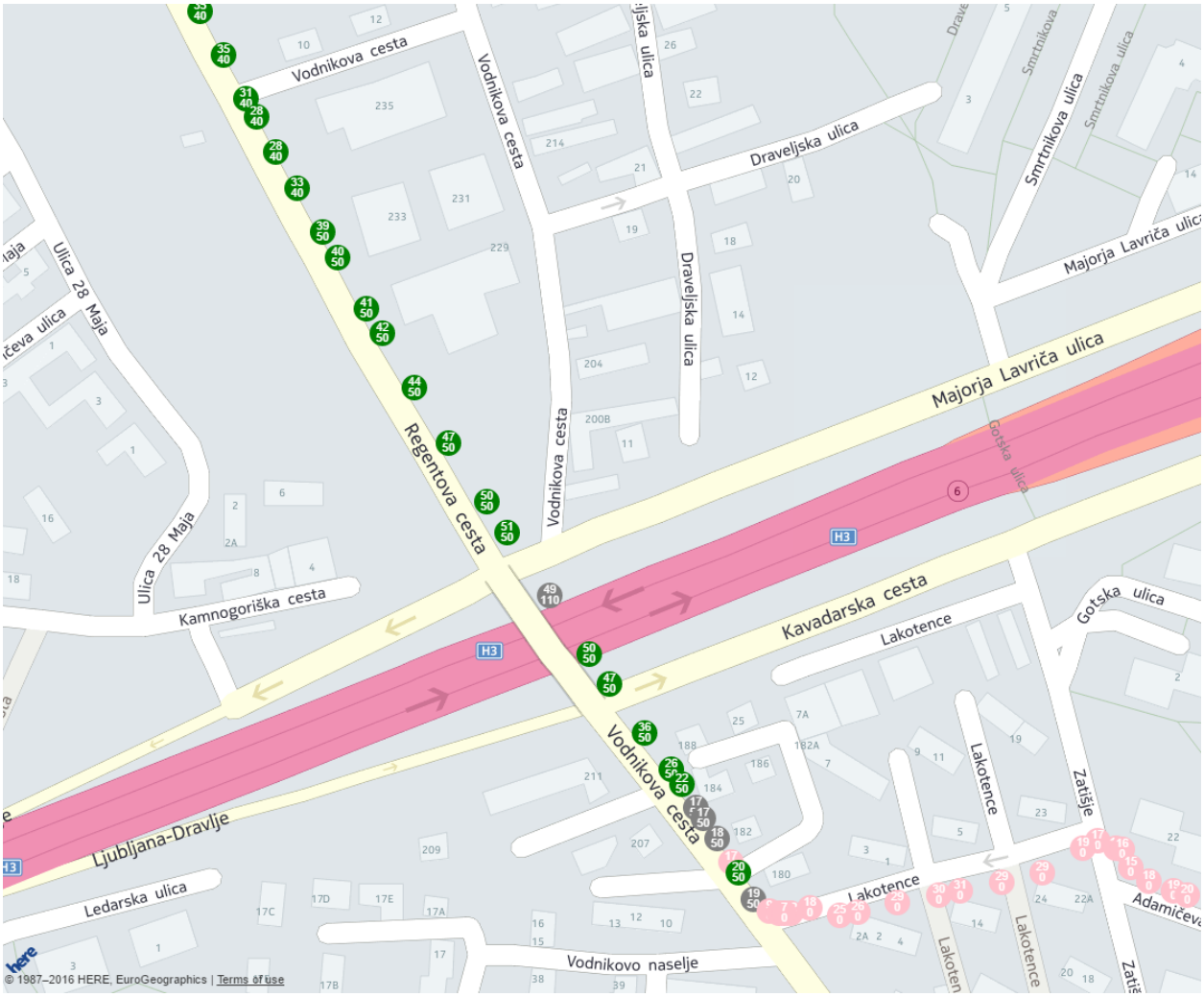



**Slika 11 – Uporabniški vmesnik: prikaz poti**




**Slika 12 – Uporabniški vmesnik: analiza hitrosti**

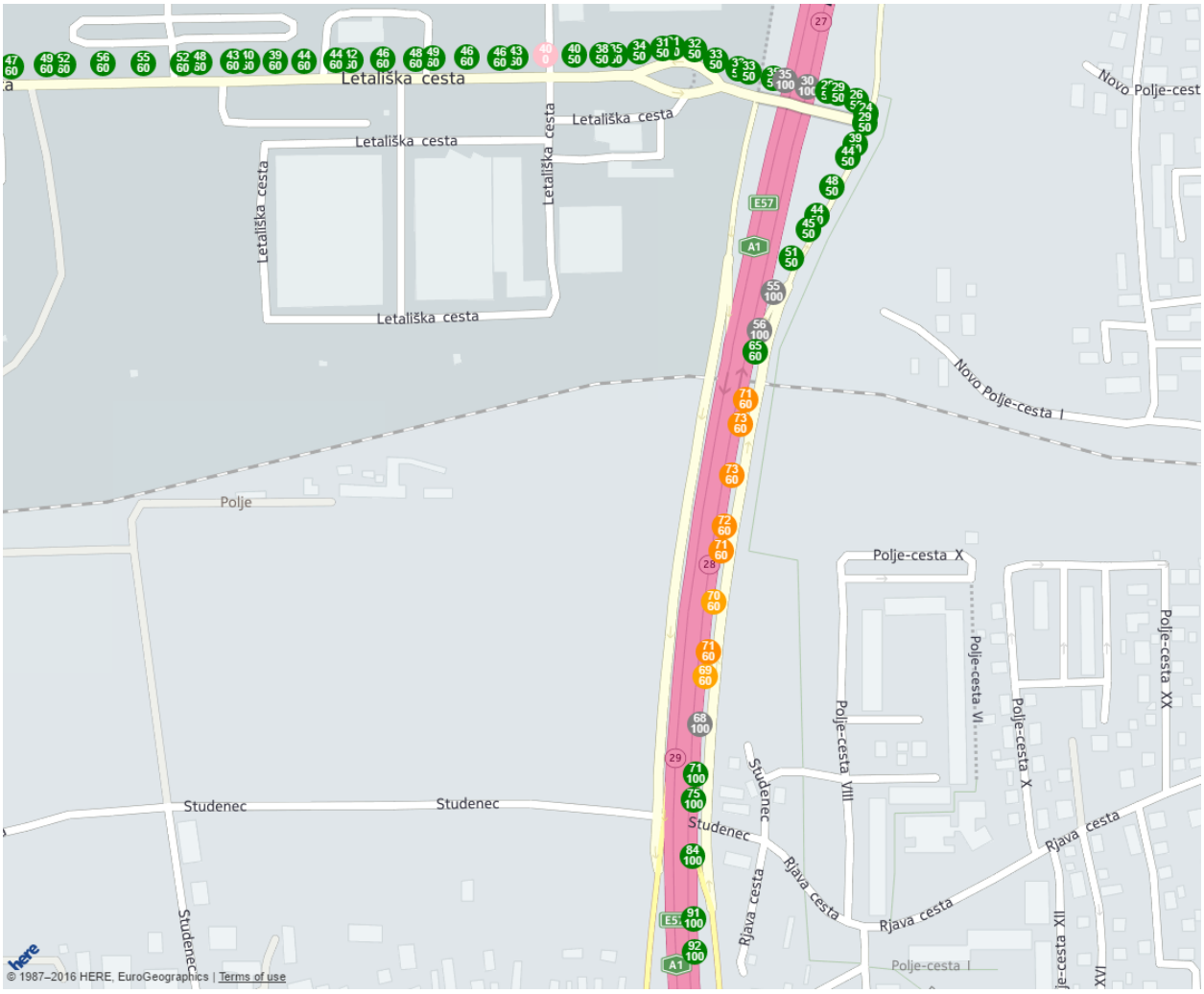

**Slika 13 – Uporabniški vmesnik: datoteke za uvoz**




**Slika 14 – Analiza poti: Vodnikova cesta**

Slika 14 prikazuje:

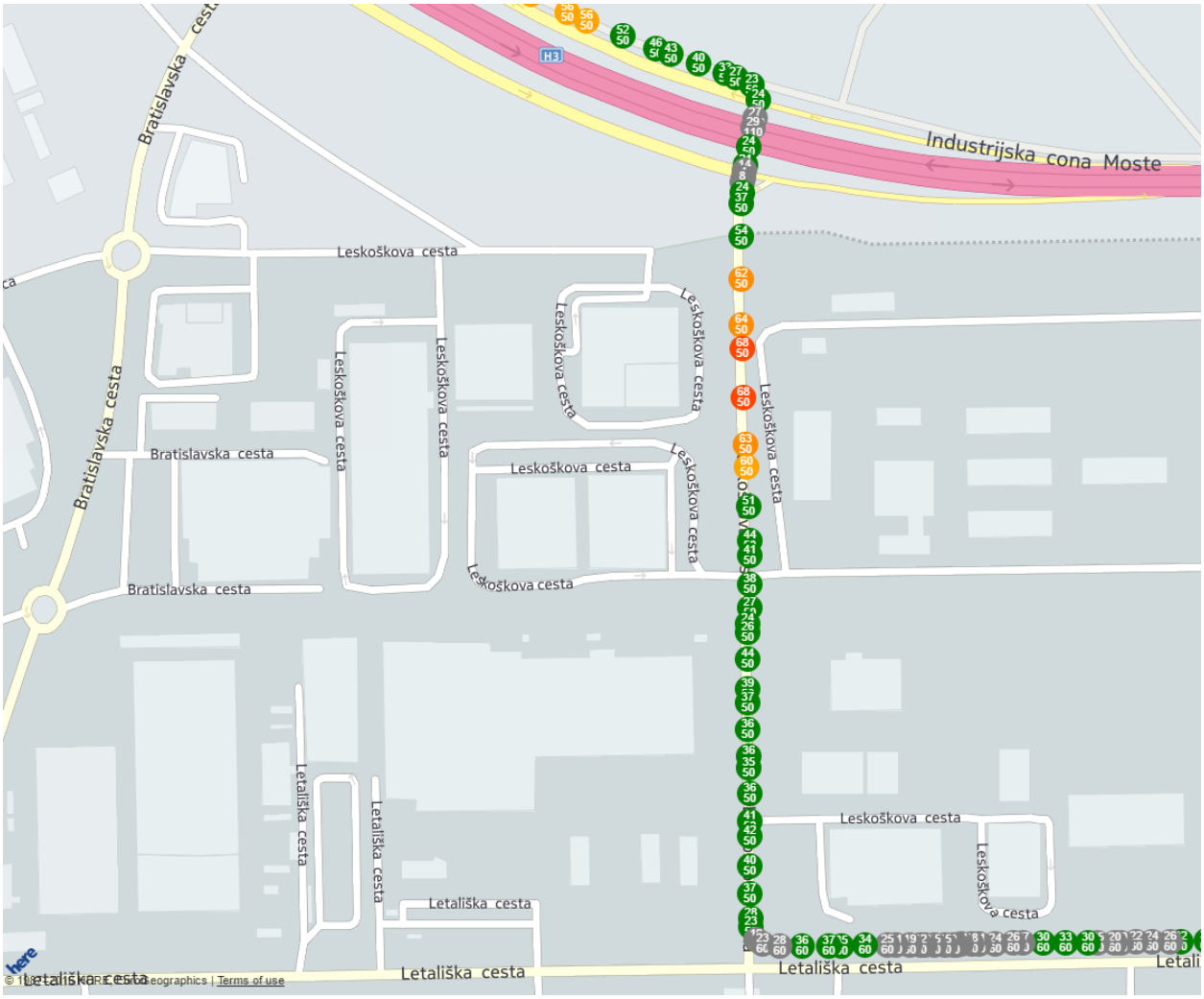
- pomanjkanje podatkov o hitrostni omejitvi (rožnata barva),
- napako pri pridobivanju hitrostne omejitve zaradi nenatančnosti lege GPS:
  - pri prečkanju nadvoza nad H3 (siva barva),
  - pri vožnji mimo stranske ulice na Vodnikovi cesti med stavbama 180 in 182 (rožnata barva).




**Slika 15 – Analiza poti: vzhodna obvoznica**

Slika 15 prikazuje:

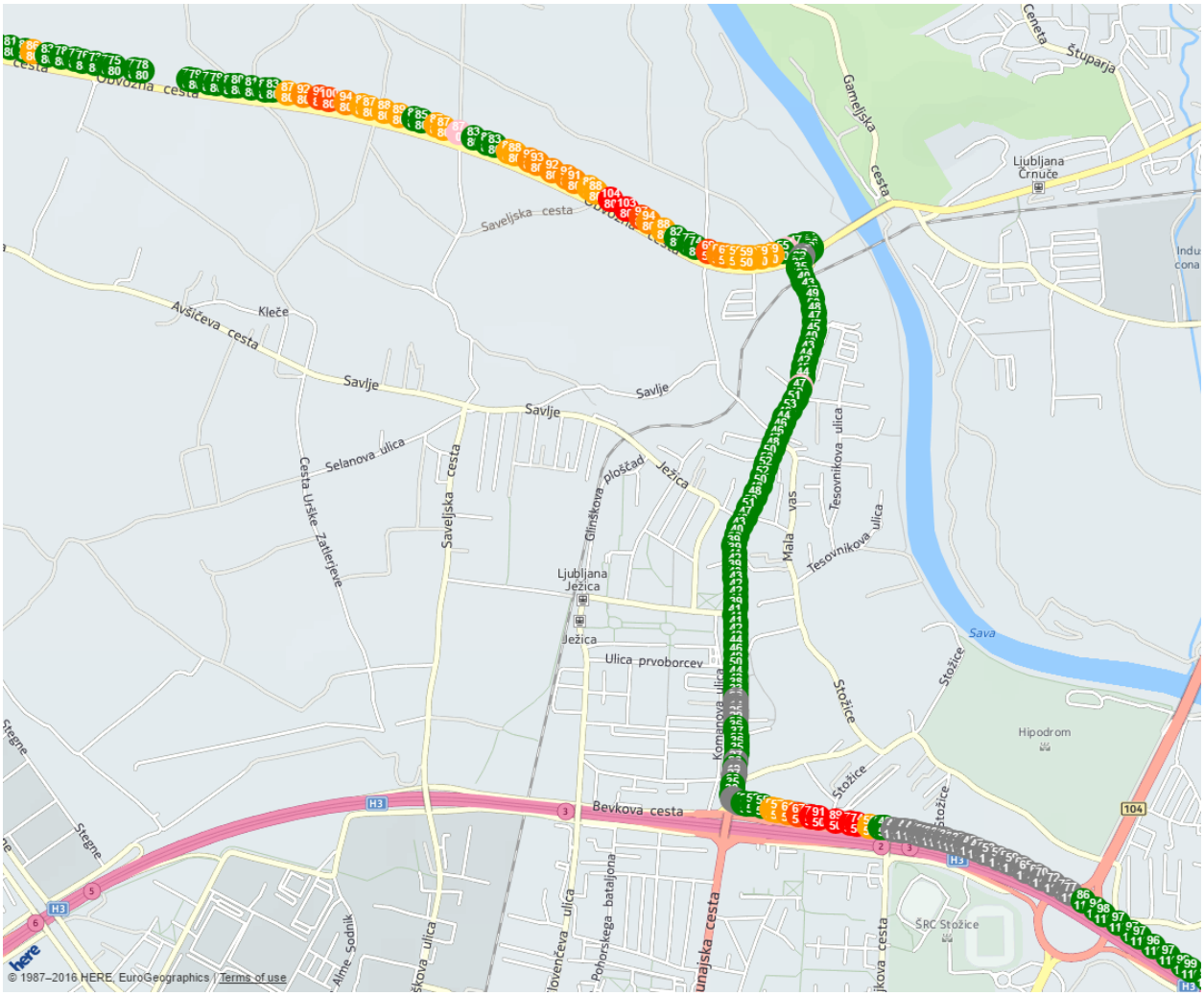
- napako pri pridobivanju hitrosti zaradi nenatančnosti lege GPS:
  - vozilo potuje severno po A1/E57, podatki o omejitvi pa so od vzporednice (oranžna barva),
  - vozilo je že zavilo levo proč od obvoznice A1/E57, podatki o omejitvi pa so še vedno za obvoznico (siva barva, zaznamki 55/100 in 56/100),
  - vozilo prečka obvoznico čez nadvoz, podatki o omejitvi pripadajo A1/E57 (siva barva, 35/100, 30/100).




**Slika 16– Analiza poti: Leskoškova cesta**

Slika 16 prikazuje:

- počasno vožnjo po Letališki cesti zaradi gneče na semaforju na križišču z Letališko cesto (siva barva, spodaj desno),
- prekoračitev omejitve na Leskoškovi cesti (oranžna in rdeča barva, sredina slike),
- napako pri pridobivanju omejitve pri vožnji čez nadvoz nad obvoznico H3 (siva barva, zgoraj).




**Slika 17 – Analiza poti: Ljubljana – sever**

Slika 17 prikazuje:

- počasno vožnjo po obvoznici H3 (siva barva, spodaj levo),
- prehitro vožnjo pri zapuščanju obvoznice H2 (oranžna in rdeča barva, sredina spodaj),
- prehitro vožnjo po Obvozni cesti (oranžna in rdeča barva, zgoraj),
- napako pri pridobivanju hitrostne omejitve pri vožnji mimo stranske ceste (rožnata barva, sredina zgoraj).

# 3 Sklepne ugotovitve

Podatkovna platforma za razvoj aplikacij na primeru uporabe spremljanja vozil in voznikovih navad zahteva širok spekter znanj. Ta segajo od skoraj strojnih, kot so poznavanje avtomobilske samodiagnostike (OBD2), senzorjev mobilne naprave in fizike vožnje, do programskih, kot so razvoj večplatformnih mobilnih aplikacij, razvoj spletnih storitev in uporaba (prostorskih) podatkovnih baz. Pri slednjih smo primorani uporabiti različne programske jezike, kot so C/C++, Java in Javascript, ter različne tehnologije, kot so Marmalade SDK, Jetty, RESTEasy, Hibernate, PostgreSQL, PostGIS, AngularJS in Here WeGo.

Predstavljena podatkovna platforma ne skuša skriti potrebe po teh znanjih, saj bi na ta račun izgubila splošnost in performančnost. Namesto tega z uporabo standardov in s premišljeno izbiro tehnologij vzpostavi osnovo, ki uporabnika (razvijalca, analitika) uvede v področje in mu ublaži začetno učno krivuljo. Z uporabo besedila magistrskega dela kot uvoda v področje in kode kot delujočega primera lahko tudi nestrokovnjak hitro pride do rezultatov.

Težave, s katerimi smo se srečali, so bile povezane z učenjem in obvladovanjem področja in tehnologij, predvsem pri izbiri večplatformnega razvojnega okolja, uporabi ključa OBD2 in analizi podatkov.

Avtor ocenjuje, da bi z uporabo te podatkovne platforme pri postavitvi okolja, učenju in raziskovanju področja prihranili od šest do 12 inženirskih mesecev. Za seznanjanje s celotno platformo in z uporabljenimi tehnologijami bi potrebovali mesec do dva.

Zaradi omejenega časa, ki je bil na voljo za razvoj platforme, so določeni deli prototipne narave, zato predlagamo naslednje spremembe oz. izboljšave:

- na mobilni aplikaciji:
  - refaktoriranje (ang. *refactoring*) z namenom zmanjšanja ponavljajoče se kode (npr. delo z nitmi),
  - samodejno branje VIN prek vmesnika OBD2,
  - zamenjava formata zapisovanja podatkov JSON s CSV (predvidevamo 30% zmanjšanje prostora, potrebnega za stisnjene podatke),
- na strežniku:
  - izboljšati analizo hitrosti zaradi nenatančnosti naprave GPS (upoštevanje ceste, po kateri se vozilo premika),
  - implementirati analizo pospeškov,
  - implementirati manjkajoče vmesnike REST (npr. brisanje voženj),
  - dodati nove vire CRUD REST (vehicle, driver, user),
  - implementirati registracijo novega uporabnika na strežniku, avtentikacijo in konfiguracijo mobilne aplikacije,
  - izboljšati uporabniški vmesnik z uporabo AngularUI [13], npr. Bootstrap,

- izboljšati prikazovanje podatkov (npr. uporaba poligonov namesto zaznamkov),
- dodati nove prikaze za pomoč pri analizi npr. grafe, ki prikazujejo različne parametre (pospeške, OBD2) skozi čas.

Avtor upa, da bo podatkovna platforma zaživela in pomagala pri raziskavah in razvoju praktičnih aplikacij na področju avtomobilizma ali celo na drugih področjih, ki uporabljajo podobno strukturo podatkov. Za zaključek naštejmo nekatere od možnih uporab podatkovne platforme.

## 3.1 Možnosti uporabe podatkovne platforme

Poleg že znanih možnosti uporabe, npr. za zniževanje zavarovalne premije za voznike (npr. Triglav Drajv [16]), spremljanje ter vzdrževanje voznega parka (Ulu [42]), navigacijo s spremljanjem prometnih razmer na cesti (Waze [44], Here WeGo [25]) in ugotavljanje stanja cestišča (Pothole[3], Nericell [8]), si oglejmo še nekatere možnosti uporabe podatkovne platforme.

### 3.1.1 Simulator za razvoj novih mobilnih aplikacij in strojne opreme

Podatkovno platformo in zbrane podatke lahko uporabimo za razvoj mobilnih aplikacij in strojne opreme v laboratoriju. Z manjšimi spremembami lahko podatke predvajamo aplikaciji ali strojni opremi, ki jo razvijamo. Razvojni cikli so krajši, saj se izognemo potrebi po vožnjah v pravem vozilu. Vožnje lahko predvajamo večkrat, dokler nismo zadovoljni z algoritmom, ki ga razvijamo. Izbiramo lahko med različnimi vožnjami, za katere že poznamo rezultate, saj smo jih prej analizirali.

### 3.1.2 Pomoč pri opravljanju vozniškega izpita

S pomočjo aplikacije shranjujemo vožnje kandidata. Z analizo podatkov ugotavljamo stopnjo napredka kandidata in predlagamo potrebo po dodatnih vožnjah. Podatke uporabimo za objektivizacijo odločitev inštruktorja in kot dnevnik voženj. Boljši vozniki lahko opravijo izpit v krajšem času.

Zanimiva bi bila razširitev mobilne aplikacije s podatki o srčnem utripu in krvnem pritisku iz pametne ure (ang. *smartwatch*). Ugotavljali bi lahko stopnjo stresa in tako dodatno ovrednotili napredek kandidata.

### 3.1.3 Napredno obračunavanje cestnine

Večina trenutnih načinov obračunavanja cestnine je pavšalnih. S pomočjo podatkov z OBD2 in senzorjev mobilne naprave bi lahko, podobno kot v zavarovalništvu, plačilo cestnine zniževali na podlagi dejavnikov, kot so: dejanski čas uporabe cest (tisti, ki se manj vozi, plača manj), obremenjenost cest (nagrajevanje za uporabo manj obremenjenih poti), količina izpustov (spomnimo se, da se večina parametrov OBD2 nanaša na izpuste) in način vožnje.

Za tako uporabo je seveda najprimernejša strojna oblika rešitve, saj je treba preprečiti zlorabo oz. ponarejanje podatkov.

### 3.1.4 Analiza stanja cestišč na slovenskih cestah

Prej omenjena projekta, Pothole in Nericell, se že ukvarjata s spremljanjem stanja cestišča, vendar v Sloveniji nimata veliko uporabnikov. Glede na stanje slovenskih cest in nezadovoljstvo voznikov bi podoben projekt v Sloveniji pritegnil kar nekaj uporabnikov.

Vozniki bi z uporabo mobilne aplikacije zbirali podatke o stanju cestišča in imeli možnost glasovanja o nujnosti prenove določenih odsekov. Spletna stran bi prikazovala podatke s pomočjo zemljevidov. Za tako aplikacijo priključek OBD2 ni potreben.

### 3.1.5 Črna skrinjica za hranjenje podatkov v primeru prometne nesreče

Z dodatnim zajemanjem videa bi lahko mobilno napravo uporabili kot črno skrinjico, ki bi shranjevala podatke o zadnjih trenutkih pred prometno nesrečo. Podobne snemalne naprave že obstajajo, vendar so omejene na zajemanje videa (in zvoka). Z uporabo mobilne naprave bi lahko shranjevali še podatke o legi in pospeških pred in med trkom. Dokazne podatke lahko dodatno obogatimo s pomočjo podatkov s ključa OBD2, npr. število vrtljajev in položaj stopalke za plin.

Uporabnost pri dokazovanju krivde je omejena. Poglavitni razlog za to je, da tako dokazno gradivo ne omogoča ugotavljanja vzroka nesreče (razen mogoče v najočitnejših primerih). Uporabno pa je pri izterjatvi morebitne odškodnine, saj prikazuje resnost dogodka. Omenimo še, da nekatere zavarovalnice v tujini znižajo zavarovalno premijo (10–15 %) ob vgradnji naprave za zajemanje videa.

### 3.1.6 Spremljanje in evidentiranje prometne signalizacije

Kot je bilo omenjeno v poglavju 2.1.2.1, za analizo voženj potrebujemo podatke o hitrostnih omejitvah. Kakovost podatkov in njihova cena sta odvisna od ponudnika podatkov. Ažurnosti in točnosti podatkov ne poznamo.

Z razširjeno mobilno aplikacije, ki bi s pomočjo kamere in računalniškega vida prepoznavala prometno signalizacijo, bi lahko zgradili svojo podatkovno bazo prometne signalizacije. Poleg hitrostnih omejitev bi lahko v bazi hranili še drugo prometno signalizacijo in jo upoštevali pri analizi vožnje. Z zbranimi podatki bi bilo mogoče obogatiti obstoječe podatkovne zbirke, kot je npr. OSM.

# 4 Priloge

## 4.1 Dodatek A – Format datotek JSON

Dogodki so zapisani v datoteki v obliki JSON (ang. *Javascript Object Notation*). Vsak dogodek je v svoji vrstici.

Vsak zapis vsebuje tip zapisa (*typ*), čas milisekundne natančnosti (*tim*) in podatke same.

Mogoči tipi zapisov so:

- *hdr*; glava datoteke, vključuje *vin* (identifikacijska številka vozila), *fil* (ime datoteke), *seq* (zaporedna številka datoteke),
- *gps*; lega GPS, vključuje *lat* (zemljepisna širina), *lon* (zemljepisna dolžina), *alt* (nadmorska višina), *hac* (horizontalna natančnost), *vac* (vertikalna natančnost), *siv* (število vidnih satelitov), *sus* (število uporabljeni satelitov), *crs* (kurz), *cac* (natančnost kurza), *hea* (smer), *hac* (natančnost smeri), *spd* (hitrost), *sac* (natančnost hitrosti),
- *acc*; pospeški, vključuje *x*, *y*, *z* (pospešek v smeri x, y, z),
- *obd*; vrednost parametrov OBD2, primer: "*0105*":"*39*" pomeni, da ima PID 0105 vrednost 39,
- *mrk*; označba (nastane ob pritisku na Mark), nima zaznamkov.

Pri tipih *gps*, *acc*, *obd* je mogoč še zaznamek *evt*, ki pove, kdaj se je modul pognal/ustavil.

Oblika datoteke je izbrana tako, da je mogoče dodajati nove tipe oz. zaznamke.

Ime datoteke je oblike YYYYMMDD_HHMMSS_<seq>.dat.gz. Največja velikost datoteke je programsko določena (1Mb). Ko je ta velikost presežena, se ustvari nova datoteka, pri kateri je zaznamek *seq* povečan za 1.

Primer:


```
{"typ":"hdr","tim":"2016-08-01T22:55:00.228+0200","vin":"LJ 16-FRI","fil":
    "20160801_225500_00000.dat.gz", "seq":"0"}
{"typ":"gps","tim":"2016-08-01T22:55:00.424+0200","evt":"start"}
{"typ":"acc","tim":"2016-08-01T22:55:00.447+0200","evt":"start"}
{"typ":"gps","tim":"2016-08-01T22:55:00.405+0200","lat":"46.0555560","lon":"14.5083330",
    "alt":"60.0000000", "hac":"20.0000000","vac":"100.0000000"}
{"typ":"acc","tim":"2016-08-01T22:55:00.452+0200","x":"0","y":"0","z":"-1000"}
{"typ":"acc","tim":"2016-08-01T22:55:00.470+0200","x":"0","y":"0","z":"-1000"}
{"typ":"obd","tim":"2016-08-01T22:55:01.589+0200","evt":"start"}
{"typ":"acc","tim":"2016-08-01T22:55:01.613+0200","x":"0","y":"0","z":"-1000"}
{"typ":"gps","tim":"2016-08-01T22:55:01.487+0200","lat":"46.0555560","lon":"14.5083330",
    "alt":"60.0000000","hac":"20.0000000","vac":"100.0000000"}
{"typ":"acc","tim":"2016-08-01T22:55:01.960+0200","x":"0","y":"0","z":"-1000"}
{"typ":"acc","tim":"2016-08-01T22:55:01.977+0200","x":"0","y":"0","z":"-1000"}
{"typ":"acc","tim":"2016-08-01T22:55:02.014+0200","x":"0","y":"0","z":"-1000"}
{"typ":"obd","tim":"2016-08-01T22:55:02.019+0200","0105":"39","010C":"2676"}
```

## 4.2 Dodatek B – Izvorna koda in testni podatki

Izvorna koda je priložena na zgoščenki:

- koda mobilne aplikacije (C/C++) je v mapi mobdlog,
- koda strežnika (Java, Javascript) je v mapi mobdsrv.

Osnovni pogoji za uporabo kode so:

- mobilna aplikacija:
  - Microsoft Visual Studio 2015 (potrebna brezplačna registracija),
  - Marmalade SDK Community Edition (potrebna brezplačna registracija),
  - MacOSX z Xcode za iOS mobilne naprave;
- strežnik:
  - JDK 1.8,
  - Maven,
  - IntelliJ IDEA (ali Eclipse),
  - brskalnik Chrome ali Firefox (za testiranje in razhroščevanje spletnega uporabniškega vmesnika),
  - uporabniški račun na Here WeGo in registracija aplikacije (appId and appCode vpišemo v config/Configuration.java).

V mapi mobsrv/data je shranjenih nekaj testnih podatkov – voženj. Vožnje so bile programsko spremenjene za potrebe simuliranja prekoračitev omejitev in prekomernega pospeševanja.

# 5 Literatura

[1] Baek S-H, Kim H-S, Jeong D-W, Kim M-J, Park Y-S, Jang J-W. "Implementation vehicle driving state system with OBD-II", MOST network, Proceedings of the 17th Asia-Pacific Conference on Communications (APCC); leto 2011, str. 709

[2] Bruce G. Simons-Morton, Marie Claude Ouimet, Rusan Chen, Sheila G. Klauer, Suzanne E. Lee, Jing Wang, Thomas A. Dingus, "Peer influence predicts speeding prevalence among teenage drivers", Journal of Safety Research 43, leto 2012, str. 397–403

[3] Eriksson J, Girod L, Hull B, Newton R, Madden S, Balakrishnan H. "The pothole patrol: using a mobile sensor network for road surface monitoring", 2008, str. 29–39

[4] Guillermo Cueva-Fernandez, Jordán Pascual Espada, Vicente García-Díaz, Cristian González García, Nestor Garcia-Fernandez, "Vitruvius: An expert system for vehicle sensor tracking and managing application generation", Journal of Network and Computer Applications 42, leto 2014, str 178–188

[5] Guillermo Cueva-Fernandez, Jordán Pascual Espada, Vicente García-Díaz, Ruben Gonzalez-Crespo, "Fuzzy decision method to improve the information exchange in a vehicle sensor tracking system", Applied Soft Computing 35, leto 2015, str. 708–716

[6] Keith Mike, Schincariol Merrick, Pro JPA 2, 2nd Edition, APress, 2013

[7] Nguyen DL, Lee M-E, Lensky A, "The design and implementation of new vehicle black box using the OBD information", Proceedings of the 7th International Conference on Computing and Convergence Technology (ICCCT); leto 2012, str. 1281.

[8] Prashanth Mohan, Venkat Padmanabhan, and Ramachandran Ramjee, Nericell: "Rich Monitoring of Road and Traffic Conditions using Mobile Smartphones", ACM Sensys, Association for Computing Machinery, Inc., november 2008.

[9] Purushothaman Jobinesh, RESTful Java Web Services, 2nd Edition, Packt Publishing, 2015

[10] Yun H-J, Lee S-K, Kwon O-C, "Vehicle-generated data exchange protocol for remote OBD inspection and maintenance", Proceedings of the 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT); leto 2011, str. 81

# 6 Drugi viri


[11] AngularJS Tutorial, https://docs.angularjs.org/tutorial

[12] AngularJS, https://angularjs.org

[13] AngularUI, https://angular-ui.github.io

[14] Apache Cordova, https://cordova.apache.org

[15] Apache Maven, https://maven.apache.org

[16] Drajv, 2015, http://drajv.triglav.si

[17] Eclipse Jetty, http://www.eclipse.org/jetty

[18] ELM Electronics, ELM327 OBD to RS232 Interpreter, https://www.elmelectronics.com/wp-content/uploads/2016/07//ELM327DS.pdf

[19] ELM Electronics, https://www.elmelectronics.com

[20] Github, Angular Here Maps, https://github.com/lukemarsh/angular-here-maps

[21] Google App Engine, 2008, https://cloud.google.com/appengine/docs

[22] Google Maps, 2005, https://maps.google.com

[23] Here Maps API for JavaScript, https://developer.here.com/develop/javascript-api

[24] Here Maps REST API, https://developer.here.com/develop/rest-apis

[25] Here WeGo, 2014, https://wego.here.com

[26] Hibernate ORM, http://hibernate.org/orm

[27] Hibernate Spatial, http://www.hibernatespatial.org

[28] JBoss RESTEasy, http://resteasy.jboss.org/

[29] Jetbrains, IntelliJ IDEA, https://www.jetbrains.com/idea

[30] JTS Topology Suite - Features, http://tsusiatsoftware.net/jts/jts-features.html

[31] Marmalade SDK, https://www.madewithmarmalade.com

[32] Nesreče, 2012, http://nesrece.avp-rs.si

[33] OBDSim, https://icculus.org/obdgpslogger/obdsim.html

[34] OpenGIS, Simple Feature Access - Part 2, SQL Option, http://www.opengeospatial.org/standards/sfs

[35] OpenStreetMap, 2004, https://www.openstreetmap.org

[36] Oracle JPA, http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html

[37] Phonearena, "Did you know how many different kinds of sensors go inside a smartphone?", http://www.phonearena.com/news/Did-you-know-how-many-different-kinds-of-sensors-go-inside-a-smartphone_id57885, 2014

[38] PostGIS, http://postgis.net

[39] Sourceforge, com2com, com2tcp, https://sourceforge.net/projects/com0com/files

[40] TomTom, 1991, http://www.tomtom.com

[41] Torque, 2010, http://torque-bhp.com

[42] ULU, 2015, http://www.ulu.io

[43] Vindecoder, http://www.vindecoder.net

[44] Waze, 2008, http://www.waze.com

[45] Wikipedia, AngularJS, https://en.wikipedia.org/wiki/AngularJS

[46] Wikipedia, Marmalade (software), https://en.wikipedia.org/wiki/Marmalade_(software)

[47] Wikipedia, OBD-II PIDs, https://en.wikipedia.org/wiki/OBD-II_PIDs

[48] Wikipedia, On-board Diagnostics, https://en.wikipedia.org/wiki/On-board_diagnostics