



**УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**



**УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
НОВИ САД
Департман за рачунарство и аутоматику
Одсек за рачунарску технику и рачунарске комуникације**

ЗАВРШНИ (BACHELOR) РАД

Кандидат: Горан Вуков

Број индекса: RA148/2011

Тема рада: Реализација снимача података о догађајима у возилу

Ментор рада: др. Иван Каштелан

Нови Сад, јун, 2015



КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, РБР:	
Идентификациони број, ИБР:	
Тип документације, ТД:	Монографска документација
Тип записа, ТЗ:	Текстуални штампани материјал
Врста рада, ВР:	Завршни (Bachelor) рад
Аутор, АУ:	Горан Вуков
Ментор, МН:	др. Иван Каштелан
Наслов рада, НР:	Реализација снимача података о догађајима у возилу
Језик публикације, ЈП:	Српски / латиница
Језик извода, ЈИ:	Српски
Земља публиковања, ЗП:	Република Србија
Уже географско подручје, УГП:	Војводина
Година, ГО:	2015
Издавач, ИЗ:	Ауторски репринт
Место и адреса, МА:	Нови Сад; трг Доситеја Обрадовића 6
Физички опис рада, ФО: (поглавља/страна/ цитата/табела/слика/графика/прилога)	7/39/0/4/14/2/0
Научна област, НО:	Електротехника и рачунарство
Научна дисциплина, НД:	Рачунарска техника
Предметна одредница/Кључне речи, ПО:	сакупљач података, ауто, андроид, „obd2“ протокол, сервис
УДК	
Чува се, ЧУ:	У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, ВН:	
Извод, ИЗ:	У оквиру задатка реализован је сервис за читавање и складиштење података о догађајима у возилу. Ови догађаји укључују пре свега поруке о грешкама, али укључују и друге информације о возилу. Реализована је и испитна апликација која омогућава и добављање складиштених података по разним критеријумима. Целокупан рад се заснива на познавању „OBD2“ система, „Java“ програмског језика и „SQLite“ базе података. У току израде рада, у сврху испитивања, коришћен је симулатор за симулирање догађаја возила.
Датум прихватања теме, ДП:	
Датум одбране, ДО:	
Чланови комисије, КО:	Председник: др. Јелена Ковачевић
	Члан: др. Небојша Пјевалица
	Члан, ментор: др. Иван Каштелан
	Потпис ментора



KEY WORDS DOCUMENTATION

Accession number, ANO :	
Identification number, INO :	
Document type, DT :	Monographic publication
Type of record, TR :	Textual printed material
Contents code, CC :	Bachelor Thesis
Author, AU :	Goran Vukov
Mentor, MN :	Ivan Kastelan, PhD
Title, TI :	Vehicle event data recorder
Language of text, LT :	Serbian
Language of abstract, LA :	Serbian
Country of publication, CP :	Republic of Serbia
Locality of publication, LP :	Vojvodina
Publication year, PY :	2015
Publisher, PB :	Author's reprint
Publication place, PP :	Novi Sad, Dositeja Obradovica sq. 6
Physical description, PD : <small>(chapters/pages/ref./tables/pictures/graphs/appendixes)</small>	7/39/0/4/14/2/0
Scientific field, SF :	Electrical Engineering
Scientific discipline, SD :	Computer Engineering, Engineering of Computer Based Systems
Subject/Key words, S/KW :	event data recorder, android, service, automotive, infotainment, OBD2 protocol
UC	
Holding data, HD :	The Library of Faculty of Technical Sciences, Novi Sad, Serbia
Note, N :	
Abstract, AB :	In this project Android service has been made for reading and storing vehicle information data. Vehicle information data presents information about vehicle trouble codes, speed etc. Test app has been made for gathering and storing data with various criteria. Project is based on OBD II system, SQLite library and Java programming knowledge. During making of project OBD II simulator has been used to simulate car behaviour.
Accepted by the Scientific Board on, ASB :	
Defended on, DE :	
Defended Board, DB :	President: Jelena Kovacevic, PhD
	Member: Nebojsa Pjevalica, PhD
	Member, Mentor: Ivan Kastelan, PhD
	Mentor's sign

Zahvalnost

Zahvaljujem se institutu RT-RK, pre svega Automotive timu: Tomislavu Maruni, Ivanu Kaštelanu, Marku Kovačeviću, Branimiru Kovačeviću, Nenadu Jovanoviću i Mladenu Kovačevu na stručnoj i nesebičnoj pomoći u toku izrade rada.

Na kraju se zahvaljujem porodici, prijateljima i kolegama na pruženoj moralnoj podršci.

SADRŽAJ

1. Uvod.....	1
2. Teorijske osnove	3
2.1 Android operativni sistem	3
2.1.1 Servisi u Androidu	4
2.1.2 Android Studio.....	6
2.2 OBD II sistem.....	6
2.2.1 OBD II PID-ovi na konektoru	8
2.3 OBD II simulator.....	9
2.4 SQLite biblioteka za rukovanje sa bazom podataka	10
2.4.1 Tipovi podataka u SQLite	10
2.4.2 Ograničenja u SQLite	11
2.4.3 Zaštita podataka u SQLite	11
3. Koncept rešenja.....	12
3.1 Analiza problema	12
3.2 Algoritam za rešenje problema	13
3.3 Osnovni problemi pri projektovanju	14
3.3.1 Organizacija baze podataka	14
3.3.2 Brzina pristupa memoriji	14
3.3.3 Veličina tabela i količina podataka.....	15
3.4 Modularnost	15
3.5 Prednosti u odnosu na ostala rešenja.....	15
4. Programsko rešenje.....	16
4.1 Snimač podataka	16

4.1.1	Servis za prikupljanje podataka	17
4.1.1.1	Niti za slanje zahteva o stanju željenih parametara.....	17
4.1.2	SQLite modul za bazu podataka	19
4.1.2.1	API baze podataka.....	19
4.1.2.2	OBD2DbHelper	20
4.1.3	GPS modul za praćenje lokacije	21
4.1.4	Modul za parsiranje kodova grešaka	22
4.2	Probna aplikacija.....	22
5.	Rezultati i verifikacija.....	25
5.1	Optimizacija rešenja	26
6.	Zaključak	28
7.	Literatura.....	29

SPISAK SLIKA

<i>Slika 1 Arhitektura Android operativnog sistema</i>	4
<i>Slika 2 Životni ciklus Android servisa</i>	5
<i>Slika 3 Prikaz „Logcat“ ispisa</i>	6
<i>Slika 4 Prikaz OBD II priključak</i>	7
<i>Slika 5 “ELM – 327” OBD II konektora</i>	9
<i>Slika 6 Prikaz grafičkog okruženja OBD II simulatora</i>	9
<i>Slika 7 Komunikacija snimača podataka sa vozilom/simulatorom</i>	13
<i>Slika 8 Modularni prikaz funkcionisanja snimača podataka</i>	16
<i>Slika 9 Prikaz GPS dijaloga</i>	22
<i>Slika 10 Izgled testne aplikacije</i>	23
<i>Slika 11 Izgled testne aplikacije sa prikazom lokacije u “Here” - Microsoft mapama</i>	24
<i>Slika 14 Prikaz testiranja snimača podataka u automobilu VW Bora</i>	26

Slika 12 Grafički prikaz vremena obrade parametara sa i bez optimizacije27

Slika 13 Grafički prikaz vremena upisa u bazu podataka sa i bez optimizacije27

SPISAK TABELA

<i>Tabela 1</i> Selekcija moda i značenje pidova	8
<i>Tabela 2</i> Korišćene funkcije iz API servisa za prihvatanje informacija	19
<i>Tabela 3</i> Spisak API funkcija baze podataka	20
<i>Tabela 4</i> Objašnjenje konstruktora i funkcija <i>OBD2DbHelper</i> klase	21

SKRAĆENICE

ADB – Android Debug Bridge, Android alat za ispravljanje grešaka

API – Application Programming Interface, Programski prilagodni sloj

ECU – Engine Control Unit, Kontrolna jedinica motora

EDR – Event Data Recorder, Snimač podataka o događajima

GPS – Global Positioning System, Globalni pozicioni sistem

JNI – Java Native Interface, Programski okvir koji omogućava spregu Java programskog jezika sa drugim programskim jezicima

OBD II – On-board Diagnostic, Dijagnostika na vozilu

PID – Performance Information Data, informacije o karakteristikama vozila

SQLite – Structured Query Language, Strukturno upitni jezik

1. Uvod

U ovom radu je realizovan snimač podataka kao Android servis i probna aplikacija za očitavanje i skladištenje podataka o događajima u vozilu. Ovi događaji uključuju pre svega poruke o greškama, ali i ostale informacije o vozilu kao što su: brzina, nivo goriva, pozicija papučice za gas, GPS lokacija, broj obrtaja motora i ostale.

Od pre deset godina zbog povećanog broja udesa usled umora vozača, uvedena je strožija kontrola vozača kao i njihovih poslovođa. Takođe, zbog nepoštovanja prava vozača od strane poslovođa počela je ugradnja uređaja ili aplikacija koje mogu da se integrišu u neke od uređaja u vozilu kao što je ova. Dok je vozilo u pokretu beleži se koliko je dugo vozilo bilo u pokretu, da li je prekoračilo brzinu ili slično. Na taj način organi zaduženi za kontrolu lako mogu ustanoviti da li je došlo do prekršaja zakona i kazniti nesavesne vozače.

Snimač podataka takođe pomaže proizvođačima i serviserima da lakše ustanove kvar na vozilu. Parametri snimanja podataka mogu biti prilagodivi te je i njihova namena različita. Ukoliko se stalno skladište podaci o vozilu, mogu se ustanoviti razlozi kvara vozila koji se pojavljuju povremeno. Svi parametri u trenutku pojavljivanja kvara će biti sačuvani i na osnovu njih može se lakše ustanoviti razlog njihovog nastajanja. Odgovarajućim izborom parametara aplikacija se može konfigurisati da radi kao crna kutija. U tom slučaju potrebno je da vreme čuvanja podataka o vozilu bude kratko i da se posle isteka tog vremena podaci prepisuju. Crna kutija će pomoći njihovim proizvođačima da vide kako se vozilo ponašalo neposredno pre sudara i da na taj način poboljšaju karakteristike vozila kao i sigurnost vozača. Na ovaj način moguće je i ustanoviti ko je krivac i izazivač sudara.

Za korišćenje realizovanog servisa kao i probne aplikacije neophodno je da ciljni uređaj poseduje Android operativni sistem i podršku za Bluetooth. Razlog za ovakav izbor je sve veća

prisutnost Android [1][2]operativnog sistema u uređajima potrošačke elektronike kao što su mobilni telefoni, tableti i računari za vozila koja tek treba da izađu na tržište. Modularnom realizacijom rešenja odvojen je servis od testne aplikacije kako bi servis mogao da radi u pozadini i sakuplja podatke dok aplikacija nije aktivna na ekranu. Na taj način je omogućeno da se na uređaju na kom je instaliran snimač podataka koriste i druge mogućnosti uređaja nesmetano dok snimač podataka upisuje podatke u pozadini. Parametri snimača se mogu podešavati i od njih zavisi zahtevnost resursa platforme na kojoj se izvršava. Najveći zahtevi su za memorijom, pre svega učestanosti njenom pristupanju kao i količinom informacija koje treba da se skladište.

U ovom zadatku je posvećena pažnja potrošnji resursa u zavisnosti od promene parametara. Za dobavljanje informacija od vozila ili simulatora događaja u vozilu (u slučaju laboratorijskih ispitivanja), korišćen je „OBD-II“ sistem koji se po standardu koristi u svim vozilima. Kako su svi proizvođači vozila shvatili da je neophodno uspostaviti način komunikacije automobila sa spoljašnosti radi lakšeg servisiranja i unapređivanja karakteristika vozila osmišljen je OBD-II sistem koji tačno definiše način komunikacije između centralnog računara automobila i spoljašnjih uređaja za bilo koju marku vozila. Svaki proizvođač ima nešto specifično za svaki model vozila, kao na primer kodove ili listu grešaka, ali je način njihovog dobavljanja za svaku marku vozila isti što je suština ovog sistema.

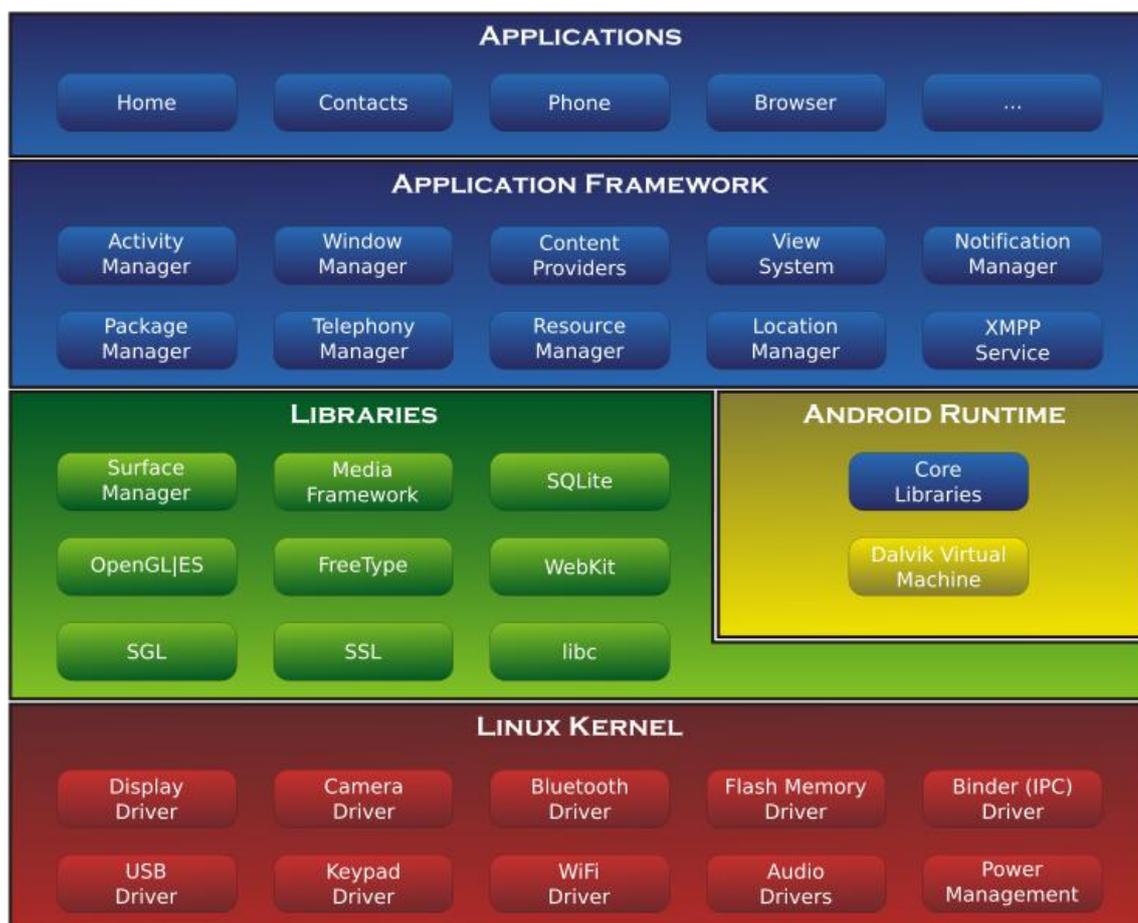
2. Teorijske osnove

U teorijskom delu su objašnjeni i analizirani glavni moduli, protokol kao i alati korišćeni u izradi snimača podataka u vozilima. U to spadaju Android operativni sistem kao i njegove glavne osobine koje smo koristili u izradi, uopšten prikaz i objašnjenje OBD II sistema, opšte informacije o SQLite programskoj biblioteci za rukovanje bazama podataka i informacije o simulatoru događaja u vozilu koji je korišćen prilikom razvijanja rešenja.

2.1 Android operativni sistem

Android operativni sistem je napravila kompanija „Android“ 2003. godine koju je 2005. godine kupila kompanija „Google“. Nakon njihovog preuzimanja operativni sistem je od 2007. ugrađivan u pametne mobilne telefone osetljive na dodir gde je doživeo naglu i veliku popularnost od strane korisnika pre svega zbog jednostvanog rukovanja i ekrana osetljivog na dodir. Posle sve većeg širenja na tržištu mobilnih telefona kompanija „Google“ je odlučila da Android učini dostupnim i na drugim platformama kao što su televizori, tableti i slični uređaji potrošačke elektronike, te je sada postao vodeći operativni sistem na „embedded“ uređajima ove vrste. Android je zasnovan na „Linux“ jezgru i programiran je u C, C++ i Java programskim jezicima. Njegova programska struktura je slojevita i prikazana je na Slika 1. Dodatna pristupačnost je u tome što je Android sistem otvorenog koda pod „Apache License“ što znači da je dopuštena slobodna izmena i distribucija softvera od strane proizvođača uređaja ili pojedinaca. Takođe i alati koje ova kompanija nudi su besplatni i ne predstavljaju dodatne troškove inženjerima koji razvijaju programsku podršku za ovaj operativni sistem pa je stoga veoma popularan i zastupljen. Zbog specifičnih zahteva platformi na kojima se koristi, Android [3] operativni sistem je optimizovan u pogledu potrošnje resursa. Pošto se koristi kod prenosnih uređaja, potrebno je da dužina trajanja rada na bateriji bude što duža. Ovaj sistem poseduje

ugrađene osnovne aplikacije kao što su: Google mail, Google search, Google book, kalendar, aplikacija za telefoniranje (engl. *dialer*) i druge. Osim ugrađenih aplikacija mnoštvo Android aplikacija se može preuzeti sa „Google Play“ marketa. Trenutan broj aplikacija na ovom servisu je oko milion, dok je broj preuzimanja preko pedeset miliona.



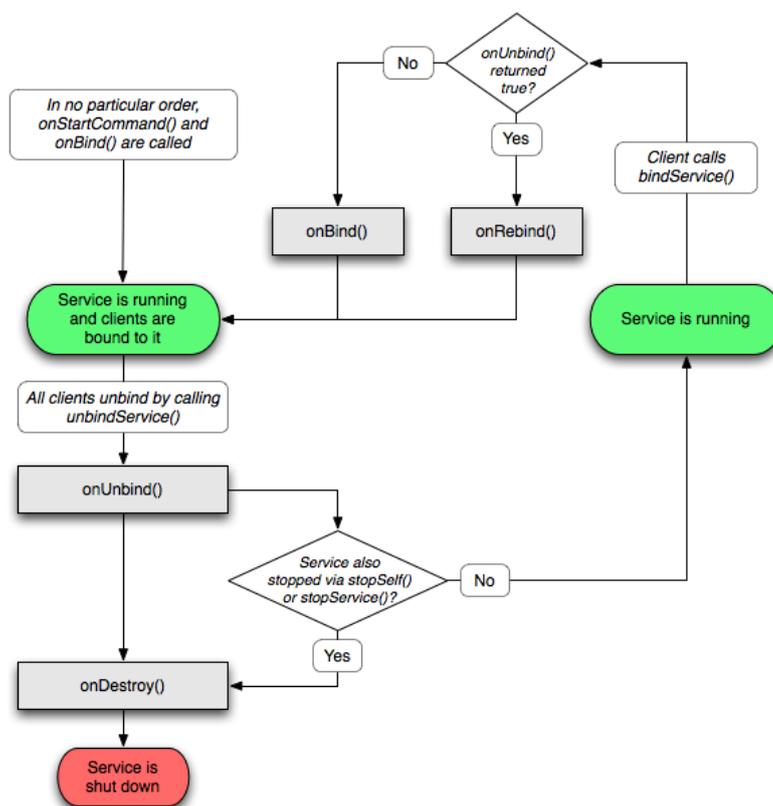
Slika 1 Arhitektura Android operativnog sistema

2.1.1 Servisi u Androidu

Servisi u Android operativnom sistemu [4] predstavljaju komponente koje rade u pozadini i nemaju grafičko okruženje. Obično su namenjeni za obavljanje dugotrajnih operacija. Postoje dve vrste Android serisa: sistemski i aplikativni. Sistemski servisi predstavljaju deo operativnog sistema i obezbeđuju spregu između aplikativnog sloja i Linux jezgra. Primeri sistemskih servisa su: alarm, servis lokacije, senzor za temperaturu, servis za telefoniranje i mnogi drugi. Većina sistemskih servisa je napisana u Javi, dok su neki sistemski servisi napisani u C/C++ programskim jezicima zbog ubrzanja fizičke arhitekture, to jest zahtevnosti za visokim performansama. Primeri ovakvih servisa su servis za rukovanje kamerom kao i servis za reprodukciju audio/video sadržaja.

Pod aplikativnim servisima podrazumevamo servise koje naknadno instaliramo na uređajima. Aplikativni servisi se mogu pokrenuti na više načina: pokretanjem iz korisničke aplikacije, iz drugog Android servisa ili putem prijemnika neusmerenih poruka (engl. *broadcast receiver*). Prijemnik neusmerenih poruka šalje poruke određenog sadržaja svim Android servisima koji su se pretplatili da ih oslušuju. Svaki predplaćeni servis tumačenjem te poruke prepoznaje kome je namenjena i preduzima određenu akciju. Prijemnik neusmerenih poruka se aktivira prilikom pokretanja uređaja ili prijemom zahteva za slanje poruke ovog tipa.

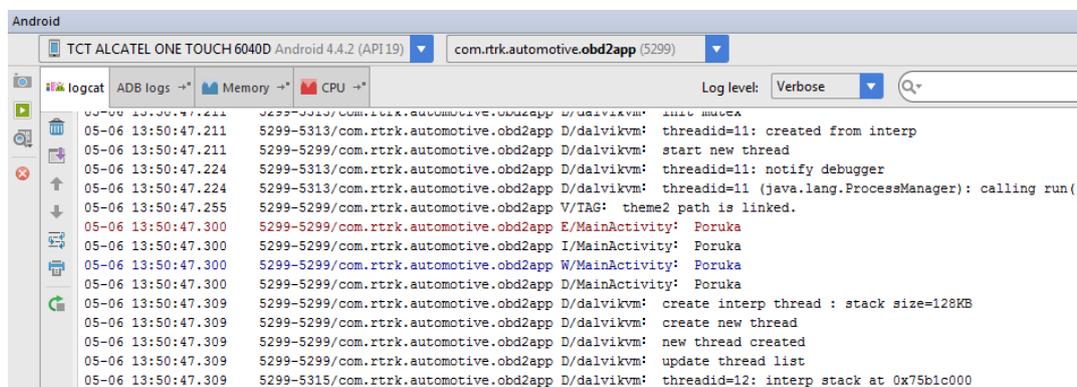
Aplikativni servisi se dele na dve grupe: lokalni i udaljeni. Lokalni servisi se izvršavaju u istom memorijskom prostoru kao i aplikacija kojoj pripadaju dok se udaljeni servisi izvršavaju u posebnom memorijskom prostoru i sa njima se komunicira putem IPC (engl. *InterProcess Communication*) mehanizma. Komunikacija između dva servisa ili servisa i korisničke aplikacije se vrši preko sprega (engl. *Interface*) koje su u Androidu predstavljene AIDL datotekama. U ovom rešenju je realizovan lokalni aplikativni servis. Način rada kao i prikaz životnog ciklusa Android servisa su prikazani na *Slika 2*. Kada se pokrene lokalni servis na njega se mogu povezati udaljeni klijenti. On će raditi sve dok svi klijenti ne prekinu vezu sa njim i dok se servis ne zaustavi pozivom funkcija za prekid njegovog rada. Na servis se mogu povezati klijenti (programi ili drugi servisi) iako on prethodno nije bio pokrenut. Sve dok svi klijenti ne prekinu vezu sa servisom on neće biti zaustavljen.



Slika 2 Životni ciklus Android servisa

2.1.2 Android Studio

Android Studio predstavlja razvojno okruženje za izradu Android aplikacija. On obezbeđuje pisanje programskog koda, pravljenje grafičke korisničke sprege, otkrivanje i uklanjanje grešaka ili praćenje izvršavanja programa, testiranje i još mnogo toga. Jednom rečju sadrži sve što je potrebno za pravljenje aplikacija za ovaj operativni sistem. Pored njegovih ugrađenih funkcija i mogućnosti moguće je i instalirati dodatke koji pomažu i olakšavaju posao programerima kao što je dodatak za kontrolu verzije (SVN, Git), rad sa određenim datotekama (.json) i drugi. Na ovaj način kompanija „Google“ koja je napravila ovaj alat, pruža mogućnost njegovog proširenja i usavršavanja u raznim pravcima. Iz ovog alata je moguće direktno instalirati i pokrenuti aplikaciju na ciljnoj platformi koja je povezana sa ovim alatom putem ADB (engl. *Android Debug Bridge*) upravljačkog programa. Alat omogućava i praćenje izvršavanja Android aplikacije u realnom vremenu korišćenjem „logcat“ servisa. Logcat servis omogućava ispis poruka koje je programer napisao na željenim mestima u programu ili ispis poruka samog sistema. Podržani su različiti tipovi poruka: poruka greške, informaciona poruka, poruka upozorenja ili poruka otkrivanja i uklanjanja grešaka. Sastoje se iz dva dela, oznake ili imena pod kojim se mogu pronaći i teksta poruke. Primer ispisa poruka je dat na *Slika 3*.



```

Android
TCT ALCATEL ONE TOUCH 6040D Android 4.4.2 (API19) com.rtrk.automotive.obd2app (5299)
Logcat ADB logs Memory CPU Log level: Verbose
05-06 13:50:47.211 5299-5313/com.rtrk.automotive.obd2app D/dalvikvm: start main
05-06 13:50:47.211 5299-5313/com.rtrk.automotive.obd2app D/dalvikvm: threadid=11: created from interp
05-06 13:50:47.211 5299-5299/com.rtrk.automotive.obd2app D/dalvikvm: start new thread
05-06 13:50:47.224 5299-5313/com.rtrk.automotive.obd2app D/dalvikvm: threadid=11: notify debugger
05-06 13:50:47.224 5299-5313/com.rtrk.automotive.obd2app D/dalvikvm: threadid=11 (java.lang.ProcessManager): calling run()
05-06 13:50:47.255 5299-5299/com.rtrk.automotive.obd2app V/TAG: theme2 path is linked.
05-06 13:50:47.300 5299-5299/com.rtrk.automotive.obd2app E/MainActivity: Poruka
05-06 13:50:47.300 5299-5299/com.rtrk.automotive.obd2app I/MainActivity: Poruka
05-06 13:50:47.300 5299-5299/com.rtrk.automotive.obd2app W/MainActivity: Poruka
05-06 13:50:47.300 5299-5299/com.rtrk.automotive.obd2app D/MainActivity: Poruka
05-06 13:50:47.309 5299-5299/com.rtrk.automotive.obd2app D/dalvikvm: create interp thread : stack size=128KB
05-06 13:50:47.309 5299-5299/com.rtrk.automotive.obd2app D/dalvikvm: create new thread
05-06 13:50:47.309 5299-5299/com.rtrk.automotive.obd2app D/dalvikvm: new thread created
05-06 13:50:47.309 5299-5299/com.rtrk.automotive.obd2app D/dalvikvm: update thread list
05-06 13:50:47.309 5299-5315/com.rtrk.automotive.obd2app D/dalvikvm: threadid=12: interp stack at 0x75b1c000
  
```

Slika 3 Prikaz „Logcat“ ispisa

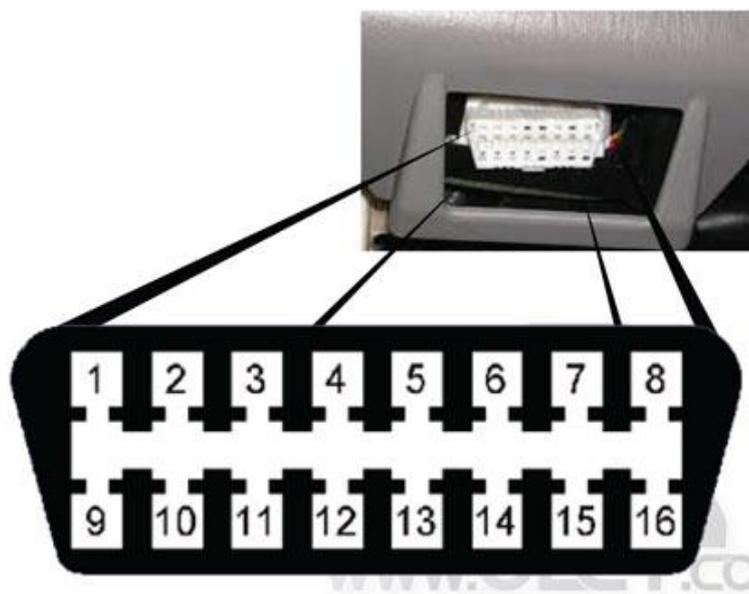
2.2 OBD II sistem

OBD (engl. *On-Board Diagnostic*) [5] je sistem koji se nalazi u svim savremenim vozilima. Primena je počela u Americi 1970. godine sa ciljem da se smanji štetnost vozila po životnu sredinu. Da bi se proizvođači vozila pridržavali niza zakona donetih u tu svrhu napravljeni su sistemi u vozilu koji se elektronski kontrolišu. Na taj način su uspevali da

kontrolišu funkcionisanje motora i dijagnostikuju njegove probleme. Senzori su merili parametre motora i prilagođavali sistem kako bi zagađenje bilo što manje. Tokom godina ovaj sistem se usavršavao i menjao.

Godine 1988. SAE (engl. *Society of Automotive Engineers*) je postavio standard kakav se priključak koristi za povezivanje sa vozilom i skup dijagnostičkih test signala. OBD II je proširenje početnog standarda SAE, koje omogućava gotovo kompletnu kontrolu nad motorom i takođe nadgleda delove šasije i tela vozila. Njegova primena je počela 1. Januara 1996.

Svi automobili koji su proizvedeni od 1996. godine pa nadalje imaju OBD II sistem. Najbolji način provere da vozilo poseduje ovaj sistem je pronalaženje OBD II priključaka u vozilu koji se po standardu nalazi najviše jedan metar od vozača i izgleda kao na *Slika 4*.



Slika 4 Prikaz OBD II priključak

OBD II poseduje 5 protokola koji se razlikuju po načinu komunikacije sa vozilom. Korišćeni protokol se može utvrditi po pinovima na OBD priključku:

- J1850 VPW – ovaj konektor mora imati metalne kontakte na pinovima 2, 4, 16, ali ne i 10
- ISO 9141-2/KWP2000 – ovaj konektor mora imati metalne kontakte na pinovima 4, 5, 7, 15 i 16
- J1850 PWM – ovaj konektor mora imati metalne kontakte na pinovima 2, 4, 5, 10, i 16
- CAN – ovaj konektor mora imati metalne kontakte na pinovima 4, 5, 6, 14 i 16, on je najnoviji, i u upotrebi je od 2008. godine.

2.2.1 OBD II PID-ovi na konektoru

Komunikacija sa vozilom se postiže preko metalnih kontakata PID-ova OBD II konektora tako što se na njega priključi odgovarajući uređaj. To može biti kao u slučaju testiranja ovog zadatka gde putem OBD II utičnice “ELM - 327 interface” [6] vidi *Slika 5.* vršimo povezivanje vozila sa mobilnim telefonom koji poseduje programsku podršku za ovaj sistem. Putem programske podrške mobilnog telefonavršimo selekciju određenih pidova na ELM - 327 utičnici i šaljemo podatke na CAN magistralu u vozilu. Centralni računar prepoznaje zahtev sa magistrale i vraća odgovor preko iste. Za dobavljanje različitih informacija potrebno je selektovati različite PID-ove to jest režime rada [7] kao što je prikazano u *Tabela 1.*

Selekcija pidova / režimi rada	Opis PID-ova
01	Trenutni podaci
02	Podaci nastali u trenutku kada je ustanovljena greška u sistemu
03	Kodovi greške
04	Uklanjanje svih kodova greške i sačuvanih vrednosti
05	Nadgledanje senzora za kiseonik (ne CAN protokol)
06	Nadgledanje senzora za kiseonik(samo CAN protokol)
07	Kodovi grešaka koji su se dogodili tokom trenutne ili poslednje vožnje tj. nepotvrđeni kodovi grešaka
08	Kontrolne operacije nad OBD II sistemom

Tabela 1 Selekcija moda i značenje pidova

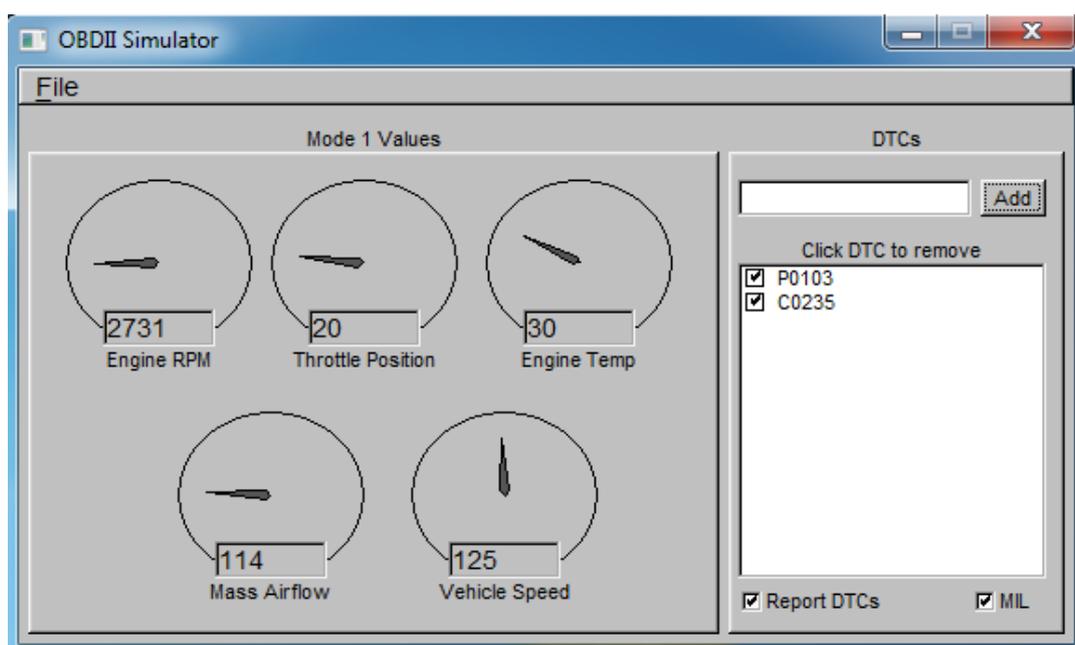
Pored izbora režima rada iz *Tabela 1.* za dobavljanje konkretnih informacija potrebna je selekcija još pidova. Na primer u slučaju da je u režim rada 01 potrebno dobiti brzinu neophodno je dodati i 0D, a ako je potrebno dobiti brzinu kada je nastala greška dodaje se 0D na pid 02. Za temperaturu je potrebno dodati 05 i slično. Kada se ovi PID-ovi šalju na simulator ili CAN magistralu vozila, svaki se mora završiti sa “/r”, to jest novim redom jer protokol tako zahteva i na taj način zna da je to jedan zahtev.



Slika 5 “ELM – 327” OBD II konektora

2.3 OBD II simulator

Prilikom testiranja i razvoja servisa sa testnom aplikacijom u laboratorijskim uslovima korišćen je simulator događaja u vozilu. Simulator simulira ponašanje vozila i njegove parametre. Ima nekoliko načina na koji simulira podatke. Može se izabrati na primer slučajan izbor vrednosti, slučajan izbor vrednosti sa međusobnom zavisnosti određenih parametara, kao na primer brzina kretanja vozila sa brzinom obrtaja motora i potrošnjom goriva, učitavanje vrednosti iz datoteke ili ručno postavljanje vrednosti. Najkorisniji način rada simulatora je poslednji jer je na vozilu u vožnji teško tačno postaviti određenu kombinaciju parametara radi testiranja. Na ovaj način jednostavno možemo napraviti svaki scenario za potrebe testiranja. Izgled grafičkog okruženja simulatora prikazan je na *Slika 6*.



Slika 6 Prikaz grafičkog okruženja OBD II simulatora

2.4 SQLite biblioteka za rukovanje sa bazom podataka

SQLite je biblioteka otvorenog programskog koda za rad sa bazama podataka. Ona podržava standardne funkcije ili instrukcije za rukovanje relacionim bazama podataka kao što je SQL sintaksa. SQLite [8][9] zahteva malu memoriju za rad u toku izvršavanja programa (maksimum 250 KByte) što je čini dobrim kandidatom za “ugrađene” (engl. *embedded*) uređaje ove vrste. SQLite biblioteka za rukovanjem bazama podataka je ugrađena u svaki Android uređaj. Korišćenje ove biblioteke ne zahteva dodatnu instalacionu proceduru. Za korišćenje funkcionalnosti ove biblioteke u aplikaciji ili servisu potrebno je stvoriti i pozvati SQL izjave za stvaranje i osvežavanje baze. Nakon toga baza će automatski biti napravljena i održavana od strane Android platforme. Nakon toga je moguće napraviti tabele kao tip organizacije podataka i funkcije odnosno API biblioteku funkcija za rukovanje sa njom. Pristupanje podacima iz baze podataka zahteva pristup sistemu datoteka što može biti veoma sporo. Zbog toga je poželjno pristupati podacima asinhrono. Pod tim se podrazumeva da ako želimo dobavljanje određenih podataka pod zadatim kriterijumom iz baze, gde ima puno elemenata, poželjno je da se to ne radi u glavnoj programskoj niti programa jer taj posao zahteva dosta vremena i aplikacija određeno vreme ne bi imala odziv, što za Android operativni sistem može značiti da aplikacija ne radi kako treba i da je treba “ubiti” to jest nasilno zaustaviti njeno izvršavanje. Jedno rešenje ovog problema je izmeštanje operacije čitanja u posebnu nit ili asinhroni zadatak (engl. *AsyncTask*) kako bi se sprečilo da operativni sistem nasilno prekine rad aplikacije jer se ne odaziva duže vreme ili da bi se sprečilo blokiranje grafičke korisničke sprege aplikacije. Baza podataka se uvek pravi na podrazumevanoj lokaciji u fascikli „DATA/data/APP_NAME/databases/FILENAME“. Putanje ovih fascikli se veoma lako dobivljaju pozivom ugrađenih Android funkcija ili njihovom kombinacijom. Putanja „DATA/data/“ fascikle se dobija pozivom funkcije „`Environment.getDataDirectory()`“, dok se ime aplikacije dobija pozivom „`getString(getApplicationContext().labelRes)`“. „FILENAME“ predstavlja ime baze podataka koju je programer naveo. Na ovaj način baza podataka se može pribaviti i kopirati, to jest prebaciti na drugi uređaj kako bi se podaci lakše analizirali. Na drugom uređaju je potrebno imati bilo koji program koji služi za čitanje baza podataka ovog tipa, kao na primer „SQLiteBrowser“. SQL upitima se mogu pribaviti podaci pod različitim kriterijumima.

2.4.1 Tipovi podataka u SQLite

SQLite ima nekoliko tipova podataka, dok se svi ostali moraju svesti ili pretvoriti u neki od ovih tipova. U SQLite verziji 3 to su: null, integer, real, text i blob. Null predstavlja praznu vrednost, odnosno da nema podataka. Integer predstavlja označen ceo broj i može biti veličine

1,2,3,4,6 ili 8 bajtova u zavisnosti od veličine podatka koji treba da se unese u polje ovog tipa. Real predstavlja označen broj sa pokretnim zarezom i veličine je 8 bajtova. Text predstavlja znakovni niz (engl. *string*) i može biti kodiran sa UTF-8, UTF-16BE ili UTF-16LE formatom znakova. Blob predstavlja podatak bilo koje vrste i smešta se isti kao što je bio pre upisa u bazu ili identičan u binarnom zapisu. Svi ostali tipovi se svode na neki od ovih tipova. Na primer SQLite ne podržava Bulov tip podataka koji je predstavljen sa tačno i netačno pa se umesto tog tipa može koristiti integer tip koji sa vrednošću 1 označava tačno, a sa vrednošću 0 netačno. Sličan primer je i sa datumom, ako je datum predstavljen u formatu “gggg:mm:dd-ss:mm:ss” može se predstaviti kao text, odnosno niz znakova. Vreme takođe može i da se predstavi kao “integer” tip u milisekundama.

2.4.2 Ograničenja u SQLite

SQLite pored ograničenja u veličini svakog tipa podataka pojedinačno ima i druga ograničenja kao što su broj kolona ili broj redova svake tabele. Tako na primer maksimalni broj kolona u svakoj tabeli mora biti 2000, to ograničenje programer može nasilno da proširi na maksimalnih 32767 elemenata. Ipak mnogiiskusni dizajneri baza podataka tvrde da svaka dobro organizovana baza nikada neće imati više od 100 kolona, što se savetuje u originalnoj dokumentaciji SQLite biblioteke. Maksimalan broj redova u jednoj tabeli je 2^{64} .

2.4.3 Zaštita podataka u SQLite

Svaki snimač podataka bi trebalo da ima modul za zaštitu podataka. Potrebno je zaštititi privatnost korisnika snimača jer se njegovi podaci mogu zloupotrebili. Snimač beleži informacije poput lokacije na osnovu čega se lako može ustanoviti kuda i u koje vreme se vozač kretao. Zbog narušavanja privatnosti korisnika podaci snimača se uglavnom koriste od strane proizvođača za unapređenje vozila, iako mogu poslužiti kao dokaz na sudu usled udesa. Primena podataka snimača zavisi od zakona zemlje u kojoj se koristi.

Zaštitu je moguće izvršiti kodovanjem podataka pri upisu u bazu podatka i dekodovanjem pri čitanju uz poštovanje privatnosti korisnika. U ovom radu nije vršena zaštita podatka jer obično kodovanje kao na primer ROT13 ili MD5 nije od prevelikog značaja, a kompleksnije biblioteke za kodovanje podataka se plaćaju. Dalji razvoj ovog rešenja može biti usmeren u ovom smeru.

3. Koncept rešenja

U ovom poglavlju je opisana analiza problema, algoritam rada kao i diskusija problema koje je potrebno rešiti.

3.1 Analiza problema

U ovom radu potrebno je napraviti snimač podataka koji čuva željene parametre vozila kao što su na primer brzina kretanja, broj obrtaja motora, lista grešaka, pozicija regulatora brzine, podaci o lokaciji, nivo goriva i slično. Cilj ovog snimača podataka je da se on integriše u putni računar koji je baziran na Android operativnom sistemu jer ovakvi uređaji zasnovani na Android platformi još nisu izašli na tržište, a sasvim je sigurno da će se uskoro pojaviti.

Snimač je realizovan kao lokalni servis u okviru probne aplikacije. Servis u sebi sadrži bazu podataka i modul koji u određenom vremenskom periodu zahteva željene podatke od simulatora događaja u vozilu ili vozila preko servisa za komunikaciju u kojem je ugrađen OBD II sistem, a zatim smešta dobijene podatke u bazu podataka. Servis je izabran iz razloga kako bi neka druga funkcionalnost ovog uređaja mogla da se koristi dok snimač radi u pozadini. Servis se pokreće iz probne aplikacije. Pošto snimač podataka u zavisnosti od njegove konfiguracije u ovoj implementaciji može imati veoma zahtevne potrebe za resursima treba omogućiti da se one pokrene sa pokretanjem sistema. Ukoliko korisnik želi da uključi snimanje parametara to može uraditi uključivanjem ovog servisa iz probne aplikacije i unošenjem parametara, a nakon određenog vremena isključiti. Potrebno je i omogućiti korisniku da izveze (engl. *export*) podatke to jest bazu podataka na prenosivu memoriju. Na taj način korisnik može otvoriti i pretraživati podatke iz bilo kog programa za prikaz SQLite baze podataka. Podatke je potrebno pretraživati i u probnoj aplikaciji po raznim kriterijumima, na primer po tipu parametra, po opsegu vrednosti parametra ili po vremenu kada su se dogodili.

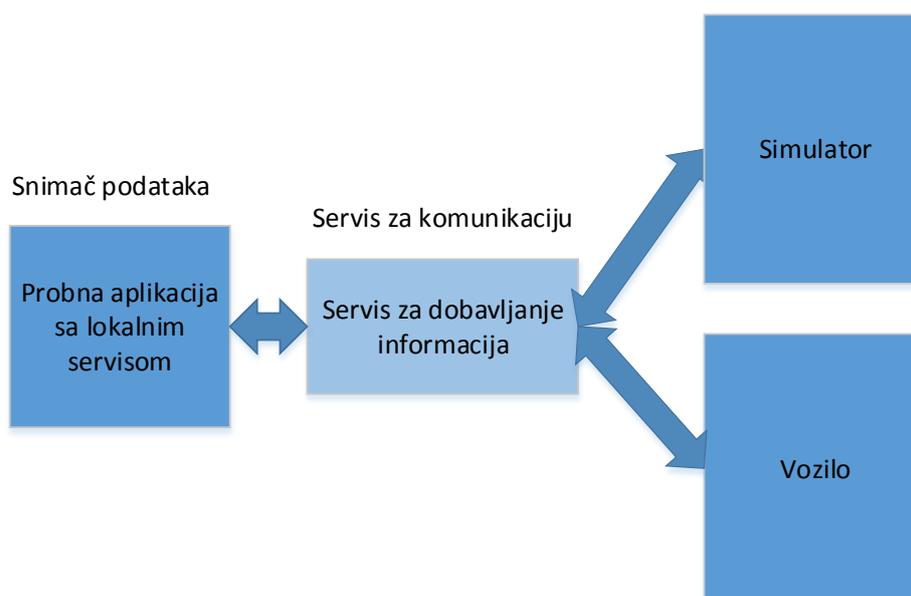
U podešavanjima servisa potrebno je omogućiti unošenje parametara kao što su učestanost upisa parametara u bazu, vremenski interval praćenja parametara. Nakon isteka zadatog vremenskog intervala najstariji podaci u bazi se prepisuju novim, tako da baza uvek sadrži najvažnije podatke. U podešavanjima je takođe potrebno omogućiti korisniku da izabere koje parametre želi da prati.

Probna aplikacije treba da omogući ispis kodova grešaka sa njihovim objašnjenjima. Aplikacija takođe treba da poseduje mehanizam za ažuriranje liste grešaka. U slučaju da se snimač podataka integriše na različite modele vozila potrebna je različita lista grešaka koja odgovara datom modelu vozila.

Ukoliko korisnik želi da vidi na mapi prikaz neke željene lokacije koja je sačuvana u bazi potrebno je pokreniti odgovarajuću aplikaciju sa mapama i prikazati je.

3.2 Algoritam za rešenje problema

Na *Slika 7* je prikazan uopšten model komunikacije snimača podataka sa vozilom odnosno simulatorom događaja u vozilu (u laboratorijskim uslovima).



Slika 7 Komunikacija snimača podataka sa vozilom/simulatorom

Snimač podataka sadrži mehanizam koji zahteva podatke po određenim kriterijumima i smešta ih u SQLite bazu podataka. Snimač takođe treba da poseduje probnu aplikaciju u kojoj je moguće isčitati podatke iz baze po raznim kriterijumima. Servis za komunikaciju Slika 6. predstavlja odvojen servis od snimača podataka i služi za komunikaciju sa vozilom ili simulatorom događaja u vozilu. U njemu je implementiran OBD II sistem. Snimač pomoću API funkcija zahteva podatke od servisa za dobavljanje informacija, a on putem OBD II sistema očitava parametre sa vozila ili simulatora događaja u vozilu i vraća ih snimaču. Simulator ili vozilo putem nekog od

izabраниh dostupnih protokola u OBD II sistemu odgovara na upite servisa za komunikaciju i daje tražene informacije o stanju u kom se nalazi.

3.3 Osnovni problemi pri projektovanju

Prilikom izrade snimača uočeni su najvažniji problemi ovakvog sistema, pa je projekat prolazio kroz različite načine implementacije. Osnovni problemi ovog rešenja su način organizacije podataka u bazi i brzina pristupa memoriji ili odziva sistema pri velikoj učestanosti upisa podataka u memoriju.

3.3.1 Organizacija baze podataka

Zbog lakšeg i bržeg pristupa podacima po željenim kriterijumima od izuzetne važnosti je da podaci budu dobro organizovani. Na primer ako se svi parametri vozila smeštaju u istu tabelu baze, implementacija funkcija za njen pristup će biti veoma jednostavna. Gledano iz drugog ugla tako organizovana tabela će imati puno redova što nije dobro rešenje. Ukoliko korisnik bude želeo da pretraži podatke iz te tablele po nekom kriterijumu i prikaže ih, moraće da sačeka određeno vreme i pri tom bi mu grafičko korisničko okruženje bilo nedostupno. Dakle, što su podaci podeljeni na više manjih celina i bolje organizovani njihova pretraga će biti brža što bi se korisniku više svidelo od čekanja. Grupisanje podataka u više manjih celina zahteva dodavanje većeg broja API funkcija za rad sa bazom podataka i bolju ali i komplikovaniju organizaciju podataka. Uzimanjem u obzir ovih aspekata dolazi se do prihvatljivog rešenja.

3.3.2 Brzina pristupa memoriji

Kako se povećava željeni broj parametara za snimanje i učestanost njihovog pribavljanja zbog upis u bazu podataka povećava se broj zahteva za pristup memoriji. U tom slučaju sistem na kom se izvršava snimač se vidno usporava i ukoliko se broj zahteva za upis brže povećava u odnosu na njihovu obradu operativni sistem će ustanoviti da se snimač nepravilno izvršava (nepravilno troši resurse) i nasilno će ga zaustaviti jer se ne odaziva. U tim situacijama potrebna je optimizacija pristupa memorije. Optimizacija može da se ogleda u prioritetu parametara ili smanjenju učestanosti upisa. Promena brzine se sigurno češće dešava u odnosu na pormenu stanja goriva. Dakle usklađivanje učestanosti upisa podataka od njihovog prioriteta svakako doprinosi optimalnijem rešenju. Drugi način optimizacije može da se ogleda u odbacivanju manje značajnih podataka. Ako se vozilo duže vreme kreće konstantnom brzinom nije potrebno uvek upisivati isti podatak u bazu već samo ukoliko je došlo do promene. Ukoliko duže vreme nije došlo do promene moguće je u određenom trenutku ipak upisati trenutnu vrednost parametra kao signal da snimač ispravno radi.

3.3.3 Veličina tabela i količina podataka

Kao što je spomenuto u poglavlju 2.4.2 SQLite poseduje razna ograničenja po pitanju veličine tabela. Iako je moguće uneti velik broj redova u svaku tabelu to nije najbolje rešenje. Jedan način izbegavanja ovog problema je ograničavanje maksimalnog vremena prikupljanja podataka. Ublažavanje ovog problema se može postići kružnim prikupljanjem podataka. Kada snimač upiše željeni maksimalan broj elemenata u bazu počinje se sa prepisivanjem najstarijih podataka. Na taj način u bazi su uvek sačuvani najsvežiji podaci u definisanom vremenskom periodu. Najbolje rešenje bi bila kompleksnija organizacija podataka za koju je potrebno dodatno vreme.

3.4 Modularnost

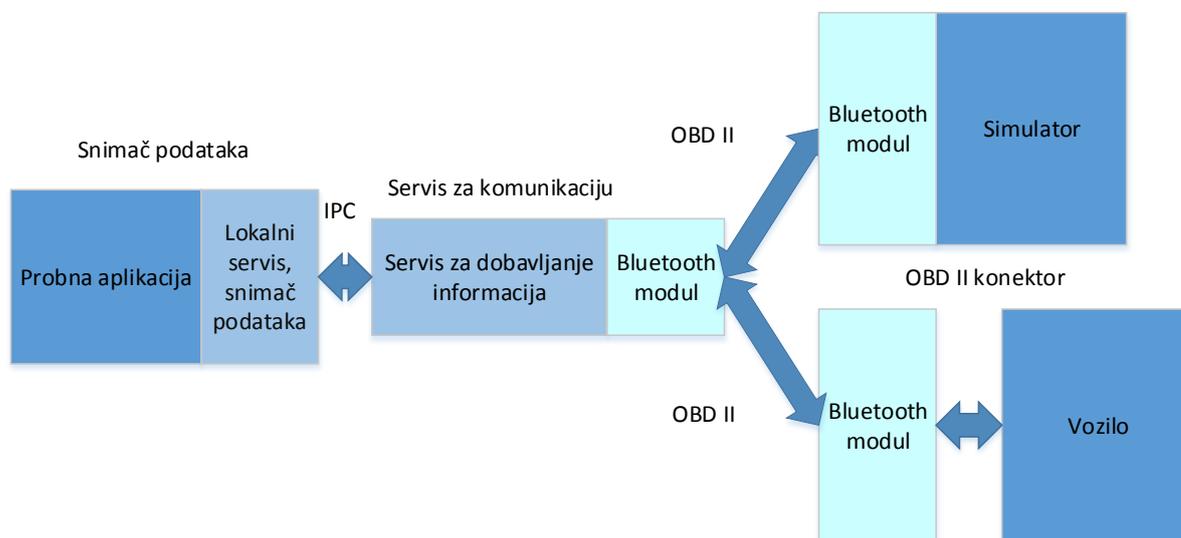
Kako bi snimač podataka bio pogodan za ispravku ili eventualnu nadogradnju funkcionalnosti potrebno je modulisati, to jest grupisati njegove module u manje celine kako bi način njegovog funkcionisanja bio što jasniji. To se radi, na primer, ukoliko je potrebno dodati funkcionalnost za prikupljanje podataka novih parametara. Ukoliko je modul sa API funkcijama za rad sa bazom podataka izdvojen i smisleno organizovan dodavanje novih tabela i funkcija ne bi trebalo da predstavlja komplikovan i dugotrajan problem.

3.5 Prednosti u odnosu na ostala rešenja

Razlog za uvođenje SQLite baze podataka je olakšanje pretraživanja podataka u odnosu na upis podataka u tekstualnu datoteku ili slična rešenja. SQLite fasciklu je moguće veoma lako učitati u bilo koji pregledač za ovaj tip podataka i vršiti pretragu podataka SQL upitima. Ovaj pristup pored dobrih osobina ima i loših, što će biti objašnjeno u narednim poglavljima.

4. Programsko rešenje

U ovom poglavlju je predstavljeno programsko rešenje snimača podataka po modulima. Snimač za komunikaciju sa vozilom odnosno simulatorom događaja u vozilu koristi poseban servis za dobavljanje informacija. Veza između servisa za dobavljanje informacija i simulatora događaja u vozilu ili vozila je ostvarena preko Bluetooth bežične tehnologije. Zbog toga svaki uređaj koji koristi ovo rešenje snimača mora da poseduje podršku za ovaj vid komunikacije. Struktura i način komunikacije snimača prikazana je na *Slika 8*.



Slika 8 Modularni prikaz funkcionisanja snimača podataka

4.1 Snimač podataka

Na *Slika 8* modulom sa leve strane je prikazan snimač tojest tema ovog rada koji je realizovan u vidu lokalnog servisa i testne aplikacije koja prezentuje njegov rad.

4.1.1 Servis za prikupljanje podataka

Modul za prikupljanje parametara je realizovan kao servis i u sebi sadrži klase sa funkcijama za rad sa bazama podataka i klase-programske niti koje vrše prikupljanje parametara. Prilikom pokretanja servisa iz testne aplikacije servis dobavlja podešavanja na osnovu kojih vrši prikupljanje podataka o vozilu. Nakon toga proverava da li baza podataka već postoji, ukoliko postoji čuvanje parametara će biti nastavljeno u prethodno već napravljen bazu a u suprotnom servis će kreirati novu bazu i nove tabele u koje je potrebno smeštati podatke. Posle uspešnog proveravanja da li je SQLite baza podataka spremna za rad servis sa spoljašnje memorije uređaja tojest memorijske kartice iz fascikle *sdcard/OBD2APP/* učitava datoteku *ErrorCode.json* koja sadrži listu svih kodova grešaka sa njenim objašnjenjima. Na ovaj način veoma je jednostavno uskladiti kodove grešaka sa različitim modelima vozila. Ukoliko na ovoj lokaciji ne postoji datoteka pod tim imenom, servis će učitati svoju datoteku sa opštim kodovima grešaka koja su standardna za većinu vozila. Ova datoteka se nalazi u folderu *assets* i integrisana je zajedno sa aplikacijom pa joj nije moguće pristupiti bez postojanja izvornog koda. Nakon toga servis-snimač pokušava da uspostavi konekciju sa servisom za prihvatanje informacija o vozilu, pogledati sliku 7. Ako je među procesorska komunikacija uspešna, snimač pokreće programske niti koje prema unetim podešavanjima šalju zahteve za podatke dok korisnik ne zaustavi servis iz probne aplikacije. Modul za prikupljanje informacija o lokaciji ne poseduje programsku nit i realizovan je na drugačiji način.

4.1.1.1 Niti za slanje zahteva o stanju željenih parametara

Ovo rešenje snimača podataka poseduje mogućnost praćenja šest parametara: trenutna brzina, nivo goriva, pozicija pedale za regulaciju brzine, broj obrtaja motora, informacije o greškama i informacije o lokaciji vozila. Sve programske niti koriste biblioteku tojest API servisa za prihvatanje informacija "ICarInfo". Niti prema unetim podešavanjima pozivaju funkcije API-ja i šalju zahteve preko servisa za prihvatanje informacija simulatoru događaja u vozilu ili vozilu. Simulator ili vozilo u najkraćem mogućem roku u zavisnosti od izabranog protokola komunikacije OBD II sistema vraća podatke o željenim parametrima. Svaka nit prvo proverava da li je došlo do greške pri komunikaciji ili centralni računar ECU vozila nema informacije o traženom parametru. Ukoliko nema "bluetooth" konekcije servis za prihvatanje informacija vraća vrednost "-1" kao kod greške. Nit na osnovu ovog koda greške upisuje u bazu podataka da u tom trenutku nije bilo komunikacije sa vozilom odnosno simulatorom. Ukoliko ECU nema informacije o traženom parametru servis za prihvatanje informacija vraća vrednost "-2" kao kod greške a nit upisuje u bazu podataka da u tim trenucima nema informacija o traženom parametru. Ukoliko nit pozivom API funkcija dobije validnu vrednost upisuje je u bazu podataka. Svaka nit

vodi računa da ukoliko nije došlo do promene vrednosti parametara između dva upita ne upisuje istu vrednost ponovo. Ipak nakon isteka predviđenog vremena koje je korisnik uneo u podšavanjima probne aplikacije nit upisuje trenutnu vrednost u bazu iako nije došlo do promena. Na taj način se može protumačiti da snimač radi iako nije došlo do promene vrednosti parametara duže vreme. Nit nakon završetka jednog ciklusa ponovo šalje zahtev po istoj proceduri na učestanosti koju je korisnik zadao u podešavanjima aplikacije. API funkcije servisa za dobaljanje informacija koje su korišćene u nitima za slanje zahteva o ostanju parametara prikazane su u *Tabela 2*.

<pre>int requestSpeed(ICarInfoSpeedNotification speedNotification)</pre> <p>Funkcija koja za parametar prihvata spregu koji će biti povratni poziv(engl. <i>callback</i>). Brzina iz prosleđene funkcije klijentu će biti vraćena kroz povratni poziv. Ako je povratna vrednost funkcije „-1“ nema „bluetooth“ konekcije, za vrednost „-2“ ECU nema informacije o traženom parametru.</p>
<pre>int requestThrottlePosition(ICarInfoThrottleNotification throttleNotification)</pre> <p>Funkcija koja za parametar prihvata spregu koji će biti povratni poziv(engl. <i>callback</i>). Pozicija papučice iz prosleđene funkcije klijentu će biti vraćena kroz povratni poziv. Ako je povratna vrednost funkcije „-1“ nema „bluetooth“ konekcije, za vrednost „-2“ ECU nema informacije o traženom parametru.</p>
<pre>int requestEngineRpm(ICarInfoEngineRpmNotification engineRpmNotification)</pre> <p>Funkcija koja za parametar prihvata spregu koji će biti povratni poziv(engl. <i>callback</i>). Broj obrtaja iz prosleđene funkcije klijentu će biti prosleđen kroz povratni poziv. Ako je povratna vrednost funkcije „-1“ nema „bluetooth“ konekcije, za vrednost „-2“ ECU nema informacije o traženom parametru.</p>
<pre>int requestFuelLevel(ICarInfoFuelLevelNotification fuelLevelNotification)</pre> <p>Funkcija koja za parametar prihvata spregu koji će biti povratni poziv(engl. <i>callback</i>). Nivo goriva iz prosleđene funkcije klijentu će biti prosleđen kroz povratni poziv. Ako je povratna vrednost funkcije „-1“ nema „bluetooth“ konekcije, za vrednost „-2“ ECU nema informacije o traženom parametru.</p>
<pre>int requestTroubleCodes(ICarInfoTroubleCodesNotification troubleCodesNotification)</pre> <p>Funkcija koja za parametar prihvata spregu koji će biti povratni poziv(engl. <i>callback</i>). Kodovi greške iz prosleđene funkcije klijentu će biti prosleđena kroz povratni poziv. Ako je povratna</p>

vrednost funkcije „-1“ nema „bluetooth“ konekcije, za vrednost „-2“ ECU nema informacije o traženom parametru.

Tabela 2 Korišćene funkcije iz API servisa za prihvatanje informacija

4.1.2 SQLite modul za bazu podataka

U ovom rešenju je korišćena SQLite biblioteka za rad sa bazama podataka. Razlog za ovakav izbor je jednostavnost rukovanja podacima i tip licence koja je besplatna to jest može se koristiti u bilo koje svrhe. Ovaj modul se sastoji iz dve klase API koji sadrži funkcije za rad tabelama *OBD2appDbManagerAPI* i *OBD2DbHelper* koja nasleđuje osnovnu klasu za kreiranje i kontrolu verzije baze *SQLiteOpenHelper*.

4.1.2.1 API baze podataka

Ovaj modul sadrži sve funkcije za rad sa tabelama u bazi podataka. Spisak funkcija dat je u *Tabela 3*. U ovom poglavlju su predstavljene funkcije samo za jedan parametar jer svi ostali parametri imaju iste date funkcije sa drugim imenom. Izetak su funkcije *open()*, *close()* i *getDatabase()* koje služe za rad sa celokupnom bazom podataka.

public void open(SQLiteDatabase db, Context context)

Ova funkcija se poziva na početku rada servisa i omogućava da baza podataka dobije ovlašćenja za čitanje i upis podataka. Bez poziva ove funkcije ne bi bilo moguće upisati niti isčitati podatke iz nje. Funkcija kao parametre prima objekat baze podataka i kontekst aplikacije.

public void close(SQLiteDatabase db)

Ova funkcija ukida ovlašćenja za upis i čitanje podataka u bazu. Poziva se na kraju rada servisa kako bi se sprečio neovlašćen pristup podacima. Kao parametar prima objekat baze podataka koju treba „zatvoriti“.

public void insertSpeedInDb(SQLiteDatabase db, Parameter parameter)

Ova funkcija vrši upis primljenog parametra u bazu podataka. Kao parametre prima objekat baze podataka i podatak koji je potrebno sačuvati.

public boolean updateEntrySpeed(SQLiteDatabase db, long rowIndex, Parameter parameter)

Ova funkcija vrši korekciju elementa u tabeli prosleđenog parametra. Kao parametre prima objekat baze podataka u kojoj se vrši korekcija, broj reda u kom se element nalazi i element koji treba umetnuti umesto postojećeg. Kao povratnu vrednost vraća uspošnost rada funkcije.

<p>public int removeAllEnteriesSpeed(SQLiteDatabase db)</p> <p>Ova funkcija vrši brisanje svih elemenata tabele. Kao parametar prima objekat baze podataka u kojoj se tabela nalazi. Kao povratnu vrednost vraća kod greške.</p>
<p>public List<Parameter> getAllParameterSpeed(SQLiteDatabase db)</p> <p>Ova funkcija vraća sve elemente iz tražene tabele. Kao parametar prima objekat baze podataka u kojoj se tabela nalazi. Kao povratnu vrednost vraća listu elemenata traženog parametra.</p>
<p>public int getParameterCountSpeed(SQLiteDatabase db)</p> <p>Ova funkcija vraća broj elemenata u traženoj tabeli. Kao parametar prima objekat baze podataka u kojoj se tabela nalazi. Kao povratnu vrednost vraća broj elemenata u traženoj tabeli.</p>
<p>public List<Parameter> getBetweenParameterSpeed(SQLiteDatabase db, int min, int max)</p> <p>Ova funkcija vraća sve elemente tabele čija je vrednost između vrednosti minimuma i maksimuma. Kao parametar prima objekat baze podataka u kojoj se tabela nalazi i minimalnu odnosno maksimalnu traženu vrednost. Povratna vrednost je lista elemenata čija je vrednost između parametara min i max.</p>
<p>public List<Parameter> getDateBetweenParameterSpeed(SQLiteDatabase db, long min, long max)</p> <p>Ova funkcija vraća sve elemente iz tražene tabele koji su se dogodili između dva vremenska intervala. Kao parametar prima objekat baze podataka u kojoj se tabela nalazi i početni kao i krajnji interval u milisekundama. Povratna vrednost je lista elemenata čija je vrednost između dva vremenska intervala.</p>
<p>public SQLiteDatabase getDatabase()</p> <p>Ova funkcija vraća objekat baze podataka ove aplikacije ukoliko postoji. Pozivom ove funkcije moguće je dobiti instancu baze podataka u svakom modulu. Povratna vrednost je objekat baze podataka snimača podataka.</p>

Tabela 3 Spisak API funkcija baze podataka

4.1.2.2 OBD2DbHelper

Ova klasa nasleđuje klasu *SQLiteOpenHelper* i bez nje je nemoguće kreirati bazu. U njoj je neophodno implementirati metode *onCreate()* i *onUpdate()*. Ova klasa sa datim metodama sama brine o otvaranju baze ako ona već postoji, njenom kreiranju ako nije prethodno kreirana i njenom ažuriranju. Pozivom konstruktora ove klase kreira se baza podataka što je prikazano u *Tabela 4*.

```
public void OBD2DbHelper(Context context) {super (context, DATABASE_NAME, null, DATABASE_VERSION);}
```

Ovaj konstruktor kao parametar prihvata kontekst aplikacije i kreira bazu podataka čije je ime *DATABASE_NAME* a verzija *DATABASE_VERSION*. Pogledati poglavlje 2.4.

```
public void onCreate(SQLiteDatabase db)
```

Ova funkcija služi za otvaranje baze ukoliko ona već postoji ili njeno kreiranje ukoliko ona ne postoji. Kao parametar prima objekat baze koji je potrebno kreirati.

```
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
```

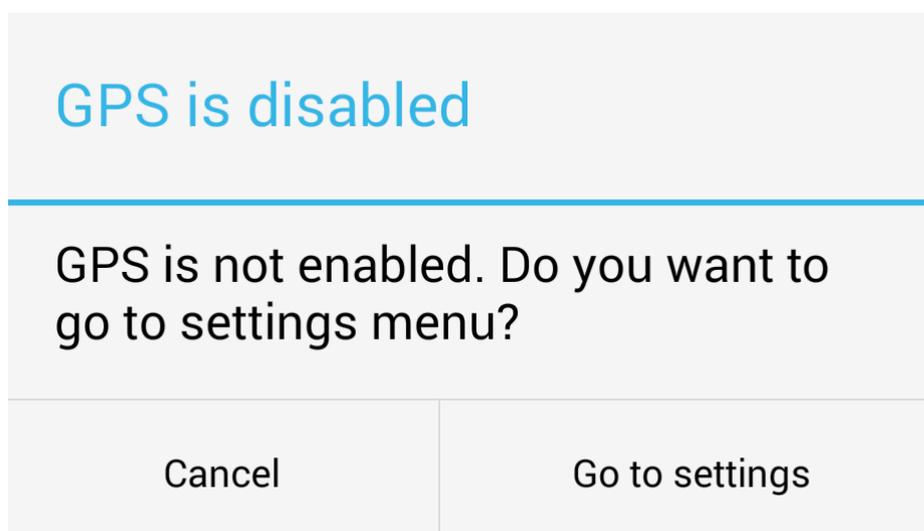
Ova funkcija služi za ažuriranje verzije baze podataka. Kao parametar prima objekat baze koju je potrebno ažurirati i brojnu vrednost stare i nove verzije.

Tabela 4 Objašnjenje konstruktora i funkcija OBD2DbHelper klase

Unutar funkcija *onCreate()* i *onUpdate()* pozivaju se SQL upiti. Na primer u funkciji *onCreate()* ćemo pozvati upit za kreiranje nove tabele. To se vrši sa pozivom funkcije *execSQL()* nad objektom baze podataka koja je prethodno kreirana. U njoj se prosleđuje tekst koji predstavlja SQL naredbu. U ovom slučaju ukoliko želimo da napravimo tabelu za smeštanje informacija o brzini učinićemo to pozivom *db.execSQL(DATABASE_CREATE_SPEED_TABLE)*.

4.1.3 GPS modul za praćenje lokacije

Ovaj modul služi za prikupljanje podataka o kretanju vozila. Uređaj na kome se bude integrisalo ovo rešenje snimača podataka moraće da poseduje podršku za GPS. Za potrebe testiranja snimača korišćen je Android mobilni telefon sa GPS podrškom. Pri startovanju servisa ukoliko detekcija lokacije nije omogućena na uređaju iscrtaće se upozoravajući dijalog kao što je prikazano na *Slika 9*.



Slika 9 Prikaz GPS dijaloga

Ovaj modul upisuje podatke u bazu podataka kada se desi promena lokacije. Ukoliko nema promene lokacije modul ne upisuje nikakve podatke. Jedno polje u bazi podataka sadrži geografsku širinu i dužinu kao i vreme u kom je lokacija zabeležena.

4.1.4 Modul za parsiranje kodova grešaka

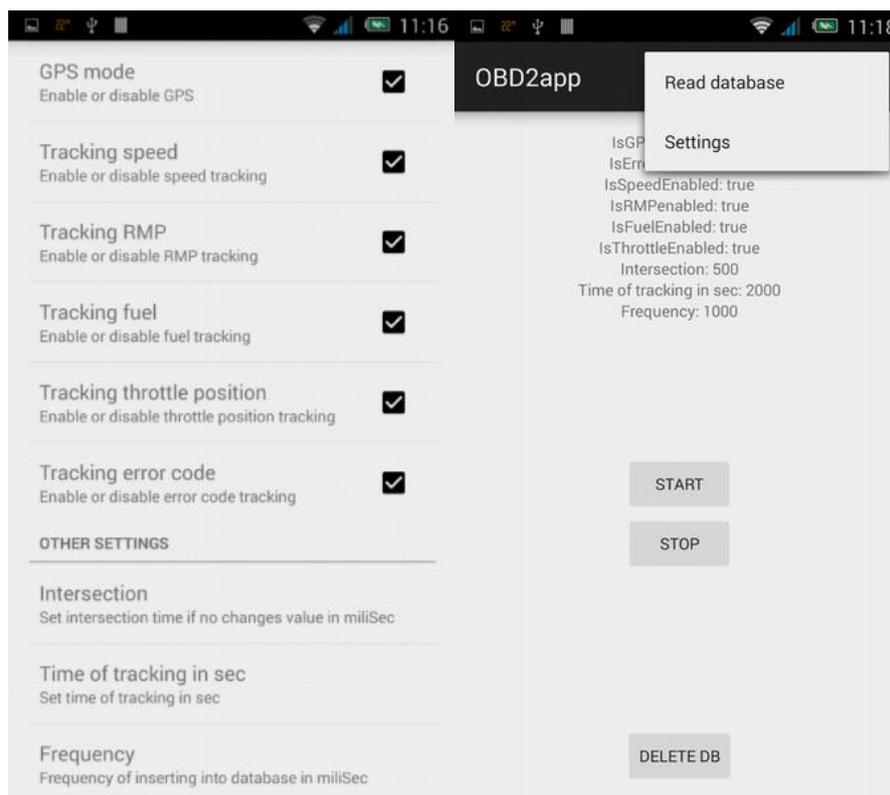
Ovaj modul vrši konverziju to jest dobavljanje (engl. *Parse*) elemenata *JSON* datoteke u listu objekata grešaka sa njihovim objašnjenjima. Modul na početku rada probne aplikacije učitava datoteku sa greškama i pozivom funkcije *loadFileAsString(InputStream stream)* pretvara je u znakovni niz, a zatim pozivom funkcije *parseJSON(String json)* koja kao parametar prima prethodno učitani niz karaktera pravi listu Java listu objekata. Svaki objekat liste sadrži dva polja, kod greške i njeno objašnjenje. Izgled *JSON* datoteke prikazan je u nastavku.

```
[
  {
    "description": "Fuel Volume Regulator Control Circuit\Open",
    "name": "P0001"
  },
  {
    "description": "Fuel Volume Regulator Control Circuit Range\Performance",
    "name": "P0002"
  },
  {
    "description": "Fuel Volume Regulator Control Circuit Low",
    "name": "P0003"
  }
]
```

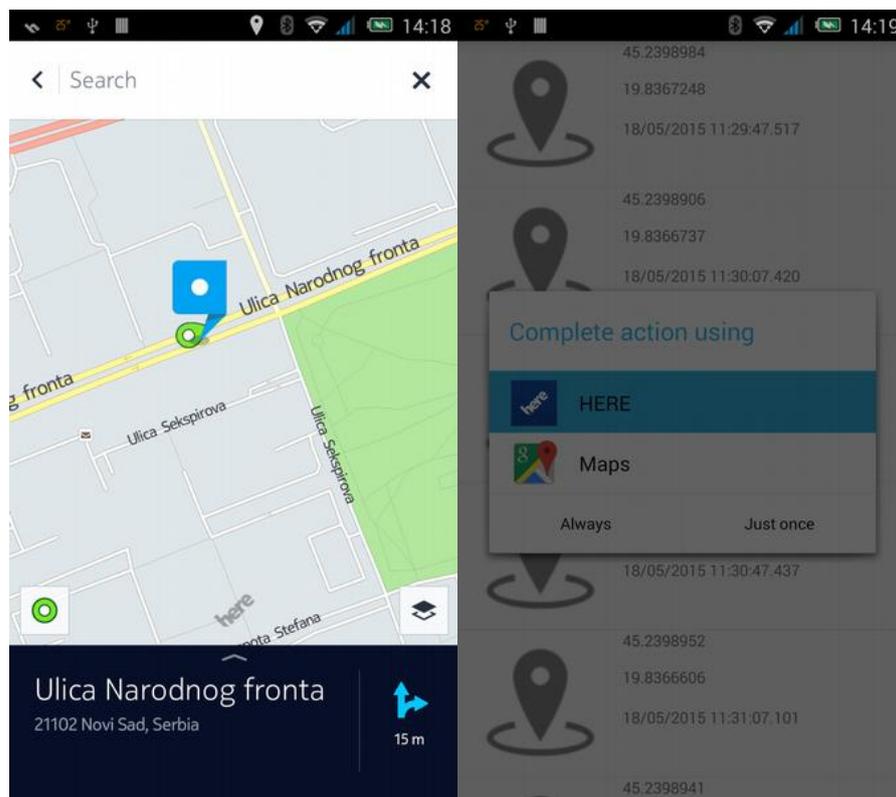
4.2 Probna aplikacija

Probna aplikacija služi za konfigurisanje parametara snimača, njegovo startovanje i zaustavljanje i prikaz dobavljanja podataka iz baze po raznim kriterijumima. U podešavanjima probne aplikacije moguće je izabrati parametre koje je potrebno pratiti. Svaki izbor podešavanja postaje odmah aktivan osim *GPS mode* i podešavanja u *Other Settings*. Za njihovu aktivaciju

neophodno je ponovo pokrenuti servis. Iz aktivnosti čitanja podataka svaki parametar je moguće pretražiti po vrednosti, u opsegu vrednosti i vremenu kada su nastali. Izuzetak su kodovi grešaka i informacije lokacije. Prilikom ispisa kodova grešaka biće pronađeno njeno objašnjenje iz liste grešaka koje smo prethodno učitali iz JSON datoteke. Ukoliko je objašnjenje za datu grešku dostupno biće ispisano zajedno sa kodom greške, u suprotnom će pisati da nema objašnjenja za traženi kod greške. Kod ispisa lokacija gde se vozilo kretalo pritiskom na neku lokaciju biće otvorena neka od aplikacija za prikaz te lokacije na mapi, kao na primer “Google maps” ili “Here&Go”. Na *Slika 10.* i *Slika 11.* Prikazan je izgled testne aplikacije.



Slika 10 Izgled testne aplikacije



Slika 11 Izgled testne aplikacije sa prikazom lokacije u "Here" - Microsoft mapama

5. Rezultati i verifikacija

Da bi se utvrdila ispravnost rada snimača podataka neophodno je generisati odgovarajuće testne slučajeve. Takođe prilikom analiziranja rezultata ustanovljeni su razni načini za optimizaciju potrošnje resursa kao osnovnog problema ovog načina realizacije.

Kao prvi način provere rada, snimač je pušten u rad na duži period i praćena je potrošnja radne memorije, to jest praćeno je da li je došlo do “curenja” memorije. Nakon toga osmišljeno je nekoliko scenarija kako bi se proverio rad snimača u određenim situacijama:

- Scenario 1: Pokušaj izvoza, pristupa, ili brisanja baze podataka ako ona prethodno nije kreirana.
- Scenario 2: Pokušaj aktiviranja GPS modula ukoliko na uređaju nije dozvoljeno korišćenje usluga lokacije.
- Scenario 3: Pokušaj izvoza baze podataka na spoljnu memorijsku karticu ukoliko ona nije u uređaju ili ako nema slobodne memorije.
- Scenario 4: Kada se nasilno prekine Bluetooth konekcija.
- Scenario 5: Kada se nasilno u toku rada onemogućí pristup informacija o lokaciji na uređaju.
- Scenario 6: Kada se u dijalogu za pretraživanje između dve vrednosti parametra unesa nevalidna vrednost, karakter umesto broja.
- Scenario 7: Ukoliko na memorijskoj kartici ne postoji datoteka sa kodovima grešaka.
- Scenario 8: Ako se izade iz aplikacije nakon pokretanja servisa da li on ispravno nastavlja sa radom.
- Scenario 9: Ako simulator događaja u vozilu ili vozilo nema informacije o traženom parametru

Nakon primene scenarija za detekciju greške u snimaču konstatovano je da on u datim slučajevima ispravno radi.

Snimač je takođe ispitan u realnim uslovima, to jest automobilu (*Slika 12*) i nisu uočene nepravilnosti u njegovom radu.

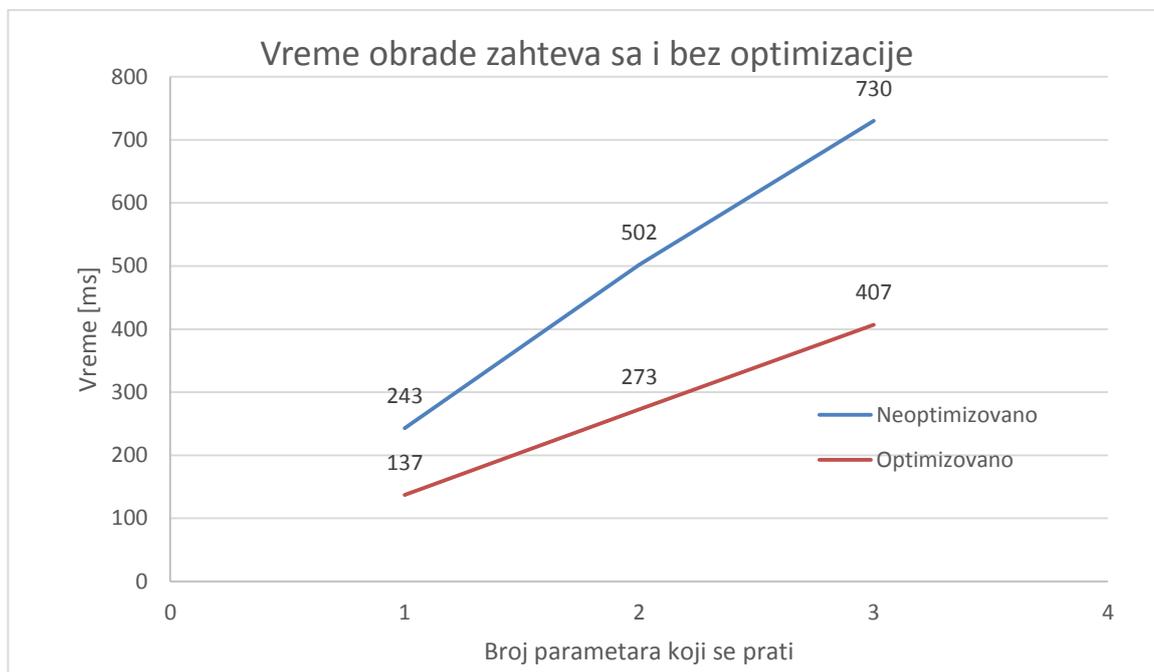


Slika 12 Prikaz testiranja snimača podataka u automobilu VW Bora

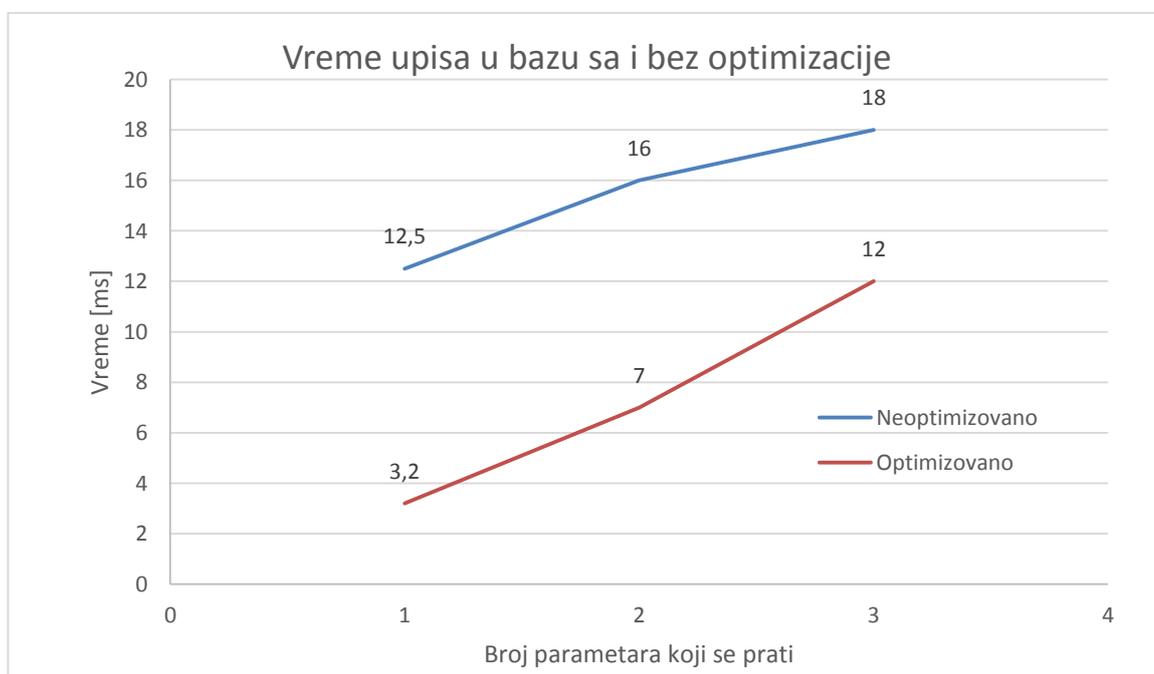
5.1 Optimizacija rešenja

Nakon završene izrade snimača uočeni su mogući načini optimizacije i ubrzanja njegovog rada. Kao što je spomenuto u predhodnim poglavljima, prilikom velikog broja parametara koji se snima i velike učestanost njihovog dobavljanja, snimač troši puno resursa i uređaj na kom se izvršava odlikuje sporost u radu. Za potrebe oprimizacije vršena su merenja vremena koja su potrebna da se izvrši upis u bazu podataka kao i vreme potrebno da se pošalje zahtev ka vozilu odnosno simulatoru i dobije odgovor. Prva optimizacija je izvršena zamenom stare verzije servisa za dobavljanje podataka. Novija verzija ovog servisa koristi bolji pristup pri prihvatanju zahteva od snimača podataka. Formira red čekanja i zahtevi se obrađuju jedan po jedan tako da je moguće prihvatiti nove zahteve dok se trenutni obrađuju. Na taj način su dobijeni bolji rezultati *Slika 11*. Starija verzija servisa za dobavljanje podataka obrađuje zahteve jedan po jedan, to jest ne prihvata novi zahtev dok se ne obradi trenutni koji se obrađuje. Rezultati sa *Slika 13*. i *Slika 14*. su srednje vrednosti na sto uzetih uzoraka. Druga optimizacija donela je manje ali vidne doprinose pri optimizaciji upisa podataka u bazu *Slika 14*. Pri starom pristupu svaki put kada se pošalje zahtev stvaraju se novi objekti koji se koriste za smeštanje podataka u bazu kao i nova programska nit za slanje novog zahteva. Trošenje vremena za stalno kreiranje novih objekata je smanjeno menjanjem vrednosti objekata koji se upisuju. Pri novim rešenju objekat

koji se smešta u bazu podataka se stvara samo jednom i svaki sledeći put se samo menjaju vrednosti njegovih polja pre upisa. Na taj način se izbegava konstantno stvaranje i brisanja objekata pri svakom zahtevu za upis u bazu.



Slika 13 Grafički prikaz vremena obrade parametara sa i bez optimizacije



Slika 14 Grafički prikaz vremena upisa u bazu podataka sa i bez optimizacije

6. Zaključak

Prilikom izrade ovog zadatka korišćena je: SQLite biblioteka za rad sa bazama podataka, Java programski jezik, Android Studio alat za razvoj Android aplikacija i OBD II sistem. Realizovan je snimač događaja u vozilu u vidu Android servisa i Android probne aplikacije. U probnoj aplikaciji je omogućeno podešavanje parametara snimača i dobavljanje skladištenih parametara po raznim kriterijumima. Servis je zadužen za stvaranje baze podataka i njenih tabela kao i prikupljanje željenih parametara od vozila ili simulatora događaja u vozilu.

Tumačenjem dobijenih rezultata je ustanovljeno da primena SQLite baze podatka pored svojih prednosti ima i mana.

Prednost ovog rešenja je jednostavnost pretraživanja to jest dobavljanja podatka po raznim kriterijumima. Ozbiljan problem predstavlja velika potrošnja resursa jer pri prikupljanju velikog broja parametara povećava se broj poziva funkcija za rad sa bazom kao i broj pristupa memoriji. U tim slučajevima uređaj odlikuje sporost u radu.

Karakteristike ovog rešenja se mogu popraviti na dva načina. Prvi način je smanjivanjem učestanosti dobavljanja i upisivanja parametara u bazu. Drugi način je kreiranje većeg broja tabela i kompleksnija organizacija podataka što zahteva dodatno vreme.

Postojeće rešenje se takođe može unaprediti po pitanju zahtevnosti resursa upisom podataka u tekstualnu datoteku. U tom slučaju se ne bi pozivale broje funkcije za rad sa bazom ali bi broj pristupa memoriji ostao približan. Ovakvo rešenje je implementirano u aplikaciji sličnog tipa "Torque" koja se može preuzeti sa Android marketa. Sa druge strane u tekstualnoj datoteci je otežano dobavljati parametre po željenim kriterijumima. Kako bi se omogućilo dobavljanje željenih parametara iz tekstualne datoteke, potrebno je napraviti kompleksan modul za učitavanje i rasčlanjivanje tih podataka kao i modul za pretragu po kriterijumima.

7. Literatura

- [1] Professional Android Application Development, Reto Meier, May 1, 2008
- [2] Professional Android 2 Application Development, Reto Meier, Mar 1, 2010
- [3] Android developer, Android 5.0 dostupno na:
<http://developer.android.com/about/versions/lollipop.html>, učitano 15.05.2015
- [4] Android services, dostupno na:
<http://developer.android.com/guide/components/services.html>, učitano 10.05.2015
- [5] OBD II Streamer family, dostupno na:
http://www.bb-elec.com/Products/Manuals/Intelligent-OBDDII-Gateway-Command-and-Response_251.pdf, učitano 10.05.2015
- [6] ELM327 OBD to RS232 Interpreter, dostupno na:
<http://www.elmelectronics.com/DSheets/ELM327DSF.pdf>, učitano 8.05.2015
- [7] OBD Modes, dostupno na: <http://www.outilsobdfacile.com/obd-mode-pid.php>, učitano 5.05.2015
- [8] SQLite for Mobile Apps Simplified, Sribatsa Das, 2014
- [9] SQLite database, dostupno na: <https://www.sqlite.org/>, učitano 5.05.2015