Android mobilna aplikacija za analizu podataka iz automobila

Živković, Ivan

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Diplomski studij

ANDROID MOBILNA APLIKACIJA ZA ANALIZU PODATAKA IZ AUTOMOBILA

Diplomski rad

Ivan Živković

Osijek, 2023.

SADRŽAJ

1. UVOD

Još od početka autoindustrije postoji problem dijagnostike kvarova. Kako su automobili postajali sve kompleksniji i prelazilo se analogne tehnologije na digitalnu, otvorile su veće mogućnosti kontrole i prikupljanja podataka. Prvo su pojedini proizvođači imali vlastite uređaje za dijagnostiku, problem je bio skupi su ti uređaji skupi usko vezani za proizvođača. Zato je uveden standard OBD2 kako bi se moglo jednostavnije pristupiti podacima svih modernih automobila. Tako danas možemo pristupiti skoro svim parametrima automobila preko jednostavnog adaptera i mobilnog uređaja. Tako su povećane mogućnosti auto entuzijastima za nadziranje i promjene na automobilu. Kao što je moguć nadzor, moguće su i brojne promjene na automobilu, od vremena koliko će dugo sjajiti osvjetljenje u automobilu nakon otključavanja pa čak i do promjene mape motora za dobitak veće snage.

Tako će se u ovome radu govoriti o izradi Android aplikacije za prikupljanje i prikaz podataka sa senzora u modernom automobilu preko ELM327 adaptera i Android mobilnog uređaja.

Prikupljat će se podaci brzine kretanja automobila, broja okretaja motora, temperature raznih senzora kao što su temperatura vode, zraka, ulja.

Rad je podijeljen na sljedeće glavne cjeline:

1. Uvod
2. Pregled postojećih rješenja
3. Opis korištenih tehnologija i alata
4. Izrada aplikacije
5. Završni izgled aplikacije
6. Zaključak

U drugoj cjelini se opisuju postojeće mobilne aplikacije za spajanje s OBD2 adapterom.

U trećoj cjelini dan je opis korištenih alata i tehnologija poput OBD2 sustava, Android studija i java programskog jezika. Opisan je povijesni razvoj OBD2 sustava i njegove mogućnosti.

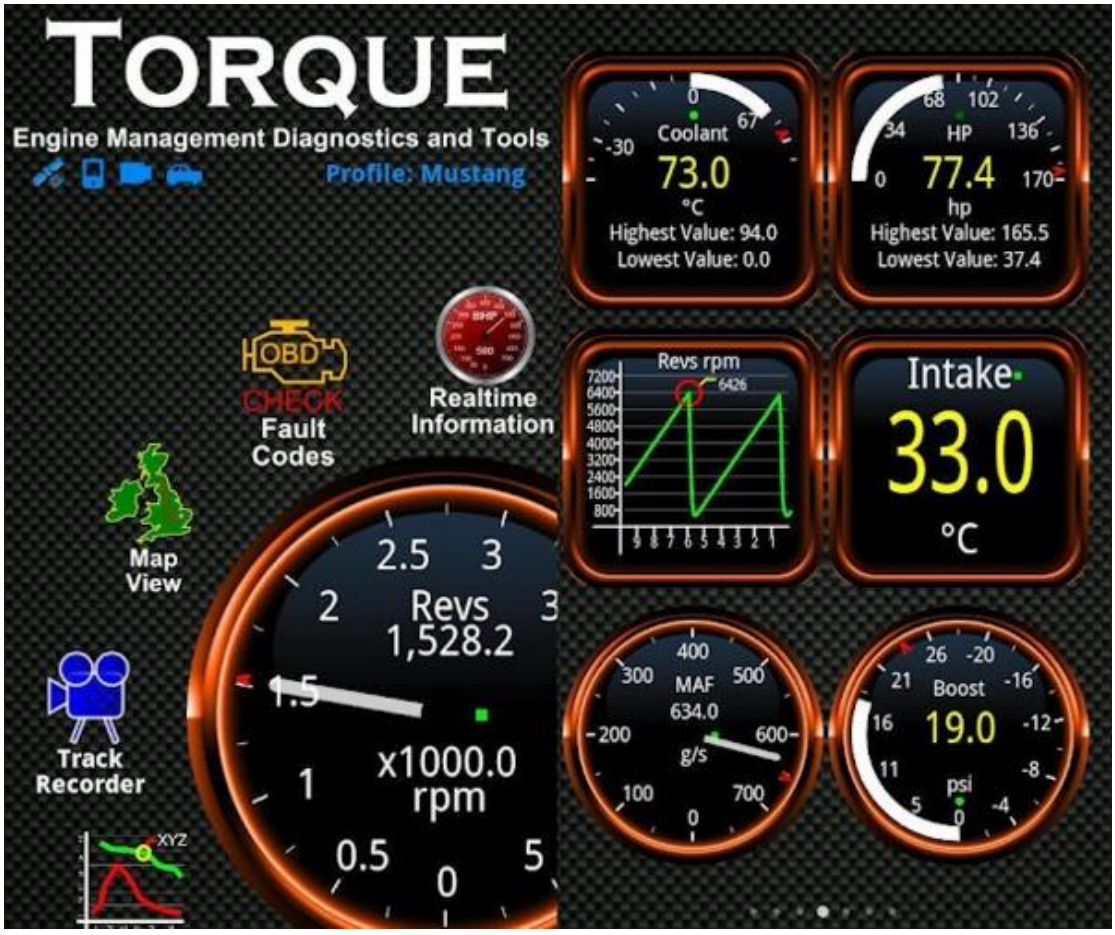
U četvrtoj cjelini opisan je postupak izrade mobilne aplikacije za prikaz parametara automobila, a u zadnjoj cjelini prikazan konačan izgled aplikacije i opisane njene mogućnosti.

2. PREGLED POSTOJEĆIH RJEŠENJA

U ovom poglavlju ćemo pregledati neke od postojećih najpopularnijih android aplikacija dostupnih na Google Play platformi koje imaju slične funkcionalnosti i kratko opisati njihove mogućnosti, kao i prednosti i mane.

2.1 Torque Pro

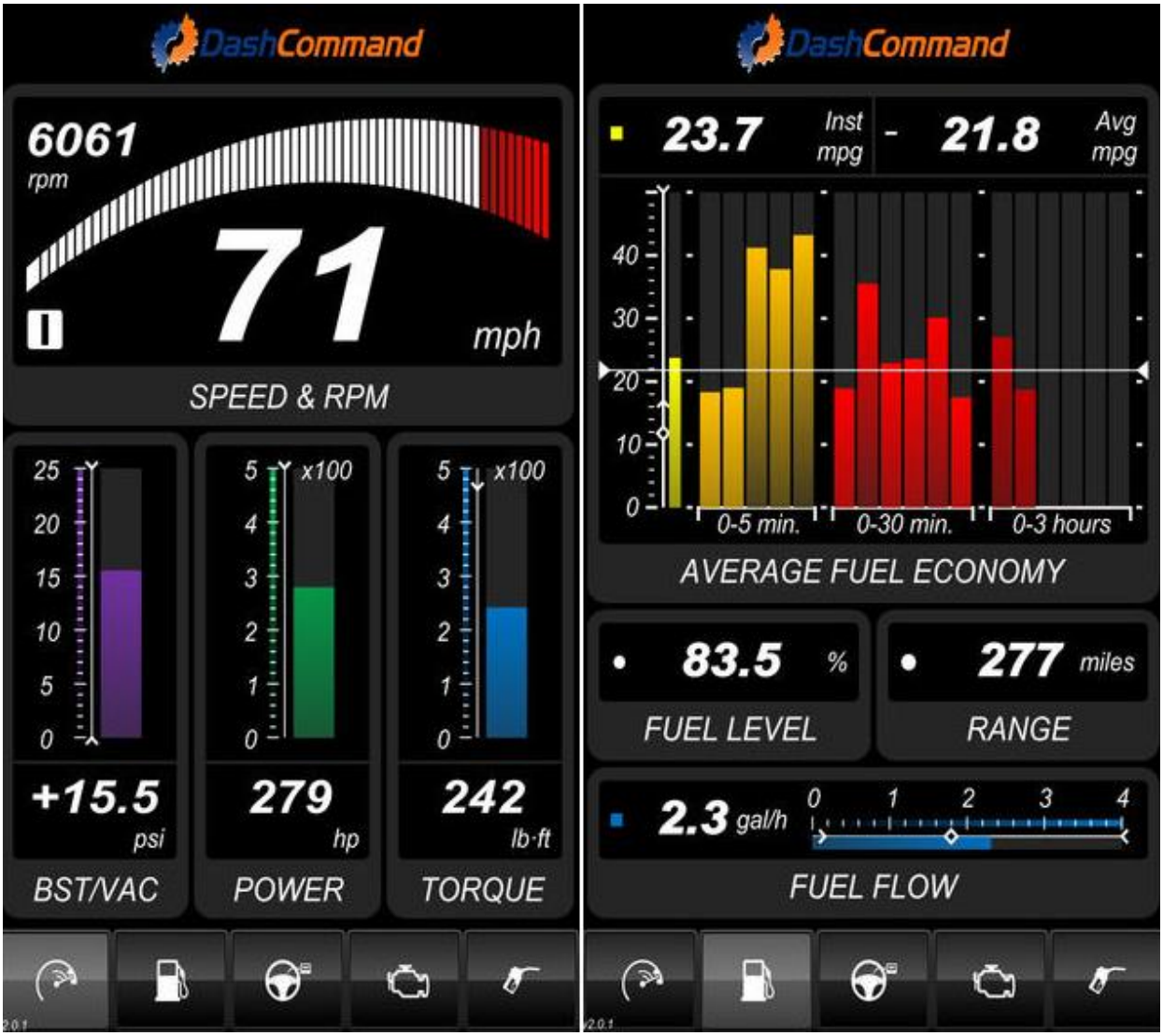
Jedna od najpopularnijih OBD2 aplikacija dostupnih za Android.
Omogućava pregled parametara senzora vozila u stvarnom vremenu, očitavanje grešaka i podataka sa senzora, kao i brisanje postojećih grešaka.
Ima mogućnost dodavanja i raspoređivanja raznih mjerača po želji i radi na svim vozilima koja koriste OBD2 standard.



Slika 2.1 *Torque Pro [1]*

2.2 DashCommand

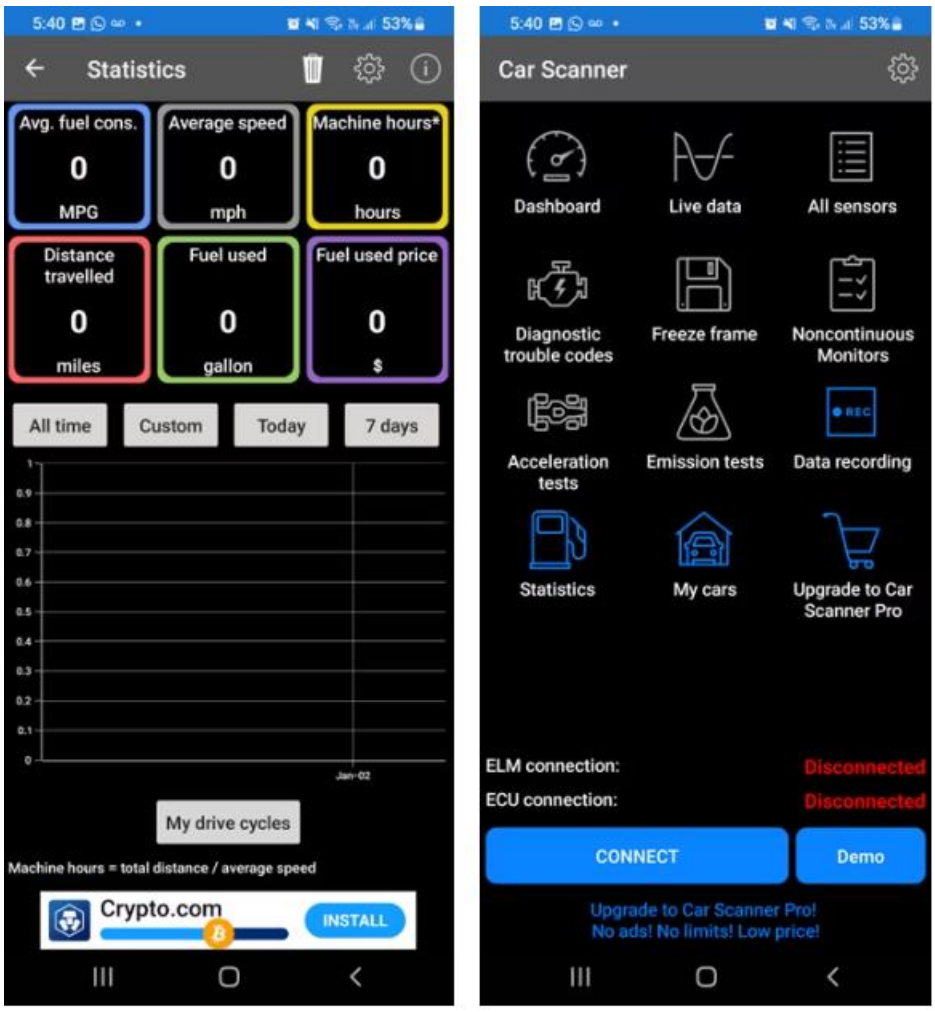
Aplikacija dostupna ne samo za Android nego i Iphone i Windows uređaje, orijentirana je više na praćenje performansi vozila.
Nudi prikaz snage, momenta, ispravljene brzine kretanja vozila, snage kočenja, potrošnje goriva, ali i parametre senzora kao što su temperatura zraka usisa, opterećenje motora i drugi.
Ima dosta kompliciran i vizualno napredan ekran s dosta raznih mjerača.



Slika 2.2 *DashCommand* [2]

2.3 Car Scanner ELM OBD2

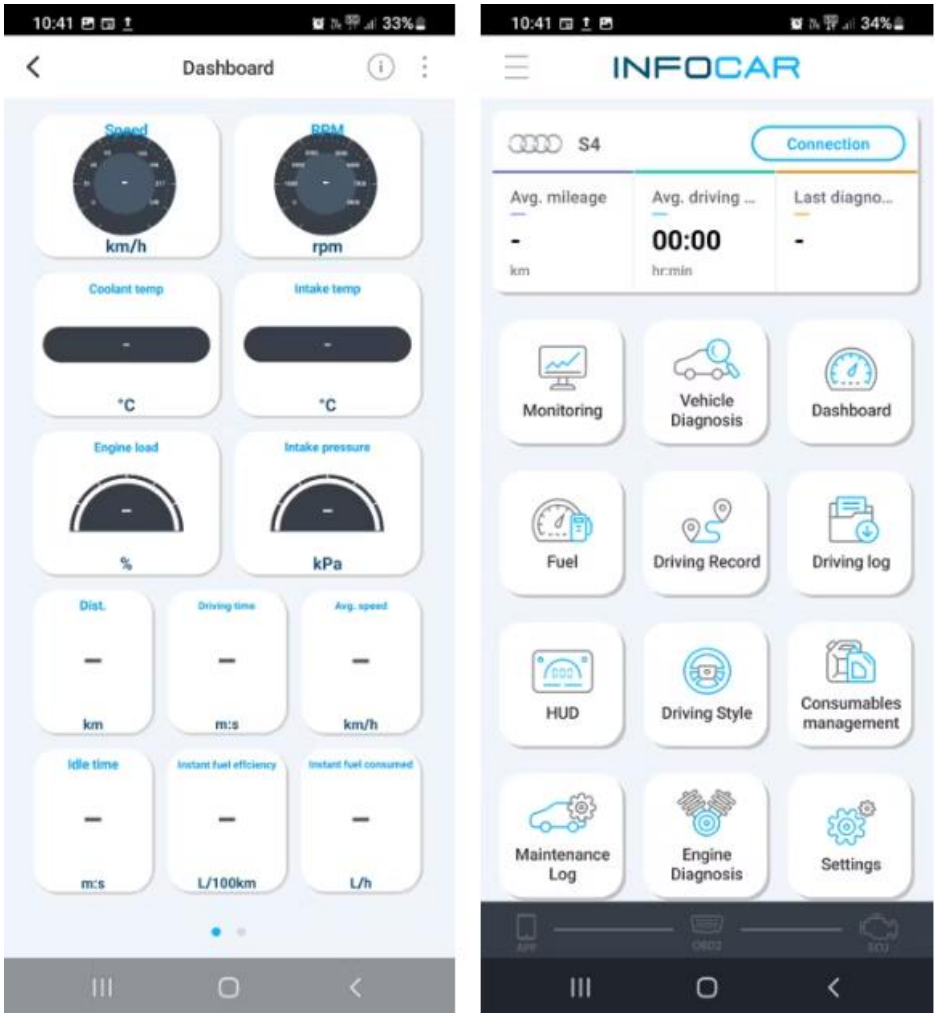
Jednostavna aplikacija s pojednostavljenim sučeljem za svakodnevne korisnike. Nudi čitanje grešaka pohranjenih u memoriju automobila i prikaze parametara senzora kao što su okretaji motora, brzina kretanja, temperature i drugih, te pravljenja statistika s parametrima po želji.



Slika 2.3 *Car Scanner ELM OBD2* [3]

2.4 Infocar

Ova aplikacija također ima funkcionalnosti kao i prethodne, očitavanje grešaka i prikaz stanja senzora, ali i jednu zanimljivu funkcionalnost pamćenja gdje ste parkirali svoj automobil. Također ima i funkciju analize stila vožnje i spremanje povijesti vožnji.



Slika 2.4 *Infocar* [4]

3. OPIS KORIŠTENIH TEHNOLOGIJA I ALATA

U ovome poglavlju će se govoriti o tehnologijama i alatima korištenih pri izradi ove Android mobilne aplikacije.

3.1 OBD sustav

OBD (eng. *On-Bord Diagnostics*) je samodijagnostički elektronički sustav automobila koji ima pristup svim podsustavima automobila te ukazuje i izvještava o problemima unutar vozila. Može se koristiti za prikupljanje mnogih podataka senzora unutar vozila i pružanje mnogih značajki vozaču, ali i tehničaru koji se može spojiti kako bi dobio podatke o vozilu i dijagnosticirao problem.

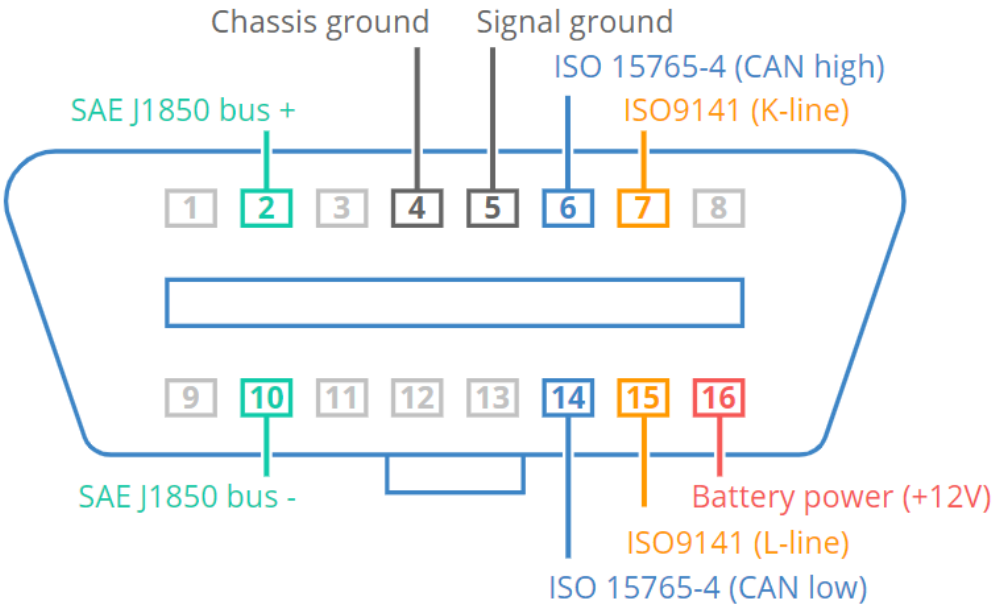
Prvi OBD sustavi se pojavljuju 1968. god od strane Volkswagen-a koji je bio u potpunosti analogan bez mogućnosti dijagnostike i služio je za kontrolu ubrizgavanja goriva tako što je mjerio okretaje motora i gustoću zraka u usisnoj grani motora kako bi odredio potrebnu količinu goriva za ubrizgati. Sljedeći njihov primjer Bosch 1974. god predstavlja svoje sisteme ubrizgavanja koji se ugrađuju u tadašnja europska vozila. Kasnije 1975. god. Počinje se koristiti u svrhu kontrole ispušnih plinova na automobilima što je jedan od glavnih razloga za njegov razvoj zato što OBD sustav ima pristup svim komponentama motora i u slučaju kvara koji može rezultirati povećanim emisijama ispušnih plinova može obavijestiti vozača. Tek 1988. god. u američkoj saveznoj državi Kaliforniji postalo je obavezno za sve novo prodane automobile da posjeduju ugrađen sustav dijagnostike, taj sustav se danas smatra OBD-1. Regulatorna namjera takvog sustava bila je potaknuti proizvođače automobila da dizajniraju pouzdane sustave kontrole emisijskih plinova koji ostaju efikasni tijekom uporabnog vijeka vozila. Nadalo se da će nametanjem godišnjeg testiranja emisija za Kaliforniju i uskraćivanjem registracije vozilima koja nisu prošla test emisijskih plinova, vozači kupovati vozila koja će pouzdanije proći test [5]. OBD-1 je bio prilično neuspješan zato što načini izvješćivanja emisijskih dijagnostičkih informacija nisu bili standardizirani, iz tog razloga se godišnji testovi nisu mogli pouzdano provoditi. Svaki je proizvođač koristio vlastiti konektor dijagnostičke veze, postavljao ga na različite lokacije, koristio različite kodove grešaka i proceduru očitavanja grešaka vozila. Tehničari su morali kupovati novi alat i kabel za svaku marku vozila ili kupovali skener koji ima niz adaptera za više vozila. Zbog

tog problema vlasnici su morali ići u ovlaštene servise pojedinih marki vozila za dijagnostiku problema. Zbog tog problema bio je pritisak na standardizaciju OBD sustava.

Slika 3.1 *OBD adapteri* [6]

Godine 1994 CARB (eng. *California Air Resources Bord)* kako bi riješio problem s korištenjem raznih vrsta priključaka kod raznih proizvođača automobila zahtijeva da se uvede standardizirani 16 žični konektor OBD-2 za sva vozila prodana u Kaliforniji od 1996. god. Te se to primijenilo i na cijelu Ameriku i od tada je u upotrebi. Društvo automobilskih inženjera SAE (eng. *Society of Automotive Engineers*)i međunarodna organizacija za razmjenu digitalnih informacija ISO (eng. *International Organization for Standardization*) su također izdali standarde za razmjenu digitalnih informacija između ECU-a (eng. *Engine Control Unit*) i dijagnostičkog alata za skeniranje kodova [7]. Taj isti standard kasnije 2001.god. postaje obavezan i u Europi. Standard OBD-2 specificira vrstu dijagnostičkog konektora, njegov raspored pinova, signalizacijske protokole i format slanja podataka. Standard SAE J1962 specificira dvije vrste ženskih OBD-2 16-pinskih konektora, A i B. Pin 16 se koristi za napajanje, kod A tipa konektora on je 12 volti i koristi se većinom u automobila, a kod B tipa on je 24 volta i koristi se u vozilima srednje i teške nosivosti. Raspored pinova OBD-2 konektora ovisi o komunikacijskom protokolu [8]. Postoji pet dopuštenih protokola s OBD-2 sučeljem. Većina vozila koristi samo jedan od njih i razlikuju se po tome koje pinove koriste na J1962 konektoru.

- SAE J1850 PWM: Koristi pulsno-širinsku modulaciju signala brzine 41,6 Kb/s, koriste se pinovi 2 i 4 za komunikaciju, 4 i 5 za masu i 16 za napajanje. Ovaj standard koriste stariji Ford automobili.
- SAE J1850 VPW: Koristi varijabilnu pulsno-širinsku modulaciju signala brzine 10,4 Kb/s, au odnosu na SAE J1850 PWM slučaj koristi samo pin 2 za komunikaciju. Koristi se u starijim GM automobilima.
- ISO 9141-2: Ovaj protokol ima asinkroni serijski prijenos podataka brzine 10,4 Kb/s. Komunikacija mu je preko jedne, dvosmjerne linije na pinu 7 i neobavezno pin 15. Uglavnom se koristio na europskim i azijskim automobilima između 2000 i 2004 god.
- ISO 14230 KWP2000 Vrlo čest protokol na automobilima novijim od 2003 god. Fizički sloj mu je identičan kao kod ISO 9141-2 pa je i njemu neobavezan pin 15 koji je korišten za neke automobile kao signal za buđenje ECU kako bi mogla započeti komunikacija preko pina 7.
- ISO 15765 CAN: Ovaj CAN (eng. *Controller Area Network*) protokol je razvio Bosch za automobilsku i industrijsku kontrolu. Za razliku od ostalih OBD protokola verzije ovog se uvelike koriste i van automobilske industrije. Ovaj protokol opisuje fizički, podatkovni i mrežni sloj nastojeći standardizirati sučelje CAN sabirnice za vanjsku ispitnu opremu. Za komunikaciju koristi pinove 6 i 14, od 2008 god. ovaj protokol je obavezan za sve nove automobile koji se prodaju u SAD-u [5].



Slika 3.2 *OBD2 pinout* [8]

OBD sustav se sastoji od središnjeg sustava, mreže senzora, priključne točke i indikatora tvoreći sustav nadzora sa standardiziranim pristupom i čitljivošću.

OBD sustav se sastoji od sljedećih komponenti:

- ECU (eng. *Electronic Control Unit*): Središnji dio OBD sustava je elektronička centralna jedinica (ECU). Ona prikuplja podatke od raznih senzora unutar vozila. Zatim te podatke koristi ili u svrhu kontrole raznih dijelova vozila kao što mlaznice za gorivo ili za praćenje problema.
- Senzori: U svim vozilima postoje senzori koji pokrivaju svako područje od motora i šasije do samog elektroničkog sustava. Svaki od tih sustava šalje kodove ECU-u precizirajući izvor i parametre signala koje ECU čita i tumači.
- DTC (eng. *Diagnostic Trouble Conde*): U slučaju da senzor šalje informaciju ECU-u koja nije unutar zadanog normalnog raspona. ECU sprema informacije kao kod koji se naziva dijagnostički kvar kod (DTC). DTC kod je zapravo popis slova i brojeva koji označavaju izvor i prirodu problema, oni su obično standardizirani. Ali mogu biti i specifični za proizvođača. Kada je DTC spremljen, ECU šalje signal određenom indikatorskom svjetlu kao znak da postoji problem. Detaljan DTC kod možemo dobiti spajajući se odgovarajućim alatom za čitanje grešaka na OBD konektor.
- MIL (eng. *Malfunction Indicator Lights*): Kada ECU primi DTC kod. Ona šalje signal kontrolnoj ploči vozila da uključi odgovarajuća indikatorska svjetla. Ova svjetla službeno su poznata kao svjetla indikacije neispravnosti (MIL), ona pružaju rano upozorenje na postojeći kvar vozila. Ako isto svjetlo treperi, to znači da je problem velik [7].

Slika 3.3 *Prikaz MIL indikatora* [9]

- DLC(eng. *Diagnostic Link Connector*): Svim podacima i DTC kodovima koje je prikupio ECU može se pristupiti preko konektora dijagnostičke veze (DLC). DLC priključak je pristupna točka za vozila s ODB sustavima i često se nalazi ispod upravljačke ploče na vozačevoj strani vozila, a može se nalaziti i na drugim mjestima u gospodarskim vozilima.

Slika 3.4 *Lokacija DLC konektora* [10]

OBD se obično koristi u širokom rasponu vozila kao jednostavan način za dijagnosticiranje problema vozila. Međutim, primjene OBD-a su se proširile kako bi pokrile specifičnija područja nadzora i održavanja vozila, posebno u zadnjih nekoliko godina. Neki od primjera su:

- Praćenje ponašanja vozača: OBD sustav se sve više koristi kao način praćenja ponašanja vozača. Primjerice u automobilima za iznajmljivanje u svrhu praćenja načina na koji vozač upravlja vozilom kako bi se spriječilo divljanje automobilom i neobzirna vožnja tako da se ograniče maksimalni okretaji motora ili onemogućavanje maksimalnog ubrzanja ako automobil nije postigao radnu temperaturu. Osim toga tvrtke mogu instalirati zapisivače podataka u svoj vozni park ili dostavna vozila kako bi pratile ponašanje svojih vozača u stvarnom vremenu, što može pomoći u smanjenju njihove odgovornosti u slučaju prometne nesreće.
- Testiranje emisija: OBD-2 je uobičajena metoda testiranja automobila na emisijske plinove. Ovi sustavi prate emisije kako bi tehničari mogli jednostavno koristiti alat za

skeniranje za provjeru kodova grešaka koji imaju veze sa emisijama kako bi osigurali da je vozilo tehnički ispravno.

- Dodatni instrumenti: Zaljubljenici u vozila i profesionalni vozači često koriste OBD sustave za praćenje mjernih podataka koji se ne prikazuju na instrument ploči u automobilu. Ti podaci se mogu prikazivati na dodatno ugrađenim instrumentima ili slati na mobilni uređaj vozača [7].
- Nadogradnja softvera: Tehničari u servisima spajanjem računala imaju mogućnost nadogradnje softvera automobila ako postoji novija verzija sustava koja poboljšava rad nekog od dijelova automobila. Primjerice nadogradnja sustava multimedije koja osigurava njen brži rad ili omogućavanje nekih funkcionalnosti automobila koje nisu bile tvornički omogućene poput automatskog zaključavanja vozila, sklapanja retrovizora i slično. Moguće su i nadogradnje ECU za učinkovitiji rad motora, optimiziranije mijenjanje brzina automatskog mjenjača za udobniju ili sportsku vožnju. U zadnje vrijeme postaje jako popularno povećavanje snage automobila izmjenom tvorničkog softvera ECU-a s prilagođenim parametrima kako bi se povećala snaga automobila.

Slika 3.5 *OBD2 adapter za izmjenu softvera ECU* [11]

3.2 CAN sabirnica

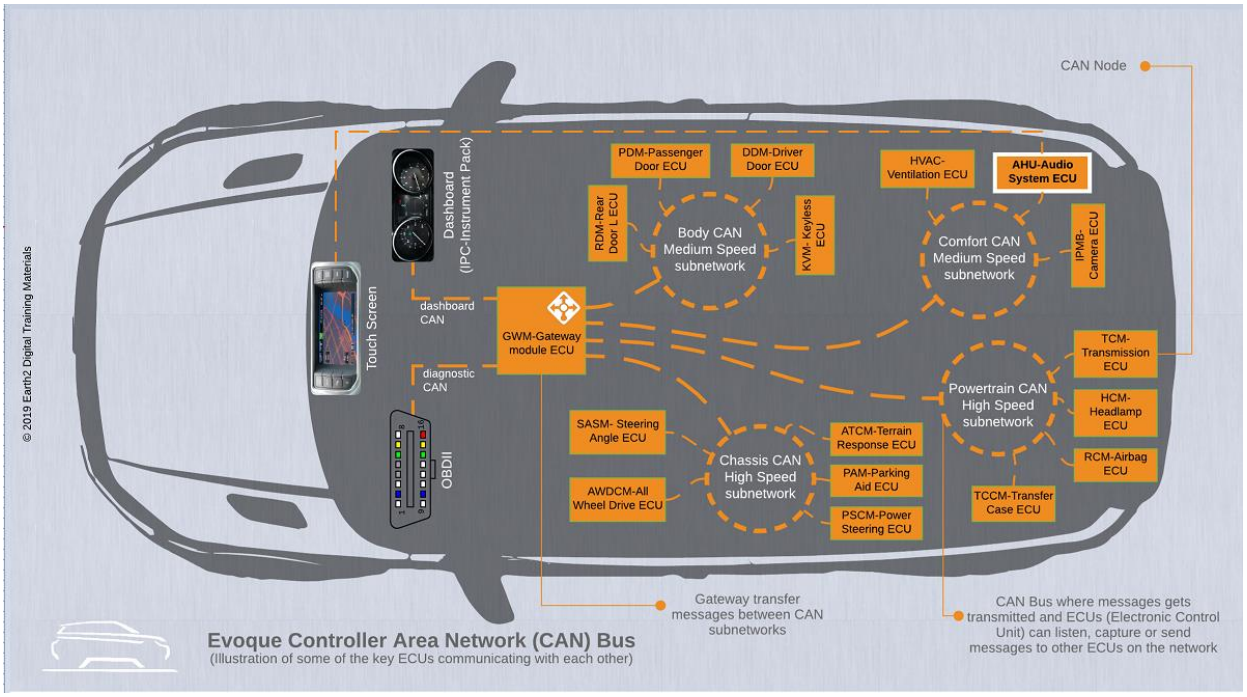
CAN sabirnica(*Controller Area Network*) je robustan standard sabirnice vozila dizajniran da omogući mikrokontrolerima i uređajima da međusobno komuniciraju jedni s drugima bez glavnog računala. To je protokol koji se temelji na porukama, koji omogućuje komunikaciju između pojedinačnih uređaja i sustava unutar mreže [12]. Izvorno dizajniran za multipleksno električno ožičenje u automobilskoj industriji zbog uštede bakra. Podaci između različitih uređaja prenose se serijski u okviru, tako da ako više uređaja odašilje u isto vrijeme, onaj uređaj najvišeg prioriteta može nastaviti, dok se ostali uređaji povlače. Okvire primaju svi uređaji, kao i uređaj za odašiljanje.

CAN sabirnica je poznata i često korištena podatkovna sabirnica. Svoju prvu pojavu je imala 1986. god. na SAE konferenciji u Detroitu. Ona je standardna metoda za komunikaciju između elektroničkih kontrolnih jedinica (ECU) unutar automobila. CAN sabirnica se sastoji od dva vodiča. CAN *High* i CAN *Low* [13]. Signali se prenose diferencijalno na tim vodičima kako bi se smanjio utjecaj elektromagnetskih smetnji. Ovaj način prijenosa signala čini CAN sabirnicu jako pouzdanom metodom komunikacije, posebno u područjima s puno elektromagnetskih smetnji.

CAN sabirnica se koristi u različite svrhe, uključujući prijenos podataka o brzini, temperaturi rashladne tekućine motora, položaju papučice gasa, položaju upravljača, senzoru sigurnosnog pojasa i drugim važnim parametrima. To omogućava različitim kontrolnim jedinicama u automobilu da međusobno razmjenjuju podatke i reagiraju na promjene u okolini. Jedna od najvažnijih prednosti korištenja CAN sabirnice u automobilima je smanjenje broja kabela potrebnih za prijenos signala. Ovo smanjenje broja kabela ne samo da smanjuje troškove proizvodnje i održavanja, nego i smanjuje ukupnu težinu automobila. CAN sabirnice su standardizirane i otvorene, što znači da se mogu koristiti na širokom rasponu uređaja i sustava, bez obzira na proizvođača. Tako se one koriste i u drugim industrijama poput industrijske automatizacije i željezničkih sustava. U tim područjima se koriste za povezivanje različitih komponenti i senzora, omogućujući im da brzo i pouzdano komuniciraju i razmjenjuju informacije.

Moderni automobil može imati između 20 do 100 elektroničkih kontrolnih jedinica (ECU) za različite podsustave. Tradicionalno, najveći procesor je upravljačka jedinica motora. Drugi se koriste za autonomnu vožnju, napredni sustav pomoći vozaču (ADAS), mjenjač, zračne jastuke, i mnoge druge kao jedinica za kontrolu značajki vrata (DCU) koja detektira otvaranje i zatvaranje

vrata, pomicanje prozora, pomicanje zrcala. Taj modul u komunikaciji s drugima obavještava vozača ako pokrene automobil ako neka vrata nisu zatvorena ili u slučaju automobila s automatskim mjenjačem, vozač uopće ne može pokrenuti vozilo.



Slika 3.6 *CAN sabirnica u automobilu* [13]

CAN sabirnica također ima i neke nedostatke. Jedan od njih je ograničenje brzine prijenosa podataka, što može biti problem u nekim primjenama. Također CAN sabirnica nije baš sigurna protiv napada hakera i drugih sigurnosnih prijetnji. Kako bi se poboljšala sigurnost, razvijene su dodatne značajke kao što je CAN sabirnica sa šifriranim porukama, što omogućava sigurniju komunikaciju između različitih komponenti. Uz rastuću potrebu za sigurnijom i učinkovitijom komunikacijom između različitih elektroničkih komponenti u modernim vozilima i drugim industrijskim primjenama, razvoj CAN sabirnice i dalje napreduje i predstavlja ključni element u suvremenim vozilima.

3.3 ELM 327

ELM327 je programirani mikrokontroler proizveden za prevođenje sučelja za dijagnostiku na vozilu (OBD) koji se nalazi u većini modernih automobila. Naredbeni protokol ELM327 jedan je od najpopularnijih standarda sučelja za spajanje računala s OBD2 sučeljem [14].

To je mali, relativno jeftin mikrokontroler koji se koristi u mnogim uređajima, ne može se usporediti s profesionalnim alatima za dijagnostiku, ali može zadovoljiti sve potrebe rekreativnih korisnika.

Mikrokontroler ELM327 funkcionira kao most između ugrađenog računala u automobilu i osobnog ili ručnog uređaja. ELM327 može komunicirati s OBD2 sustavom i prenositi podatke putem USB, Wi-Fi ili Bluetooth veze, ovisno o implementaciji [15].

ELM327 podržava nekoliko protokola Društva automobilskih inženjera (SEA) i Međunarodne organizacije za standardizaciju (ISO), a neki od najpopularnijih su: SAE J1850 , ISO 9141-2 , ISO 15765-4 CAN, ISO 14230-4 i SAE J1939 . ELM327 adapter se može koristiti za dijagnostiku automobila ili kamiona uz upotrebu dodatnog softvera. Može se povezati na računala, pametne telefone, tablete i druge uređaje.

Postoje tri glavne metode spajanja, žičana USB veza i bežične WI-FI i Bluetooth veze.

Prednost spajanja USB vezom je to što je jeftiniji, nema šanse za prekidom veze i nema mana osim što je kabel u većini slučajeva dosta kratak pa se treba koristiti u blizini samog OBD2 konektora. Bežični adapteri imaju prednost što se mogu spajati na veći broj uređaja pomoću WI-FI i bluetooth veze i kompaktniji su. Ovaj adapter nam pruža mogućnost pristupa kodovima grešaka u vozilu i parametrima senzora. S obzirom na to da je komunikacija dvosmjerna, također možemo i brisati kodove grešaka kako bismo riješili problem. Možemo ga koristiti za prikaz brzine kretanja u stvarnom vremenu, trenutnu potrošnju goriva, temperature svih tekućina i slično.

Za korištenje ovog adaptera s pametnim telefonom potrebni su sljedeći koraci:

1. Instalacija dijagnostičke mobilne aplikacije na pametni telefon
2. Spojiti adapter na OBD-2 konektor koji se obično nalazi ispod volana.
3. Uključiti automobil ili okrenuti ključ u poziciju kontakta.
4. Upariti mobilni uređaj s adapterom ako se koristi bežična veza ili ga spojiti na računalo u slučaju USB veze.
5. Pokrenuti dijagnostičku aplikaciju.

Ako je sve odrađeno kako treba, na zaslonu će se prikazati informacije o automobilu i senzori i indikatori će raditi. Nakon toga možemo vidjeti stanje raznih sustava vozila, očitati greške ECU-a, a i obrisati pojedine greške.

Slika 3.7 *ELM327 adapter* [16]

3.4 Android studio

Android Studio je službeno integrirano razvojno okruženje za Google-ov operativni sustav Android, izgrađeno na JetBrains IntelliJ IDEA softveru i dizajnirano posebno za razvoj Androida [17].

Android Studio nudi mnoštvo značajki koje poboljšavaju produktivnost pri izradi aplikacija, neke od njih su:

- Fleksibilan sustav zasnovan na Gradle-u.
- Brz i bogat emulator.
- Jedinstveno okruženje u kome se može razvijati za sve Android uređaje.
- Predlošci koda i GitHub integracija.
- Alati i okviri za testiranje.
- Podrška za C++ i NDK [18].

Android Studio nudi alate za profiliranje performansi tako da možete lakše pratiti upotrebu memorije i CPU-a svoje aplikacije, pronaći slobodne objekte, locirati curenje memorije, optimizirati performanse grafike i analizirati mrežne zahtjeve.

Android Studio koristi Gradle kao temelj sustava izgradnje, s više mogućnosti specifičnih za Android koje pruža Android dodatak za Gradle. Ovaj sustav izgradnje radi kao integrirani alat iz izbornika Android Studija i neovisno o naredbenom retku.

Može se koristiti za prilagođavanje, konfiguriranje i proširivanje procesa izgradnje, za izradu više APK-ova za svoju aplikaciju s različitim značajkama koristeći isti projekt i module, ponovnu upotrebu koda i resursa. Sve to možemo postići bez mijenjanja osnovnih izvornih datoteka koristeći Gradle.

3.5 Java programski jezik

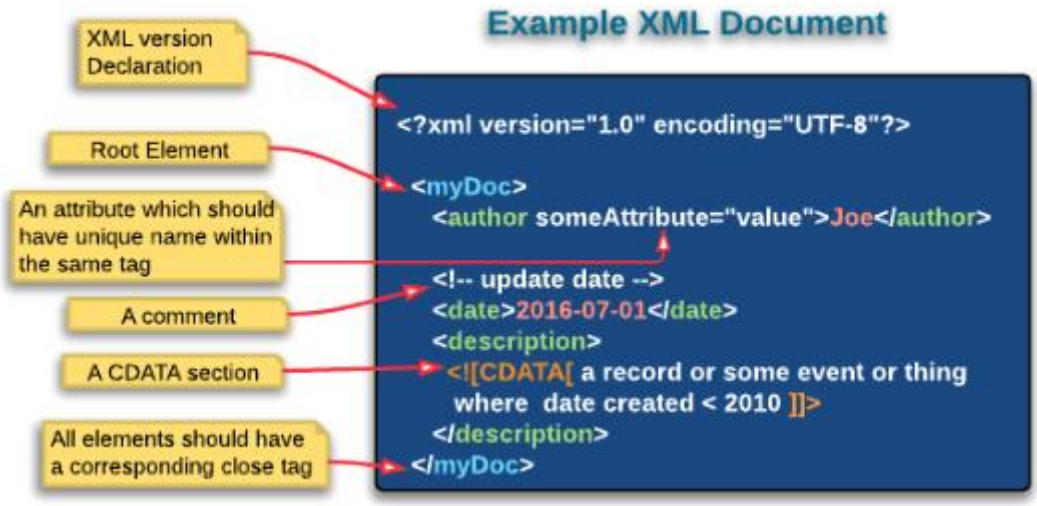
Java je objektno orijentirani programski jezik koji su razvili James Gosling, Patrick Naughton i drugi inženjeri u tvrtki Sun Microsystems. Razvoj je počeo 1991., kao dio projekta Green, a objavljen je u studenom 1995. Java je bila široko popularna u cijelom svijetu, prvenstveno zbog velikog niza značajki koje pruža. Javino obećanje „Napiši jednom i pokreni bilo gdje“ bio je jedan od glavnih čimbenika uspjeha Jave u posljednjih nekoliko desetljeća. Java Mobile Edition je napravljen za izradu aplikacija koje se mogu izvoditi na mobilnim uređajima, te je Java usvojena kao primarni razvojni jezik za izradu android mobilnih aplikacija.

Java programi su sigurni jer se izvode unutar okruženja *sandbox*. Programi napisani u javi se kompiliraju u među kod *bytecode*, taj *bytecode* se zatim izvršava unutar konteksta Java Virtualnog stroja [19].

Mobilno izdanje Jave zove se Java ME. Java ME temelji se na Javi SE i podržava ga većina pametnih telefona i tableta. Java Platform Micro Edition (Java ME) pruža fleksibilno, sigurno okruženje za izgradnju i izvršavanje aplikacija koje su usmjerene na ugrađene i mobilne uređaje. Aplikacije koje su izrađene pomoću Java ME su prenosive, sigurne i mogu iskoristiti izvorne mogućnosti uređaja. Java ME rješava izazov izvršavanja aplikacija na uređajima koji nemaju dovoljno memorije, zaslona i snage [19].

3.6 XML

Extensible Markup Language (XML) je jezik za označavanje i format datoteka za pohranu podataka, prijenos i rekonstrukciju proizvoljnih podataka. Definira set pravila za kodiranje dokumenata u formatu čitljivom i za ljude i za stroj [20]. Velika mu je prednost što je temelj XML formata standardiziran, to znači da ako dodijelimo ili prenosimo XML datoteke preko različitih sustava, lokalno ili internetom, primatelj i svi drugi mogu jednostavno analizirati i pristupiti njegovim podacima zbog standardizirane XML sintakse. U Androidu se XML koristi za implementaciju podataka vezanih za korisničko sučelje, ono je izgrađeno korištenjem hijerarhije različitih glavnih spremnika, izgleda i objekata odnosno *widgeta.*



Slika 3.8 *Primjer XML dokumenta* [21]

4. IZRADA APLIKACIJE

Ideja ove aplikacije je korisniku omogućiti prikaz parametara automobila po vlastitoj želji, od raznih temperatura, brzine, napona, protoka zraka i slično. Također mu pružiti mogućnost spremanja podataka odabranih senzora tijekom vožnje te njihov prikaz na grafu.

4.1 Simulator

Za lakšu izradu i testiranje android aplikacije tijekom njene izrade bilo je potrebno pronaći dobar simulator rada automobila. Za tu svrhu bila je zadužena android mobilna aplikacija ECU Engine Pro. Ova aplikacija pretvara android uređaj u virtualni automobil s povezanim OBD2 bluetooth adapterom. Za njeno korištenje potrebna su 2 android uređaja uparena s bluetoothom, jedan koji pokreće aplikaciju simulatora, a drugi za pokretanje android OBD2 aplikacije. Na ovaj način svaku promjenu u aplikaciji možemo testirati na licu mjesta bez potrebe da se spajamo na automobil. ECU simulacija motora radi kao poslužitelj podataka, tako da čeka zahtjev za podacima (OBD2 naredbe) od vanjskog testera, zatim obrađuje i odgovara na zahtjev. ECU simulacija motora emulira OBD2 standard: ISO 15765-4 CAN 11/500 Kb/s.

Slika 4.1 *ECU Engine Pro Simulator*

4.2 Izrada programskog rješenja

4.2.1 Definiranje globalnih varijabli

Prvi korak je bilo definiranje globalnih varijabli koje su potrebne za olakšani pristup potrebnim podacima u bilo kojem trenutku i bilo gdje u aplikaciji. Na slici 4.2 se vidi primjer definiranja PID (eng.) kodova koji se šalju OBD-u za dobivanje određenih parametara vozila. Prva dva znaka predstavljaju način rada a druga dva PID kod. Primjer „010C“, „01“ predstavlja način rada u stvarnom vremenu, a „0C„ PID kod za dohvaćanje broja okretaja motora.

```
String VOLTAGE = "ATRV",
        PROTOCOL = "ATDP",
        RESET = "ATZ",
        PIDS_SUPPORTED20 = "0100",
        ENGINE_COOLANT_TEMP = "0105",  //A-40
        ENGINE_RPM = "010C",  //((A*256)+B)/4
        ENGINE_LOAD = "0104",  // A*100/255
        VEHICLE_SPEED = "010D",  //A
        INTAKE_AIR_TEMP = "010F",  //A-40
        MAF_AIR_FLOW = "0110", //MAF air flow rate 0 - 655.35   grams/sec ((256*A)+B) / 100  [g/s]
        ENGINE_OIL_TEMP = "015C",  //A-40
        FUEL_RAIL_PRESSURE = "0122", // ((A*256)+B)*0.079
        INTAKE_MAN_PRESSURE = "010B", //Intake manifold absolute pressure 0 - 255 kPa
        CONT_MODULE_VOLT = "0142",  //((A*256)+B)/1000
        AMBIENT_AIR_TEMP = "0146",  //A-40
        CATALYST_TEMP_B1S1 = "013C",  //(((A*256)+B)/10)-40
        STATUS_DTC = "0101", //Status since DTC Cleared
        THROTTLE_POSITION = "0111", //Throttle position 0 -100 % A*100/255
        OBD_STANDARDS = "011C", //OBD standards this vehicle
        PIDS_SUPPORTED = "0120"; //PIDs supported
```

Slika 4.2 *Definiranje PIDa*

Na sljedećoj slici se vidi primjer definiranja *LiveData* varijabli koje u sebi čuvaju izmjerene podatke, a njihovim promatranjem se na svaku njihovu promjenu ažurira korisničko sučelje, tamo gdje je to potrebno.

```
public MutableLiveData<Integer> liveRpm = new MutableLiveData<>();
public MutableLiveData<Integer> liveSpeed = new MutableLiveData<>();
public MutableLiveData<Integer> liveFuel = new MutableLiveData<>();
public MutableLiveData<Integer> liveEngineTemp = new MutableLiveData<>();
public MutableLiveData<Integer> liveInFlowTemp = new MutableLiveData<>();
public MutableLiveData<Integer> liveOilTemp = new MutableLiveData<>();
public MutableLiveData<Integer> liveEngineLoad = new MutableLiveData<>();
public MutableLiveData<Integer> liveThrottlePos = new MutableLiveData<>();
public MutableLiveData<Integer> liveMassAirFlow = new MutableLiveData<>();
```

Slika 4.3 *Definiranje LiveData varijabli*

4.2.2 Uspostavljanje veze s OBD adapterom

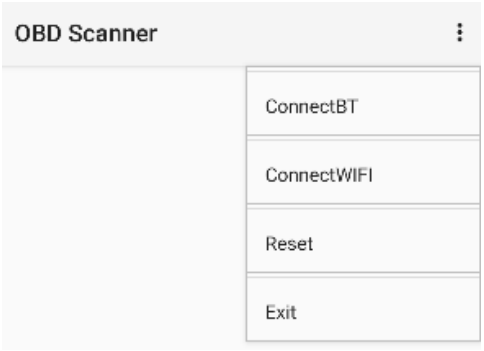
Kako bi se prethodno objašnjeni kodovi mogli poslati OBD adapteru, potrebno je uspostaviti vezu s njime. Za to se koriste komunikacijski servisi. Ovdje je povezivanje objašnjeno na primjeru *bluetooth* veze. Da bi se ostvarila veza potrebno je kreirati *BluetoothHandler*. On radi na osnovu stanja (NONE, LISTEN, CONNECTING, CONNECTED), koja definiraju što je u kojem trenutku potrebno obaviti. Također potrebno je definirati kako se obavlja pisanje i čitanje s bluetooth uređaja. Na slici 4.4 se vidi kako se podaci pročitani s uređaja prosljeđuju funkciji *analyMsg* na daljnje obrađivanje.

```
case MESSAGE_WRITE:
    byte[] writeBuf = (byte[]) msg.obj;
    String writeMessage = new String(writeBuf);
    break;

case MESSAGE_READ:
    String tmpmsg = clearMsg(msg);
    analysMsg(msg);
    break;
```

Slika 4.4 *Prosljeđivanje podataka funkciji analyMsg*

Kako bi se veza s uređajem mogla uspostaviti, potrebno je od korisnika da odabere bluetooth uređaj na koji se želi povezati. Pritiskom na *ConnectBT* tipku na izborniku (Slika 4.5) korisniku se ispisuju svi dostupni prethodno upareni uređaji, te pritiskom na neki od njih započinje proces uspostavljanja veze.



Slika 4.5 *Izbornik*

Na slici 4.6 je prikazano sve što se događa pritiskom na tipku *ConnectBT*. Prvi korak je provjeriti je li bluetooth na uređaju korisnika uključen i je li korisnik pristao dati aplikaciji prava korištenja bluetooth funkcija. Za to se koristi *PermissionService* pozivanjem metoda *checkBtPermission* i *checkScanPermission* što je također vidljivo na slici. Zatim se poziva metoda *startActivityForResoult* koja čeka da korisnik odabere neki od uparenih bluetooth uređaja nakon čega se uspostavlja veza pozivanjem metode *connectDevice*.

```
case R.id.menu_connect_bt:
    if (!mBluetoothAdapter.isEnabled()) {
        Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        mPermService.checkBtPermission();
        startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
        return false;
    }
    if (mBtService == null) setupChat();

    if (item.getTitle().equals(getString(R.string.connectbt))) {
        mPermService.checkBtPermission();
        mPermService.checkBtScanPermission();
        serverIntent = new Intent( packageContext: this, DeviceListActivity.class);
        startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE);
    } else {
        if (mBtService != null) {
            mBtService.stop();
            item.setTitle(R.string.connectbt);
        }
    }
    return true;
```

Slika 4.6 *Potrebni koraci za povezivanje preko bluetootha*

ConnectDevice metoda iz rezultata dobiva adresu bluetooth uređaja, na osnovu koje se uspostavlja veza i omogućuje slanje i primanje podataka od uređaja.

```
private void connectDevice(Intent data) {
    tryconnect = true;
    String address = data.getExtras().getString(DeviceListActivity.EXTRA_DEVICE_ADDRESS);
    BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(address);
    try {
        mBtService.connect(device);
        currentdevice = device;
    } catch (Exception ignored) {}
}
```

Slika 4.7 *Metoda ConnectDevice*

4.2.3 Slanje naredbi

Komunikacija s uređajem je napravljena tako da se nakon povezivanja u beskonačnoj petlji uređaju šalju naredbe te zatim čitaju podaci koje nam uređaj vrati. Na slici 4.8 se vidi da ako rad s uređajem nije inicijaliziran, prvo mu se šalje naredba za uspostavljanje osnovne komunikacije s uređajem, a zatim mu se pozivanjem metode *sendDefaultCommands* šalju sve potrebne naredbe na osnovu kojih uređaj zna koje podatke vratiti.

```
if (!initialized) {
    sendInitCommands();
} else {
    checkPids(tmpmsg);
    if (!m_getPids && m_dedectPids == 1) {
        String sPIDs = "0100";
        sendEcuMessage(sPIDs);
        return;
    }
    try {
        analysPIDS(tmpmsg);
    } catch (Exception e) {
    }
    sendDefaultCommands();
}
```

Slika 4.8 *Uspostava komunikacije s uređajem*

Za slanje naredbi koristi se lista s popisom naredbi, koja se konstantno ažurira i prilagođava potrebama aplikacije. Na slici 4.9 se vidi primjer dodavanja nekoliko naredbi u listu, nakon čega se prolazi kroz tu listu i šalju se naredbe uređaju jedna nakon druge. Trenutnu poziciju na listi, odnosno informaciju o tome koja će se naredba iduća poslati sadržava varijabla *whichCommand*, koja se prilikom ponovnog generiranja liste naredbi resetira na nulu, što je također vidljivo na slici 4.9.

```
whichCommand = 0;
commandslist.clear();
commandslist.add( index: 0, ENGINE_COOLANT_TEMP);
commandslist.add( index: 1, AMBIENT_AIR_TEMP);
commandslist.add( index: 2, VOLTAGE);
```

Slika 4.9 *Popis naredbi*

4.2.4 Rukovanje primljenim podacima

Za svaku prethodno poslanu naredbu, uređaj nam vraća odgovarajuće podatke u heksadecimalnom zapisu. Kako bi se dobili potrebni podaci, potrebno je iz heksadecimalnog zapisa izvući potrebne informacije, te ih pretvoriti u korisne podatke. Svaka poruka koju nam uređaj vrati u sebi sadrži 3 podatka, a to su PID, odnosno informacija o tome na koju od poslanih komandi je ta poruka poslana kao odgovor, te A i B parametre koji sadrže korisne informacije. Na slici 4.10 se vidi kako se analizom od jedne dobivene poruke dobiju prethodno spomenuti podaci, te se s tim podacima poziva metoda *calculateEcuVlaues*.

```
int A = 0, B = 0, PID = 0;
if ((dataRecieved != null) && (dataRecieved.matches( regex: "^[0-9A-F]+$"))) {
    dataRecieved = dataRecieved.trim();
    int index = dataRecieved.indexOf("41");
    String tmpmsg = null;
    if (index != -1) {
        tmpmsg = dataRecieved.substring(index);

        if (tmpmsg.startsWith("41")) {
            int lenght = tmpmsg.length();
            PID = Integer.parseInt(tmpmsg.substring(2, 4),  radix: 16);
            A = Integer.parseInt(tmpmsg.substring(4, 6),  radix: 16);
            if(tmpmsg.length() == 8) B = Integer.parseInt(tmpmsg.substring(6, 8),  radix: 16);

            calculateEcuValues(PID, A, B);
```

Slika 4.10 *Obrada podataka*

Svaka PID, odnosno naredba, ima propisanu matematičku formulu na osnovu koje se dobiju podaci iz parametara A i B. Na slici 4.11 se vidi kako se na osnovu dobivenog PID-a određuje način na koji se iz parametara A i B računaju korisni podaci. Ovdje je dan primjer računanja opterećenja motora, iz kojeg se također izračunava i prosječna potrošnja goriva. Nakon čega se ažuriraju vrijednosti prethodno objašnjenih *LiveData* globalnih varijabli, na osnovu kojih se podaci dalje ispisuju ili grafički prikazuju korisniku.

```
switch (PID) {

    case 4://PID(04): Engine Load

        // A*100/255
        val = (A * 100 / 255);
        int calcLoad = (int) val;
        liveEngineLoad.setValue(calcLoad);
        double FuelFlowLH = (mMaf * calcLoad * mEnginedisplacement / 1000.0 / 714.0) + 0.8;
        if(calcLoad == 0)
            FuelFlowLH = 0;
        avgconsumption.add(FuelFlowLH);
        Log.i(TAG,  msg: "calculateEcuValues: fuelCons" + calculateAverage(avgconsumption));
        liveFuel.setValue((int) calculateAverage(avgconsumption));

        break;
```

Slika 4.11 *Računanje opterećenja motora*

4.2.5 Korisničko sučelje

Za prikaz različitih zaslona, odnosno fragmenata, korišten je *viewPager*. Pomoću njega se na jednostavan način može prikazati više različitih fragmenata, te na jednostavan način rukovati s prijelazom između fragmenata. Na slici 4.12 se vidi da je za implementaciju *viewPagera* potrebno pronaći *viewPager* element unutar *layouta* pomoću ID-a, zatim mu postaviti adapter koji će biti odgovoran za prijelaz između fragmenata. Zatim je potrebno definirati koja će se radnja desiti u slučaju prijelaza na drugi fragment, u ovom slučaju se poziva metoda *updateCommandsListBasedOnPage*, koja na osnovu trenutne pozicije, odnosno fragmenta, ažurira listu naredbi tako da u nju smjesti naredbe koje daju informacije potrebne na tome fragmentu.

```
ViewPager viewPager = findViewById(R.id.viewPager);
MyPagerAdapter adapterViewPager = new MyPagerAdapter( activity: this, getSupportFragmentManager());
viewPager.setAdapter(adapterViewPager);
viewPager.addOnPageChangeListener(new ViewPager.SimpleOnPageChangeListener() {
    @Override
    public void onPageSelected(int position) {
        super.onPageSelected(position);
        updateCommandsListBasedOnPage(position);
    }
});
viewPager.setCurrentItem(PAGER_INITIAL_PAGE);
```

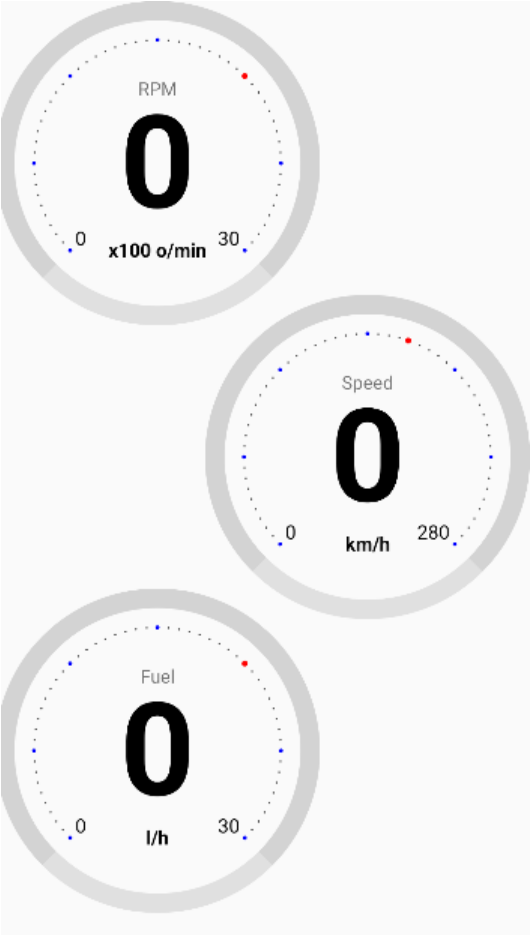
Slika 4.12 *ViewPager*

Također je dodan element *CircleIndicator* koji grafički prikazuje trenutnu poziciju *viewPagera*. Na slici 4.13 se vidi da je za njegovo postavljanje dovoljno samo dohvatiti referencu na element, te mu predati prethodno definirani *viewPager* čiju će trenutnu poziciju prikazivati.

```
CircleIndicator indicator = findViewById(R.id.indicator);
indicator.setViewPager(viewPager);
```

Slika 4.13 *CircleIndicator*

Fragmenti su kreirani tako da prva tri fragmenta sadrže satove za prikaz podataka, dok će zadnji fragment sadržavati graf za usporedbu dvije vrijednosti. Na slici 4.14 je prikazan izgled prvog fragmenta.



Slika 4.14 *Izgled prvog fragmenta*

Svaki fragment je odgovoran za prikaz određenih podataka. Za prikaz je potrebno dohvatiti reference satova, kako bi se na njima mogli prikazati podaci. Zatim je potrebno nadgledati globalne *LiveData* varijable, te na svaku njihovu promjenu, promijeniti vrijednost na satu, što je vidljivo na slici 4.15.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_first, container,  attachToRoot: false);
    KdGaugeView rpmGauge = view.findViewById(R.id.rpmGauge);
    activity.liveRpm.observe(activity, rpmGauge::setSpeed);
    KdGaugeView speedGauge = view.findViewById(R.id.speedGauge);
    activity.liveSpeed.observe(activity, speedGauge::setSpeed);
    KdGaugeView fuel = view.findViewById(R.id.fuelGauge);
    activity.liveFuel.observe(activity, fuel::setSpeed);
    return view;
}
```

Slika 4.15 *Nadgledanje LiveData varijabli*

Kod fragmenata s grafom je slična situacija, ali tu se umjesto čistog prikaza podataka, podaci prvo obrađuju kako bi se mogli prikazati na grafu. Na slici 4.16 se vidi da je potrebno inicijalizirati *spinnere* za odabir podataka koje želimo prikazati na grafu. Također treba promatrati odabrane podatke na osnovu kojih će se dodavati novi podaci u graf.

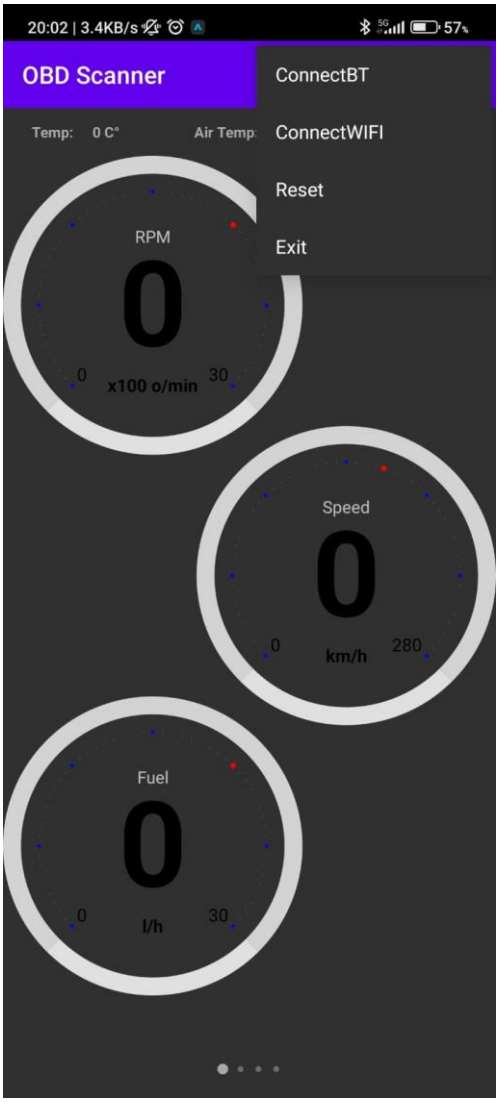
```
private void initializeSpinners(View view) {

    ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(activity,
            R.array.spinner_drop_items, R.layout.spinner_item);
    Spinner spinnerParam1 = view.findViewById(R.id.spinnerParam1);
    spinnerParam1.setAdapter(adapter);
    spinnerParam1.setOnItemSelectedListener(this);
}
```

Slika 4.16 *Inicijalizacija Spinnera*

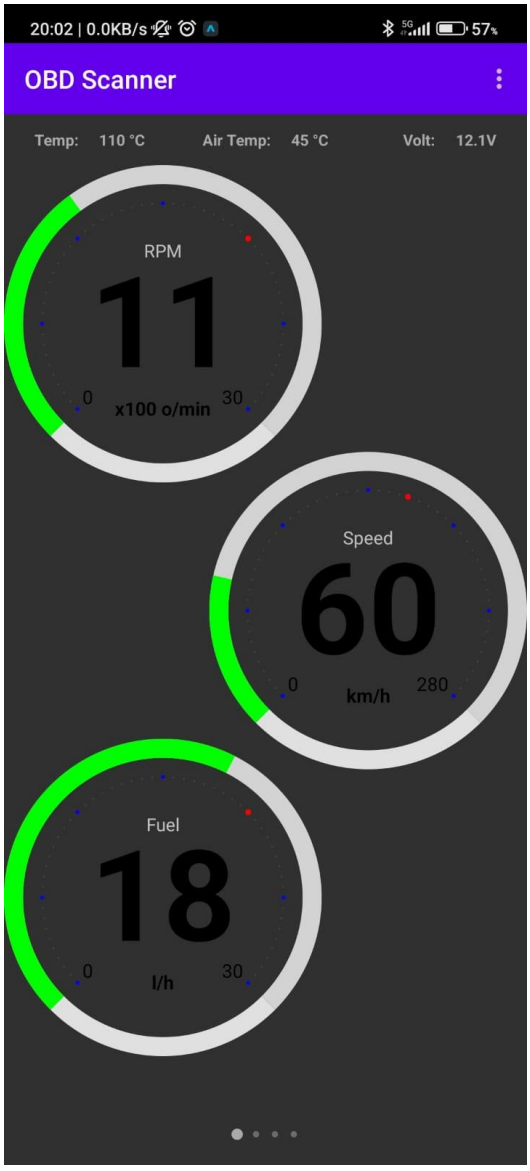
5. Završni izgled aplikacije

Kada otvorimo aplikaciju u gornjem desnom kutu nalazi se tipka koja služi za otvaranje dodatnog izbornika. U tom izborniku imamo opcije za spajanje na OBD2 adapter preko bluetooth i WI-FI veze preko tipki *ConnectBT* i *ConnectWIFI*. Imamo tipku za resetiranje aplikacije i za izlaz iz aplikacije.



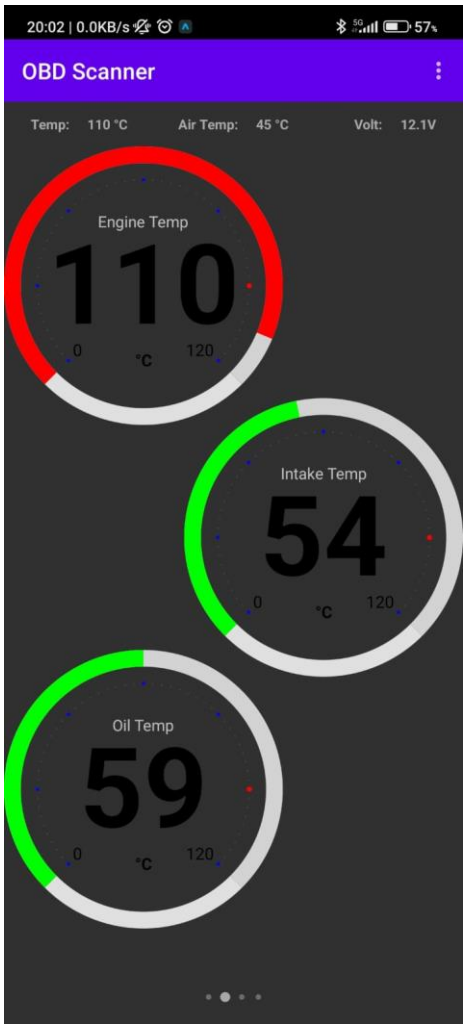
Slika 5.1 *Dodatni izbornik*

U gornjem dijelu zaslona se nalazi traka s nazivom aplikacije i tipkom izbornika. Ispod nje se nalazi traka s numeričkim prikazom trenutne temperature rashladne tekućine motora, temperaturom zraka u usisnoj grani i naponom. Na ostatku zaslona se nalaze grafički prikazi u obliku satova i prikazuju okretaje motora, trenutnu brzinu kretanja automobila i razinu goriva (Slika 5.2).



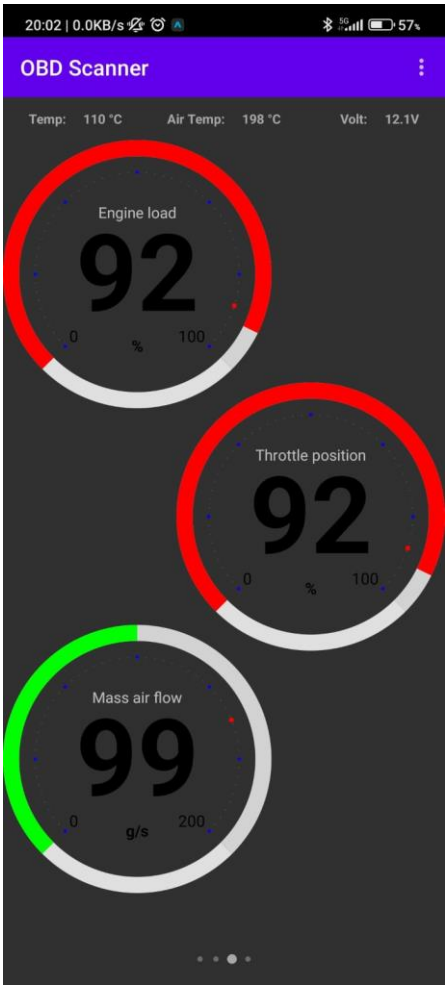
Slika 5.2 *Izgled prvog zaslona*

Na drugom zaslonu su također prikazane traka s nazivom i s temperaturama motora, usisa i naponom. Na njemu prvi sat grafički prikazuje temperaturu rashladne tekućine motora, drugi sat temperaturu zraka usisa motora, a treći temperaturu ulja u motoru. Satovi imaju zeleno i crveno područje, zeleno je kada je mjerena veličina unutar normalnih vrijednosti, a postaje crvene boje kada se približi maksimalnoj vrijednosti. Na slici 5.3 prikazan je primjer temperature rashladne tekućine motora koja postaje crvena na preko 100°C



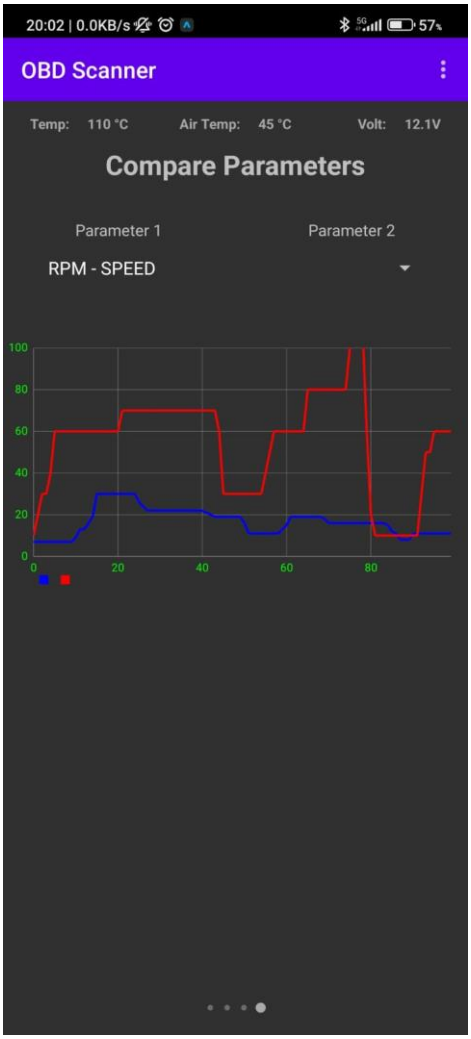
Slika 5.3 *Prikaz drugog zaslona*

Treći zaslon kao i prva dva ima iste trake, a razlikuje se u prikazu vrijednosti na satovima. Na njemu se prikazuju opterećenje motora i položaj papučice gasa izraženi u postocima, dok posljednji sat prikazuje protok zraka kroz usisnu granu motora (Slika 5.4).



Slika 5.4 *Izgled trećeg zaslona*

Četvrti zaslon je predviđen za prikaz odabranih vrijednosti na grafu. Ovdje imamo mogućnost usporedbe vrijednosti odabranih parametara u vremenu. Pritiskom na tipku Parameter 1 odabiremo prvi parametar, a tipkom Parameter 2 drugi parametar s kojim ga želimo usporediti (Slika 5.5).



Slika 5.5 *Izgled četvrtog zaslona*

6. ZAKLJUČAK

Cilj OBD2 razvoja sustava bio je kontrola ispušnih plinova automobila, ali s godinama i razvojem tehnologije njegove mogućnosti su jako porasle. Danas je preko njega moguć pristup svim značajkama automobila, senzorima i upravljačkim jedinicama. U ovom projektu se koristi njegova mogućnost spajanja s adapterom ELM 327 koji se spaja na android mobilni uređaj preko wi-fi ili bluetooth bežične veze.

Android aplikacija *OBD2 Scanner* služi za prikaz nekih bitnih parametara automobila poput broja okretaja motora, brzine kretanja i raznih temperatura kao i grafičku usporedbu odabranih parametara. Omogućava vozaču praćenje vrijednosti senzora koje nisu prikazane na standardnoj instrument ploči automobila poput temperature rashladne tekućine motora koja se sve rjeđe nalazi na novim automobilima, a jako je bitna stvar.

Pored ovih prednosti, nedostatak je nemogućnost izmjene parametara koje želimo prikazivati na satovima u aplikaciji. Iako je prikazana većina bitnih parametara, uvijek postoji korisnik koji želi prikazati nešto drugo, a ovdje nema mogućnosti za tim.

LITERATURA

[1] Ankit Banerjee, Torque, Android app lets you ta pinto the brains of your car, https://www.androidauthority.com/torque-app-android-bluetooth-obd2-95772/ [15.06.2022.].

[2] Dashcommand, https://www.apppicker.com/reviews/23555/dashcommand-app-review-2021 [15.06.2022.].

[3] Elm obd2, https://www.makeuseof.com/best-obd2-vehicle-diagnostic-apps/ [15.06.2022.].

[4] Infocar, https://www.makeuseof.com/best-obd2-vehicle-diagnostic-apps/ [15.06.2022.].

[5] https://en.wikipedia.org/wiki/On-board_diagnostics#OBD-I [08.01.2023].

[6] https://cablematic.com/en/products/kit-obd2-autocomtruck-8-adapters-for-wabco-knorr-benz-scania-man-renault-iveco-volvo-OB082/ [08.01.2023].

[7] https://www.noregon.com/what-is-obd/ [17.01.2023.].

[8] Martin Falch, OBD2 explained, https://www.csselectronics.com/pages/obd2-explained-simple-intro [17.01.2023].

[9] https://kereta.info/car-education-warning-indicator-lights-part-1/ [18.01.2023]

[10] Victor Barreto, History of on-bord diagnostics, https://www.geotab.com/blog/obd-ii/ [18.01.2023].

[11] https://euroklasse.com/products/mhd-wireless-obdii-wifi-flash-adapter [18.01.2023]

[12] Darek Fanton, What is CAN bus and why i sit so important, https://www.onlogic.com/company/io-hub/what-is-can-bus/ [16.06.2022].

[13] Adam Ali, What is CAN bus and why do you need to care, https://www.earth2.digital/blog/what-is-vehicle-can-bus-ecu-evoque-adam-ali.html [16.06.2022].

[14] https://en.wikipedia.org/wiki/ELM327

[15] Jeremy Laukkonen, ELM327 Microcontroller Car Diagnostics, https://www.lifewire.com/elm327-microcontroller-car-diagnostics-534688 [17.06.2022].

[16] https://robu.in/product/elm327-obd2-v2-1-bluetooth-interface-auto-car-diagnostic-scanner/ [20.06.2022].

[17] https://en.wikipedia.org/wiki/Android_Studio [20.06.2022].

[18]https://developer.android.com/studio/intro?fbclid=IwAR3GwNQwrAj4BJcTF3AOuwzkRgQ1q6_s0EdkbTlm7guH701BaBsjiFleG2M [20.06.2022].

[19] Joydip Kanjilal, How java is used din android app development, https://www.developer.com/mobile/java-mobile/java-mobile-programming-for-android/ [20.06.2022].

[20] https://www.logicbig.com/tutorials/misc/xml/xml-basics.html [21.06.2022].

[21] XML https://www.logicbig.com/tutorials/misc/xml/xml-basics.html [21.06.2022].

SAŽETAK

Tema ovog diplomskog rada je mobilna aplikacija za analizu podataka iz automobila. U ovome radu opisane je postupak izgradnje mobilne aplikacije. Za izradu programskog rješenja odabrana je platforma Android zbog njene rasprostranjenosti i jednostavnosti korištenja. Na početku je dan prikaz postojećih rješenja dostupnih na Google Play Store. Zatim su opisane korištene tehnologije pri izradi poput OBD2 sustava koji ima pristup svim podsustavima automobila i ELM 327 adaptera koji se ovdje koristi za prikupljanje vrijednosti senzora. Potom je prikazan postupak izgradnje programskog rješenja. Na kraju je prikazan završni izgled aplikacije i objašnjene su njene prednosti i mogućnosti, ali nedostaci.

Ključne riječi: aplikacija, android, ELM327, java, OBD2

ABSTRACT

Title: Mobile application for car data analysis

The topic of this thesis is a mobile application for analyzing car data. This paper describes the process of building a mobile application. The Android platform was chosen for the creation of the software solution due to its widespread use and ease of use. At the beginning, a presentation of the existing solutions available on the Google Play Store is given. Then the technologies used in the creation are described, such as the OBD2 system that has access to all car subsystems and the ELM 327 adapter that is used here to collect sensor values. Then the procedure of building the software solution is shown. At the end, the final appearance of the application is shown and its advantages and possibilities, but also its disadvantages are explained.

Keywords: application, android, ELM327, java, OBD2