Дипломска работа

Прототип на апликација за поврзување на OBD2 диагностика на моторни возила и Андроид мобилен уред

Ментор:

Проф. Димитар Трајанов

Изработил:

Ангел Ќосески

Индекс бр. 111235

angel.koseski@gmail.com

Скопје, Септември 2016

Содржина

1. Апстракт

Оваа дипломска работа има за цел да ги покаже можностите кои произлегуваат од поврзвувањето на мобилен уред Андроид со вградена диагностика на модерни моторни возила. Ќе се прикаже користење на протоколот за комуникација OBD2 (Onboard-Diagnostic 2). Ќе го разработи целокупниот процес на комуникација. Почнувајќи со воспоставување на bluetooth врска со ELM327 чипот и потоа испраќање/интерпретирање на разни команди кон возилото. Податоците добиени од возилото ќе бидат прикажани во реално време, но ќе постои исто така можност за преглед на патувања со податоци зачувани во база на мобилниот уред. Андроид проектот ќе биде структуриран и имплементиран по принципи за квалитетен софтверски проект, оддржлив и скалабилен. Ќе покаже користење на MVP (Model-View-Presenter) принципот и DI (Dependency injection) во андроид проект за сепарација на одговорностите помеѓу модулите и лесно тестирање.

2. Вовед

Мобилниот телефон е еден од најкористените уреди во нашето секојдневие. Постојано го носиме со нас и на него завршуваме многу од нашите задачи. Идејата за мобилен телефон произлегла од потребата на лугето за преносно комуникациско средство, со кое што ќе можеме да комуницираме со пријатели без локациски зависности.
Мобилните технологии и користениот хардвер се во постојана промена и развој. Така на пример, со појавата на паметните телефони (Smartphones) во последната деценија, користењето и интеракцијата драматично се промени.
Порано телефоните се користеа само за разговори и кратки СМС пораки, во денешно време на нашите мобилни уреди можеме да извршуваме безброј многу функционалности. Имаме комплетно функционален компјутер во рака. Постојано сме поврзани со интернет, комуницираме преку мрежи, пишуваме електронски пораки, вршиме плаќања и многу други работи. Ни нуди можност за многу брз пристап до сите информации.
Покрај преносот на податоци, имаме интеракција со нашата околина со помош на камера, сензори за околината и локациски сензори (GPS).
Паметните телефони станаа неопходен уред во секојдневниот живот на луѓето.

Развојот на технологијата е само еден дел од промените кои се случени во последните децении. Автомобилската индустрија доживеала промени од сличен ранг.
Со интегрирање на модерни технолошки платформи во моторни возила, автомобилските произветидели креираат возила кои се неспоредливи со тие од пред десет години. Производителите како VW, Audi i BMW имаат вградено високо перформантни компјутери во секое од своите најнови возила, кои се често и пософистицирани од модерните компјутери во нашите домови. Возилата се поврзани со интернет, комуницираат во мрежа, ја снимаат околината и прават пресметки за самоуправување и спречување на незгоди.
Едно модерно возило има над 100 сензори кои постојано испраќаат информации за нивната состојба.

Од голем интерес на автомобилски ентузиасти е читање на разни параметри од сензори за своето возило и моменталната состојба.
Во оваа дипломска работа ќе направам една прототип апликација за Андроид паметен телефон, која ќе се поврзи со диагностиката на модерно возило и ќе ни прикаже сензорски вредности во реално време. Исто така има можност за складирање на податоци во база и анализирање на патувања.

3. OBD

OBD или On-Board-Diagnostics (вградена диагностика) е автомобилски термин кои се однесува на диагностика и известување од страна на моторно возило. OBD системите му даваат пристап на сопственикот или техничкото лице до разните под-системи од автомобилот. Обемот на диагностички информации кои се достапни варира во голема мерка од неговото воведување во првите автомобилски компјутери во 1980тите години.
Раните OBD верзии, при детекција на некој дефект во возилото само ќе осветлеа индикатор ламбичка во инструмент таблата но не обезбедуваа дополнителни информации за природата на дефектот. Модерни имплементации на системот користат стандардизирана дигитална комуникациска порта за да прикажат податоци во реално време, поркај можноста за детекција и алармирање на дефекти за брза поправка.

Првична имплементација на компјутерски систем во моторно возило е направена од Volkswagen 1968год во нивниот автомобил Тип3 за нагодување на параметрите поврзани со системот за директно вбризгување на бензин.
Во следните години многу други производители вградувале компјутерски системи во нивните автомобили. Кодирање на системите се вршело преку повеќе различни стандарди кои секој производител си ги дефинирал за себе. Некој од тие стандардни интерфејси биле:

- ALDL
- M-OBD
- OBD-I
- OBD-1.5
- OBD-II
- EOBD
- JOBD
- ADR 70/91

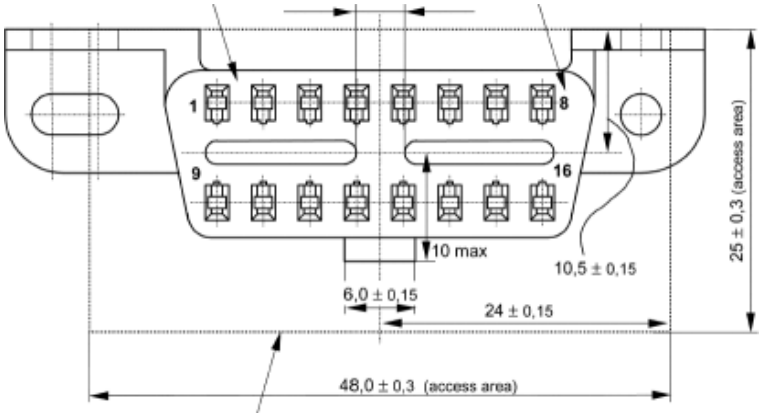
Поради воведување на мерки за заштита на околината и потреба од мерење на емисии и загадување на секое моторно возило, од 1996год е воведен закон кој што ги обврзува сите произведувачи на автомобили да обезбедат еднаков интерфејс за поврзување на диагностичка опрема за тестирање. Преносот на податоци преку овие интерфејси треба да го следи OBD-II стандардот.

3.1 Што е OBD-II?

Како и со сите други вградени системи, во автомобилскиот свет имаме специфичен јазик за комуникација со возилото. OBD-II стандардот е подобрување на сите постари стандарди во аспект на можности и стандардизација.
Тој ги специфицира достапните електрични сигнализирачки протоколи и форматот на пораките, исто така за прв пат го специфицира и типот на диагностички конектор и неговите пинови. Ни нуди листа на параметри од возилото кои што можеме да ги пратиме, и начинот на енкодирање на пораките од истите.

Во други зборови, OBD е јазикот за комуникација со контролната единица на моторт (или ECU - Engine Control Unit).

3.1.1 Диагностичен конектор



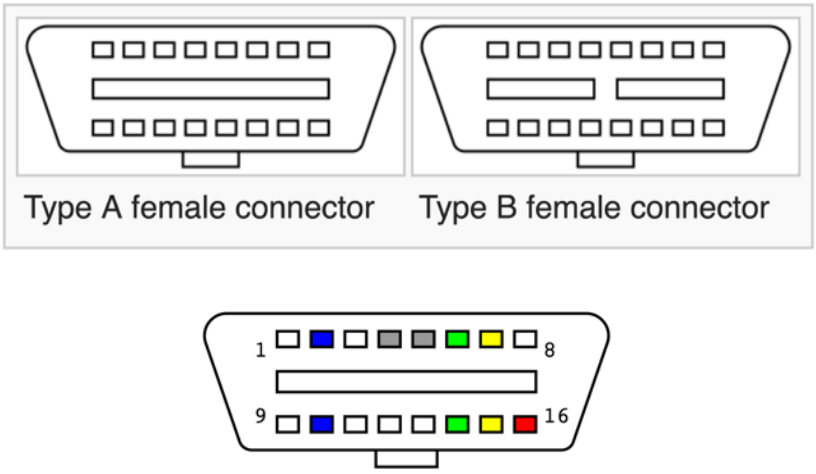
слика 1. Стандардизиран конектор за OBD-II

Пристапот до податоците и комуникацијата се врши преку **DLC** (Data Link Connector).
На слика 1. гледаме шема на диагностичкиот конектор за OBD-II. Прикажаните димензии морат да бидат запазени од сите произведувачи. Стандардизираниот конектор исто така вклучува пин, кој што нуди електрична струја од батеријата на автомобилот за да може да се снабди алатката за скенирање и да нема потреба од посебно напојување.
Обично конекторот е сместен на лесно достапно место во внатрешноста на возилото за да може да се поврзе алатка за скенирање без напори и отварање на пластики и штрафови. Постои член во стандардот кој што налага дека конекторот мора да биде на далечина не поголема од 0.60м од воланот на

возилото. Најчести локации се на пластиките између управувачките педали и воланот или во средната конзола на возилото (под подпирачот за рака или пред менувачот).

Како резултат на стандарзијацијата е можно со една алатка да се читаат податоци на сите разни возила. Првичната намена била за проверка на емисии и загадување, но иако само тоа било задолжително според закон, скоро сите производувачи го искористиле конектор и како главен конектор во возилото преку кој сите системи се диагностицираат и репрограмираат.

SAE J1962 стандардот дефинира два типа на хардверски интерфејси за OBD-II конекторот, наречени типА и типБ. И двата се конектори од женски тип, со 16 (2x8) пинови во форма на D. Како што се гледа на слика 2, разликата помеѓу двата типа е во пластичниот хоризонтален сепаратор, и во тоа што типА се користи во возила во електричен напон од 12V волти, а типБ во возила со 24V волти.



слика 2. Разлика на типА и типБ OBD-II конектори и нумерирање на пинови

Излезот на пиновите и нивните функции исто така се дефинирани во SAE J1962 со следната табела.

1	Manufacturer discretion: • GM: J2411 GMLAN/SWC/Single-Wire CAN • VW/Audi/BMW: Switched +12V to tell a scan tool whether the ignition is on. • Ford, FIAT: Infotainment CAN High	**9**	Manufacturer discretion: • BMW: TD (Tachometer Display) signal aka engine RPM signal. • GM: 8192 bit/s ALDL where fitted. • Ford: Infotainment CAN-Low
2	Bus Positive Line of SAE J1850 PWM and VPW	**10**	Bus Negative Line of SAE J1850 PWM only (not SAE J1850 VPW)
3	Manufacturer discretion: • Ford: DCL(+) Argentina, Brazil (pre OBD-II) 1997-2000, USA, Europe, etc. • Ford: Medium Speed CAN-High • Chrysler: CCD Bus(+)	**11**	Manufacturer discretion: • Ford: DCL(-) Argentina, Brazil (pre OBD-II) 1997-2000, USA, Europe, etc. • Ford: Medium Speed CAN-Low • Chrysler: CCD Bus(-)
4	Chassis ground	**12**	Manufacturer discretion: • GM: Diagnostic codes to DIC (1994-2004 Corvette)
5	Signal ground	**13**	Manufacturer discretion: • Ford: FEPS – Programming PCM voltage
6	CAN-High (ISO 15765-4 and SAE J2284)	**14**	CAN-Low (ISO 15765-4 and SAE J2284)
7	K-Line of ISO 9141-2 and ISO 14230-4	**15**	L-Line of ISO 9141-2 and ISO 14230-4
8	Manufacturer discretion: • BMW: Second K-Line for non OBD-II (Body/Chassis/ Infotainment) systems. • FIAT: Infotainment CAN-Low.	**16**	Battery voltage: • Type "A" 12V/4A • Type "B" 24V/2A

3.1.2 OBD-II Протоколи

Постојат пет сигнални протоколи кои се дозволени преку OBD-II интерфејсот. Скоро сите возила имплементираат само еден од нив, и врз база на тоа кој пинови се присутни на J1962 конекторот можеме лесно да препознаеме кој протокол го имплементира одредено возило.
Подоле е прикажан краток преглед на сите протоколи и опис на користените пинови од DLC за секој од нив.

SAE J1850 PWM

Сигланот е Pulse-Width-Modulation кој се праќа на брзина од 41.6kbps. Овој протокол е обично користен од возила на марката Ford.

Feature	Description
BUS +	Pin 2
BUS -	Pin 10
12V	Pin 16
GND	Pins 4, 5
Bus State:	Active when BUS + is pulled HIGH, BUS - is pulled LOW
Maximum Signal Voltage:	5V
Minimum Signal Voltage:	0V
Number of bytes:	12
Bit Timing:	1' bit - 8uS, '0' bit - 16uS, Start of Frame - 48uS

SAE J1850 VPW

Сигланот е Variable-Pulse-Width кој се праќа на брзина од 10.4kbps. Овој протокол е обично користен од возила на марката GM.

Feature	Description
BUS +	Pin 2
12V	Pin 16
GND	Pins 4, 5
Bus State:	Bus idles low
Maximum Signal Voltage:	+7V
Decision Signal Voltage:	+3.5V
Minimum Signal Voltage:	0V
Number of bytes:	12
Bit Timing:	1' bit -HIGH 64uS, '0' bit -HIGH 128uS, Start of Frame - HIGH 200uS

ISO 9141-2

Користен од европски и азиски возила и Crysler. Брзината е 10.4kbps и има асинхрона серијална комуникација.

Feature	Description
K Line (bidirectional)	Pin 7
L Line (unidirectional, optional)	Pin 15
12V	Pin 16
GND	Pins 4, 5
Bus State:	K Line idles HIGH. Bus is active when driven LOW.

Maximum Signal Voltage:	+12V
Minimum Signal Voltage:	0V
Number of bytes:	Message: 260, Data: 255
Bit Timing:	UART: 10400bps, 8-N-1

ISO 14230 KWP2000

Овај е Keywoard-Protocol-2000. Друга асинхрона серијална комуникација која тече на брзина од 10.4kbps. Исто така користена од Crysler и европски и азиски возила.

Feature	**Description**
K Line (bidirectional)	Pin 7
L Line (unidirectional, optional)	Pin 15
12V	Pin 16
GND	Pins 4, 5
Bus State:	Active when driven LOW.
Maximum Signal Voltage:	+12V
Minimum Signal Voltage:	0V
Number of bytes:	Data: 255
Bit Timing:	UART: 10400bps, 8-N-1

ISO 15765 CAN

Овој протокол е користен од сите американски возила од 2008 год., и во многу европски возила од 2003 год. Претставува комуникациски метод со користење на две жици, и може да врши комуникацјиа со брзина до 1Mbps.

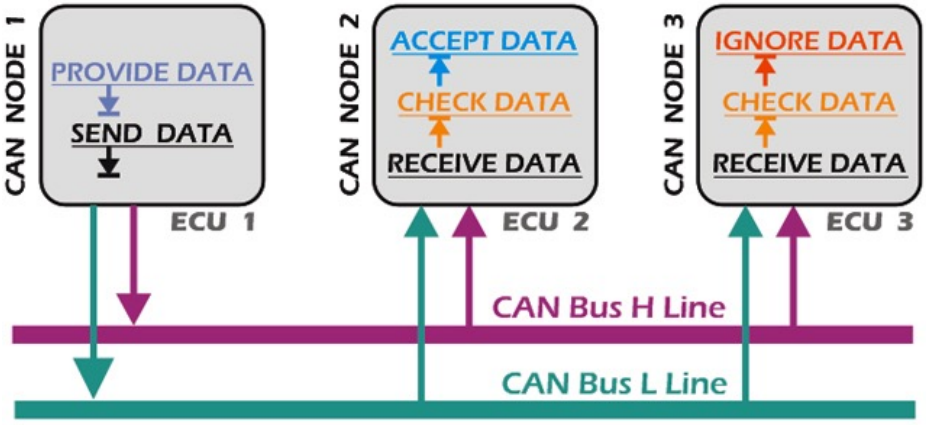
Feature	Description
CAN HIGH (CAN H)	Pin 6
CAN LOW (CAN L)	Pin 14
12V	Pin 16
GND	Pins 4, 5
Bus State:	Active when CANH pulled HIGH, CANL pulled LOW. Idle when signals are floating.
CANH Signal Voltage:	+3.5V
CANL Signal Voltage:	+1.5V
Maximum Signal Voltage:	CANH = +4.5V, CANL = +2.25V
Minimum Signal Voltage:	CANH = +2.75V, CANL = +0.5V
Number of bytes:	L
Bit Timing:	250kbit/sec or 500kbit/sec

3.1.3 CAN-BUS

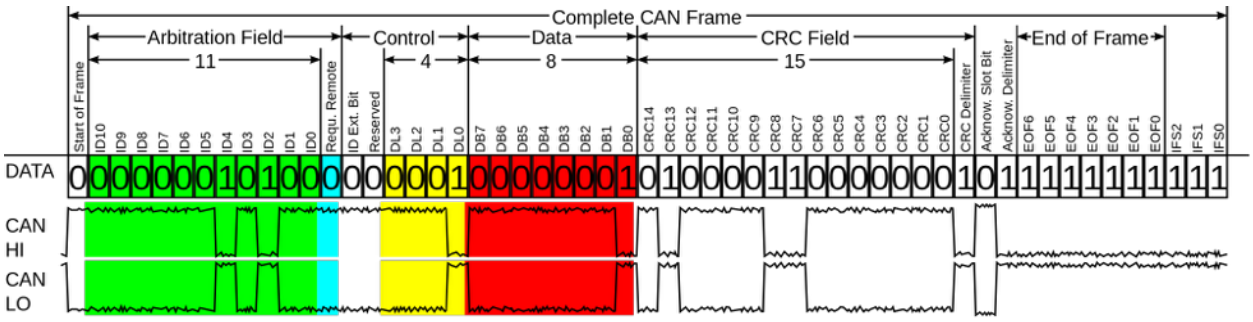
Контролната единица на автомобилот (ECU) може да претставува единечен модул, или колекција на повеќе модули. Тоа е мозокот на возилото кој прати и контролира многу главни функции. Може да биде стандарден од произведувачот, ре-програмабилен, или да биде поврзан со други ECUs за извршување на повеќе функции.
Модерни возила имаат огромен број на контролни единици, некој со поважни функции од други. Има стално една главна контролна единица за контрола на моторот и повеќе други со помал број на функционалности (Медиа-контролер, контролер за стакла, сензори итн.). Едно модерно возило има од прилика 100 микро-контролери за сите под-системи.
Со репрограмирање на параметри во главното ECU, можеме да ги сменеме возните карактеристиките на возилото со подобрување на перформансии или економија.

Во сите модерни возила, секое ECU е микро-контролер и скоро сите комуницираат со **CAN (Controller-Area-Network)** протоколот.
CAN е автомобилски BUS стандард, дизајниран за да им овозможи на микро-контролери да комуницираат меѓусебно без потреба од хост компјутер. Тој е протокол базиран на пораки врз таканаречен ‘автопат’, првично дизајниран за користење во автомобилската индустрија, но денес се користи и во многу други контексти.
‘Автопатот’ или BUS е обична жица од еден до друг крај на автомобилот, на која се врзани сите микро-контролери наречени јазли.
Секое ECU во автомобилот претставува јазол, кој е поврзан на ‘автопатот’ и има свој единствен идентификациски број.
Врз автопатот се испраќаат огромен број на пораки во секој момент, сите јазли ги слушаат сите пораки, но се заинтересирани само за тие кои што го имаат нивниот идентификациски број. Исто така, секој јазол може да поставува пораки врз автопатот.

CAN пораките имаат стандардизиран формат, со идентификациски броеви, типови на пораки, податоци и чексуми.



слика 3. CAN-BUS



слика 4. CAN порака

3.1.4 Параметри и команди

OBD-II PIDs (On-board diagnostic Parameter IDs) претставуваат кодови за барање на податоци од возилото и одредени контролни единици.
Дефинирани се многу параметри во ОБД стандардот, но произведувачи исто така дефинираат диполнителни параметри специфични за своите возила.

Не е задолжително сите произведувачи да ги поддржуваат сите PIDs, но мора да се поддржани параметрите задолжени за инспекција на емисии и загадување.

Вообичаено, автомобилски техничар користи алатка за скенирање на PIDs поврзана за OBD-II конекторот.

1. Се внесува параметарскиот број во алатката.
2. Алатката за скенирање ја испраќа командата во мрежата на возилото
3. Контролната единица на CAN-BUS која што е адресирана ја прифаќа пораката и враќа резултат адресиран кон испраќачот.
4. Алатката за скенирање го прима резултатот и го прикажува на техничарот

DTC (Diagnostic-Trouble codes) се кодови за диагностика на дефекти во возилото. Може да бидат генерички, или специфично дефинирано од одреден производител. Форматот е следен:

X X X X X

- Првата бројка го дефинира типот на дефектот
  - Pxxxx за моторот
  - Bxxxx за каросерија
  - Cxxxx за шасија
  - Uxxxx за клас 2 мрежа
- Втората бројка укажува дали кодот е генерички или не
  - x0xxx за генерички кодови
  - x1xxx за специфични кодови дефинирани од произведувачот
- Третата бројка ни кажува кој систем е референциран со дефектот
  - xx1xx/xx2xx дефект во Air/Fuel Ratio
  - xx3xx дефект во системот за палење
  - xx4xx дефект во системот за емисии
  - xx5xx контрола на брзина, лер
  - xx6xx дефект компјутерски системи
  - xx7xx/xx8xx дефект во мењач
  - xx9xx влезно/излезни сигнали иконтроли
- Четвртата и петтата бројка го кажуваат специфицниот дефектен код
  - xxx00 to xxx99 - базирани врз системите референцирани во третата бројка

Постојат десет начини на работа опишани во OBD-II стандардот.

Mode (hex)	Опис
01	Приказ на моментални податоци во реално време
02	Приказ на замрзнати податоци
03	Приказ на зачувани DTC - дефектни кодови
04	Чистење на зачувани дефектни кодови DTC
05	Тест резултати од мерачи на кислород. O2 сензори
6	Тест резултати на други системи и компоненти
7	Приказ на моментални кодови за дефект. Зачувани во тек на моменталното или последното возење.
8	Контролни операции на вградени компоненти и системи.
09	Барање на информации за возилото. Марка, тип итн.
0A	Избришани кодови за дефект

Како пример ќе покажене неколку стандардни параметри кои се дефинирани во J1979 стандардот. Очекуваниот одговор на секој од нив е познат, исто и информацијата за како да се претвори тој огговор по препознатлив формат за човекот. Битно е да се повтори дека не се поддржани сите параметри од сите возила, и дека секој производител на автомобили си дефинира свои параметри кои не се во стандардот.

Од примерите можеме да забележеме дека начините на работа 1 и 2 се речиси исти. Разликата е во тоа што начин 1 ни дава реални податочи од моментална временска точка, а начин 2 замрзнати податоци од временска точка кога бил зачуван последниот диагностички дефект код.

Податоците се праќаат во блокови од 4 бајти со следна нотација:

A								B								C								D							
A7	A6	A5	A4	A3	A2	A1	A0	B7	B6	B5	B4	B3	B2	B1	B0	C7	C6	C5	C4	C3	C2	C1	C0	D7	D6	D5	D4	D3	D2	D1	D0

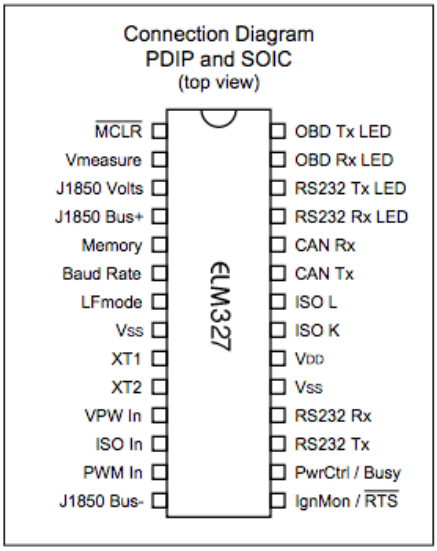
0B	1	Intake manifold absolute pressure	0	255	kPa	A
0C	2	Engine RPM	0	16,383.75	rpm	$\frac{256A+B}{4}$
0D	1	Vehicle speed	0	255	km/h	A
0E	1	Timing advance	-64	63.5	° before TDC	$\frac{A}{2}-64$
0F	1	Intake air temperature	-40	215	°C	$A-40$

слика 5. Пример на неколку OBD параметри

3.2 Што претставува ELM327 чипот

Според стандардот имаме интерфејс за трансфер на податоци од автомобилско возило до уред за читање. Добиените податоци во необработена форма не се директно читливи од компјутер или паметен уред. ELM327 чипот е дизајниран да служи како мост помеѓу OBD-II портата и стандарден RS232 серијален интерфејс. Во други зборови, тој е OBD на RS232 интерпретер.
Покрај тоа што знае автомацки да препознава и интерпретира девет различни обд протоколи, чипот исто така ни нуди поддршка за брза комуникација, мод за спијање со ниска енергетска потрошња, и J1939 стандард за камиони и автобуси. Исто така е комплетно отворен за модификација, ако сакаме да ни служи подобро за нашите потреби.

Главни функционалности се:
- Енергетска контрола со Standby мод
- Универзален сериал интерфејс (RS232)
- Автомацско барање и препознавање на протокол
- Комплетно конфигурабилен со AT команди
- CMOS дизајн за ниска енергетска потрошња



слика 6. ELM327 чип

3.2.1 Користење на ELM327

Чипот очекува да комуницира со компјутер преку RS232 серијална конекција. Иако модерните компјутери обично не се обезбедени со сериски приклучок каков што е потребен, постојат неколку начини на кои што може да се создаде

"виртуелен сериски порт”. Најчестите уреди се USB кон RS232 адаптери, но постојат неколку други, како на пример компјутер картички, Ethernet уреди, или Bluetooth сериски адаптери.
Не е важно како физички ќе се поврзиме со ELM327, ќе треба начин да се праќаат и примаат податоци. Наједноставниот начин е да се користи една од многуте "терминал" програми кои се достапни (HyperTerminal, ZTerm, итн), за да се овозможи внесување на знаци директно од нашата тастатура.
Правилно поврзани и вклучени, на ELM327 ќе светнат четирите LED ламбички во секвенца (како тест), а потоа ќе ја испрати пораката:

ELM327 v2.1
>

Покрај добивањето на идентификациски број за верзијата на овој IC, примањето на оваа низа е добар начин да се потврди дека компјутерската конекција и поставениот терминалски софтвер се точни (сепак, во овој момент сеуште не е направена комуникација до возилото, па состојбата на таа врска сеуште е непозната).
Карактерот ‘>’ кој е прикажан на втората линија покажува дека уредот е во состојба на мирување, подготвен да прима знаци на RS232 портата.
Карактери испратени од компјутерот можат да бидат наменети за интерна употреба и конфигурација на ELM327 чипот, или за преформатирање и пренесување на возилото. Чипот брзо може да препознае за што и кого се наменети добиените карактери со следење на содржината на применатa порака.
Команди кои се наменети за внатрешна употреба на ELM327, ќе започнат со ликовите “AT”, а ОБД команди за возилото смеат да содржат ASCII кодови со хексадецимални цифри (0 до 9 и од A до F).
Мора да се има предвид дека ELM327 е преведувач/интерпретер кој не прави обиди за проценка на испратените ОБД пораки за валидност - тој само гарантира дека хексадецимални цифри се применi, комбинирани во бајти, а потоа испратени на OBD портата, исто така не знае дали испратената порака до возилото била грешка.

3.2.2 Селекција на протокол и начин на работа

Еден од најопштите “AT” команди кој што се користени е следниот:

ELM327 v2.1
> AT Z

Гледаме според почетните знаци на “AT Z” командата дека оваа е наменета за внатрешна конфигурација на елм327 чипот. Со оваа команда го ресетираме интегрираното коло во почетна состојба. Често се користи ако сме правеле некој

поспецифични конфигурации на чипот, па сакаме да се вратиме на фабричко поставување.
Ако бајтите кои ќе ги испратиме до ELM327 не почнуваат на буквата "A" и "T", се претпоставува дека тие се OBD команди за возилото. Секој пар на ASCII бајти ќе биде тестиран за да се осигураме дека тие се валидни хексадецимални цифри, а потоа ќе бидат обединети во бајти на податоци за пренос кон возилото.
OBD командите всушност се испратуваат кон возилото скапувани во пакети на податоци. Повеќето стандарди бараат три бајти како глава (header bytes) и бајт за проверка на грешка (error checksum) биде вклучен во секоја OBD порака, добро е што ELM327 чипот ги додава тие екстра бајти автомацки за нас со секоја команда. Почетната конфигурација на чипот за овие дополнителни бајти обично е соодветна за повеќето барања, но ако сакате да ги промени, постои механизам за тоа преку конфигурациски команди на чипот. Повеќето OBD команди се само еден или два бајти долги, но може да бидат и подолги.

Пред да почне комуникација со возилото мора да биде точниот протокол поставен во чипот. Протоколот се поставува во зависност од тоа на кој протокол работи диагностичкиот компјутер во возилото. Командата за тоа е:

> AT SP H

Оваа команда се користи за да го поставите ELM327 чипот на работа со коректен протокол кој е даден од страна на "H", а исто така да го зачувате како нов стандарден протокол во интерна меморија на чипот.
ELM327 поддржува 12 различни протоколи (две може да бидат дефинирани од корисникот). Тие се:

- 0 - Automatic
- 1 - SAE J1850 PWM (41.6 kbaud)
- 2 - SAE J1850 VPW (10.4 kbaud)
- 3 - ISO 9141-2 (5 baud init, 10.4 kbaud)
- 4 - ISO 14230-4 KWP (5 baud init, 10.4 kbaud)
- 5 - ISO 14230-4 KWP (fast init, 10.4 kbaud)
- 6 - ISO 15765-4 CAN (11 bit ID, 500 kbaud)
- 7 - ISO 15765-4 CAN (29 bit ID, 500 kbaud)
- 8 - ISO 15765-4 CAN (11 bit ID, 250 kbaud)
- 9 - ISO 15765-4 CAN (29 bit ID, 250 kbaud)
- A - SAE J1939 CAN (29 bit ID, 250* kbaud)
- B - USER1 CAN (11* bit ID, 125* kbaud)
- C - USER2 CAN (11* bit ID, 50* kbaud)

Првиот протокол кој што е прикажан (0), е лесен начин да му се каже на ELM327 чипот дека протоколот на возилото не е познат, и дека треба да се изврши пребарување низ сите протоколи додека е најден точниот. Тоа предизвикува ELM327 да ги обиде сите протоколи, ако е потребно, во потрага по оној протокол кој што може да се иницијализира правилно. Кога е пронајден валидниот

протокол, и функцијата на меморијата е овозможено, тогаш тој протокол ќе се зачува во интерната меморија и ќе се запише во стандардното поставување. Кога ELM327 чипот ќе се конфигурира на автомацко пребарување низ протоколи, ако на следен обид не успее да се поврзе со зачуваниот протокол, повторно ке почне да бара нов валиден. ELM327 корисниците често ја користат командата SP 0 за да се ресетира протоколот за пребарување, пред да се започне (или рестартирање) со конекција.
Ако друг протокол (освен 0) е избран со оваа команда (на пр. SP 3), тој протокол ќе стане стандарден и зачуван, и ќе биде единствениот протокол кој ќе се користи од страна на чипот.
При неуспешно поврзување со избран протокол, тоа ќе резултира со следниот одговор: "BUS init: ... ERROR". Потоа нема други протоколи да бидат испробани од чипот. Ова е корисна опција (команда) ако знаеме дека возилото користи само еден единствен протокол, но може исто така да ни предизвика многу проблеми ако не ја разбираме.

3.2.3 Извршување и интерпретација на команди

Стандардот ни налага дека секоја OBD команда или порака испратена до возилото мора да следи одреден формат. Првиот бајт од пораката, познат како 'mode' или начин на работа, го опишува типот на податоците кој се побарани, а вториот бајт (а по потреба и трет и повеќе), го означува податокот пој е побаран. Бајтите кој следат после 'mode' бајтот, се познати како идентификациски параметри или 'parameter identification - PIDs'. Сите идентификациски параметри се детално опишани во документи како SAE J1979 или ISO 15031-5.
Како што кажавме погоре, првиот бајт го претставува начинот на работа или типот на податок. Сите можни типови се опишани во табелата во подсекција '3.1.4 Параметри и команди'.
Во секој 'mode', параметарот 'PID 00' е резервиран за да ни одговори кои идентификациски параметри се поддржани од одреденото возило во одредениот 'mode'.

За да почнеме со комуникација со возилото мора да ги следиме следните чекоти. Траба првично да се осигураме дека ELM327 интерфејсот е правилно поврзан кон возилото и има електрична енергија. Многу возила нема да дадат одговор ако не е вклучен клучот на 'ON' позицијата. Добра практика е на почеток стално да го ресетираме чипот со командата:

> AT Z

Би требало да светнат четрите LED ламбички на чипот и да ни одговори со:

> ELM327 v2.1

Следен чекор е да го поставиме точниот протокол, тоа најлесно ќе го направиме со автомацко пребарување преку командата:

> AT SP 0

Тоа е се што треба да направиме за конфигурација и за да сме спремни да примаме податоци од возилото.
Како пример ќе ја покажеме командата за испрашување на возилото за моменталните температура на средството за ладење на моторот. Тоа се прави со комадата:

> 01 05

Прво кажуваме дека сакаме да добиеме податок од тип 01 (моментални податоци на возилото во реално врмее), а потоа со идентификацискиот параметар 05 означуваме дека сакаме да добиеме податок за моменталните температура на средството за ладење на моторот. Возилото ќе ни одговори со:

41 05 7B

41 означува дека ова е одговор на барање од тип 01 (01 + 40 = 41), додека вториот број (05) го повторува идентификацискиот параметар кој е побаран. Следниот бајт (7B) всушност ја содржи информацијата која ја бараме.
Ако се претвори 7B од хексадецимален во декаден број, добиваме 123. Тоа ја претставува моменталната температура во степени Celsius, но со преместена нула за приказ на негативни темпетарури. За добивање на точната вредност, треба да одземеме само 40, со што се добива **83'C**.

Покажавме само еден пример за барање и обработка на податок од возилото, пристапот за останатите идентификациски параметри е многу сличен.

3.2.4 Bluetooth-Interface на ELM327

Многу олеснително е користење на bluetooth за конекција на компјутер (или телефон) до ELM327 чипот. Постојат елм327 чипови со вграден bluetooth примач/испраќач наречени "OBD Bluetooth-donggle". Еден пример на таков е прикажзан на следната слика:
Со користење на овај тип на приклучот сме многу пофлексибилни и не мора да имаме кабли во возилото. Комуникацијата исто се извршува со серијална комуникација но само преку bluetooth-interface.

слика 7. ELM327 OBD Bluetooth-donggle

4.0 Андроид оперативен систем

Двата главни оперативни системи кои се користат во денешните паметни телефони се Android (развиен од Google) и iOS (развиен од Apple).
Андроид е мобилен оперативен систем развиен од Google, базиран на linux јадрото и првично дизајниран за уреди со екрани на допир, како мобилни паметни телефони и таблети. Корисничкиот интерфејс на андроид се базира на директна манипулација со помош на гестикулатии со допир, како на пример кликање, влечење и зумирање со прсти. За внес на текст и пишување се користи виртуелна тастатура која се појавува на екранот по потреба.

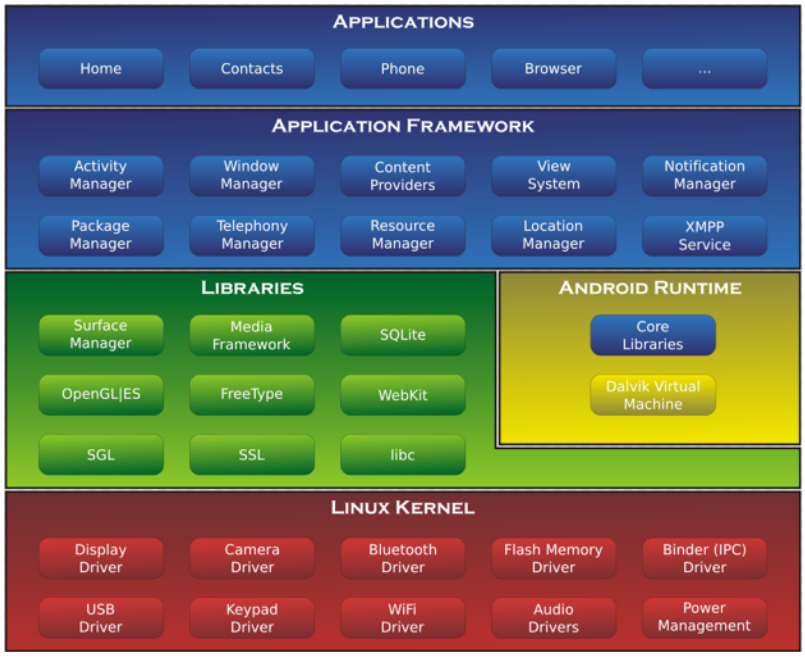
Андроид платформата е продукт од Open Handset Alliance, група од организации кои соработуваат помеѓу себе за да изградат подобар софтвер, хардвер и телекомункации. Оваа група која е предводена од Google, вклучува мобилни оператори, производувачи на уреди, провајдери на софтверски решенија и маркетинг компании. Од страна на Google, андроди платформата и софтверскиот код се достапни за јавноста под Open-Source лиценцата.
Првиот андроид уред на маркетот беше G1 уредот произведен од HTC и пропишан од T-Mobile. После една година на шпекулации уредот стана достапен. Во тоа време тимот на андроид исто така работеше на развој на SDK (Software Development Kit) за креирање на апликации. Како што се ближеше датумот на објавување на G1 уредот, андроид тимот ја објави првата верзија од SDK, со што започна развојот на првите апликации за новата платформа.
За да ја поттикне иновацијата, Google спонзорираше две рунди од “Android Developer Challenges” односно предизвици за андроид програмерите, каде што најдобрите апликации беа наградени со по милиони долари. После неколку месеци од објавувањето на G1 уредот, објавен беше и така нареченият андроид маркет Google Play Store, овозможувајќи им на корисниците да пребаруваат и преземаат апликации директно на нивните телефони. Во последни неколку години, андроид оперативниот систем е интегриран во паметни часовници (Android Wear), телевизори (Android TV) и медиа урери во автомобили (Android Auto). Развојот на апликации за сите видови на уреди се одвива преку истите алатки користејќи го Android SDK.

Алатките за развој исто така се достапни под open-source лиценцата и може да се превземат бесплатно.

Андроид оперативниот систем има најголем број на инсталации од сите оперативни системи од било кој вид. Тој е најпродавам оперативен систем за таблети од 2013год, и е доминантен на секое поле во сферата на мобилни паметни телефони.

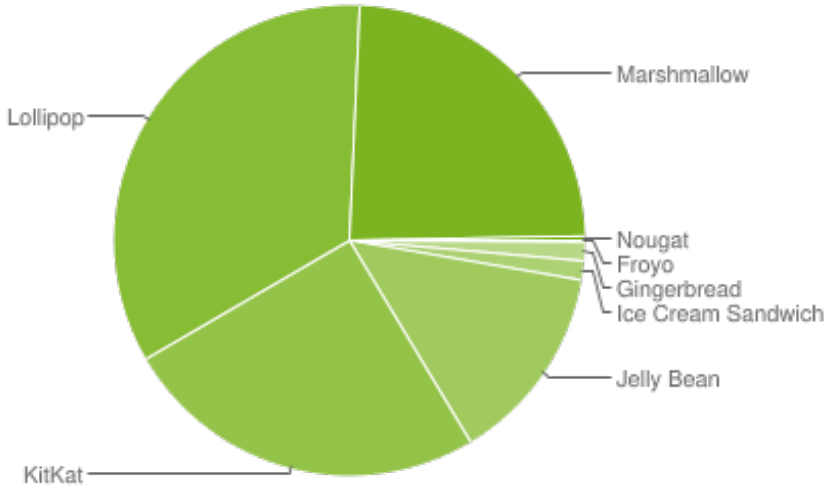
4.1 Архитектура на платформата

Андроид ни нуди богат спектрум на функционалности и начин на пристап до хардверот во мобилниот уред на лесен начин. Со користење на моќностите од андроид платформата, интеракција со околината преку сервисите за локација, акцелометар, сензори за топлина, близина и многу други, но и опции за конекција преку WiFi, Bluetooth како и безжична интернет конекција преку мобилната мрежа (GPRS, EDGE, 3G, 4G), не е замисливо колкав број на разни апликации можат да бидат креирани.
Јадрото на андроид платформата е linux kernel врз кој потоа има други посредници, библиотеки и APIs напишани во програмсиот јазик C, и апликациски софтвер кој работи врз апликацискатa платформа и се состои од Java библиотеки. Развојот на linux јадрото тече независно од платформата. Апликациите за андроид главно се развиваат во Java програмскиот јазик и потоа извршувани врз јава виртуелната машина.



слика 8. Архитектура на андроид платформата

За потреби од многу високи перформансии, постои можноста некој делови да се развиваат во програмскиот јазик C со упортеба на NDK (Native Development Kit). Важно е да се напомени дека секоја инстанца на андроид апликација се извршува на посебна виртуелна машина, која не е јава виртуелна машина (JVM) туку Dalvik, исто така open-source.
Секоја инстанца на Dalvik виртуелната машина е потоа хостирана во linux процес, како што е прикажано на сликата подолу.
Андроид оперативниот систем има голема фрагментација од аспект на верзии и големини на екрани (таблети, телефони, часовници итн.). При креирање на апликации многу е битно да се дефинира кој верзии од платформата ќе бидат поддржани бидејќи не сите функционалности се компатабилни наназад (backwards-compatible). Историјата на верзиите на андроид оперативниот систем започна со Android Alpha која беше објавена во ноември 2007 година. Првата комерцијална верзија достапна за јавноста е Android 1.0, таа беше објавена во септември 2008 година. Оперативниот систем континуирано се развива и подобрува од страна на Google и Open-Handset-Alliance.
Најновата достапна верзија е Android 7.0 или позната како “Android Nougat”, таа беше објавена во септември 2016год. Почнувајќи од април 2009 година, верзиите на андроид се именувани според храна (слатки), и по азбучен ред, почнувајќи од Android 1.5 или “Cupcake”. Следи листа на сите Андроид верзии:



слика 9. Процентуална застапеност на различните верзии

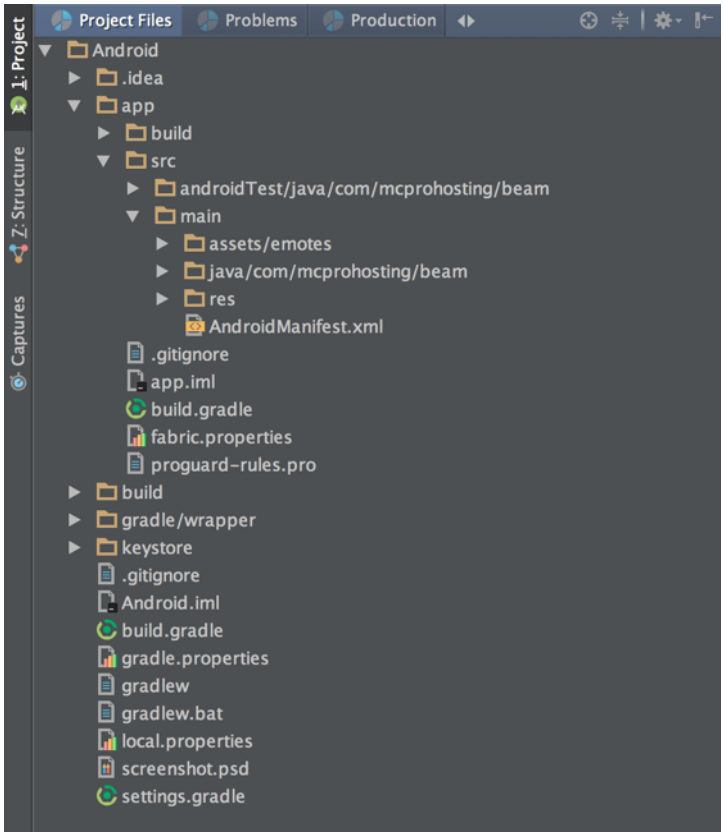
- **Cupcake (1.5)**
- **Donut (1.6)**
- **Eclair (2.0 – 2.1)**
- **Froyo (2.2 – 2.2.3)**
- **Gingerbread (2.3 – 2.3.7)**
- **Honeycomb (3.0 – 3.2.6)**

- **Ice Cream Sandwich (4.0 – 4.0.4)**
- **Jelly Bean (4.1 – 4.3.1)**
- **KitKat (4.4 – 4.4.4)**
- **Lollipop (5.0 – 5.1.1)**
- **Marshmallow (6.0 – 6.0.1)**
- **Nougat (7.0 – 7.x)**

На претходната слика може да ја видиме процентуалната користеност на сите верзии во денешно време.

4.2 Структура на Андроид проект

Еден андроид проект мора да биде структуриран во специфичен формат за да можат сите датотеки да бидат спакувани во апликација. Сите датотеки имаат свое место во одредени именици со специфични имиња. Ако не се следат тие правила, алатката за градење на апликацијата нема да може да ги најде и ќе ни фрли грешки. Постојат неколку именици и датотеки кои се задолжителни за градење на секоја апликација.
Главната структура е прикажана на следната слика:



слика 10. Структура на андроид проект

Некој од поважните именици и датотеки се следните:

module-name/
Еден модул е колекција на source-code датотеки и конфигурации за градење со кој што можеме да ја поделиме апликацијата во посебни функционални единици. Проектот може да има еден или повеќе модули и исто така еден модул може да користи друг модул како зависност. Секој модул може да биде од тип на апликација или библиотека, и може независно да биде развиван, тестиран, дебагиран и билдан. Повеќе модуле се користат обично кога сакаме да креираме библиотека од дел на нашиот код и да ја користеме во различни типови на апликацијата како на пример со различни ресурси за таблет и телефон.

build/
Тука стојат сите датотеки кои произлегуваат од ‘build’ чекорот. Сите ‘build output files’.

libs/
Се состои од приватни библиотеки.

src/
Ги состои сите датотеки со код и ресурси за својот модул. Тие се групирани во следните именици.

> **androidTest/**
> Се состои од код за инструментациски тестови кои се извршуваат на реален уред или емулатор.
>
> **main/**
> Тука стојат сите датотеки со код (source-code files) и сите датотеки за ресурси (resources files) во соодветни java/ и res/ именици. Датотеките со код се соодветно структурирани во јава пакети.
>
> > **AndroidManifest.xml**
> > Ја опишува природата на апликацијата.
> >
> > **java/**
> > Тука се сите Java датотеки. Java source-code files.
> >
> > **jni/**
> > Датотеки со 'native code' напишан во програмскиот јазик C со користење на Java-Native-Interface (JNI).
> >
> > **gen/**
> > Се состои од генерирани датотеки од алатките за развој, AndroidStudio IDE и R.java фајлот.
> >
> > **res/**
> > Апликациски ресурси, слики, икони и датотеки за изглед (layout files).

assets/
Тука се сместени сите асети на апликацијата што треба да се пакуваат исто така во апликацијата. Датотеките од овај фолдер може да се пристапат од страна на апликацијата исто како file-system. На пример ова е добро место за текстури, податоци за игри, статични HTML и други.

test/
Датотеки за тестирање - Unit-tests.

build.gradle (module)
Тука се наоѓа конфигурацијата за за градење на апликацијата и овај модул.

4.3 Алатки за развој и AndroidStudio

AndroidStudio е официјалната интегрирана развојна околина (IDE) за развој на Android апликации, базирана врз овнова на InteliJ IDEA развојната околина. Врз моќниот код едитор и алатките за развој на InteliJ IDEA, Андроид студио ни нуди многу дополнителни можности кои ја подобруваат продуктивноста при градење на андроид апликации. Некој од нив се:

- Флексибилен систем за градење базиран на Gradle. Gradle Build-System.
- Емулатор кој е брз и богат со функционалности.
- Унифицирана средина за развој на сите типа андроид уреди.
- Instant Run за правење на промени во апликацијата инстантно и без градење на ново APK.
- Код шаблони и интеграција со Github.
- Широка рамка на алатки за тестирање
- C++ поддршка за развој на делови во 'native' код
- Вграден супорт за Google Cloud Platform

Во самиот AndroidStudio исто така се вклучени сите алатки од Android SDK, и ни нуду можност за обновување на истите.
Андроид студио корисничкиот интерфејс е прикажан на слика подоле.

1. Лентата со алатки овозможува да се извршуваат широк спектар на активности, вклучувајќи стартување на вашата апликација и пуштање на Андроид алатки.

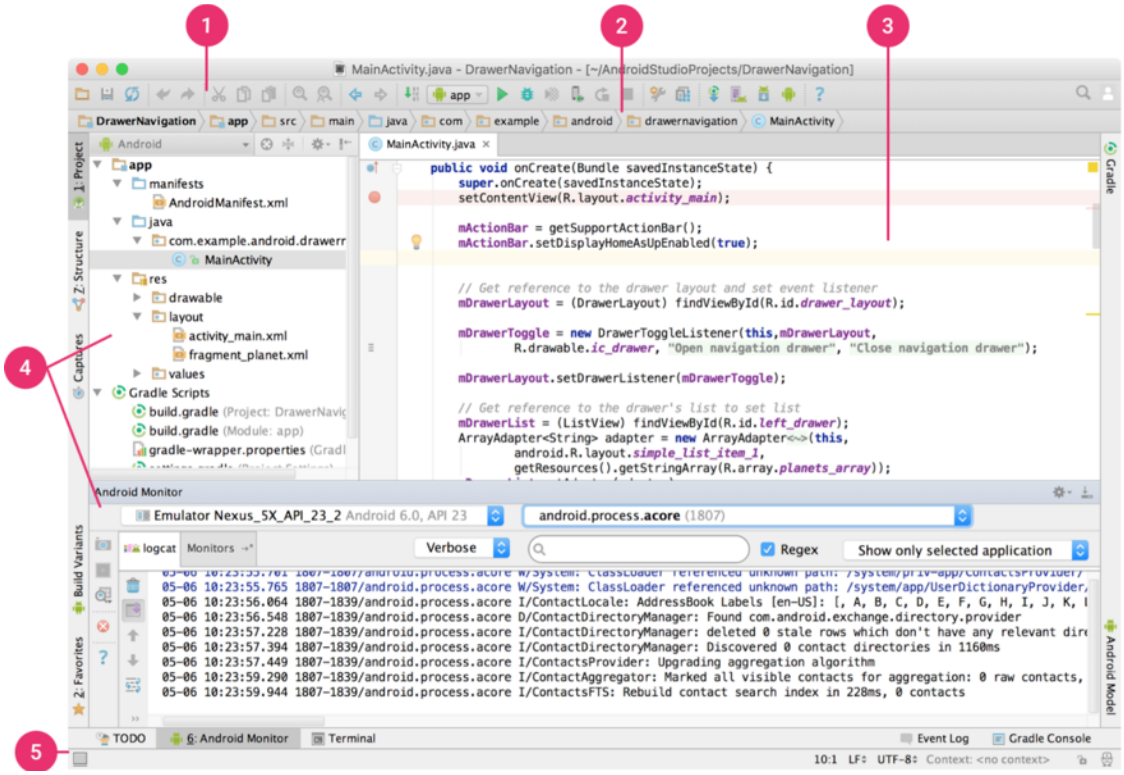
2. Лента за навигација ви помага да се движите низ вашиот проект и да отворите датотеки за уредување. Таа обезбедува повеќе компактен поглед на структурата видлива во прозорецот на проектот.

3. Прозорецот на уредувачот е местото каде што ќе се создава и менува код. Во зависност од моменталниот тип на датотека, уредник може да се промени. На

пример, кога гледате датотеката за распоред на UI елементи, уредувачот прикажува Layout-Editor.

4. Windows алатката ви дава пристап до специфични задачи како управување со проекти, пребарување, контрола на верзии, и многу повеќе. Можете да ги проширите и затворете.

5. Статусната лента го прикажува статусот на вашиот проект и самиот IDE, како и сите предупредувања и пораки.



слика 11. Android Studio - развојна околина

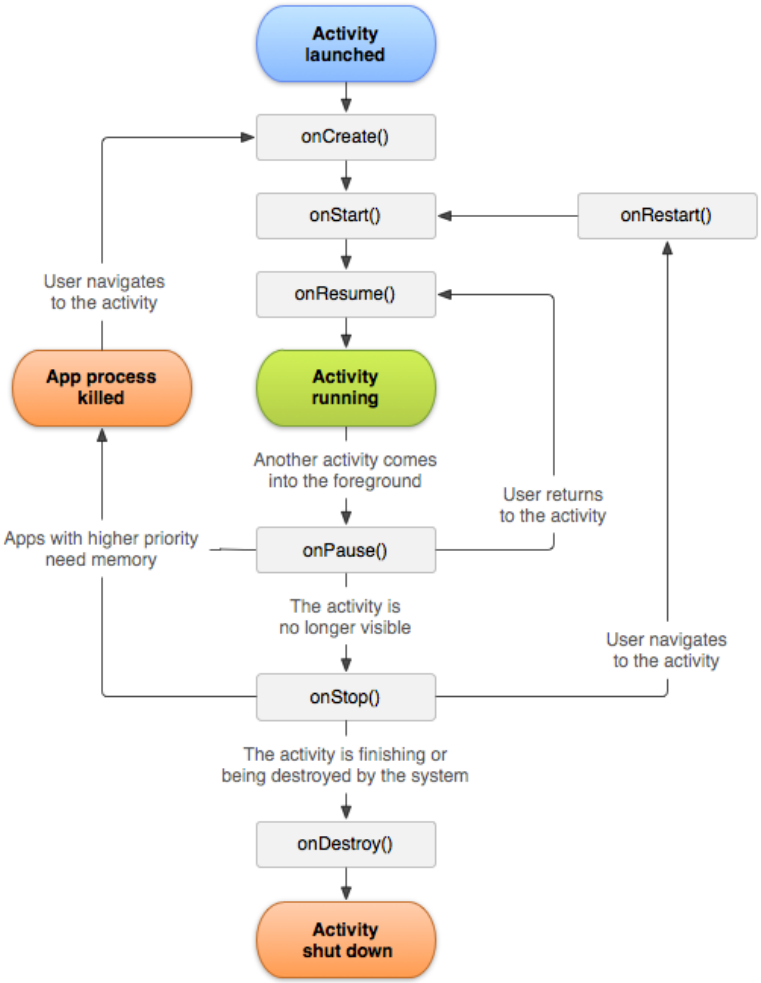
4.4 Компоненти на Андроид апликација

4.4.1 AndroidManifest.xml

Како што беше спомнато погоре, ја опишува природата на апликацијата. Главниот фајл кој ги опишува и специфицира нејзините компоненти. Тука се дефинирани сите главни параметри како пермисии, верзија, име на апликацијата, активности, сервиси итн. Овај фајл е користен исто така од google play-store маркетот за да имовозножи инсталација на соодветни поддржани телефони.

4.4.2 Активности (Activity)

Activity или активност е единечна, фокусирана работа која што корисникот ја извршува на сбојот уред. Скоро сите активности во една апликација имаат интеракција со корисникот и му прикажуваат свој изглед (layout).
Една активност може најпросто да се објасни како еден екран, на кој што имаме одредена функционалност и изглед, и на тој екран правиме интеракција со допир или пишување на виртуелна тастатура. Примери за активности се звонење на телефон, правење фотографија, праќање на е-маил, гледање на мапа итн. На секоја активност и е даден прозорец за да го исцрта својот кориснички интерфејс. Прозорецот типично го исполнува целиот екран, но може да биде и помал и да “лебди” врз други прозорци.

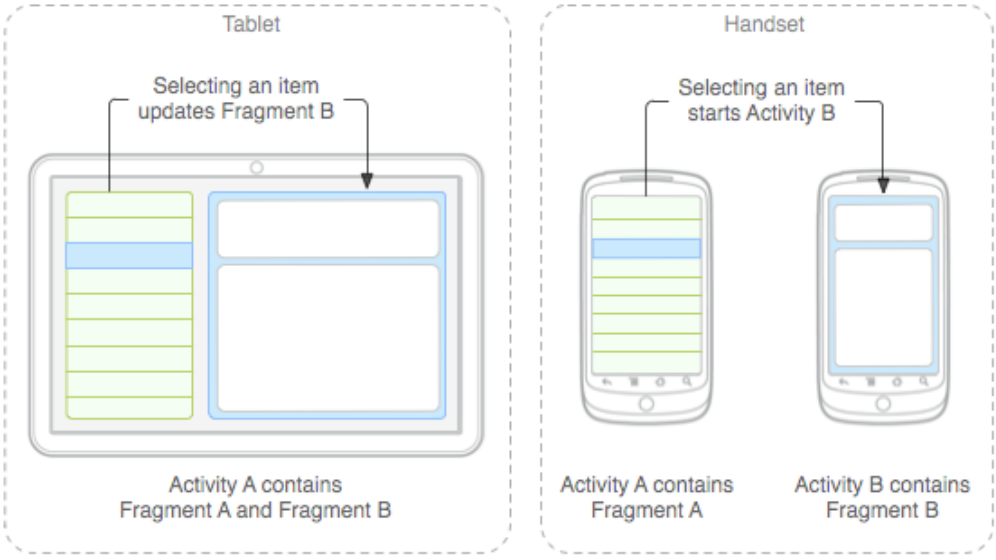


слика 12. Животен циклус на една активност

Една апликација најчесто има повеќе активности кои дефинираат одреден тек на движење и функционалности во апликацијата. Секоја апликација мора да дефинира така наречена стартна активност (Launcher-Activity), која ќе се стартува при клик на апликацискатa икона од главното мени на уредот. Потоа секоја активност има можност да стартува друга активност, и исто така да се движи назад во претходната.
Активност во андроид платформата е дефинирана со класата Activity. За креирање на активности во нашата апликација, мора класите да наследуваат од Activity класата. Со помош на таа класа, потоа поставуваме изглед на екран и интеракција. Активности имаат животен циклус кој е предефиниран од платформата и може да се креира, стопира, паузира и уништува. Треба соодветно да се грижиме за состојбата и животниот циклус на секоја активност како на пример ослободување на ресурси при стопираење и слично. Activity класата ни нуди методи кои се повикуваат при промена на состојба, на сликата е даден целосен тек на животен циклус на една активност.
Активности се најбитните компоненти во една андроид апликација.

4.4.3 Фрагменти (Fragment)

Фрагментите се воведени во Android Honeycomb (3.0) и нудат можност за користење во имплементација на една активност за подобро модуларизирање на кодот и градење на пософистицирани корисничќки интерфејси за поголеми екрани. Со тоа ни помагаат во скалирање на апликациите за големи и мали екрани. Фрагментите треба да се користат за ре-искористување на код и изглед од повеќе активности. Еден фрагмент не може да постои сам по себе, туку мора да биде хостиран и менаџиран од страна на една активност.



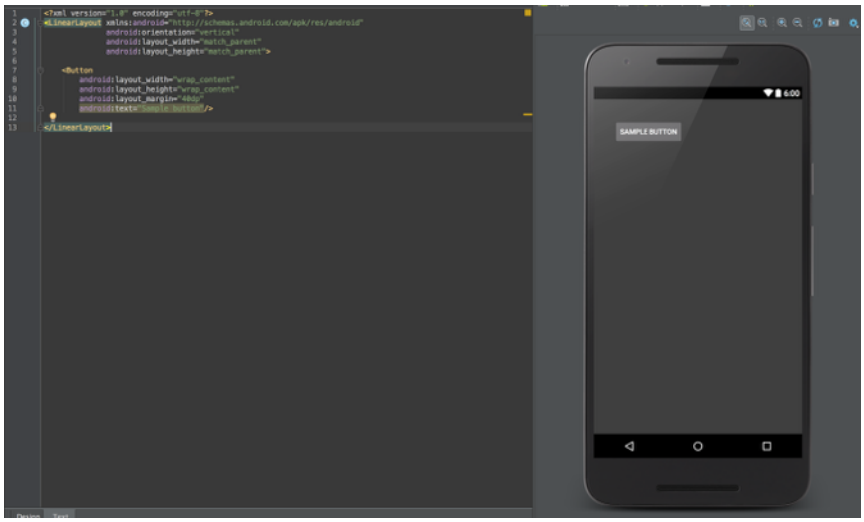
слика 13. Пример за користење на фрагменти

Фрагмент репрезентира однесување или дел од кориснички интерфејс на активност. Може да се комбинираат повеќе фрагменти во една активност за да се изгради мулти панел кориснички интерфејс а воедно еден фрагмент ќе може повторно да се искористи во повеќе други активности. Фрагментот може да се замисли како модуларна секција од активност, која има свој животен циклус, свои настани и исто така може да се додаде или отстрани додека една активност се уште работи.
За креирање на фрагмент, треба нашата класа да наследува од Fragment класата која ние е достапна од платформата.
Постојат многу различни начини на користење на фрагменти, еден таков пример е прикажан на слика погоре.

4.4.4 Изглед (Layout)

Изгледот на секој екран во една андроид апликација мора да биде специфициран на одреден начин. Распоредот на погледот ја дефинира визуелната структура на корисничкиот интерфејс за активност или фрагмент. Постојат два начини со кој што може да се дефинира распоред на поглед:
- Дефиниција на поглед преку XML. Андроид платформата има XML вокабулар за креирање на компоненти од кориснички интерфејс. Вокабуларот за декларирање на распоредите ја следи структурата на имињата на класите и методите, каде што имињата на елементите одговараат на имињата на класите и имињата на атрибутите одговараат на имињата на методите.
- Дефинирање на поглед преку код. Со креирање на сопствени класи кои наследуваат од класите понудени од платформата (View, ViewGroup, LinearLayout, FrameLayout etc.) можеме да направиме многу софистицирани погледи. Во код можеме да правиме комплицирани пресметки и аминации за комплициран кориснички интерфејс. Со тоа се креираат погледи и групи на погледи за време на извршување и можност за модификации.

Најчесто се користат XML датотеки за изгледот бидејќи е најбрзо и наједноставно. За покомплексен изглед кој што не може да се креира статичи, мораме да го пишуваме во код. XML датотеката мора да ги следи сите правила за коректно форматиран XML. Секој фајл за распоред мора да содржи точно еден елемент – корен (root element), кој што мора да биде објект од поглед или група од погледи. Од како елементот корен е дефиниран, може да се додаваат дополнителни објекти како деца, за да се изгради потребната хиерархија што ќе го дефинира посакуваниот изглед. На следната слика е прикажан XML код од изглед кој што користи вертикална положба (LinearLayout), и во него содржи текст поле (TextView) и копче (Button).



слика 14. Креирање на изглед на екран во андроид-студио

4.4.5 Ресурси (Resources)

Ресурси во андроид апликација се сите слики, икони, позадини, текстови, бои и стилови.
Претходно, каде што ја опишавме структурата на андроид проект, видовме дека има посебен именик каде што треба да бидат сместени сите ресурси. Тоа е “module-name/src/main/res” именикот. Сите ресурци секогаш треба да се сместени надворешно, односно да не бидат вметнати во кодот кој се компајлира. Одделувањето на ресурси овозможува дефиниција на специфични ресурси за ордерени конфигурации на уреди и локализација на текстови. Со цел да се овозможи компатибилност за различни конфигурации ресурсите мора да бидат организирани во директориумот res/, користејќи различни поддиректориуми што ги групираат ресурсите по тип и конфигурација. Неколку примери за поддиректориуми се:

- **drawable-hdpi** Слики и икони наменети за екран со висока резолуција.
- **drawable-mdpi** Слики и икони наменети за екран со средна резолуција.
- **values** Именик за текстови, димензии и стилови.
- **values-de** Истиот именик но локализиран за уред со германски јазик.
- **layout** Именик за сите XML датотеки за изглед. И тие претставуваат ресурси.
- **color** Именик за дефинирање на бои.

Како што гледаме од структурата на именици за ресурсите, постои можност да дефинираме еден ист ресурс за стандардна конфигурација и специфична (јазик, големина, ориентација итн.). За пример, кога стандардниот изглед е зачувам во директориумот res/layout, може да се дефинира различен изглед кој ќе биде користен кога уредот е во хоризонтална положба со тоа што тој изглед ќе биде сочувам во директориумот res/layout-land/. Андроид автоматски ги користи ресурсите доколку тековната конфигурација на уредот се совпаѓа со името на директориумот.

4.4.6 Сервиси (Services)

Андроид сервис е компонента која работи во позадина, без директна интеракција со корисникот. Сервисот нема кориснички интерфејс и затоа не е врзан со животниот циклус на активности. Сервиси се користат за повторливи и потенцијално долги операции, како на пример интернет преземања, проверка за нови податоци, обработка на податоци, слушање музика, ажурирање на содржината и други слични услуги.
Сервисите кои се стартуваат од нашите апликации работат со повисок приоритет од неактивни или невидливи активности и поради тоа е помалку веројатно дека Андроид системот ќе ги уништи. Исто така може да биде конфигурирани да се рестартираат ако тие се прекинат од страна на системот во момент кога повторно доволно системски ресурси се на располагање.
Сервисите по дифолт се извршуваат на иста нишка како и главната нишка на апликацијата (UI Thread). Задолжително е да креираме механизам во самиот сервис кој што ќе ја извршува задачата асинхроно на позадинска нишка (Background Thread). Со тоа нема да се блокира корисничкиот интерфејс додека спуштаме некоја слика од интернет во сервис. За цел на асинхроно извршување постојат неколку механизми понудени од платформата, како на пример AsyncTask, Thread, JobScheduler, Executor и други.

4.4.7 Сензори, GPS и Bluetooth

Секој модерен андроид уред има вградени сензори кои што можеме да ги користеме во нашите апликации. За пристап до одреден сензор прво треба да го прашаме корисникот за пермисии, а потоа да го пристапиме преку понудената класа “SensorManager”. Добивање на инстанца од “SensorManager” се прави преку повикот “getSystemService(SENSOR_SERVICE)”. Одкако е добиена инстанца на специфицен сензор, со барање преку сензор-менаџерот, треба да регистрираме т.н слушач или “SensorEventListener”. Овај слушач ќе биде информиран стално кога вредности на сензорот ќе се променат. За да избегнеме непотребно користење на батеријата, треба да го отстраниме слушачот при паузирање на активностите.

Пример за добивање локациска вредност од GPS сензор:

```
1. // Get the location manager
2. LocationManager locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
3. Criteria criteria = new Criteria();
4. String provider = locationManager.getBestProvider(criteria, false);
5. Location location = locationManager.getLastKnownLocation(provider);
```

4.4.8 Меморија и пристап

Андроид платформата ни дозволува секоја апликација да има можност за чување на апликациски податоци во фајл-системот. За секоја апликација, при инсталација оперативниот систем креира соодветни именици ексклузивно наменети за чување на податоците од нашата апликација. Именикот што се креира е следниот:

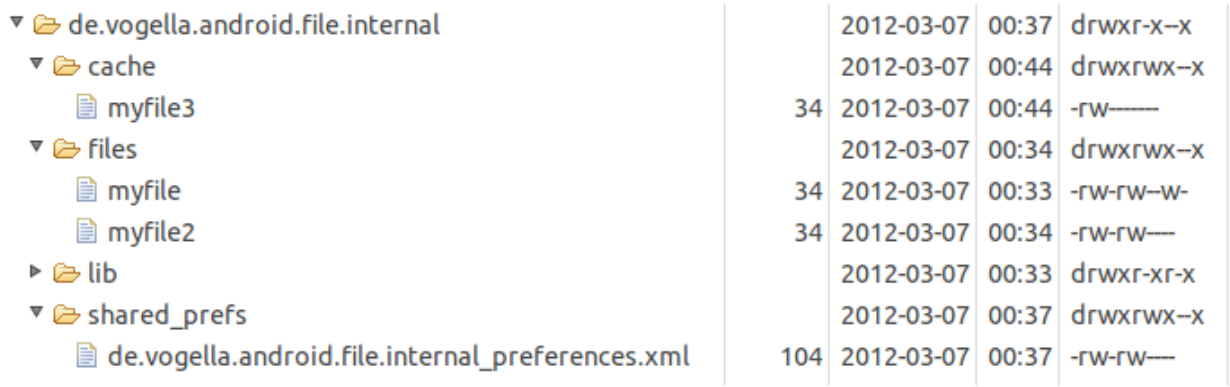
data/data/[application package]

Постојат неколку начини на кој што можеме да чуваме податоци:

1. Датотеки (Files) - Креирање и промена на датотеки зачувани на фајл-системот.
2. Преференци (Preferences) - Зачувување и читање на клуч-вредност парови од примитивни податочни типови.
3. SQLite база на податоци - Класична имплементација на SQL база на податоци, исто зачувана на фајл-системот.

Датотеките се чуваат во папката “files”, а поставувања за апликација се зачувани како XML датотеки во папката “shared_prefs”.
Ако нашата апликација креира база на податоци (SQLite), базата на податоци ќе биде зачувана во главниот директориум, во папката “databases”.

На следната слика гледаме пример на податочен систем кој содржи датотеки, податоци од кеш, преференци.



слика 15. Фајл-систем на апликацијата

Битно е да се напомени дека само нашата апликација има пристам до овај главен директориум. Може да креираме додатки под-директориуми во главниот, и за нив да дадеме пермисии до други апликации ако треба да споделиме некој ресури.

4.5 Gradle Build System

Процесот на пакување или билдање на андроид апликација се врши преку алатката “Gradle”. Gradle е т.н “Build-Tool” кој е дизајниран да поддржува комплексни сценарија при креирање на андроид апликации. Некој од нив се:

- Дистрибуција на истата апликација со кастомизација за повеќе клиенти или компании. (Multi-Distribution)
- Поддршка за креирање на повеќе пакети апк од апликацијата наменети за различни типови на уреди со реискористување на делови од кодот. (Multi-APK)
- Менаџмент на зависности до библиотеки.

Gradle во главно е алатка за Java проекти но исто така поддржува голем број на други намени. Тоа е алатка која се извршува од командна линија (CommanLineInterface), но AndroidStudio околината има многу добра интеграција на Gradle и ги извршува соодветните команди за нас при стартување на апликацијата.
Конфигурацијата на Gradle за пакување на нашата апликациа се пишува во датотеката: build.gradle

Еден пример на таков build.gradle фајл е следниот:

```java
android {
    ...

    buildTypes {
        release {
            minifyEnabled true
            shrinkResources true
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}
```

5. Android OBD2 Reader

Претходно опишавме што е точно диагностика на модерни автомобили, и во кратки црти ги објаснавме најбитните компоненти од андроид оперативен систем. Од многу интересен аспект е поврзување на овие две технологии. Тоа ќе го покажам со креирање на андроид апликација која ќе се поврзе преку bluetooth со OBD-II читач и ке комуницира со возилото.
Андроид апликацијата, **Android OBD Reader**, има голем број на функционалности кои ќе бидат објаснети подетано во под-точки подоле.
Со воспоставување на таа врска, ни се отвараат голем број на други можности за обработка и анализа на податоците од возилото. За секој автомобилски ентузијаст тоа би било од голем интерес.

5.1 Структура на проектот

Проектот е структуриран по стандардите на андроид платформата. Сите датотеки се сместени во соодветни папки и добро организирани.
Именувањето на класите и други компоненти е направено внимателно и добро смислено, со тоа проектот е многу лесно разбирлив и оддрзлив.

При развој е користен верзиски контролен систем VCS Git, хостиран на приватно Bitbucket Repository. Со тоа се овозможува голема сигурност при развојот и лесно навраќање на постари верзии од кодот по потреба.

5.1.1 MVP - Model-View-Presenter

MVP патернот претставува “Software Design Pattern”, што е еден вид на структурирање софтверски проект. Самото име ни кажува дека проектните датотеки и класи треба да се организираат во три групи, модели, погледи и презентери.
Подетално објаснување за секоја група:

- **Погледи** - Тоа е еден слој во проектот или апликацијата кој што е наменет за прикажување на податоци и реагирање на корисничи интеракции. Во андроид апликација тоа би може да биде една од следните компоненти: Активност, Фрагмент, Диалог или друго.
- **Модели** - Тие е слој за пристап до податоци како на пример пристап до податоци во база на податоци или пристап до REST сервис на сервер. Исто така во таа група спаѓаат сите обични POJO класи.
- **Презентери** - Слој кој што го пополнува погледот со податоци од моделот. Исто така е задолжен за извршување на акции и пресметки во позадински нишки.

Во андроид проект, MVP е начин да се разделат позадински задачи од активности/фрагменти/погледи и со тоа да се независни од скоро сите настани поврзани со животниот циклус на погледите. Со примена на овај “pattern”, проектот станува поедноставен, доверливоста и стабилноста на апликацијата се зголемува до десет пати, кодот е пократок и оддржувањето е подобро и многу лесно.
Исто така од големо значење е тестирањето. Развивање и имплементирање на UnitTests и IntegrationTests е многу битно не само за проект од мобилната сфера, туку за било каков софтверски проект. Ако еден андроид проект има целосна примена на MVP, многу е лесно да се пишуваат UnitTests за сите презентери и модели, бидејќи тие се одделени и независни од платформските компоненти. За тестирање на погледите и платформските компоненти потоа се користат интеграциски тестови кои што се извршуваат на реален уред или емулатор.

5.2 Воспоставување на Bluetooth конекција помеѓу ELM327 и Андроид уред

Секој модерен паметен телефон во денешно време има можност за bluetooth конекции. Објаснавме исто така дека постојат ELM327 чипови на пазарот со интерфејс за bluetooth конекција.
Ако нашиот андроид уред го поврземе со ELM327 (pairing), ќе се креира серијален комуникациски канал помеѓу двата уреда. Баш тоа ни треба за понатамошно праќање/примање на податоци од возилото.

За отварање на bluetooth конекција на андрод уредот прво треба да го земеме системскиот адаптер и преку него да отвореме конекција. Тоа се прави на следниот начин:

```
BluetoothAdapter btAdapter = BluetoothAdapter.getDefaultAdapter();
BluetoothDevice dev = btAdapter.getRemoteDevice(BLUETOOTH_DEVICE_NAME);
BluetoothSocket sock = BluetoothManager.connect(dev)
```

Со извршување на овие повици имаме успешно креирано сокет за комуникација. Тој сокет ни нуди влезно/излезни стримови (input/output streams) врз кои што ќе пуштаме и примаме податоци.
Испрашување на возилото за моменталните вртежи на моторот, се прави со следниот блок:

```
OutputStream out = sock.getOutputStream();
out.write((“01 0C” + ”\r”).getBytes());
out.flush();
```

```
// читање на одговор
InputStream in = sock.getInputStream();
String result = IOUtils.read(in, buffer);
```

Со овие команди испративме бајти до возилото и го вчитавме одговорот од автомобилот назад во бафер, секоја команда има свој начини на калкулирање и пресметување на одговорот за добивање на разбирлива вредност.
За тие потреби, имам напишано класа за секоја OBD команда, која знае точно кој бајти треба да се испратат до возилото за добивање на информации, и потоа исто така има метода за пресметка и калкулирање на одговорот.

5.3 Приказ на податоци во реално време

Креирав функционалност во апликацијата која ни дозволува да пратиме одредени параметри од возилото во реално време. За да се реализира тоа, прво креирав класи и инстанци за сите команди и информации кои што би сакал да ги гледам. Потоа се воспоставува bluetooth конекција на начинот објаснет во

слика 16. Android OBD-Reader - Realtime data

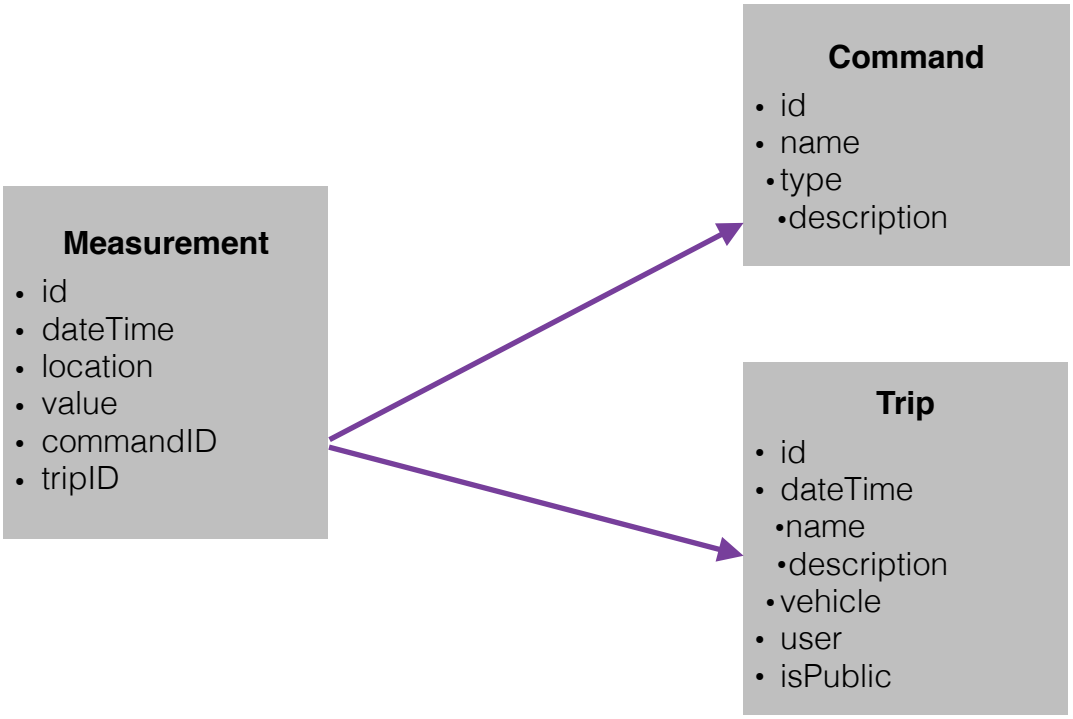
предходниот параграф, и врз таа конекција ги пуштаме сите команди од интерес во циклус на одреден временски интервал. Возилото ни одговара на секој круг од циклусот и добиените вредности ги зацувуваме во мапа.
Имам креирано визуелизација на вредностите од мапата врз екранот на телефонот. Сите вредности се освежуваат во одреден временски интервал, и добиваме еден вид на “Dashboard” со информации за моменталната состојба на нашето возило. Во овој пример имам одбрано само неколку параметри, но листата може да се дополне по желба.

5.4 База на податоци

Чување на податоци на диск е исто така интегрирано во апликацијата. Има можност за логирање на добиените вредности од возилото и исто така нивно зачувување на фајл систем од уредот.
При секоја комуникација со возилото добиваме неколку информации кои што се од интерес. Нормално тука спаѓа и информацијата која што е побарана од возилото, но исто така и вредности од сензори на мобилниот уред. Се чуваат следните информации:

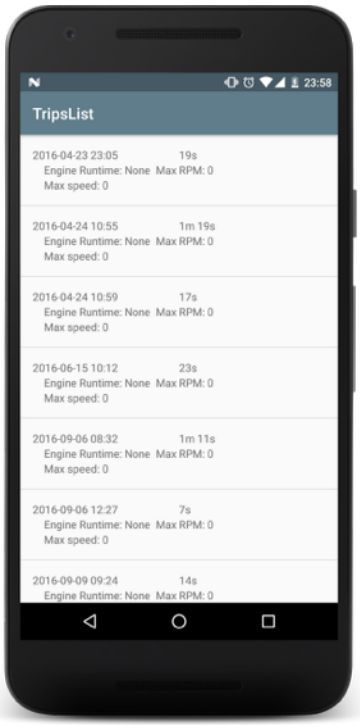
- Време и датум
- Географска должина
- Географска ширина
- Надморска висина
- Идентификациски број на возилото
- Идентификациски број на параметарот побаран од возилото
- Вредност на податокот

Логирање и чување на податоци од возилото е од големо значење, бидејќи потоа може множеството на податоци да биде анализирано и визуелизирано за разни цели. Првичната имплементација за чување на податоците која ја имплементирав во апликацијата беше во CSV фајл. Тоа претставува текстуален фајл во кој што вредностите се одделени со кома. Ова претставува многу лесен начин за складирање на податоци, без компликации за креирање на SQL база.
Но, со раст на обемот на податоци, или снимање на многу долги патувања, складирањето во CSV фајл нема воопшто да е ефикасно и ќе треба да се бара друг начин, најсоодветно би било SQL база на податоци.
SQL базата има голема ефикасност и ни нуди можност за лесно пребарување и групирање на податоците. Иницијалниот дизајн на базата од аспект на табели би можело да изгледа вака:



5.5 Преглед на зачувани патувања

Апликацијата нуди функционалност за преглед на зачувани патувања и нивно анализирање. Секое патување има неколку параметри кои го карактеризираат и голем број на мерења асоцирани со него. Приказ на листата со зачувани патувања во апликацијата се прави со кликање на копчето “Trips”.



слика 17. Android OBD-Reader - Trips

5.5.1 Графички приказ на параметри

Со кликање на едно патување од листата можеме да ги видиме деталите за тоа патување. Имплементирано е графичко претставување на неколку параметри од тоа патување. Графот е исцртан врз база на времето во кое е земено мерењето и вредноста на податокот. Со ваков графички приказ имаме инстантен увид во состојбата на возилото во одреден временски интервал и лесна анализа и истражување.
На слика може да се видат графовите за брзината на движење и надворешната температура.

5.5.2 Мапа на интензитет

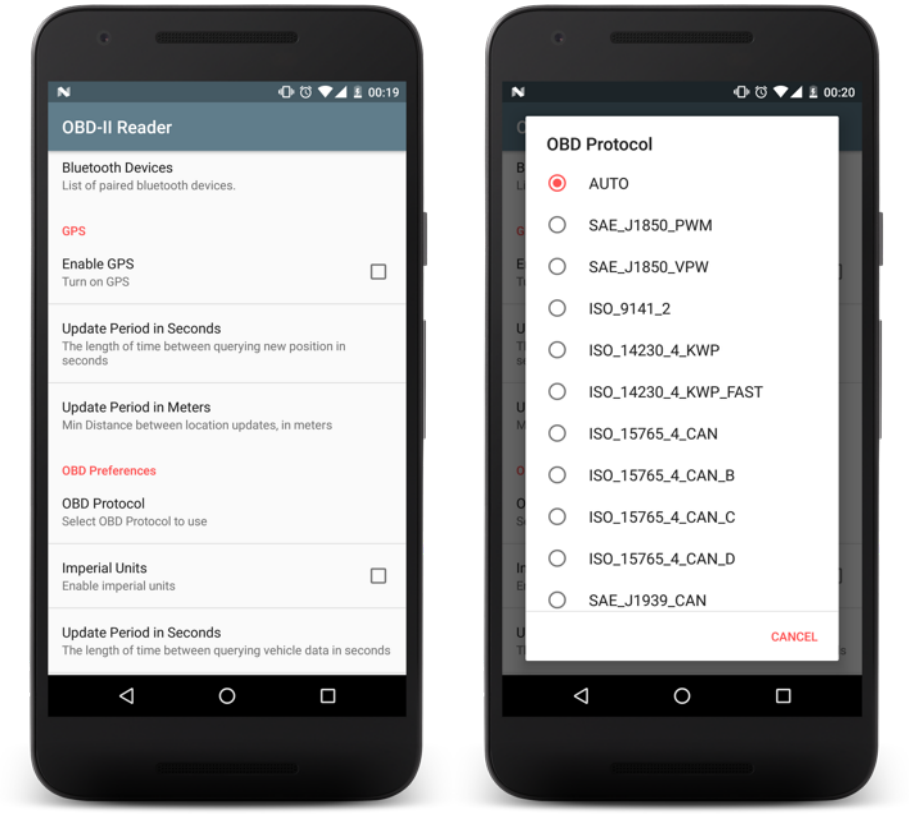
Мапа на интензитет претставува обојување на делови од мапа во зависност од вредноста или множеството на точки на таа географска локација. Со мапирање на многу точки добиваме слика за движењето и интензитетот на одредени параметри во одредена географска локација.
Бидејќи ние извршуваме прибирања на податоци во автомобилско возило кое се движи, можеме било кој параметар да го визуелизираме на мапа на интензитет. Еден пример за такво нешто е прикажан на сликата.

слика 18. Android OBD-Reader - Trip Details

5.6 Подесувања

За лесно подесување на кориснички преференци, креиран е екран за подесување “Settings”. На екранот за подесувања може да се постават многу различки работи, некој од нив се:

- OBD протокол
- Локациски сервиси (GPS)
- Временски интервал на освежување на податоците од возилото
- Листа на параметри кои ги испраќаме до возилото
- Селекција на Bluetooth уред
- Користени мерни единици
- Серверска адреса
- Локација за чување на базата



слика 19. Android OBD-Reader - Settings

6. Заклучок и идна работа

Со креирање на оваа апликација покажавме само една од многуте можности кои произлегуваат при поврзување на две современи и развиени технолошки области. Висико перформантните компјутери во автомобилите константно се усовршуваат и надградуваат. Истото се случува и со индустријата на преносни мобилни телефони.
Покажавме колку многу е развиена Андроид платформата и какви се можности ни нуди за интеракција со околината. Мобилната технологија секојдневно трпи промени и подобрувања, производителите на паметни телефони стално истражуваат нови хардверски и софтверски компоненти со што им нудат се поголем и поголем број на функционалности на своите корисници.
Моменталните паметни телефони достапни за јавноста се толку мгогу развиени, што тоа било скроз незамисливо во времето кога единствена цел на телефонот му била јавувањето. Луѓето своите телефони секој ден ги вклучуваат како алатка во се повеќе секојдневни активности, и тоа ќе продолжи.

Исто како и индустријата на мобилни телефони, денешните автомобилите се развиваат година за година и гледаме како произведувачите вметнуваат се повеќе технолошки напредни уреди во своите возила. Пред кратко време, единствена намена на возилата им била да служат како превозно средство, денес автомобилите се технолошко развиени предмети од висок ранг.
Сето тоа не е без негативна страна. Иако автомобилите се многу развиени, не се доволно заштитени и имаат многу дупки за напади од хакери. Тоа е главен проблем, произведувачите се префокусирани кон развој на нови феунцоналности и интеграција на хардвер, што забораваат да вложат доволно време во сигурносни мерки на вградените компјутери. Автомобилите се многу ранливи на напади, што може да води до катастрофални последици, само во последната година видовме неколку презентирани напади врз модерни возила.
Денешниот свет не функционира без компјутерите, аналогно и автомобилите не се повеќе механички направи, тие се дигитални, поврзани со околината, управувани и контролирани од микро-чипови. Што е следно, и во која насока се движи автомобилската индустрија? Многу луѓе се едногласни во одговорот, електрика. Растот на електрични возила е во почетна фаза моментално во светот и се очекува да има бум во следните години. Голем дел од механичките делови во електричните возила се заменети со дигитални компјутери, сензори, чипови. Еден од најголемите играчи во таа класа е Tesla Motors.
Инженерите на тесла развиваат ауто-пилот контролиран од микро-чипови, сензори и електрични мотори за само-управување на возилото. Базиран на софтверска вештачка интелигенција, постигнува фасцинантни резултати.
Оваа сфера е сеуште во почетни фази на развои, има огромем број на интересни можности и отварање на различни нови работни места. Дефинитивно треба да се вложи време и работа што побрзо.

7. Референци

- ELM327 Datasheet
  https://www.elmelectronics.com/wp-content/uploads/2016/07/ELM327DS.pdf
- Onboard-Diagnostic
  https://en.wikipedia.org/wiki/On-board_diagnostics
- OBD-II Parameter IDs
  https://en.wikipedia.org/wiki/OBD-II_PIDs
- Vehicle Hacking blog
  http://www.canhack.org/board/
- CanHack Forum
  http://www.canhack.org/board/
- OBD Java API
  https://github.com/pires/obd-java-api
- OBD Tips by ELM electronics
  https://www.elmelectronics.com/help/obd/tips/#327_Commands
- Android Developers
  https://developer.android.com/index.html
- Android References
  https://developer.android.com/reference/packages.html
- Vogella Android tutorials
  http://www.vogella.com/tutorials/android.html
- Car hacking at DefCon
  https://www.youtube.com/watch?v=OobLb1McxnI
  https://www.youtube.com/watch?v=KX_0c9R4Fng