

ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

ДИПЛОМНА РАБОТА

Тема: Мобилно приложение – сервизна книжка за автомобил

Дипломант:

Симеон Шопкин

Научен ръководител:

маг. инж. Кирил Митов

С О Ф И Я

2 0 1 6

Съдържание

УВОД.....	6
ПЪРВА ГЛАВА	7
Проблематика и сходни приложения	7
1.1. Проблематика.....	7
1.2. Настоящата дипломна работа.....	7
1.3. Сходни приложения.....	7
1.3.1. Car service Free.....	7
1.3.2. aCar – Car Management	8
1.3.3. Fuel Buddy.....	9
1.3.4. Cars Manager	10
1.4. Обобщение:	10
ВТОРА ГЛАВА	12
Основна структура на приложението и аргументация за избраните средства и технологии	12
2.1. Функционални изисквания към програмния продукт и развойната среда	12
2.1.1. Изискванията, на които трябва да отговаря приложението са следните:	12
2.1.2. Изискванията, на които трябва да отговаря развойната среда са следните:	12
2.2. Избор на операционна система, език за програмиране и софтуерни средства	12
2.2.1 Операционна система.....	12
2.2.2. Език за разработка	13
2.2.3. Софтуерни средства	13
2.3. Структура на приложението.....	13
2.4. Структура на базата.....	15
2.4.1. Първа версия	15
2.4.2. Втора версия	16
2.4.3. Трета версия	17
2.5. Wireframe-ове на приложението.....	18
ТРЕТА ГЛАВА	21
Програмна реализация на приложението.....	21
3.1. Определяне етапите на разработка	21
3.2. Създаване на приложението.....	22
3.2.1. Основна функционалност	22
3.2.2. Безплатна версия	26
3.3. Основни компоненти	45

ЧЕТВЪРТА ГЛАВА.....	46
Ръководство на потребителя	46
4.1. Инсталация.....	46
4.2. Добавяне на запис.....	47
4.3. Показване на различни статистики.....	48
4.4. Редакция на объркан километраж	48
4.5. Набиране на номер за пътна помощ.....	48
4.6. Изпращане/импортиране на базата.....	49
ЗАКЛЮЧЕНИЕ	50
Постижения в дипломната работа.....	50
Бъдещо развитие.....	50
Използвана литература	51

УВОД

Целта на настоящата дипломна работа е създаване на мобилно приложение за Android, което представлява автомобилна сервизна книжка. Чрез приложението трябва да е възможно управлението на данни за извършени дейности, импортьт и експортьт на базата, както и предоставянето на различни статистики, посредством информацията, която е въвел потребителят.

В днешно време голяма част от хората използват за транспорт лични или служебни пътни превозни средства. Превозът на товари също в голям аспект разчита на лекотоварни или тежкотоварни автомобили.

За повечето собственици на автомобили е трудно да помнят кога, какво и за каква сума е извършено по автомобила. Запазването на тази информация е свързано със съхранението на много касови бележки, документи и гаранции, което е изключително неприятна дейност. За да се разбере колко пари струва дадена дейност за автомобила за определен период от време са необходими много сметки.

Тук се намесват приложенията, които помагат на собствениците (водачите) на автомобили. Те вършат цялата работа, свързана с показването на различна информация, вместо хората, като им предоставят възможност да оползотворят времето си по подходящ начин.

През последните години се наблюдава ръст на използването на мобилни технологии за различни цели. Превръщането на телефоните в „умни“ става чрез операционната система.

„Android“ е една от най - широко разпространените и най - бързо развиващите се мобилни операционни системи в днешно време, управлявана от един от големите брандове в света на технологиите - „Google“. Linux базирана, отвореният код на системата неминуемо допринася за нейното разрастване.

ПЪРВА ГЛАВА

Проблематика и сходни приложения

1.1. Проблематика

Всеки собственик на автомобил знае, че е трудно да следи постоянно разхода на автомобила, да помни кога е сменял последно маслото, да помни кога, къде и за какви пари е извършвал различни ремонти по автомобила. Също така на всеки се е случвало да забрави кога последно е заредил гориво и на каква цена. Тук се намесват приложенията „помощници“ на собствениците на автомобили. Тяхната цел е да помагат в различните видове дейности и изчисления, оставяйки на водачите да се наслаждават на удоволствието от шофирането.

1.2. Настоящата дипломна работа

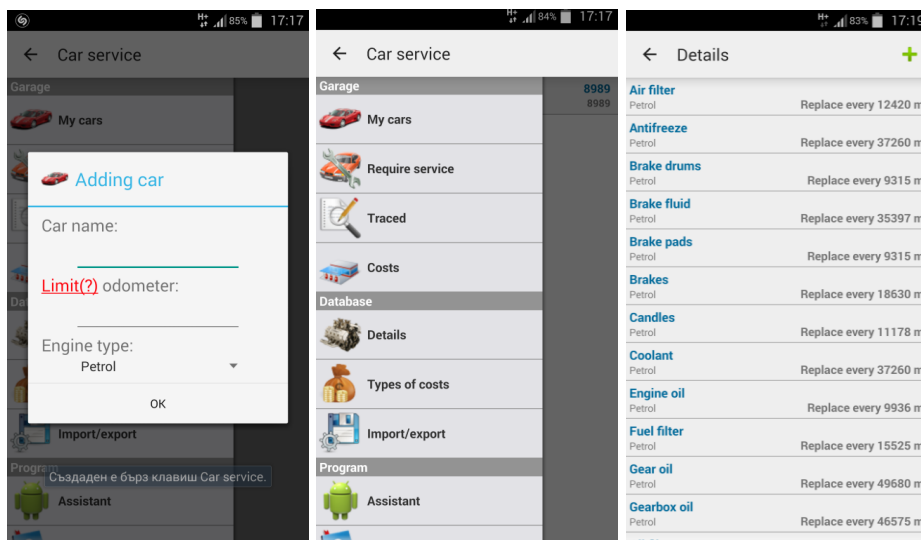
Настоящата дипломна работа има за цел да реши проблемите и неудобствата, които създава поддръжката и следенето на различните дейности и разходи по автомобила.

1.3. Сходни приложения

1.3.1. Car service Free

“Car service free” е регионалното водещо (с най-голям брой сваляния за България) приложение в “GooglePlay”. Свалено е 50 хиляди пъти, с оценка 3,6 от 591 гласа. Приложението поддържа категории като: гориво, необходими дейности, цени на горивата. Прекалено наблягане на цените и видовете горива без да може да се използват, като повече време отнема самото въвеждане на информацията. Приложението не предлага възможност за въвеждане на какъвто и да било тип дейност, а само информира на колко километра какво трябва да се прави. Няма история на извършените дейности. Локална - не е възможен експорт на информацията. Интерфейсът е объркващ, потребителят не разбира защо е нужно да въвежда различните цени на горивата, като той не ползва повече от два вида гориво. С “car service” се работи бавно и

непрецизно. На фиг. 1.1, 1.2 и 1.3 са показани някои от основните екрани на приложението.



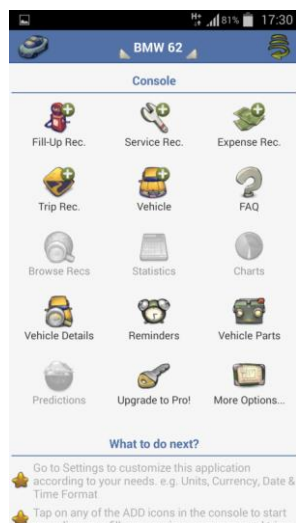
Фигура 1.1

Фигура 1.2

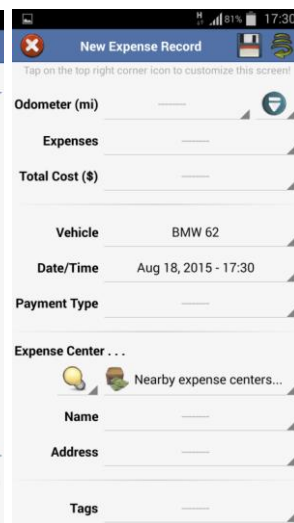
Фигура 1.3

1.3.2. aCar – Car Management

“aCar – Car Management” е приложението с най-голям брой сваляния – 1 милион, като оценката от гласувалите близо 20 хиляди души е 4,5. Безплатната версия на програмата няма статистика. Безплатната версия е с много намалена функционалност. Платената версия е на цена от почти девет лева към януари 2016г. Не е възможен експорт на базата с данни. Трудно преминаване между различните мерни единици – галон/литър, миля/километър и тн. Дизайнът тук е прекалено детайлизиран, което не е в крак с обявеното от Google за “добро” и препоръчително. На фиг. 1.4 и 1.5 са показани някои от основните екрани на приложението.



Фигура 1.4



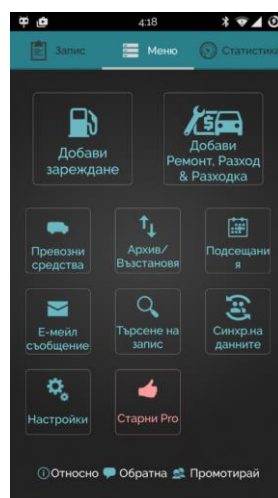
Фигура 1.5

1.3.3. Fuel Buddy

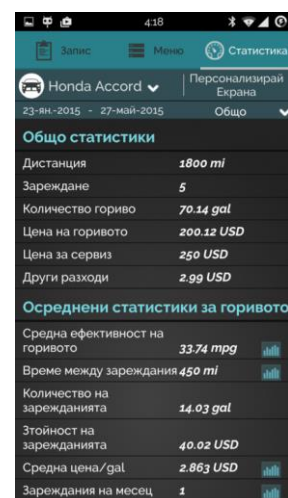
“Fuel Buddy” е приложение със 100 хиляди сваляния и оценка от 4,4 на базата на 3000 гласували. Приложението се откроява с възможността си за гласово попълване на стойности, както и с подсказките за извършване на някакъв тип дейност. За жалост приложението е прекалено финансово ориентирано, отново е невъзможен експорт на базата данни, а също така разполага с малък набор от необходими дейности. Дизайнът е по-прост, но отново е малко объркващ. Менютата се губят. Със закупуване на платената версия ефективността се повишава незначително. На фиг. 1.6, 1.7 и 1.8 са показани някои от основните екрани на приложението.



Фигура 1.6



Фигура 1.7



Фигура 1.8

1.3.4. Cars Manager

Cars Manager е приложение с 50 хиляди сваляния и оценка 4,0 от 700 гласа. Разполага с изчистено меню както и с възможност за износ на базата данни в DropBox или като "csv" файл. Между отделните екрани на приложението се преминава единствено чрез тоолбара, в който от Google препоръчват да има само заглавие и две икони (назад и падащо меню). Не е възможен импорт на база данни. Размерът на иконите е прекалено малък. Отново прекалено детайлизирани икони, както и тъмни цветове. Само и единствено на руски език. На фиг. 1.9, 1.10 и 1.11 са показани някои от основните екрани на приложението.



Фигура 1.9



Фигура 1.10



Фигура 1.11

1.4. Обобщение:

Според мен и направеното проучване, нито едно от разгледаните приложения не разполага с прост и съвременен дизайн. При всяко едно от тях е прекалено с детайлизирането, което в някои случаи е объркващо. Нито едно от разгледаните приложения няма вариант за експорт и импорт на базата

данни. Много малко от разгледаните приложения имат възможността да подсказват за изтичащи задължителна застраховка, годишен технически преглед или каско. Само едно от разгледаните приложения предлага и подсказка за смяна на маслото на базата на средното месечно изминато разстояние. Нито едно от приложенията няма връзка с пътна помощ или дори не предоставя възможност за такава. Статистиките за всяко приложение са сложни, прекалено детайлизирани и изискващи много информация, която се попълва бавно.

Да се създаде приложение, което да помага в наблюдението на разходите за автомобила. Да предоставя на потребителите различни по вид статистики в безплатната си версия. Ако собственикът продаде своя автомобил, то той да може да покаже или изпрати различните видове дейности, които е извършвал, което да даде възможност на купувача да си направи различни изводи, за това дали колата е подържана и експлоатирана по правилния начин. Да бъде възможен експортът и импортът на базата данни. Да изисква минимално количество информация, от която да предоставя максимален брой статистики. Приложението трябва да предоставя възможност за преглед на историята (зареждания, извършени ремонти и т.н.), не само статистики. Да е с прост и съвременен дизайн. Основните цветове да са светли и приятни. Детайлите да бъдат намалени до минимум.

ВТОРА ГЛАВА

Основна структура на приложението и аргументация за избраните средства и технологии

2.1. Функционални изисквания към програмния продукт и развойната среда

2.1.1. Изискванията, на които трябва да отговаря приложението са следните:

- Приложението трябва да има лесен и разбираем потребителски интерфейс
- Възможност за въвеждане на данни за извършени ремонтни дейности по колата
- Възможност за въвеждане на данни за заредено гориво
- Показване на статистики за похарчени средства относно различни дейности за различни периоди
- Показване на статистика за разход на гориво за различни периоди
- Изпращане и получаване на базата данни
- Приходи от реклами

2.1.2. Изискванията, на които трябва да отговаря развойната среда са следните:

- Осигурява среда подходяща за разработка на приложението
- Лицензирана със свободен лиценз или платен лиценз
- Лесна за ползване

2.2. Избор на операционна система, език за програмиране и софтуерни средства

2.2.1 Операционна система – Операционната система се избира да бъде Android, защото тя е водеща по продажби от 2012 година насам според IDC

(<http://www.idc.com/prodserv/smartphone-os-market-share.js>). Тя заема към 2015 година 82,8% от пазара, следвана от IOS с 13,9%. Средният процент на Android за последните три години е 82,47%. Също така устройствата, които ползват Android като операционна система, имат и по-нискобюджетни модели, което позволява по-голям брой потребители да имат достъп до приложението. Изборът за минимална версия на Android също не беше труден, избра се Android 4.0.3 (Ice Cream Sandwich), защото той се използва от 94% от устройствата.

2.2.2. Език за разработка - Езикът за програмиране се реши да бъде „Java“, защото той е официалният език, който е обявен от Google Inc., както и вече имам известен опит с него. Също така бих искал и да напредна малко повече в тази област.

2.2.3. Софтуерни средства – За разработката на приложението бе избран Android SDK, който е безплатен и е на разположение, както за „Windows“, така и за „Linux“. Основната работна среда е „Android Studio“. Това е официалната среда за разработка на „Android“ приложения, която също така е безплатна.

2.3. Структура на приложението

Приложението е разделено на 12 екрана (activities) и два помощни класа. Всеки екран (activity) представлява различна страница на приложението.

2.3.1. MainActivity – Съдържа шест основни бутона за препратки към други activity-та, както и информация за текущия километраж с възможност за редакция.

2.3.2. FuelActivity - Отговаря за записа на данни в таблицата от базата, която е за горивото. Задължителната информация е цена и количество и стойността на километража, а това дали резервоарът е пълен до горе – по избор. Датата, по подразбиране е днешна, но може да се променя. (Фиг. 2.1)

2.3.3. ServiceActivity – Отговаря за извършени ремонти и поддръжка на автомобила. Цената, типа извършена дейност и километража са задължителни.

Допълнителното поле - бележка. Датата по подразбиране е днешна, но може да се променя. (Фиг. 2.3)

2.3.4. InsActivity – Отговаря за въвеждане на застраховките. Задължителни полета са цена, валидност и километраж, а допълнително може да се добави бележка. Датата по подразбиране е днешна, но може да се променя.

2.3.5. WashActivity - Отговаря за въвеждане на информация относно почистване на автомобила. Задължителни полета са тези за цената и километража, а допълнително – това за бележка. Датата по подразбиране е днешна, но може да се променя.

2.3.6. StatisticActivity – Показва различни статистики за автомобила, като разход, колко пари се отделят за поддръжка, изминато разстояние за определен период от време.

2.3.7. SOSActivity – Бутони за връзка с пътна помощ в различните континенти. Допълнителен начин за изкарване на средства от приложението.

2.3.8. ViewAll – История на записите от дадена таблица, подредени по ред на записване, като последният запис е най-отгоре.

2.3.9. ViewOne – Подробно описание на запис с възможност за изтриване или редакция.

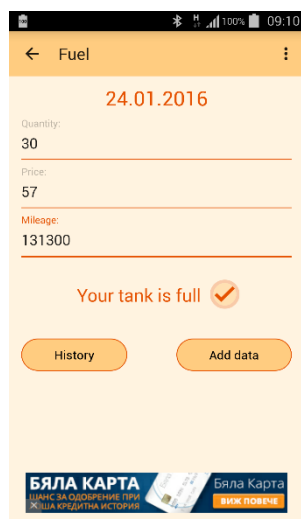
2.3.10. EditActivity – Редакция на даден запис (всички негови полета).

2.3.11. SettingsActivity – Състои се от три падащи менюта, от които потребителят избира какви да бъдат мерните единици за точност и разстояние, както и каква да бъде валутата.

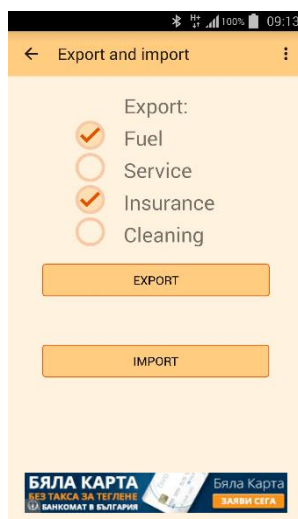
2.3.12. ExplmpActivity – Отговаря за изпращането и импортирането на базата, като потребителят има възможност да избере коя точно информация да прати. (Фиг. 2.2)

2.3.13. DatabaseHelper – Клас който отговаря за работата на приложението с базата данни.

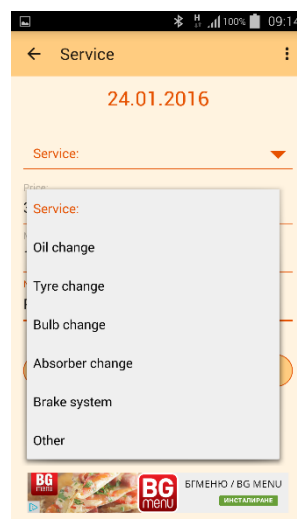
2.3.14. ActivityHelper – Функции, които се ползват от повече от един екран, с цел избягване на повторения.



Фигура 2.1



Фигура 2.2



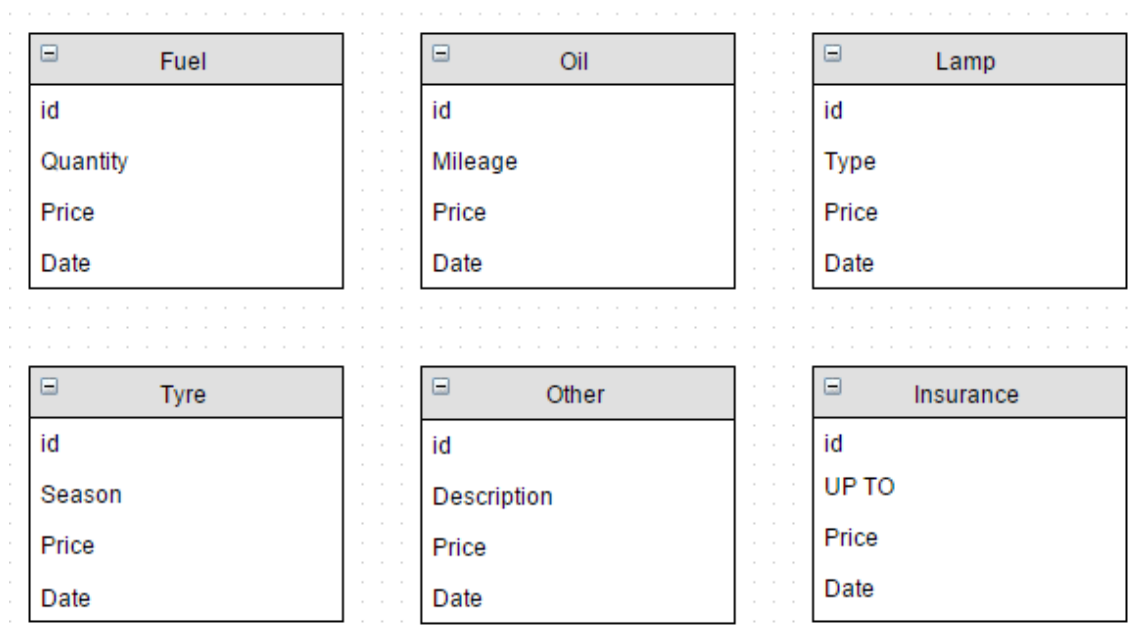
Фигура 2.3

2.4. Структура на базата

Базата има три основни версии.

2.4.1. Първа версия

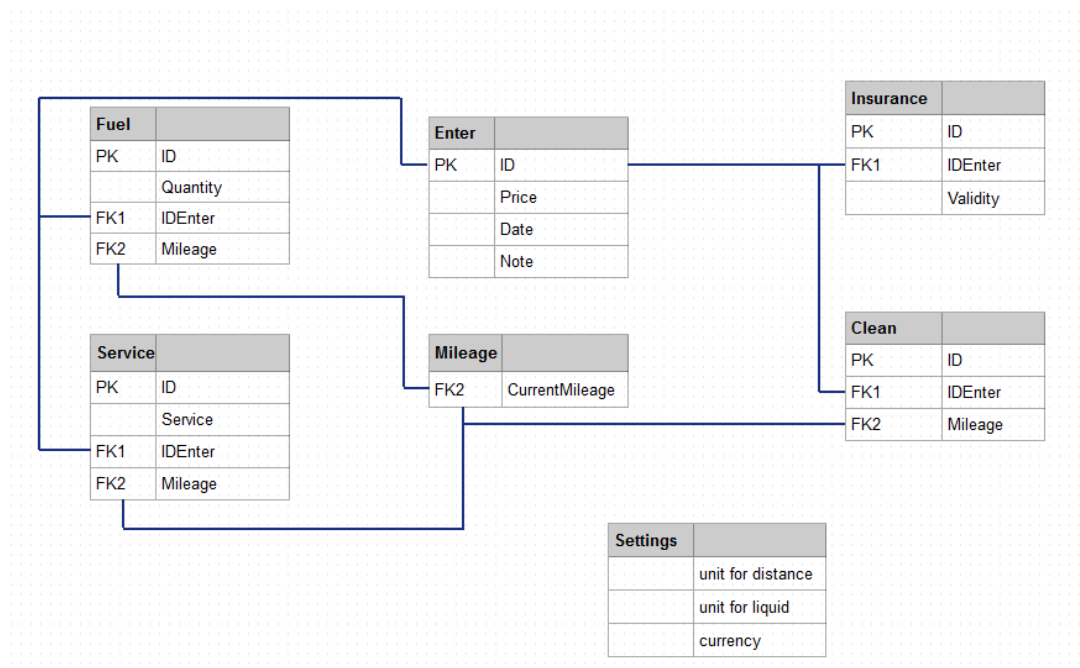
Според първоначалната идея, на приложението му бяха достатъчни шест независими таблици. По една таблица за гориво, масло, крушки, гуми, други и застраховки. Между таблиците нямаше никакви връзки. (Фиг. 2.4)



Фигура 2.4

2.4.2. Втора версия

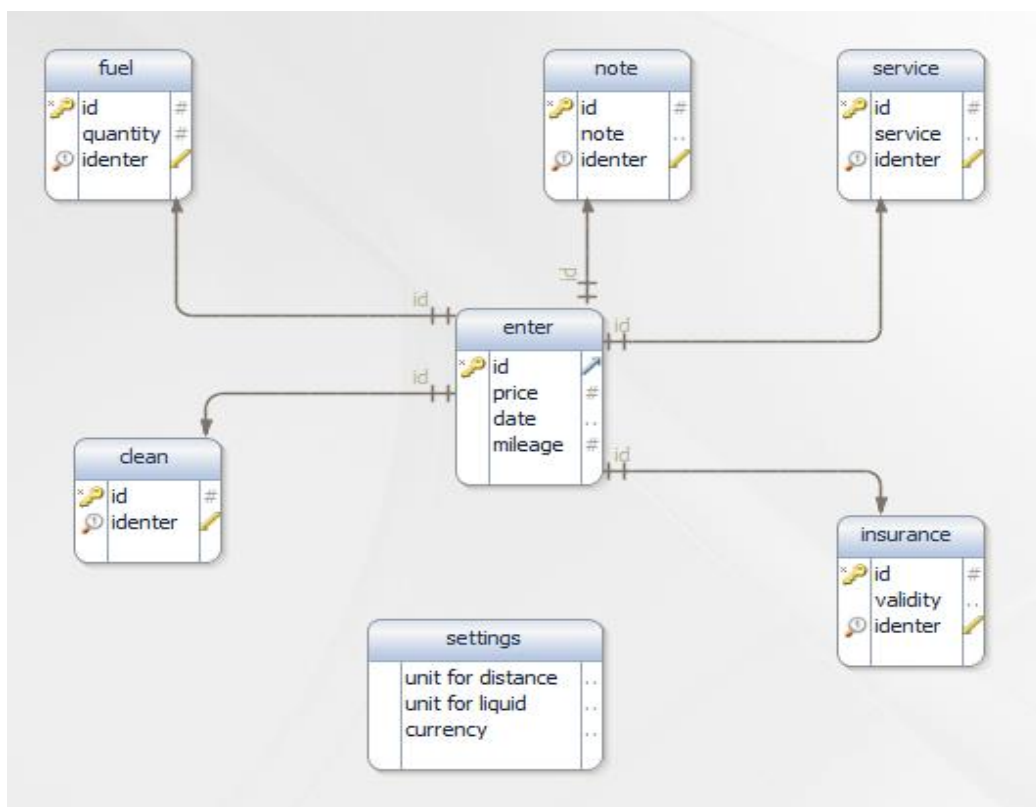
В процеса на работа се наложиха някои промени и оптимизации по базата. Някои от таблиците трябваше да се махнат, а на тяхно място се поставиха нови. Таблиците Lamp, Oil и Tyre, се обединиха в таблицата Service. За да бъде запазен външният вид на приложението трябваше да се добави един екран за почистване на автомобила и съответно трябваше да се създаде таблица за този екран. Добавена бе и таблица за настройките, в която се пази информация за избраните мерни единици и валута. Бе добавена и таблица Mileage, в която редът (беше само един) се презаписваше при въвеждане на километрите. Тази таблица трябваше да представлява текущия километраж, който се използва, за да не пише потребителят целия километраж, а само да променя последните цифри. Общите полета от всички таблици трябваше да се изнесат в нова таблица. Тази таблица бе наречена Enter, тъй като общите полета представляват информацията, която потребителят въвежда при добавяне на нов запис. Id-то на таблица Enter е първи вторичен ключ за таблиците Fuel, Insurance, Service, Clean. Стойността на реда от таблица Mileage беше втори вторичен ключ за същите таблици. (Фиг. 2.5)



Фигура 2.5

2.4.3. Трета версия

За по-добра функционалност на приложението се наложи още малко оптимизиране на базата. Тъй като Note стана единственото незадължително поле от таблицата Enter и фактът, че повечето записи не би трябвало да съдържат Note, полето Note е изнесено в отделна таблица, където всеки запис си има id, някакво съдържание. Като вторичен ключ id-то на таблица Enter, като запис се прави единствено, ако потребителят е написал някаква бележка, като по този начин се избягват множество празни полета в таблицата Enter. След това се прецени, че таблицата Mileage е напълно излишна и би било достатъчно, ако в таблицата Enter се добави нова колонка която е mileage. Съответно това доведе и до промяна на таблиците Fuel, Insurance, Service, Clean, като от всяка от се премахна колоната за Mileage, тъй като тази информация вече я има в таблицата Enter. Последната версия на базата изглежда по следния начин: (Фиг. 2.6)

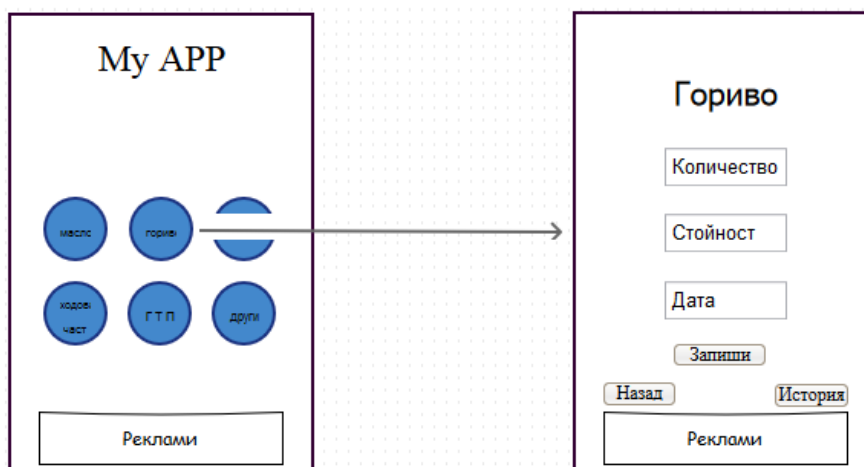


Фигура 2.6

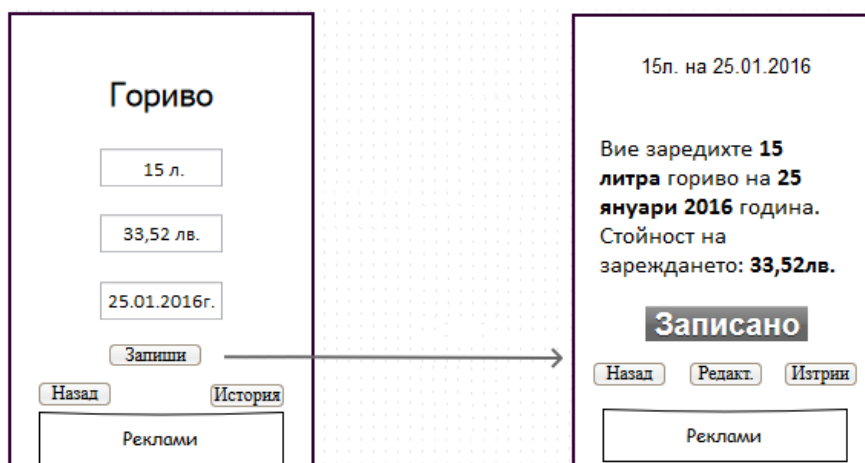
2.5. Wireframe-ове на приложението

Wireframe-овете показват архитектурата на приложението, как би трябвало да изглежда приложението и какви опции би трябвало да има приложението. Помагат с дизайна и спестяват време.

2.5.1. Стартиране на приложението и добавяне на запис (фиг. 2.7, фиг. 2.8):

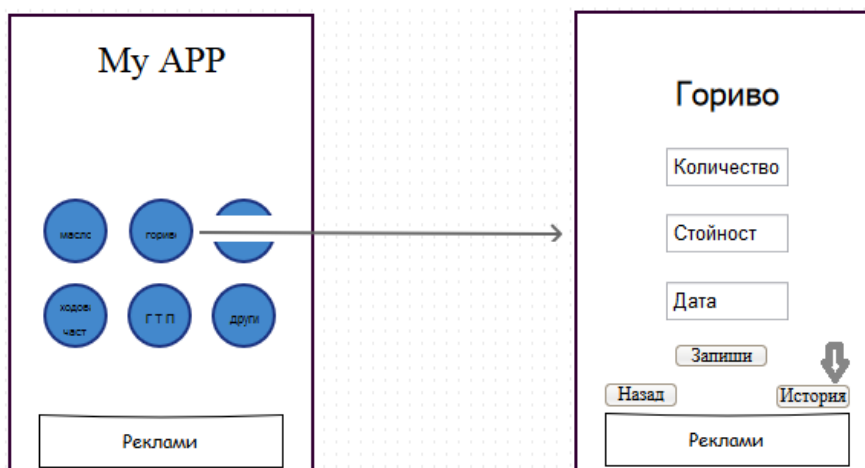


Фигура 2.7

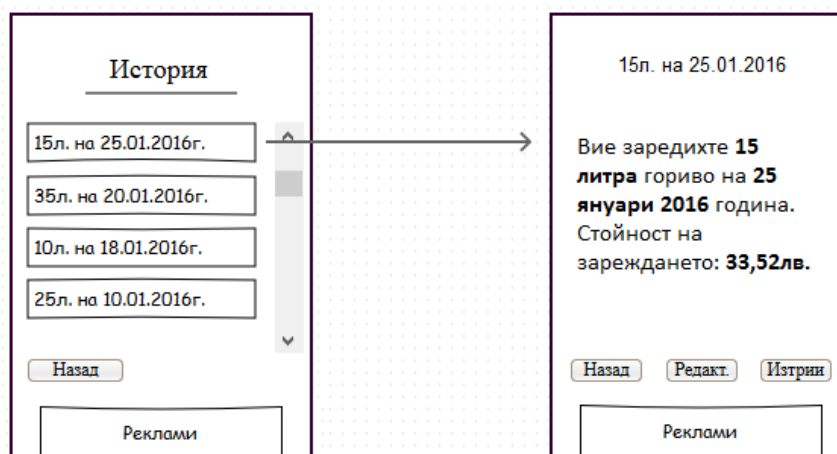


Фигура 2.8

2.5.2. Проверка на предишните записи (фиг. 2.9 , фиг. 2.10):

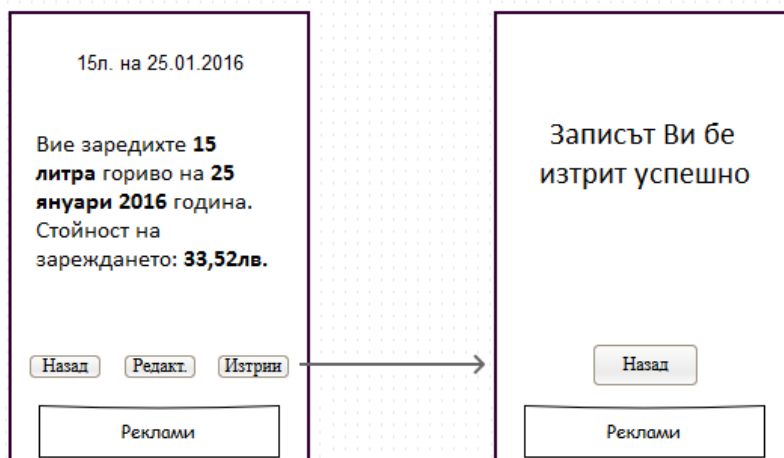


Фигура 2.9



Фигура 2.10

2.5.3. Изтриване на запис (фиг. 2.11):



Фигура 2.11

ТРЕТА ГЛАВА

Програмна реализация на приложението

3.1. Определяне етапите на разработка

След като беше ясна идеята за приложението, трябваше да се направи график за работа, както и да се определят етапите за разработка. Определиха се три главни етапа:

3.1.1. Първи етап – Основна функционалност

През този етап трябваше да се създаде основната функционалност на приложението. Възможно трябваше да бъде добавянето на различни дейности, извършени по колата, както и наличието на база, в която да се съхраняват. Също така трябваше да е възможно и изпращането на базата данни по някакъв начин на друго устройство.

3.1.2. Втори етап – Безплатна версия

Този етап представлява безплатната версия на приложението, която трябваше да отговоря на изискванията от заданието. При нея освен функционалността от първия етап трябваше да се добавят следните неща:

- Дизайн – съвременен и удобен за потребителя
- Реклами – добавяне на реклами, като източник на средства
- Различни статистики – създаване на различни статистики, използвайки данните, които е въвел потребителят

3.1.3. Трети етап – Платена версия

Този етап представлява платената (пълната) версия на приложението. При нея трябваше да се премахнат рекламите и да се добави още функционалност, като подсказки за застраховките, поддръжка на повече от един автомобил, „направи си сам“ – връзка на приложението с youtube.

3.2. Създаване на приложението

3.2.1. Основна функционалност

В началото с Android Studio много лесно се създадоха екраните Main, Fuel (фиг. 3.7.), ViewAll и ViewOne. MainActivity е началният екран на приложението. В началото се направи опит бутоните да се подредят в кръг (фиг. 3.6), но резултатът не беше сполучлив и поради тази причина се върнаха в първоначалния си вариант.

За работа на приложението с базата данни се създаде допълнителен клас DatabaseHelper.java (виж т. 2.3.13.).

От самото начало се реши бутоните да не са обикновени бутони с текст, а да са бутони – картинки. За да изглежда още по добре, при натискане на бутона, картинката трябваше да си сменя цвета. Това всъщност става, като за всеки бутон се добави нов .xml файл , в който се описват трите състояния на бутона - свободно, натиснато и задържан. Като всъщност тези действия не променят цвета на бутона, а зареждат нова картинка.

След това се замени EditText (виж т. 3.3.5.) полето за въвеждане на дата с фабричния DatePicker, но резултатът не беше сполучлив, затова се върна EditText-а.

ViewAll екрана се състоеше от едно TextView (виж т. 3.4.1.) - History, едно ListView (виж т. 3.4.2.) и два бутона – за експорт или импорт на съответната информация (фиг. 3.8.).

Вече в приложението беше възможно да се въведе запис за гориво, той да бъде записан в базата данни и да бъде видян. Също така при добавяне на последващ запис, той се поставя под предишните записи в ViewAll екрана.

След това трябваше тази информация да се записва във файл, който в последствие да бъде изпратен. Пробвах се няколко варианта за записване на файл, докато се откри, че за да се запише файл извън директорията на

програмата трябва в Manifest-а трябва да се добави разрешение за писане, което изглежда така (фрагмент 3.1):

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

Фрагмент 3.1

По този начин вече приложението успешно записваше съответната таблица от базата във файл, който се прашаше, чрез някои от възможните за даденото устройство методи.

Вече базата се праша, но все още потребителят може само да въвежда различна информация, без да я премахва. За изтриване на файл, се създаде функцията delete(String id), в DatabaseHelper-а. Тя изглежда така (фрагмент 3.2):

```
public void delete(String id) {  
    SQLiteDatabase db = this.getWritableDatabase();  
    try {  
        db.delete(TABLE_NAME, "ID = ?", new String[]{id});  
    }  
    catch (Exception e) {  
        e.printStackTrace();  
    }  
    finally {  
        db.close();  
    }  
}
```

Фрагмент 3.2

След изтриване на записа стана възможно приложението да препраща потребителя към историята с всички записи.

Когато потребителят иска да прегледа даден запис за горивото, то ViewOne екрана разделя количеството на СТОЙНОСТ и ЕДИНИЦА и при показване на екрана изглежда : Вие заредихте СТОЙНОСТ ЕДИНИЦА гориво на ДАТА. Това ви струваше СУМАТА или СУМАТА/СТОЙНОСТ за ЕДИНИЦА.

Пример: Вие заредихте 15 литра гориво на 12.01.2016г. Това ви струваше 30 лева или 2.00 лева за литър.

Пример: Вие заредихте 15 галона гориво на 12.01.2016г. Това ви струваше 30 евро или 2.00 евро за галон.

Тъй като бе имплементирана само едната категория - гориво, бе обмислено как да се направят по по-оптимизиран начин и другите категории, които са с подобна функционалност, но без повторения, доколкото е възможно. В началото се появиха две идеи. Първата беше за един екран и вътре с много и дълги if-ове, а втората беше с много почти еднакви екрани. Реши се, че е по - прегледно за всяка форма за запис да си има отделен екран. След това се създаде още един помощен клас – ActivityHelper (виж т. 2.3.14.), чиято цел е да държи в себе си еднаквите функции на различните екрани и всеки един екран да наследява този клас. Това помага да бъдат избегнати повторенията в отделните екрани, но също така позволява и всеки екран да бъде стилизиран по различен начин и/или да притежава индивидуалност.

Историята на приложението досега показваше последния запис най-отдолу, в последствие най-новия запис идва най-горе с командата показана на фрагмент 3.3:

```
Collections.reverse(List<?> list);
```

Фрагмент 3.3

Добави се първоначално още една таблица, за да се направи функциониращ още един екран, като се избра това да бъде LampActivity.

За да функционира правилно и за да може всеки екран да използва отделна таблица от базата данни се наложи да се променят и двата Helper класа, като новото е, че на всяка една от функциите трябва да се подаде като параметър и таблицата, с която да работи, а не както е досега, името на таблицата да бъде hardcoded-нато в Helper класа, който отговаря за базата.

Направи се след добавяне на запис да се отваря екрана със записаната информация.

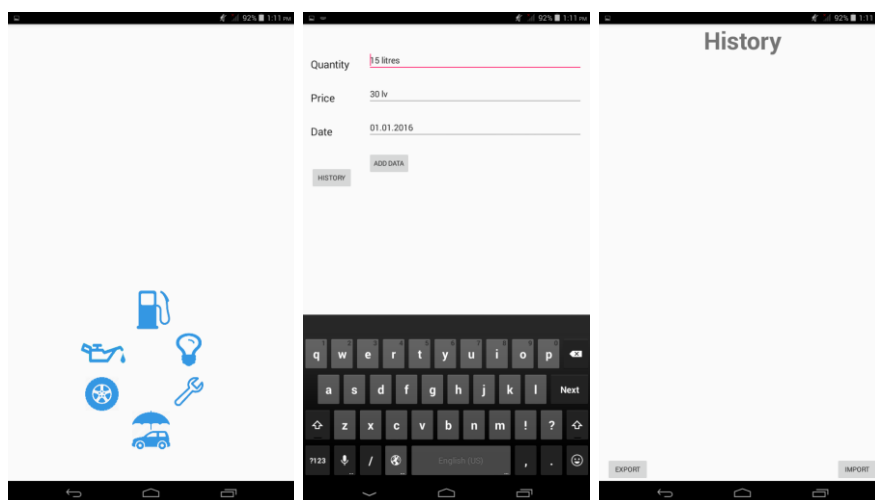
Оптимизираха се двата екрана, изнасяйки всичко, което е възможно в ActivityHelper класа.

Наложи се и промяна на самия метод за импортване на файл.

Относно външния вид, приложението на голям екран изглеждаше ужасно, а и не беше възможно да се местят елементи от Design опцията на AndroidStudio-то. За да се оправят тези две неща се наложи да се промени основния таг в xml-ите за дизайн в папката layout от CoordinatorLayout, на RelativeLayout.

След това дойде ред да се създадат и всички останали екрани за останалите бутони. Екраните бяха лесно създадени, но се наложи да се промени ViewOne екранът, за да може текстът да показва по правилен начин запазената в таблицата на екрана информация, като бе направен опит да се придаде на всеки екран по някаква индивидуалност.

Вече първия етап бе изпълнен. Приложението изглеждаше така (фиг. 3.6 , фиг. 3.7 , фиг. 3.8) :



Фигура 3.6

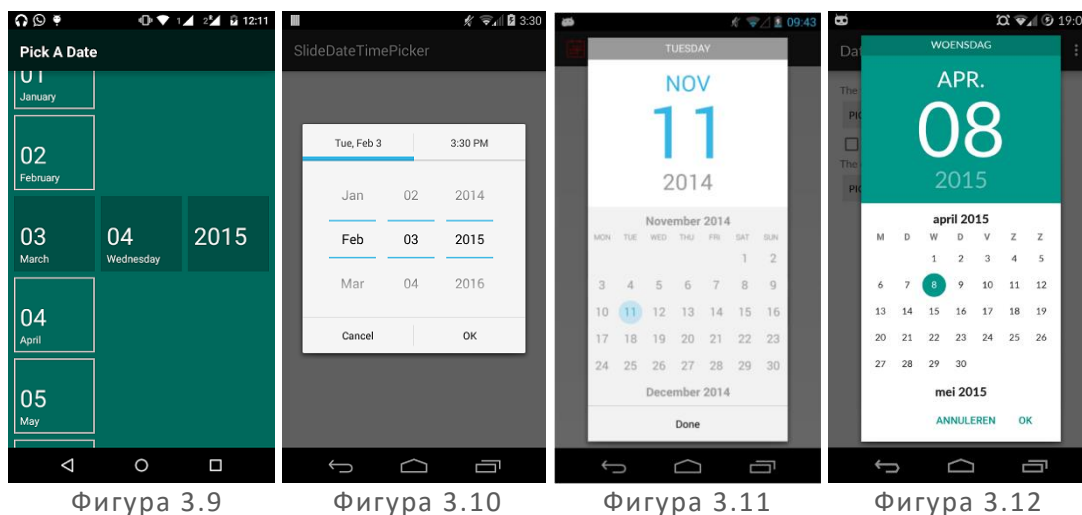
Фигура 3.7

Фигура 3.8

3.2.2. Безплатна версия

3.2.2.1 Календар

За начало на втория етап бе поставена за цел промяна на дизайна. Започна се със смяна на начина, по който потребителят въвежда датата, като от писане символ по символ се направи автоматично да взема днешната дата, като е възможно да бъде променена. Поради факта, че обикновените (вградените) DatePicker-и са с много лош външен вид, се реши да се използва някой с променен дизайн. След като се разгледаха няколко варианта – WinKal (Фиг. 3.9), SlideDatePicker (Фиг. 3.10), DatePickerAndroid 4.0+ (Фиг. 3.11) и MaterialDatePicker (Фиг. 3.12).



Фигура 3.9

Фигура 3.10

Фигура 3.11

Фигура 3.12

Избра се MaterialDatePicker, който прилича на DatePickerAndroid 4.0+, с това изключение, че дава опцията да се променят цветовете и има възможност да се променя дали първо да излиза календарът с месеца и дните или първо годината, после месецът и тн. Лицензът на MaterialDatePicker е Apache, което позволява неговата употреба.

3.2.2.2. Реклами

Трябваше да се добавят банери с реклами на всички екрани и затова се реши да се разгледа какви възможности има. Съществуваха варианти като Ad Revenue, Appnext, AdMob и AppFlyer, като официалният вариант предложен от

Google е AdMob. Реши се да се използва официалния вариант, първо защото вграждането му в приложението е лесно и второ, защото е най-добре оптимизирано за Android приложения. Добавянето става като в build.gradle файла се добави фрагмент 3.4:

```
compile 'com.google.android.gms:play-services-ads:8.3.0'
```

Фрагмент 3.4

и в отделните layout xml-ли се добави изгледа на реклама (фрагмент 3.5):

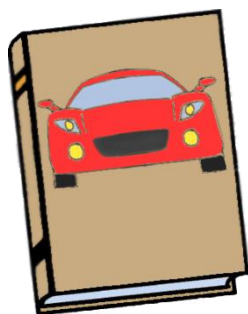
```
<com.google.android.gms.ads.AdView  
    android:id="@+id/adView"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignParentBottom="true"  
    ads:adSize="BANNER"  
    ads:adUnitId="@string/banner_ad_exp"  
</com.google.android.gms.ads.AdView>
```

Фрагмент 3.5

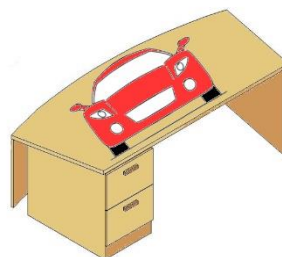
Където adSize е размерът на рекламата, който се избира при създаването и от сайта <https://apps.admob.com> , а adUnitId е уникален ключ за всяка реклама, който бива генериран при създаването ѝ. Това гарантира нейната уникалност, също така при създаването на рекламата може да се избере какво да е съдържанието (текст и/или картинка), дали да се обновява и на какво време, както и основно стилизиране, като фон, цвят на текста и тн.

3.2.2.3. Икона и лого

Като всяко едно приложение и на това, трябваше да се направи икона. Първата идея бе иконката да бъде бюро с кола на него. Тази идея дойде от името на приложението (CarDesk), защото приложението може да се интерпретира като „счетоводител“ за автомобил, но след това се промени на това, което е - сервизна книжка за автомобил. Направени бяха и двата варианта, за да се прецени кое би стояло по-добре.



Фигура 3.12



Фигура 3.13

Избран бе вариантът с книжката, защото е по-уместен.

След иконата трябваше да се измисли и лого. Реши се то да бъде един голям текст (името на приложението) и един малък (нещо което допълва името). Най-добре допълва името: your car manager. Тоест логото трябваше да бъде: CarDesk - your car manager. Пробвах се няколко варианта, но най-добре се получи (фиг. 3.14):



Фигура 3.14

Добави се и фон, който в началото беше различен за всеки един екран. За началния се избра на бананово жълто, а за другите - светло зелено, а за история и показване на един запис в светло синьо.

3.2.2.4. Toolbar

Добави се и ToolBar (виж т. 3.4.4.). ToolBar-а в layout файла изглежда така (фрагмент 3.6):

```
<android.support.v7.widget.Toolbar  
    android:id="@+id/toolbar"  
    android:layout_width="match_parent"  
    android:layout_height="?attr/actionBarSize"  
    android:background="@color/navigationBarOrange"  
    android:theme="@style/ThemeOverlay.AppCompat.ActionBar"/>
```

Фрагмент 3.6

Задават се размери и тема по подразбиране, а цвета на фона бе променен, като използваният цвят е с индекс 500 (както препоръчва Google) от основния цвят.

ToolBar-а трябва да се инициализира и в java файла на съответния екран. Това става посредством фрагмент 3.7.

```
Toolbar toolbar = (Toolbar)
findViewById(R.id.toolbar);
setSupportActionBar(toolbar);
```

Фрагмент 3.7

В него най-важен беше back бутонът. За да сработи обаче трябваше да се добавят и Parent връзки в манифеста. Провери се как се добавя back бутон и се намери (фрагмент 3.8):

```
getSupportActionBar().setDisplayHomeAsUpEnabled(true);
```

Фрагмент 3.8

По този начин на ToolBar-а се появява стрелката за назад, без да се задава връзка към картинка със стрелка.

Така вече toolbar-а имаше стрелка назад, но възникна проблем. След натискане на бутона приложението връщаше в главното меню, но това беше възможно само веднъж. След това приложението спираше. В опитите да бъде решен този проблем се пробва да се сменя фрагмент 3.8 с фрагмент 3.9

```
toolbar.setNavigationIcon(getResources().getDrawable(R.drawable
e.ic_action_back));
```

```
mActionBar.setNavigationOnClickListener(new
View.OnClickListener() {
@Override public void onClick(View v) {
startActivity(new Intent(MainActivity.this,
InsActivity.class));
}
});
```

Фрагмент 3.9

Тоест вместо фабричната стрелка се поставя такава, свалена от сайта на Google икони - <https://design.google.com/icons/> , и на нея да се задава да стартира предишното activity при натискане. И този вариант не сработи, затова се потърси какъв може да е проблемът. Имаше различни варианти, едни се отнасяха за манифеста, други за xml-а, но никой не беше полезен. Накрая се

откри, че ако има ToolBar и в него има различни бутони, трябва да се добави в activity-to (фрагмент 3.10):

```
@Override
public boolean onOptionsItemSelected(MenuItem menuItem) {
    onBackPressed();
    return true;
}
```

Фрагмент 3.10

След това в ActivityHelper класа се изведе инициализацията на NavigationBar-а и задаването на заглавие и back бутон. Наложил се промяна на действията, които трябва да се извършват при натискане на бутона. Функцията `onBackPressed()`, връщаше потребителя на предишния екран, което в някои случаи, не беше желателно, затова вместо `onBackPressed()`, се стартира новият екран, към който трябваше да се отиде. Това беше нужно, защото по този начин, ако има някаква въведена информация, то тя се изчиства и на потребителя се предоставя възможност да започне всичко отначало, без да се налага да трие или променя нещо.

3.2.2.5. Бутони

След това фабричните бутони (фиг. 3.15) се замениха с такива, които имат по-добър външен вид. Разгледах се няколко варианта на бутони, но най-добрите на външен вид бяха FancyButtons (фиг. 3.16). Техният лиценз е MIT, което позволява употреба им. Те разполагат с голям набор от атрибути, които могат да бъдат променяни, като: фон, цвят на текста, закръгляне в ъглите, дебелина на очертаващата линия. Добавянето им към проекта става чрез добавяне на `compile 'com.github.medyo:fancybuttons:1.5@aar'` в `build.gradle` файла. А в `layout` файловете чрез (фрагмент 3.11):

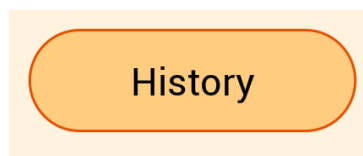
```
<mehdi.sakout.fancybuttons.FancyButton
    android:id="@+id/button_history"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingLeft="40dp"
    android:paddingRight="40dp"
    android:paddingTop="10dp"
    android:paddingBottom="10dp"
    ads:fb_borderColor="@color/accentOrange"
```

```
ads:fb_borderWidth="1dp"  
ads:fb_defaultColor="@color/navigationBarOrange"  
ads:fb_focusColor="@color/lightOrange"  
ads:fb_radius="30dp"  
ads:fb_text="@strings/history"  
ads:fb_textColor="@color/black"/>
```

Фрагмент 3.11



Фигура 3.15



Фигура 3.16

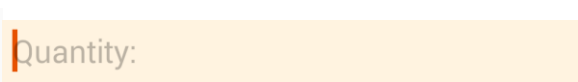
3.2.2.6. Полета за въвеждане на текст

След като вече бутоните имаха добър и съвременен външен вид, се обърна внимание и на полета за въвеждане на текст. До този момент и те бяха такива (Фиг. 3.17) каквито ги предоставя AndroidStudio. Отново се провериха няколко варианта, но бе избран този, който най-добре спазва идеята за MaterialDesign, която въвежда Google. Дори и самото име насочва към тази идея – MaterialEditText (Фиг. 3.18). Лицензът е Apache. Избрани бяха те, защото позволяват много голям процент персонализиране, фон с избран от нас цвят, подсказките преминават над полето за въвеждане (Фиг 3.19) и много други. И тук добавянето става чрез build.gradle файла:

```
compile 'com.rengwuxian.materialedittext:library:2.1.4'.
```



Фигура 3.17



Фигура 3.18

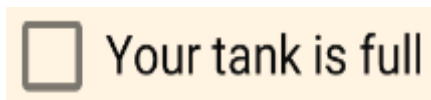


Фигура 3.19

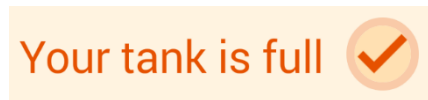
3.2.2.7. Маркиращо се поле

Във FuelActivity-то има един checkbox (виж т. 3.4.6.), който все още беше фабричният (Фиг. 3.20), и вече не се връзваше на новия дизайн. Провери се, дали не е възможно да се промени и checkbox-а с някой, който е по-съвременен. Отново от няколкото разгледани варианта се избра, този който

най-добре се връзва с останалата част. В този случай това беше AnimCheckBox (Фиг. 3.21). И неговият лиценз е Apache. Избран бе, заради приятното преминаване от маркирано към немаркирано състояние и обратното.



Фигура 3.20



Фигура 3.21

3.2.2.8. Падащи менюта

Единственото, което все още беше с оригиналния си дизайн, е Spinner-а (Виж т. 3.4.7.), в който стояха сервизните дейности на колата. Отново се прегледаха няколко възможни варианта и бе избран този, който най-добре съвпада с досегашния дизайн и това е MaterialSpinner (Фиг. 3.22 и Фиг. 3.23).

Добавянето в проекта отново става чрез едно `dependencies - compile`

```
'com.github.ganfra:material-spinner:1.1.1'
```

в `build.gradle` файла.



Фигура 3.22



Фигура 3.23

Дизайнът от към избор на различните елементи вече беше готов. Възниква следният проблем. Когато потребителят въвежда информация, рекламата излиза на нивото на полетата за попълване, което създава известно неудобство, защото е много лесно да се натисне реклама, вместо полето за попълване, което от своя страна не е много приятно за потребителя. Този проблем се оказва често срещан и съответно имаше много решения, но това което беше най-удачно е целият RelativeLayout от layout на екраните да се постави в едно scrollView. По този начин рекламата остава видима докато потребителят попълва дадена информация, но същевременно не му пречи да попълва информацията.

С така подбраните елементи първоначално се сглоби дизайнът на екраните за горивото (Фиг. 3.24), ремонтите (Фиг. 3.25), и това за застраховките (Фиг.

3.26). Към момента в приложението беше възможно да се въвежда дадена информация в съответните полета, които са с добър външен вид и улесняват потребителя.

Фигура 3.24

Фигура 3.25

Фигура 3.26

3.2.2.9. Екрана за застраховките

Трябваше да се обърне внимание на въвеждането на застраховките, тъй като там се налага да се въвеждат две дати – дата на сключване на застраховката и дата на изтичане на застраховката. Трябваша два DatePicker-а. Дата на валидност на застраховката, която не трябва да е по-ранна от тази на сключване, за това се реши, че ще е най-добре при избор на валидност, да се взема датата на сключване на застраховката и да се проверява дали е по-ранна от тази, която избира потребителят, ако е по-ранна, автоматично се присвоява на дата за сключване на застраховката, ако не е се задава на полето за валидност. След това се откри проблем при сравняването на датите. В някои определени случаи, не можеше да сравни правилно двете дати. За това се потърси друг начин, който би могъл да се ползва за сравнение. Откри се, че може датата от бутона да се раздели на отделни елементи и да се сравнява всеки от тях. Датата имаше вид: 16.12.2015, следователно разделителя е "." и се разделя чрез `.split(".")`. Обаче приложението започна да дава проблеми. След няколко опита се установи, че грешката идва от разделянето, и то не е

.split("."), а е .split("\\. "). Самото сравнение става по този начин (фрагмент 3.12):

```
@Override
public void onDateSet(DatePickerDialog view, int year, int
monthOfYear, int dayOfMonth) {
    String date = dayOfMonth + "." + ++monthOfYear + "." +
year;
    int buttonYear = Integer.parseInt
(dateButton.getText().toString().split("\\. ")[2]);
    int buttonMonth = Integer.parseInt
(dateButton.getText().toString().split("\\. ")[1]);
    int buttonDay = Integer.parseInt
(dateButton.getText().toString().split("\\. ")[0]);
    if (year > buttonYear) {
        editValidity.setText(date);
    } else if ((monthOfYear >
buttonMonth) && (year == buttonYear)) {
        editValidity.setText(date);
    } else if ((dayOfMonth >
buttonDay) && (monthOfYear == buttonMonth) && (year == buttonYear)) {
        editValidity.setText(date);
    } else {
        dateButton.setText(date);
    }
}
```

Фрагмент 3.12

Идеята е следната - в променливите започващи с button се пази датата на сключване на застраховката, а в променливите year, monthOfYear и dayOfMonth е новата дата, която избира потребителят. Проверяваме първо годината, ако тя е по-голяма значи потребителят въвежда валидност, после месеца и годината, накрая деня, месеца и годината, ако и денят е по-малък, значи потребителят въвежда дата на сключване и задаваме тази дата като дата на сключване.

Реши се, че lamp и tyre таблиците са напълно излишни, както и техните activity-та. Цялата тяхна функционалност се изпълняваше и от other activity-то и таблицата other.

Също така основно за приложението са статистиките, поради което в таблицата за гориво трябваше да се добави още една колонка, в която да се пази информация дали е избрано, че потребителят е заредил "до горе". Това се

налага за точното пресмятане на разхода за изминатото разстояние. Също така трябваше в почти всички останали таблици да се добави и на колко километра е извършена дадена дейност. Това също ще е от полза и за изписването на текущия километраж в началната страница.


Всеки един екран може да променя текущия километраж само в положителна посока, чрез презаписване на стойността - `currentMIleage` в `Other` таблицата (Фиг. 2.5), а ако потребителят е допуснал някаква грешка, то той може да "върне" текущия километраж само от полето в началната страница.

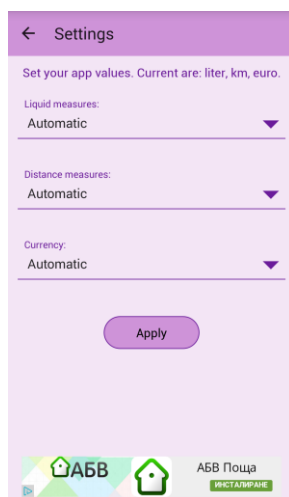
Прецени се, че смяната на маслото също може да бъде въведена като опция на екрана `Other` и таблицата `Other`. По този начин приложението обаче ставаше с пет бутона на началния екран - гориво, застраховки, сервизация, статистики и настройки, което не изглежда добре. Шестте бутона стояха много добре, което наложи да се обмисли какво още прави един собственик на автомобил, различно от сипване на гориво, сключване на застраховки и ремонтите по него. Всеки автомобил се нуждае от периодично почистване, което наложи смяна името на таблицата `Oil` на `Clean`. Почистването се превърна в шестия бутон на началната страница. По този начин, добрият външен вид си оставаше непокътнат.

3.2.2.10. Настройки

Тъй като беше дразнещо всеки път потребителят да пише валутата и мерната единица, тази информация се изведе в отделна таблица и отделен екран.

За да се създаде, `TyreActivity`-то се преименува на `SettingsActivity`, както и името на едната таблица в базата. Дизайнът трябва да бъде опростен, затова се избраха три падащи менюта и един бутон за запис. Едното за мерната единица за течност, другото за мерната единица за разстояние и последното за валутата. Използваха се три `MaterialSpinner`-а, заради добрия им външен вид. За цветово оформление се избра цвят, различен от този за останалите екрани, защото ако потребителят без да иска натисне бутона на `SettingsActivity`-то да му

направи впечатление, че не се намира там, където трябва. За иконка се използва , която е от <https://www.google.com/design/icons/> и е безплатна. За запис на информацията в базата се добави функцията `updateSettings` от `DatabaseHelper` класа, която да се вика при натискане на бутона. Обаче възникна следният проблем. Ако потребителят инсталира за пръв път програмата, то тази таблица е празна и приложението няма да работи. Трябваше в главния екран на приложението (`MainActivity`) да се добави една проверка, за това дали тази таблица е празна, и ако е така, да се изпълни `enterSettings` със стойности по подразбиране (според езика и местоположението на телефона). И така настройките бяха готови (Фиг. 3.27).



Фигура 3.27

Оказа се, че би било най-добре ако всички екрани на приложението имат еднакъв фон, а с различен да бъде само този за настройките. След направено проучване, като основен цвят за приложението стана оранжевото, а за настройките – лилаво.

3.2.2.11. Ремонтни дейности

Името на `Other` екрана и таблица се промени на `Service`, защото наименованието е по-точно. След това трябваше да се направят напълно функциониращи останалите екрани.

Започна се екрана за поддръжката. За да функционира правилно трябваше да се промени таблицата отговаряща за нея. Същото беше и при таблиците за

останалите activity-та. Поради факта, че таблиците на горивото и поддръжката имат една колона повече, се добави метод, който отговаря за записването на данните от тези две таблици, който е подобен на досега съществуващия. Наименуван бе insertDataFS , като FS идва от Fuel and Service. Наложиха се малки промени по activity-та, които не бяха нещо особено (промяна на методите за взимане на данни и записването им в базата). Наложиха се и промяна в метода за вкарване на данни и за останалите две activity-та. За показване на информацията от записите на гориво и ремонти се промени и ViewOne екрана на ViewFS, тъй като в него се взима повече информация от базата. Към този момент съществуваше идеята, че ще трябва да се създаде още един екран, за да показва информация от останалите две екрана. След това се наложи да се направи get() функция, която да взема стойностите от таблицата за настройките.

3.2.2.12. Статистики

Започна се activity-то за статистиките. Кръстих го SearchActivity (Фиг. 3.28). Иконката е от тези, които Google предоставя. Началният екран е с шест бутона (Фиг. 3.29). Първа стъпка е дизайна, който трябваше да е опростен - два spinner-а и две полета за текст, едното за парите другото за разхода. Първият spinner е за това каква информация да бъде показана. Вторият е за какъв период да бъде показваната информация. Едното поле за текст показва колко пари са похарчени за колата през месеца, а второто - какъв е разходът, но то се появява само ако е избрана таблицата за гориво. Ако от spinner-а за информацията бъде избрано горивото и от spinner-а за период бъде избран „Точно“, то показва статистиката за реално използва стойностите между две зареждания на резервоара „до горе“. След това се реши, че ще е добре ако добавя и един бутон който да е за обновяване. Започна се с експерименти относно запитвания по таблиците и тн. Запитванията в началото се правеха с query(), но в последствие с rawQuery().

Тъй като статистиките използват датите, които е въвел потребителят, се наложи тези дати да бъдат записани по определен начин, а не както до сега.

Т.е. дата от „18.05.2015“ трябваше да стане „2015-05-18“, за да са коректни запитванията относно базата.

Най – интересно бе запитването до базата, което показва колко гориво е използвал потребителя между две зареждания. Сложността идва от факта, че ние искаме сумата на горивата между два записа за гориво без количеството на първия запис. Информацията която имаме след като съберем таблиците Fuel и Note изглежда така (фрагмент 3.13)

id	service	enterid	note
2	50	25	Your tank was full up.
3	20	26	NULL
4	20	35	Your tank was full up.
8	30	36	NULL
9	15	37	NULL
10	20	42	Your tank was full up.

Фрагмент 3.13

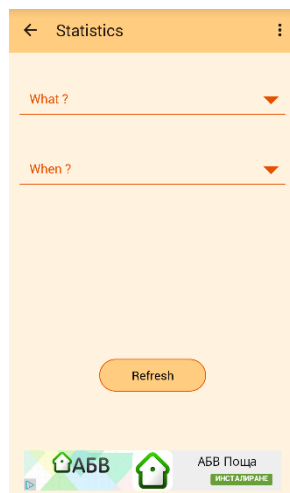
Заявката която показваше точното гориво изглежда така (фрагмент 3.14):

```
SELECT SUM(fq.service) AS total
FROM
(SELECT fq2.*,
(SELECT COUNT(*)
FROM (SELECT f.id, f.service, f.enterid, n.note
      FROM fuel_table f
      LEFT JOIN note_table n
      ON f.enterid = n.enterid) fq1
WHERE fq1.note = 'Yes' AND fq1.id >= fq2.id) AS numNotesAhead
FROM
(SELECT f.id, f.service, f.enterid, n.note
      FROM fuel_table f
      LEFT JOIN note_table n
      ON f.enterid = n.enterid) fq2) fq
WHERE numNotesAhead = 1
```

Фрагмент 3.14

За различните периоди от време, тази заявка се поставя в цикъл.

Дойде време и да се оформи екрана-то за почистване на автомобила. Той замества този за смяна на маслото, което отиде в сервизните дейности. Състои се от - дата, цена, километри на които е извършено почистването и някаква бележка (коя автомивка). Отново се използваха MaterialDesign елементите и иконката от Google за CarWash, която е и в GoogleMaps.



Фигура 3.28



Фигура 3.29

3.2.2.13. ToolBar

За да бъде приложението в крак с тенденциите бутоната, който беше за настройките, трябваше да отиде в toolbar-а. В началото се предполагаше, че това става с някакъв set-ър, както е за заглавието и стрелката за назад, но се оказва, че за да се появи елемент за меню (настройки) се налага да се създаде файл, в папката menu, намираща се в папката res. Този файл изглежда така (фрагмент 3.15):

```
<menu
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto">

    <item android:id="@+id/action_settings"
        android:title="@string/action_settings"
        app:showAsAction="never"/>
</menu>
```

Фрагмент 3.15

След като е създаден файлът трябва да се Overrid-не функцията `onCreateOptionsMenu` и в нея да се посочи създаденият по - горе файл, който да се използва за меню. За да „разбере“ приложението, че е натиснат бутонът за меню, а не стрелката за назад, трябваше да се направят промени и в Overrid-натия вече метод `onOptionsItemSelected`, като от Android препоръчват това да бъде направено с switch цикъл, в който да се switch-ва по id на натиснатия елемент, следователно ако id-то е `action_settings`, изпълняваме опциите, които трябва да се изпълнят след натискане на бутоната

за настройки, а в default-а стои това, което трябва да се случи при натискане на стрелката за назад.

Вече има меню, което стои в горния десен ъгъл(Фиг. 3.30).

След като вече от началния екран се премахна бутона за настройките, бутоните станаха пет и отново идеята за шестте бутона се разпадна. След обмисляне на нуждите на един шофьор и възможностите едно приложение, се разбра че много малка част от шофьорите знаят номер на пътна помощ. Затова се създаде бутон, който с едно натискане да ви предложи да набере номер на пътна помощ. Този бутон е просто надпис SOS. Това е и вариант за допълнителни доходи от приложението, чрез договор с някоя от фирмите за пътна помощ. Обаче се появи проблем, ако потребителят без да иска натисне бутона, то не би било приятно да излезе от приложението и да зареди номер за набиране, затова се създаде нов екран, в който да има бутони за пътна помощ в четирите континента, към които се цели приложението, тъй като във всеки континент има пътна помощ, която отговаря за целия континент. При създаването на бутоните в SOSActivity-то трябваше да се добави в манифеста фрагмент 3.16:

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

Фрагмент 3.16

, което позволява на приложението да използва набирането на телефонен номер през него.

Някой екрани изискваха да се въведе текущият километраж на автомобила, което не беше трудно и задължително, но пък беше скучно всеки път да въвеждаш еднакви цифри (поне първите три - 192***) на автомобила, затова се направи така, че от началната страница на приложението да може да се задава текущият километраж на автомобила, а от другите екрани, само да се променя в положителна посока. Затова отговаря една таблица от базата, в която само се презаписва информацията, а след инсталиране на приложението стойността на километража в тази таблица се задава да е 0. Ако потребителят

допусне грешка и въведе километри, които са по малко от предишните приложението автоматично взема последно въведените, а пък ако въведе километри които са по - големи от текущите, той може да ги промени от началната страница (Фиг.3.30).



Фигура 3.30

Преминаването към последната версия на базата данни(Фиг. 2.6) доведе до някои промени. Промените бяха най-вече свързани с DatabaseHelper класа, но се наложиха промени и по самите екрани. На някои от тях се наложи да се добави по един ред, на други да се променят методите, които използват. Голяма промяна претърпя и екрана за статистиките. Тъй като повечето негови методи използват, методи които трябва да бъдат променени. Заедно с това се промени и информацията, която си предават екраните един на друг. Вместо цялата информация за един запис, то те да предават само неговото Id и името на таблицата.

Тъй като беше променен начина, по който се записва информацията в базата се наложи да се добави функция, която върши обратното на това, което до сега е вършила dateToBase - dateToShow (реално просто сменя реда на записа на датата и „-“ ги заменя с „.“).

3.2.2.14. Редактиране на запис

Дойде ред да се направи и екрана за редактиране на въведената информация. Той представлява пет EditText-a, които са MaterialDesign. Тези пет EditText-a

напълно покриват информацията, въведена за всички категории, като ако е избрано да се редактира запис от тези за почистването на автомобила, автоматично единият EditText се скрива, а на потребителя е предоставена възможността да промени всяка една от въведените стойности, без да има празни полета. И тук дата се избира чрез календар.

За да функционира правилно екрана, се наложи да добавя и няколко метода за обновяване на информацията.

3.2.2.15. Изпращане/получаване на базата

До този момент, ако потребителят иска да прати дадена информация от таблицата, то той трябва праща една по една всяка таблица от ViewAllActivity-то. Това обаче беше много усложнено от новата база, а и ако потребителят би желал да прати всичката информация, то той трябва да прати общо 4 файла, което е излишно. Решението на този проблем е създаването на нов екран (Фиг. 3.31), който да отговаря за изпращане и импортирането на базата данни. Този екран се състои от четири checkbox-а, чрез които потребителят да избира каква информация да изпрати. Ако потребителят е маркирал някое от полетата задължително се изпращат таблиците enter и note, като към тях се добавя и избора на потребителя. Информацията се форматира като json, който изглежда така (фрагмент 3.17):

```
{
  "fuel_table": [
    {"service": "Value", "id": "Value", "enterId": "Value"},
    {"service": "Value", "id": "Value", "enterId": "Value"} ],
  "ins_table": [
    {"service": "Value", "id": "Value", "enterId": "Value"},
    {"service": "Value", "id": "Value", "enterId": "Value"} ],
  "enter_table": [
    {"id": "Value", "price": "Value", "date": "Value", "mileage": "Value"},
    {"service": "Value", "price": "Value", "date": "Value", "mileage": "Value"} ],
  "note_table": [
```

```
{“enterId”: “Value”, “note”: “Value”},  
{“enterId”: “Value”, “note”: “Value”}],  
}
```

Фрагмент 3.17

Кодът чрез който се извлича информацията и се създава файла е (фрагмент 3.18):

```
@TargetApi (Build.VERSION_CODES.KITKAT)  
public void sendData(DatabaseHelper myDb, String tableName) throws  
JSONException {  
    Cursor res;  
    if ("enter_table".equals(tableName)) {  
        res = myDb.getAllEnter();  
        finalObject.put("enter_table", textEnterTable(res));  
        res = myDb.getAllNotes();  
        finalObject.put("note_table", textNoteTable(res));  
    } else if ("clean_table".equals(tableName)) {  
        res = myDb.getAllClean();  
        if (res.getCount() != 0) {  
            finalObject.put("clean_table", textCleanTable(res));  
        }  
    } else {  
        res = myDb.getAllTables(tableName);  
        if (res.getCount() != 0) {  
            finalObject.put(tableName, textTable(res));  
        }  
    }  
}  
  
public JSONArray textEnterTable(Cursor res) {  
    JSONObject row;  
    JSONArray tableAsArray = new JSONArray();  
    while (res.moveToNext()) {  
        row = new JSONObject();  
        try {  
            row.put("id", res.getString(0));  
            row.put("price", res.getString(1));  
            row.put("date", res.getString(2));  
            row.put("mileage", res.getString(3));  
        } catch (JSONException e) {  
            e.printStackTrace();  
        }  
        tableAsArray.put(row);  
    }  
    return tableAsArray;  
}  
  
public JSONArray textTable(Cursor res) {  
    JSONObject row;  
    JSONArray tableAsArray = new JSONArray();  
    while (res.moveToNext()) {
```

```
        row = new JSONObject();
        try {
            row.put("id", res.getString(0));
            row.put("service", res.getString(1));
            row.put("enterId", res.getString(2));
        } catch (JSONException e) {
            e.printStackTrace();
        }
        tableAsArray.put(row);
    }
    return tableAsArray;
}

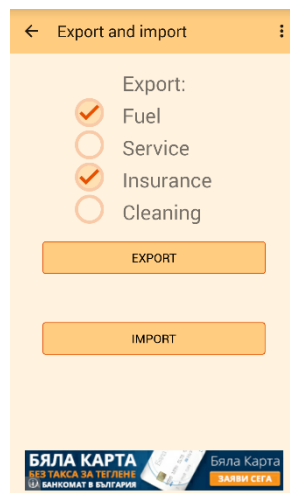
@TargetApi(Build.VERSION_CODES.KITKAT)
public void generateFile(String FilePath, String sBody){
    try (Writer writer = new BufferedWriter(new OutputStreamWriter(
        new FileOutputStream(FilePath), "utf-8"))) {
        writer.write(sBody);
        Toast.makeText(this, "Saved", Toast.LENGTH_SHORT).show();
    } catch (FileNotFoundException e) {
        Toast.makeText(this, "FileNotFoundException",
            Toast.LENGTH_SHORT).show();
    } catch (UnsupportedEncodingException e) {
        Toast.makeText(this, "UnsupportedEncodingException",
            Toast.LENGTH_SHORT).show();
    } catch (IOException e) {
        Toast.makeText(this, "IOException",
            Toast.LENGTH_SHORT).show();
    }
    sendFile();
}
```

Фрагмент 3.18

Този файл се запазва като base.txt. И се изпраща през някои от възможните за телефона начини, а от другото устройство трябва да бъде import-нат.

Новата конструкция на файла наложи да се промени и методът за импортване на данните в приложението.

Най-уместно беше до това activity да се стига от менюто в горния десен ъгъл, където бяха настройките. За да бъде възможно това, трябваше да се промени файл menu.xml и да се добави още една възможност в switch-а. Така вече менюто съдържаше два елемента.



Фигура 3.31

3.3. Основни компоненти

3.3.1. TextView – Показва на потребителя даден текст с възможност за индиректна редакция

3.3.2. ListView – Показва списък от елементи с възможност да се скролва между тях

3.3.3. DatePicker – Елемент, чрез който избираме дадена дата

3.3.4. ToolBar – Лента с инструменти, която се препоръчва от Google да стои в горната част на екрана

3.3.5. EditText – Елемент, който позволява въвеждане и редакция на текст

3.3.6. CheckBox – Елемент, който позволява на потребителя да избере една или повече опции от даден списък

3.3.7. Spinner – Елемент, който предлага бърз начин за избор на една стойност от даден списък

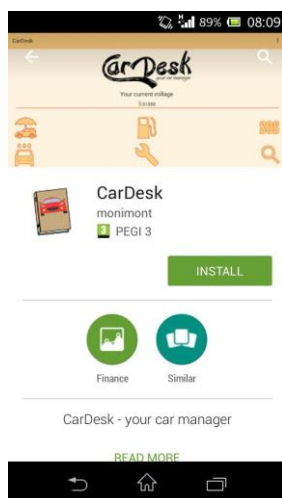
ЧЕТВЪРТА ГЛАВА

Ръководство на потребителя

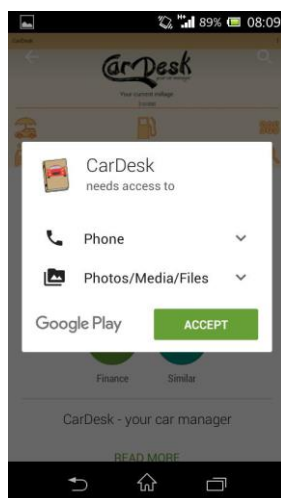
4.1. Инсталация

4.1.1. Инсталация от телефон

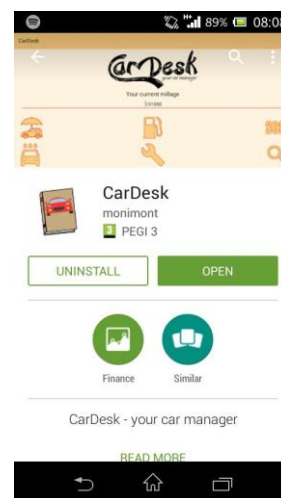
Стартираме приложението GooglePlay, което идва заедно с операционната система Android. В търсачката въвеждаме името на приложението – CarDesk. От резултатите избираме приложението и натискаме бутона „Инсталиране“ (фиг. 4.1). Прочитаме и ако сме съгласни приемаме изискванията („Permissions“) на приложението (фиг. 4.2). Под името на приложението се появява прозорец, който ни уведомява за процеса на инсталацията. След като всичко е готово, избираме „Отвори“, с което процесът е приключен (фиг. 4.3).



Фигура 4.1



Фигура 4.2



Фигура 4.3

4.1.2. Инсталация през браузър

Стартираме браузъра, който ползваме. В полето за въвеждане на сайт пишем: <https://play.google.com/> . В горната част на сайта има търсачка, в която въвеждаме "CarDesk" и избираме „Търсене“. От кутийките с приложения избираме "CarDesk" . След това избираме "Install". И приложението се сваля на устройствата, които ползват акаунта, който е въведен и на компютъра и на устройствата.

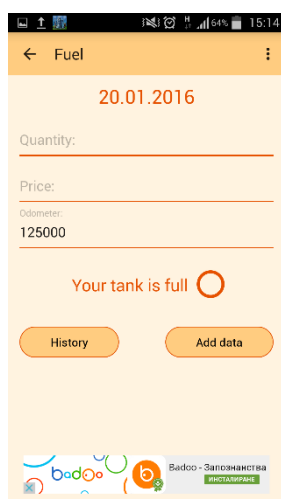
Внимание! Приложението няма да може да се свали, ако устройството, на което се опитвате да го инсталирате няма интернет достъп. При липса на такъв приложението ще се свали едва, когато на устройството се осигури достъп до интернет.

4.2. Добавяне на запис

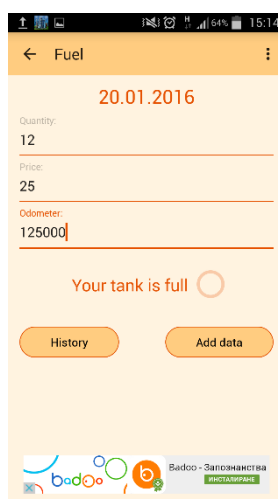
Стартираме приложението и избираме опция от началния екран.

4.2.1. Запис за гориво

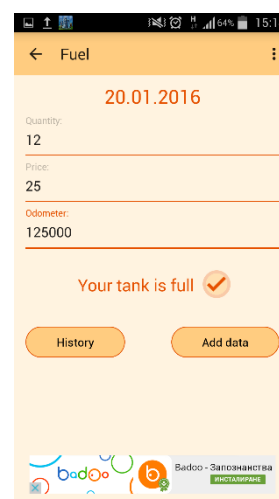
Когато сме избрали, че ще въвеждаме гориво приложението отваря нов екран (Фиг. 4.4), в който потребителят има възможност да въведе количество, цена, километри на автомобила (Фиг. 4.5), и/или да промени датата, и/или да отбележи дали е заредил „до горе“ (Фиг. 4.6).



Фигура 4.4



Фигура 4.5



Фигура 4.6

4.2.2. Запис за ремонт

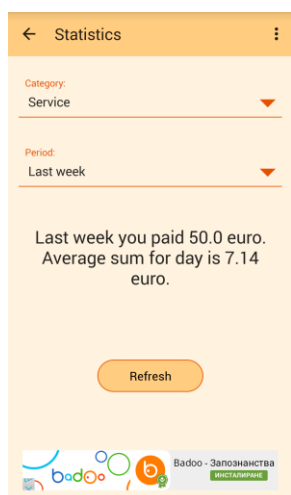
Добавянето на запис за ремонтна дейност по автомобила е аналогично на това за горивото. Разликата е в това, че потребителят избира дейността от падащото меню.

4.2.3. Запис за застраховки

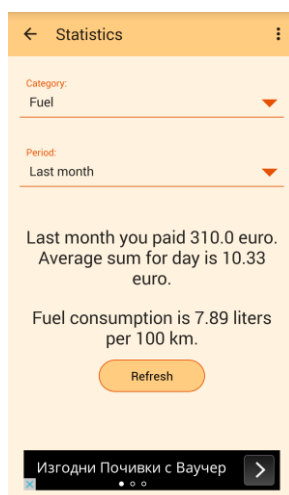
Добавянето на запис за направена застраховка е подобно на останалите, с разликата, че потребителят въвежда валидност.

4.3. Показване на различни статистики

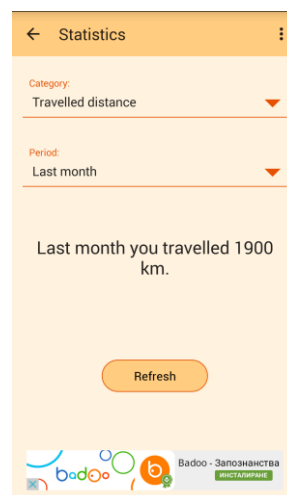
За показване на различни статистики, потребителят трябва да избере от главния екран „лупата“, което ще стартира нов екран с две падащи менюта. Едно за категория и едно за периода. За да функционира правилно е желателно да има въведени повече данни в приложението. На фигура 4.7 е показана статистика за изминатото разстояние за последния месец, на фигура 4.8 е показано колко пари сме дали за ремонтни дейности последната седмица и на фигура 4.9 е показано колко сме похарчили за гориво и какъв е бил разходът за последния месец.



Фигура 4.7



Фигура 4.8



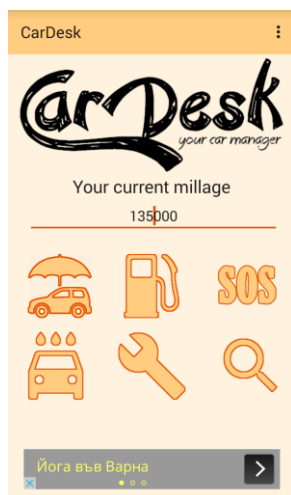
Фигура 4.9

4.4. Редакция на объркан километраж

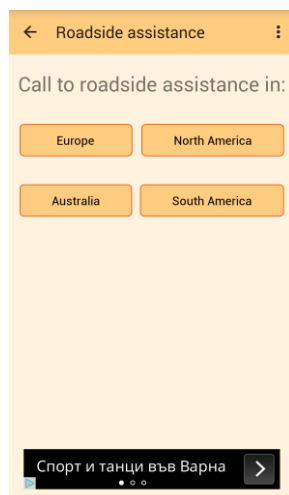
Ако потребителят е объркал стойността на километража в положителна посока (въвел е по-голям от действителния), то той може да го оправи, като направи нов запис с по-големия километраж и след това редактира само километража.

4.5. Набиране на номер за пътна помощ

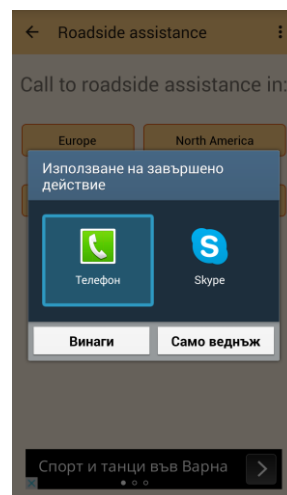
За повикване на пътна помощ, от началния екран избираме SOS (Фиг. 4.10). От новоотворения екран избираме континента (Фиг. 4.11), в който се намираме и метода за набиране (Фиг. 4.12).



Фигура 4.10



Фигура 4.11

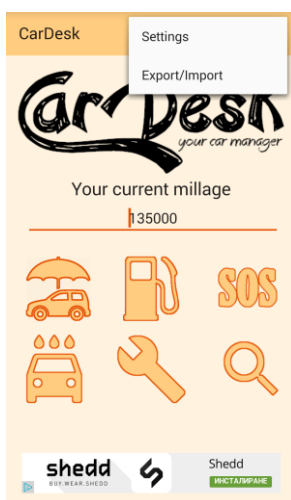


Фигура 4.12

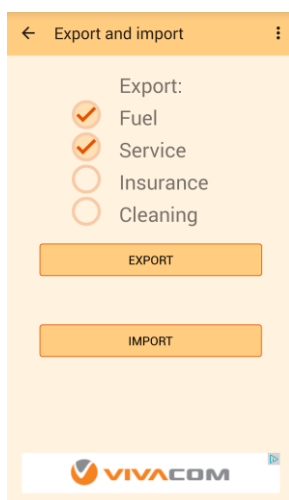
4.6. Изпращане/импортиране на базата

За изпращане на базата е необходимо от менюто в горния десен ъгъл да изберем „Export/Import“ (Фиг. 4.13). Маркираме коя информация искаме да изпратим (Фиг. 4.14) и натискаме бутона „EXPORT“. Избираме чрез кой метод да изпратим файла (Фиг. 4.15) и така процесът за изпращане е приключен.

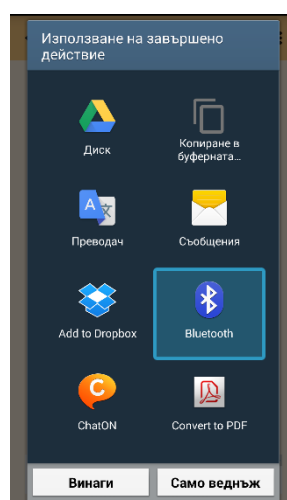
За импортиране на базата натискаме бутона „IMPORT“ и избираме генерирания/получения файл.



Фигура 4.13



Фигура 4.14



Фигура 4.15

ЗАКЛЮЧЕНИЕ

Постижения в дипломната работа

В дипломната работа успешно е разработено приложение, което представлява сервизна книжка. Разработеното приложение покрива всички изисквания на дипломната работа, като неговата функционалност е разширена. Софтуерният продукт е широко приложим от всеки собственик и/или водач на автомобил, като той вече е и достъпен на пазара за Android приложения – Google Play.

Бъдещо развитие

Бъдещото развитие включва изработване на платената версия на приложението (Виж т. 3.1.3.). По този начин ще се увеличи броят на потребителите, които биха го ползвали.

Използвана литература

1. Статистически проучвания за пазара на мобилни устройства - <http://www.idc.com/prodserv/smartphone-os-market-share.jsp>
2. Препоръки за дизайн - <https://design.google.com/>
3. Информация за елементи и тяхната употреба
<https://www.google.com/design/spec/components/bottom-sheets.html>
4. Android - <http://developer.android.com/index.html>
5. Java - <http://code.tutsplus.com/series/learn-java-for-android-development--mobile-22888>
6. Репозитори - <https://github.com/SShopkin/DiplomApp>