

**ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ**  
**към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

## **ДИПЛОМНА РАБОТА**

Тема: Мобилно приложение за автомобилна сервизна книжка

Дипломант:

*Валентин Валентинов Стоянов*

Научен ръководител:

*маг. инж. Кирил Митов*

СОФИЯ

2017



# СЪДЪРЖАНИЕ

<b>Увод</b>	<b>6</b>
<b>ПЪРВА ГЛАВА</b>	<b>7</b>
<b>Проучвателна част. Преглед на съществуващи подобни приложения и преглед на известните развойни средства и среди</b>	<b>7</b>
1.1. Преглед на съществуващи подобни приложения	7
1.1.1. Car Maintenance Reminder Lite	7
1.1.2. Car service Free	8
1.1.3. Car Manager	10
1.1.4. Обобщение	11
1.2. Развойни среди	11
1.2.1. Android Studio	11
1.2.2. SDK	11
1.2.3. ADT	12
1.3. Развойни средства	12
1.3.1. Компоненти на мобилно приложение за Android	12
<b>ВТОРА ГЛАВА</b>	<b>14</b>
<b>Изисквания към програмния продукт. Избор на езика за програмиране и софтуерните средства. Описание на основната структура на приложението и базата данни</b>	<b>14</b>
2.1. Изисквания към програмния продукт	14
2.2. Избор на операционна система, език за програмиране и софтуерни средства	15
2.2.1. Избор на операционна система	15
2.2.2. Избор на език за програмиране и софтуерните средства	15
2.3. Структура на приложението	15
2.3.1. Пакети в проекта	15

2.3.2. Зависимости на проекта	16
2.4. Структура на базата данни	20
<b>ТРЕТА ГЛАВА</b>	<b>22</b>
<b>Програмна реализация на приложението</b>	<b>22</b>
3.1. Главен екран на приложението	22
3.2. Функционалности	23
3.2.1. Добавяне и редактиране на записи	23
3.2.1.1. Базов клас	23
3.2.1.2. Добавяне на автомобил	25
3.2.1.3. Добавяне на дейност към автомобила	25
3.2.2. Преглеждане на записи	28
3.2.2.1. Под формата на лист	28
3.2.2.2. Единично преглеждане	29
3.2.3. Статистики	30
3.2.3.1. Цени на горивата	30
3.2.3.2. Разходи по дейности	31
3.2.3.3. Други	31
3.2.4. Импорт и експорт на превозни средства	32
3.2.4.1. Импорт	32
3.2.4.2. Експорт	33
3.2.5. Известяване	34
<b>ЧЕТВЪРТА ГЛАВА</b>	<b>36</b>
<b>Ръководство на потребителя</b>	<b>36</b>
4.1. Стартиране на приложението след инсталация	36
4.2. Добавяне на запис	37
4.2.1. Превозно средство	38

4.2.2. Сервизна услуга	39
4.2.3. Застраховка	40
4.2.4. Разход	41
4.2.5. Презареждане	42
4.3. Преглеждане на записи и статистики.	43
4.3.1. Записи	44
4.3.2. Статистики	48
4.4. Настройки	49
4.5. Импорт и експорт	50
4.5.1.Импорт	50
4.5.2.Експорт	51
<b>ЗАКЛЮЧЕНИЕ</b>	<b>53</b>
<b>ИЗПОЛЗВАНА ЛИТЕРАТУРА</b>	<b>54</b>

## Увод

Целта на настоящата дипломна работа е създаване на мобилно приложение за автомобилна сервизна книжка. В приложението трябва да е възможно да се добавят много превозни средства, да се импортира и експортира информация, да се добавят различни дейности, които са извършени по автомобила, както и да се водят разнообразни статистики на база въведените записи.

В днешното забързано ежедневие, често ни се налага да помним най-различни работи. Така неведнъж се случва да се забрави, какво е било и/или трябва да се извърши по автомобила ни. Запазването на такава информация би могло да се случи на хартиени носители, но постепенно данните стават трудно обработваеми, поради техния голям обем. По-ефективният и по-модерният начин за съхранение, е да се използва съвременната техника, до която почти всеки човек има достъп, а именно смартфоните.

Изпълнените задачи са: успешно се добавят превозни средства и дейности извършени по тях; показване на статистики и графики в приложението; импорт и експорт на коли чрез Bluetooth; известяване.

## **ПЪРВА ГЛАВА**

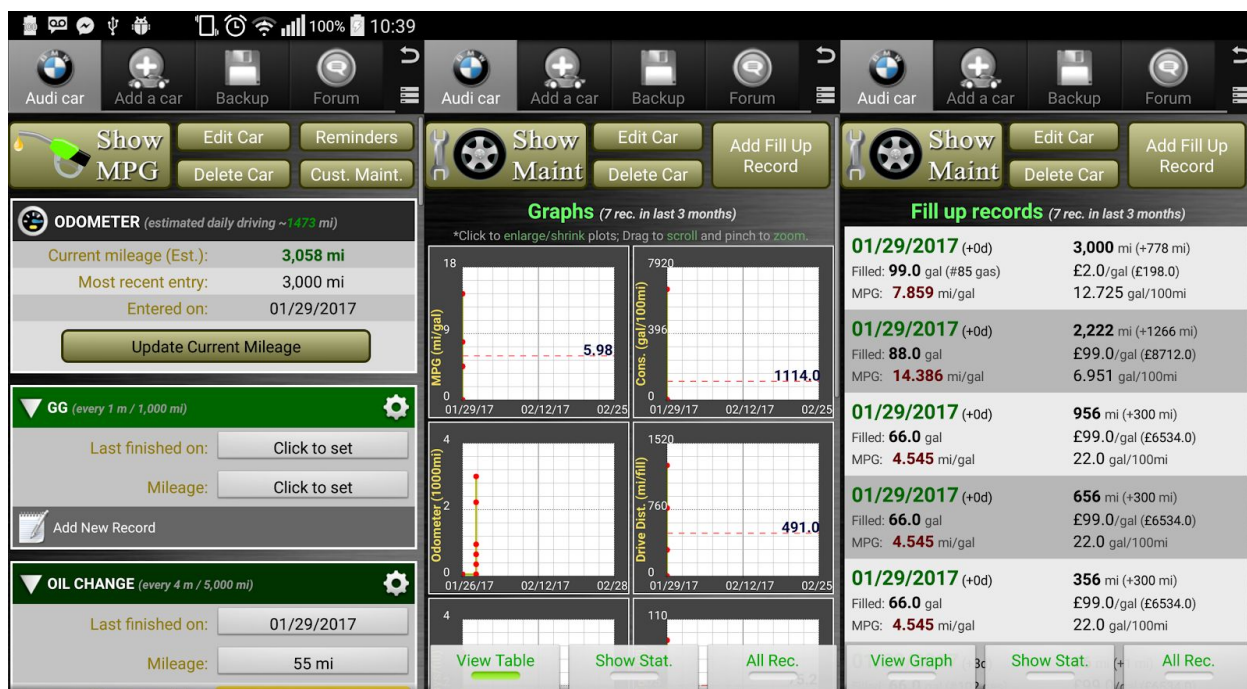
### **Проучвателна част. Преглед на съществуващи подобни приложения и преглед на известните развойни средства и среди**

#### **1.1. Преглед на съществуващи подобни приложения**

##### **1.1.1. Car Maintenance Reminder Lite**

“Car Maintenance Reminder Lite” е приложение в Google Play, което има 100,000 - 500,000 инсталирания и е с оценка 4,0 от 909 гласа. В приложението има различни опции за сервизни услуги, като е добавена и възможността да се добавят нови такива. Също така, може и да се добавят презарежданията на автомобила. След това приложението създава 10 графики, в които се показват различни неща. Интерфейсът е доста остарял и леко объркващ, което може би е вследствие на това, че сме свикнали на по-новите приложения.

Приложението предлага само експорт на базата, като има и възможност за импорт, но само в платената версия. Самият експорт става към CSV формат, като този файл се записва в паметта на телефона и потребителят трябва сам да отиде на местоположението му. На фиг. 1.1 може да се видят някои от основните екрани на приложението.



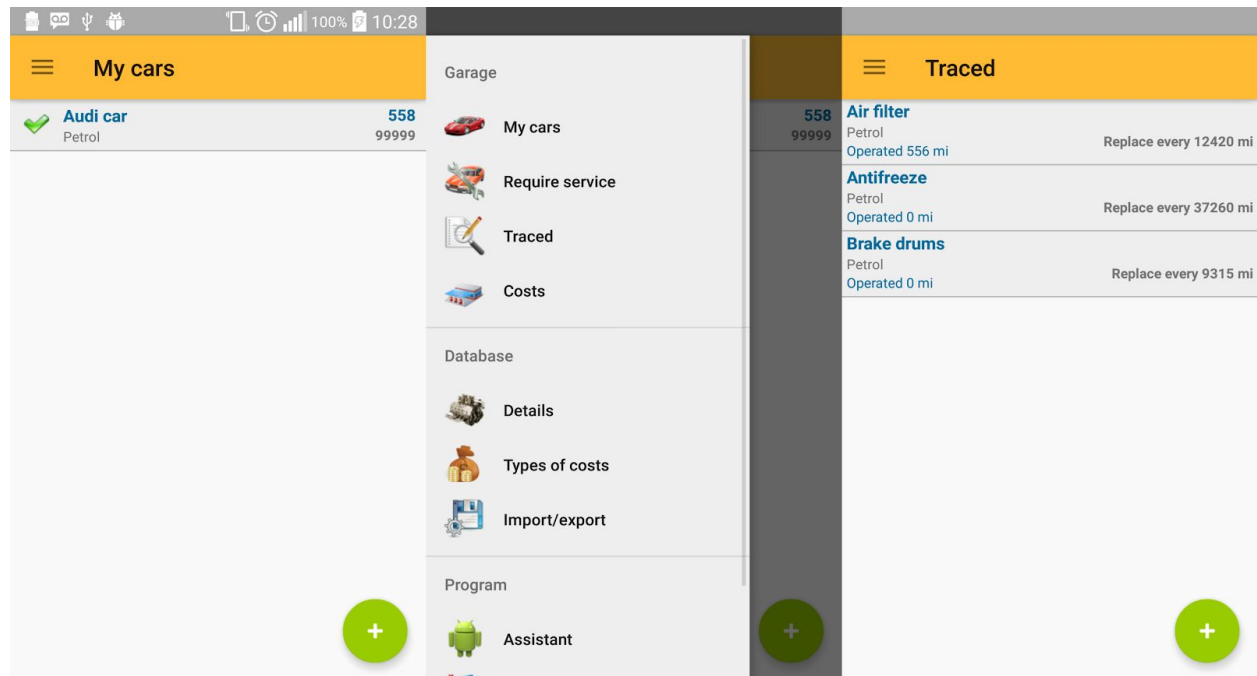
Фиг. 1.1

### 1.1.2. Car service Free

“Car service Free” е приложение в Google Play, което има 100,000 - 500,000 инсталирания и е с оценка 3,7 от 771 гласа. В приложението могат да се записват само сервизни услуги, които са предложени, като възможността за известие за подмяна е само на база километраж, без начин за добавяне на дата. Предоставена е опция за импорт и експорт на базата, но проблемът е, че това се случва малко неясно, тъй като приложението създава файл във файловата система, без да покаже по някакъв начин къде се намира този файл или поне как се казва. Импортът е на същия принцип, трябва да се добави файл на определено местоположение, след което да се влезне в приложението



и да се натисне съответният бутон, който да добави информацията, като изтрива всичко и записва новите данни. Интерфейсът е леко объркващ, като тук той е малко по-съвременен, макар и отново да не са спазени повечето от насоките на Google. На фиг. 1.2 може да се видят някои от основните екрани на приложението.

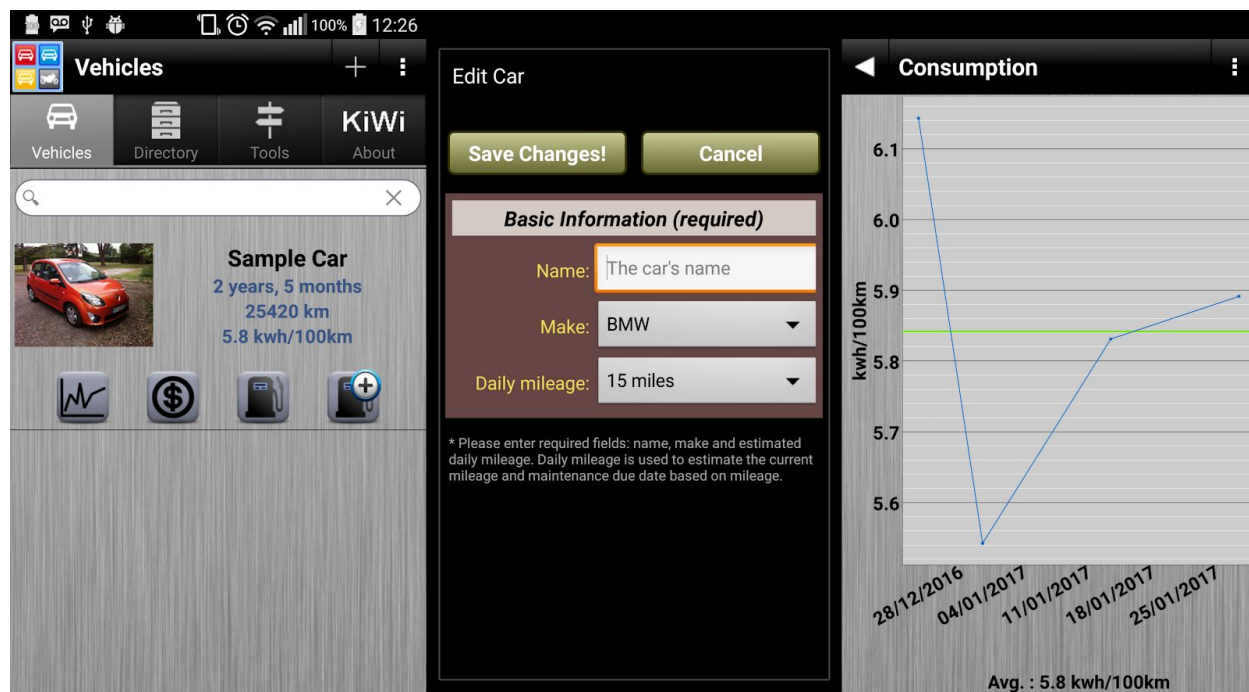


Фиг. 1.2

### 1.1.3. Car Manager

“Car Manager” е приложение в Google Play, което има 10,000 - 50,000 инсталирания и е с оценка 3,7 от 269 гласа. В апликацията има възможност за

добавяне на дадени дейности, но не се предлага нещо по-специфично, за сметка на това, има добавени доста графики, съставени на базата на въведената информация. Приложението е с остарял интерфейс и не предлага възможност за импорт и експорт на информация. Също така нямаме възможност за повече от един резервоар. На фиг.1.3 може да се видят някои от екраните в приложението.



Фиг. 1.3

#### 1.1.4. Обобщение

Като цяло в магазина на Google Play има доста подобни приложения, но повечето от тях не са спазили много от новите насоки, които от Google предлагат, при разработване на Android приложение. Това води до

затруднение при работата с тях.

## **1.2. Развойни среди**

### **1.2.1. Android Studio**

Android Studio е базирано на IntelliJ IDEA и е официалната интегрирана среда за разработка(IDE) за създаване на мобилни приложения на Android.

Android Studio предлага следните функционални възможности:

- Бърз емулатор
- Instant Run опция, която прилага промените по кода без да има нужда да се създава нов APK файл
- Интеграция на система за контрол на версиите
- Gradle приложение за автоматизация на сглобяването на проектите
- Редактор

### **1.2.2. SDK**

SDK(Software Development Kit) е добавка, която включва набор от инструменти за разработка на мобилни приложения на Android. То може да се добави както на Android Studio, така и на Eclipse. Чрез него, лесно могат да се добавят библиотеки и инструменти, които спомагат за по-лесното разработване на приложение, което да работи на различните версии на Android.

### 1.2.3. ADT

ADT(Android Development Tools) е добавка за Eclipse, която предоставя развойна среда за разработване на приложения за Android, но вече не се поддържа. Затова от Android силно се препоръчва да се използва Android Studio.

## 1.3. Развойни средства

### 1.3.1. Компоненти на мобилно приложение за Android

- **Activity** - това е екран със потребителски интерфейс, който управлява взаимодействието на потребителя със смартфона
- **Fragment** - представлява част от интерфейса в дадено activity
- **Service** - това е услуга, чието основното предназначение е да изпълнява задачи на зададен фон
- **Broadcast receiver** - това е компонент, който позволява на системата да предоставя определени събития, които да бъдат обработени, до приложението
- **View** - елементи от интерфейса като полета, бутони и други.
- **Intent** - това е съобщение, чрез което се заявява намерение да се извърши нещо
- **Resources** - това са всички допълнителни ресурси като изображения, символни низове и други

- **Layouts** - чрез тях се структурират изгледите в потребителския интерфейс по определен начин
- **Manifest** - това е файл, в който се описват компонентите на приложението

## **ВТОРА ГЛАВА**

**Изисквания към програмния продукт. Избор на езика за програмиране и софтуерните средства. Описание на основната структура на приложението и базата данни**

### **2.1. Изисквания към програмния продукт**

Създаване на мобилно приложение за Android платформата, което да има съвременен и разбираем потребителски интерфейс, както и да поддържа множество коли. В апликацията трябва да има възможност за въвеждане на информация за: извършени ремонтни дейности по колата, направени застраховки, презареждания на превозното средство и други разходи като такси, глоби и други. Също така, трябва да се добави възможност за известяване в определено време или на зададена стойност на километража. Трябва да има и статистики, които да показват разходите по извършените дейности и цените на горивата. Приложението трябва да предоставя възможност за изпращане и получаване на превозни средства при смяна на собственика на колата.

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

#### **2.2.1. Избор на операционна система**

За операционна система беше избрана Android, поради наличието на подходящи устройства, които са съвместими за разработване на приложение

за тази платформа.

### 2.2.2. Избор на език за програмиране и софтуерните средства

За създаването на приложението се използва езикът за програмиране Java, защото това е официалният език за програмиране на Android, който се препоръчва от Google, а и вече имам известен опит, който искам да доразвия. Работната среда е Android Studio, защото това е официалното IDE, което Google препоръчва, също така, то е и безплатно.

## 2.3. Структура на приложението

### 2.3.1. Пакети в проекта

В приложението разполага с 10 пакета.

- **activities** - съдържа всички Activity класове, както и помощните такива. Това са всички екрани в приложението.
- **adapters** - съдържа всички адаптери, които се използват за управлението на контейнерите в приложението.
- **app** - в този пакет има клас, в който се конфигурира Realm базата за цялото приложение.
- **broadcasts** - този пакет разполага с всички приемници на приложението. Чрез тях можем да реагираме при някакво събитие.
- **dialogs** - съдържа всички диалогови прозорци в приложението.
- **fragments** - в този пакет са всички използвани фрагменти в дипломната

работа.

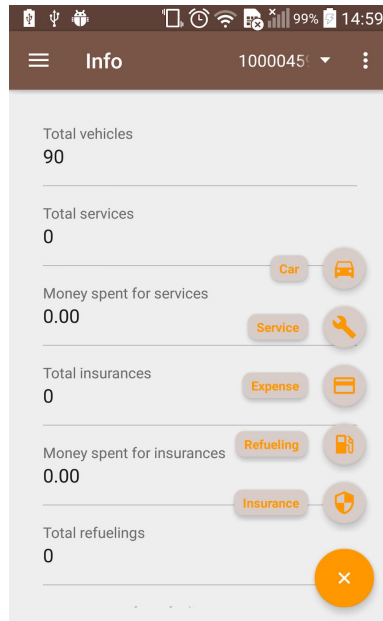
- **preferences** - съдържа всички класове, с които се достъпват SharedPreferences(начин за записване на информация под формата на ключ и стойност).
- **realm** - тук се съдържат всички модели на базата данни.
- **services** - това са всички услуги, които се използват в приложението.
- **utils** - този пакет съдържа всички помощни класове с предимно статични методи.

### 2.3.2. Зависимости на проекта

В приложението са използвани 11 външни библиотеки.

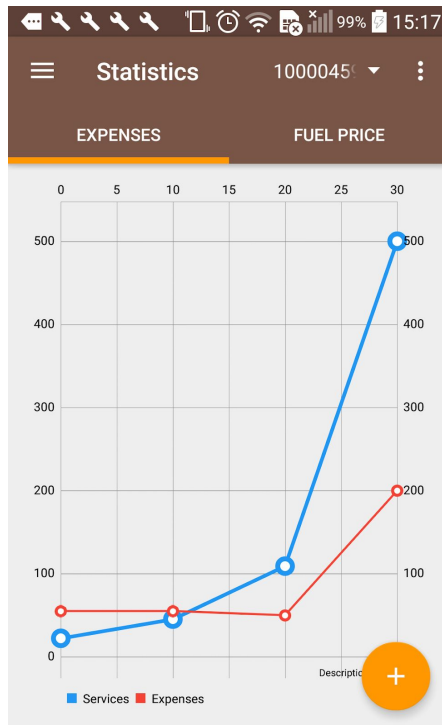
- **'org.apache.commons', name: 'commons-lang3', version: '3.0'** - предлага помощни класове, които се ползват за валидация на входните данни в проекта.
- **'io.github.yavski:fab-speed-dial:1.0.6'** - предоставя Floating Action Button(FAB) и меню към него, отговарящи на Material Design. Съответният бутон може да се види на фиг.2.3





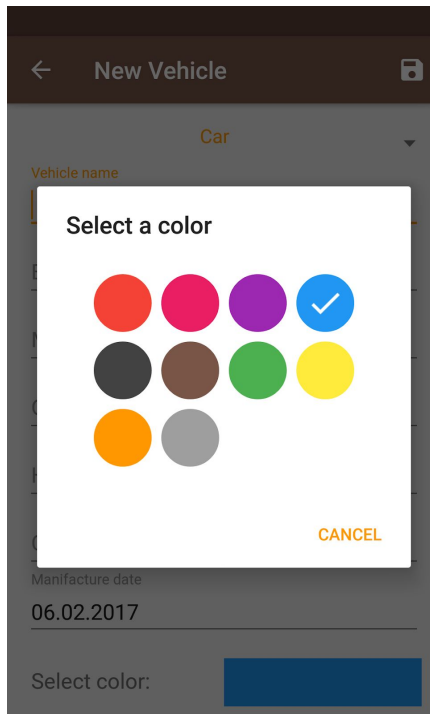
Фиг.2.3

- **'com.github.PhilJay:MPAndroidChart:v3.0.1'** - библиотека, която улеснява чертането на графики, които се използват в приложението за статистики. На фиг.2.4 може да се види как изглеждат графиките.



Фиг.2.4

- **'com.thebluealliance:spectrum:0.7.1'** - предлага диалогов прозорец, от който може да се избере цвят. На фиг.2.5 може да се види как изглежда диалогът.



Фиг.2.5

- **'io.realm:realm-gradle-plugin:2.3.0'** - база данни, в която ще се съхраняват данните в приложението
- **'com.github.thorbenprimke:realm-recyclerview:0.9.25'** - предлага оптимизиран, за работа с базата Realm, контейнер, който улеснява използването на предлагания от Android, RecyclerView.
- **'io.realm:android-adapters:1.4.0'** - съдържа адаптери, които са оптимизирани за работа с базата Realm.
- **'com.google.firebase:firebase-ads:10.0.1'** - спомага монетизацията на приложението чрез реклами.
- **'com.google.code.gson:gson:2.8.0'** - улеснява превръщането на обекти в JSON формат и обратното.

- **'com.android.support:appcompat-v7:25.1.1'** - предлага класове, които улесняват поддръжката на по-стари версии на Android
- **'com.android.support:design:25.1.1'** - библиотека, която съдържа основните практики за дизайн в Android

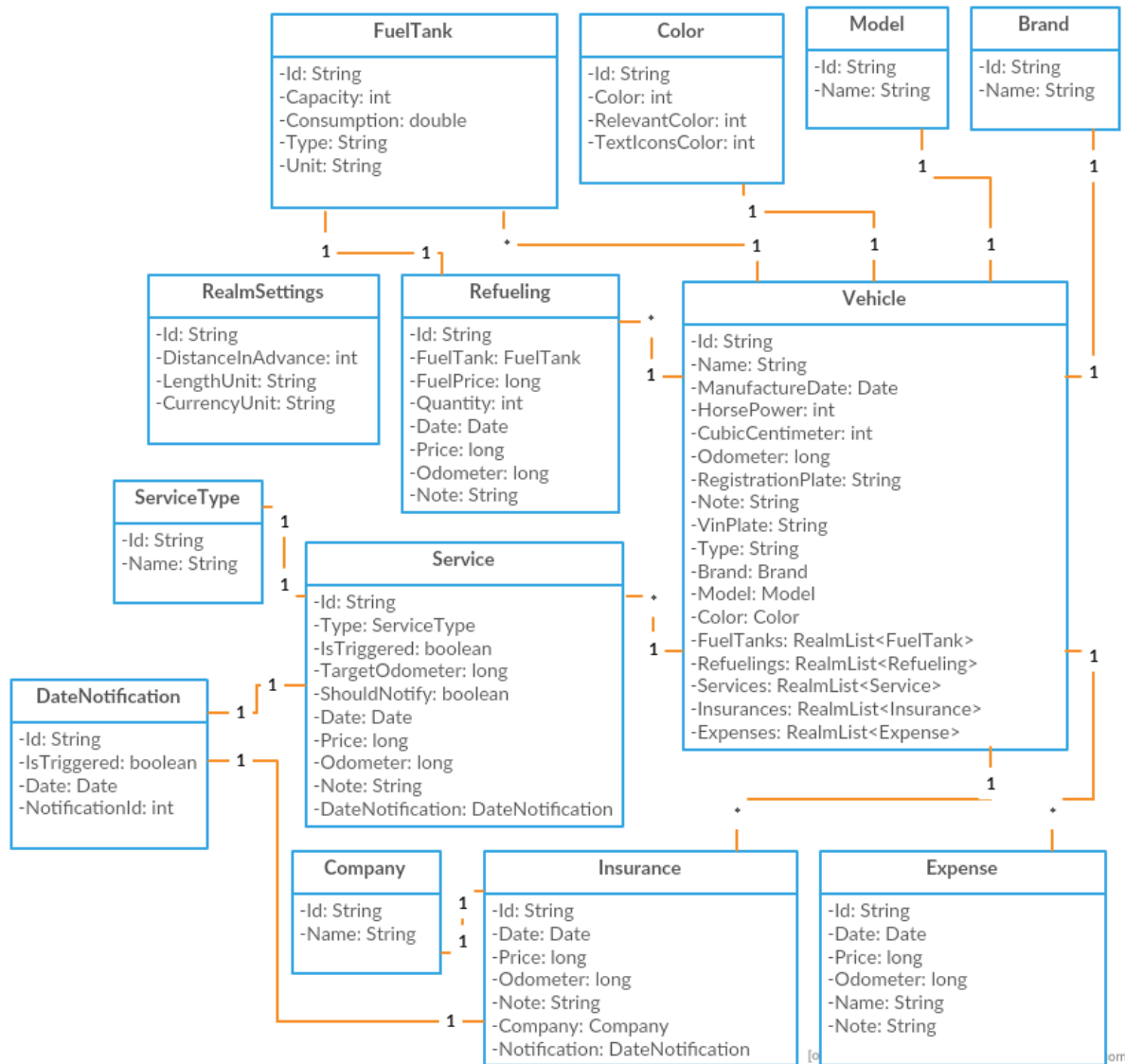
## 2.4. Структура на базата данни

За база данни в приложението се използва обектно-ориентираната база данни Realm, защото е безплатна, използва се лесно, бърза е и е добре документирана. За да се запази обект в тази база, трябва да бъде наследен RealmObject класа.

Базата на приложението се състои от 13 обекта, а на фиг.2.6 може да се види структурата ѝ.

За да може приложението да поддържа множество коли, е създаден обект Vehicle, който има някои спецификации за колите като цвят, марка, модел, регистрационен номер, конски сили и други. Понеже една кола може да има повече от един резервоар, е направен и обект FuelTank, който има тип на горивото, консумация, капацитет. Всяка кола може да има много извършени сервизни услуги, презареждания, разходи и застраховки, затова са добавени и тези обекти. Понеже и сервизните услуги и застраховките могат да известяват на определена дата, е добавен и DateNotification обекта. Всяка сервизна услуга(Service) има тип на ремонта, затова е създаден и ServiceType обекта. Всички застраховки са направени от застрахователна компания,

затова е добавен и Company обекта. В RealmSetting обекта се съхраняват настройките на приложението(пример: различни мерни единици).



Фиг.2.6

## **ТРЕТА ГЛАВА**

### **Програмна реализация на приложението**

Нека разгледаме текущата дипломна работа, като опишем главния екран и отделните функционалности на приложението.

#### **3.1. Главен екран на приложението**

В този екран има `NavigationDrawer`. Това е плъзгащо се меню отстрани, което има елементи, при натискането, на които, лесно се навигира до различни фрагменти от приложението(например: за извършените сервизни услуги, застраховки, разходи, статистики и други).

В екрана, също е добавена и лента с инструменти, в която има `Spinner`(падащо меню), от което се избира превозно средство, и `overflow menu`, от което може да се изберат настройки за приложението, както и импортиране на превозно средство. При избор на настройките, изскача диалогов прозорец, откъдето може да се промени мерната единица за дължина и валута, а и въвеждане на стойност за по-ранно подсещане при известяване на база изминато разстояние. Ако се избере опцията за импорт, то се показва нов екран, откъдето да се избере файл, който да бъде обработен и записан, но това ще бъде разгледано по-долу.

Добавен е и `FloatingActionButton`, като при натискането му се появява меню, откъдето лесно може да се добави нова кола, сервизна услуга, застраховка, разход или презареждане.

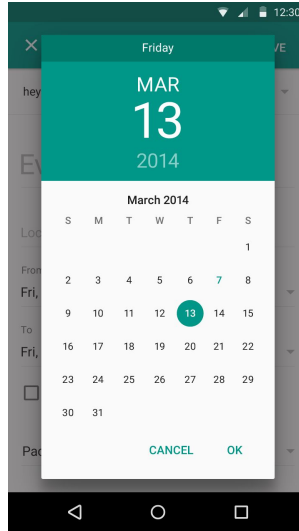
## **3.2. Функционалности**

### **3.2.1. Добавяне и редактиране на записи**

За добавяне и редактиране на записи, е създаден базов клас `NewBaseActivity`, който се наследява от всички 5 екрана: `NewVehicleActivity`, `NewServiceActivity`, `NewInsuranceActivity`, `NewRefuelingActivity` и `NewExpenseActivity`, които съответно отговарят за добавяне и редактиране на превозни средства, сервизни услуги, застраховки, презареждания и други разходи.

#### **3.2.1.1. Базов клас**

Базовият клас `NewBaseActivity` се съдържат общите неща, за всички екрани, свързани с добавяне и редактиране на записи. Тъй като всеки екран има текстови полета(`TextInputEditText`) за датата, стойността на километража и за бележка и лента с инструменти, в която има бутон за връщане назад и за записване, то те са в базовия клас, като той отговаря за тяхната инициализация, проверка при запис и слага слушател за кликуване върху полето за дата, за да може да се покаже `DatePicker`(може да се види на фиг. 3.1), от който да се избере датата, вместо да се въвежда от клавиатурата.



Фиг. 3.1

При записването на новата информация, класът проверява дали има сервизни услуги, за които потребителят трябва да бъде известен, че трябва да ги прегледа при достигане на определена стойност на километража. На фрагмент 1.1, може да се види заявката към базата, която да върне съответните сервизни услуги. В заявката се търсят тези сервизни услуги, които трябва да известяват, не са били показани на потребителя и чиято стойност километража за известяване е по-малка или равна на въведената нова. След това за всяка услуга се създава известие, с което потребителят да бъде уведомен(как става това, ще бъде описано по-долу).

```
long targetOdometer = odometer + settings.getDistanceInAdvance();
RealmResults<Service> services = myRealm
    .where(Service.class)
    .equalTo(RealmTable.SHOULD_NOTIFY, true)
    .equalTo(RealmTable.IS_ODOMETER_TRIGGERED, false)
    .notEqualTo(RealmTable.TARGET_ODOMETER, 0)
    .lessThanOrEqualTo(RealmTable.TARGET_ODOMETER,
        targetOdometer)
    .findAll();
```

Фрагмент 1.1



### 3.2.1.2. Добавяне на автомобил

Добавянето или редактирането на автомобил става в NewVehicleActivity екранът. Той разполага със: десет текстови полета за: име, марка, модел, дата на производство, текуща стойност на километража, конски сили, кубически сантиметри, регистрационен номер, номер на двигателя и допълнителна бележка; едно падащо меню(Spinner) за избор на типа на превозното средство(бус, кола, камион, мотор); четири бутона: за връщане към предишния екран, за запазване на колата, за избор на цвят и за добавяне на резервоар. При натискане на бутона за избор на цвят, се появява диалогов прозорец, от който може да се избере измежду 11 цвята. Ако се натисне бутона за добавяне на резервоар, се появява диалов прозорец с падащо меню за избор на типа на горивото и две полета, в които да се въведе капацитетът му и разходът на гориво.

### 3.2.1.3. Добавяне на дейност към автомобила

Нека разгледаме как се добавя ново презареждане. За целта, се отваря NewRefuelingActivity екрана, който разполага с шест текстови полета(TextInputEditText), в които се въвежда дата, час, стойност на километража, цена, количество и допълнителна бележка. Също така, има и падащо меню(Spinner), към което е прикачен адаптер с типовете на горивото, според резервоарите, с които разполага автомобила. Във фрагмент 3.2 може да се види как става това.

```
private ArrayList<String> getFuelTypeNamesFromResults() {  
    ArrayList<String> types = new ArrayList<>(results.size());  
    for (FuelTank fuelTank : results) {  
        types.add(fuelTank.getType());  
    }  
}
```

```

        return types;
    }
    ArrayList<String> fuelTypeNames = getFuelTypeNamesFromResults();
    ArrayAdapter<String> adapter = new ArrayAdapter<>(getApplicationContext(),
        R.layout.textview_spinner, fuelTypeNames);
    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
    spnFuelTanks.setAdapter(adapter);

```

Фрагмент 3.2

Екранът разполага и с бутон(ToggleButton), от който може да се избере дали резервоарът е пълен, като по този начин, няма да има нужда да се въвежда количеството на зареденото гориво. Във фрагмент 3.3, може да се проследи как става това.

```

toggleButton.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton
        compoundButton, boolean isChecked) {
        if (isChecked) {
            FuelTank fuelTank = getFuelTankFromSpinner();
            TextUtils.setTextToTil(tilQuantity,
                String.valueOf(fuelTank.getCapacity()));
            tilQuantity.setEnabled(false);
        } else {
            TextUtils.setTextToTil(tilQuantity, "");
            tilQuantity.setEnabled(true);
        }
    }
});

```

Фрагмент 3.3

След като бъде попълнена информацията и се натисне бутонът за записване от лентата с инструменти, то тя бива валидирана чрез функцията `isValid()`, която връща булева стойност, според това дали данните отговарят на изискванията или не. Ако отговарят, то те се записват в базата данни, като се извиква метода, показан във фрагмент 3.4.

```
myRealm.executeTransactionAsync(new Realm.Transaction() {  
    @Override  
    public void execute(Realm realm) {  
        //SAVE  
    }, new Realm.Transaction.OnSuccess() {  
        @Override  
        public void onSuccess() {  
            //ShowMessage that it is saved successfully  
        }  
    }, new Realm.Transaction.OnError() {  
        @Override  
        public void onError(Throwable error) {  
            showMessage("Something went wrong...");  
            error.printStackTrace();  
            finish();  
        }  
    }  
});
```

Фрагмент 3.4

Записването на останалите дейности по автомобила(сервизни услуги, застраховки и други разходи) е аналогично. Въвеждат се данни в текстови полета(`TextInputEditText`), след което те биват валидирани и записани.

## 3.2.2. Преглеждане на записи

### 3.2.2.1. Под формата на лист

В главният екран на приложението, от плъзгащото се меню отстрани(Navigation Drawer), могат да се прегледат различните записи на дейностите, извършени по автомобила.

Нека разгледаме, как се осъществява това със сервизните услуги. При избор на тях от менюто на главния екран се отваря нов фрагмент ListFragment. Той получава от екрана, уникалния идентификатор на автомобила, за да може да се изпрати заявка към базата. Също така, фрагментът получава и идентификаторът на избраната опция от главния екран, за да може се знае данни за коя дейност да бъдат показани. Във фрагмент 3.5, може да се види как става това.

```
Bundle bundle = getArguments();  
int id = bundle.getInt(MainActivity.FRAGMENT_TYPE);  
String vehicleId = bundle.getString(RealmTable.ID);
```

Фрагмент 3.5

Фрагментът разполага с RecyclerView. Това е изглед, като за разлика от ListView, всеки негов елемент се преизползва и това го прави много по-ефикасен при показването за по-голям набор от данни.

За всяка дейност, е създаден и адаптер. В случая, той се казва ServiceRecyclerViewAdapter. Той получава лист от сервизни услуги и за всяка от тях, се показват типа на сервизната услуга, цената, датата и известяването.

Преглеждането на всички останали записи е аналогично, като за всяка дейност има отделен адаптер клас.

### 3.2.2.2. Единично преглеждане

При кликване върху елемент от RecyclerView изглежда, се отваря екран, в който може да се види цялата информация за записа. Ако елементите са коли, то се отваря ViewVehicleActivity, а за всичко останало - ViewActivity. И двата екрана разполагат с бутон(FloatingActionButton), при натискането, на който се съответния екран от т.3.2.1 за редактиране на записа.

- ViewVehicleActivity - този екран получава идентификатора на колата, след което в текстови изгледи(TextView) се показва информацията.
- ViewActivity - този екран получава идентификатора на автомобила, на дейността и типа на дейността(застраховка, разходи, презареждане или сервизна услуга), след което според типа на дейността се добавят динамично изгледи. Във фрагмент 3.6, може да се види как става това.

```
private void displayView(String title, String value) {  
    View view = getLayoutInflater()  
        .inflate(R.layout.item_view_activity, null);  
    TextView tvTitle = (TextView) view  
        .findViewById(R.id.tv_item_view_title);  
    TextView tvValue = (TextView) view  
        .findViewById(R.id.tv_item_view_value);  
    tvTitle.setText(title);  
    tvValue.setText(value);  
    LinearLayout linearLayout = (LinearLayout)  
        findViewById(R.id.content_view);  
    linearLayout.addView(view);  
}
```

Фрагмент 3.6

### 3.2.3. Статистики

При избор на опцията Statistics от менюто в главния екран на приложението се показва нов фрагмент - StatisticsFragment, който разполага с

ViewPager и TabLayout. ViewPager позволява на потребителя да плъзга наляво и надясно по екрана, като по този начин се сменят страници. TabLayout предоставя оформление, което има раздели. Към ViewPager се добавят два фрагмента - за цените на горивата(FuelPriceFragment) и за разходите по извършените дейности(LineChartFragment).

### 3.2.3.1. Цени на горивата

В този фрагмент, по отделно за всеки тип горива, се изпраща заявка до базата за всички презареждания с типа гориво. Във фрагмент 3.7, може да се види как се случва това за зададен тип на горивото дизел.

```
RealmResults<Refueling> refuelings = myRealm
    .where(Refueling.class)
    .equalTo(RealmTable.FUEL_TYPE, "Diesel")
    .findAll();
```

Фрагмент 3.7

След това, за всяко презареждане се взимат цените на горивата, след което се добавят към графиката. Във фрагмент 3.8, може да се проследи как става това.

```
ArrayList<Entry> dDataSet = new ArrayList<>();
int i = 0;
for (Refueling refueling : refuelings) {
    float f = MoneyUtils.longToFloat(new BigDecimal(
        refueling.getFuelPrice()));
    dDataSet.add(new Entry(i, f));
    i += 10;
}
LineDataSet dataSet = new LineDataSet(dDataSet, "Diesel");
```

```
LineData lineData = new LineData(dataSet);  
lineChart.setData(lineData);
```

Фрагмент 3.8

Имплементацията за останалите типове на горивата е сходна.

### 3.2.3.2. Разходи по дейности

Във фрагментът, се изпраща заявка до базата данни за всички дейности, извършени по избраната кола от падащото меню в главното меню. След това за всяка дейност, се взима цената, след което се добавя към графиката. Имплементацията наподобява кода от фрагмент 3.8.

### 3.2.3.3. Други

При избор на опцията Info от менюто в главния екран, се отваря нов фрагмент - InfoFragment. В него отново се добавят изгледи динамично, а кодът за това е сходен като този във фрагмент 3.6. Фрагментът показва:

- Броят на: превозните средства, сервизните услуги, застраховки, презареждания, разходи.
- Разходите за гореизброените дейности
- Всички разходи сумирани

Във фрагмент 3.9, може да се види как става това за застраховките, а за останалите дейности е аналогично.

```
RealmQuery<Insurance> insurances = myRealm.where(Insurance.class);  
displayView("Total insurances", String.valueOf(insurances.count()),  
            inflater);
```

```

bigDecimal = new BigDecimal(
    insurances.sum(RealmTable.PRICE).toString());
BigDecimal totalCost = new BigDecimal(0);
totalCost = totalCost.add(bigDecimal);
displayView("Money spent for insurances", String.format(text,
    MoneyUtils.longToString(bigDecimal)), inflater);

```

Фрагмент 3.9

### 3.2.4. Импорт и експорт на превозни средства

#### 3.2.4.1. Импорт

При импорт на превозно средство, се отваря нов екран, от който потребителят трябва да избере файл(виж фрагмент 3.10), от който да се извлече автомобил(ако е възможно) и той да бъде записан. Във фрагмент 3.11, може да се види как става това.

```

Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);
intent.addCategory(Intent.CATEGORY_OPENABLE);
intent.setType("*/*");
startActivityForResult(intent, READ_REQUEST_CODE);

```

Фрагмент 3.10

```

Uri uri;
if (data != null) {
    uri = data.getData();
    try {
        InputStream inputStream = getContentResolver()
            .openInputStream(uri);
    }
}

```



```

String content = FileUtils
    .getContentFromInputStream(inputStream);
Vehicle vehicle = new Gson().fromJson(content, Vehicle.class);
if (vehicle != null) {
    //save vehicle
    importVehicle(Vehicle vehicle);
} else {
    showMessage("Couldn't save vehicle");
}
} catch (JsonSyntaxException e) {
    e.printStackTrace();
    showMessage("Couldn't get vehicle");
} catch (FileNotFoundException e) {
    e.printStackTrace();
    showMessage("File not found");
}
}
}

```

Фрагмент 3.11

### 3.2.4.2. Експорт

При експорт на автомобил, първо се отправя заявка за превозното средство към базата. След това обектът се преобразува в JSON формат, който се запазва в променлива от тип String(виж фрагмент 3.12) и се записва във файл на устройството. Тогава файлът се праща с Bluetooth до друго устройство(виж фрагмент 3.13).

```

String content = new Gson().toJson(vehicle);

```

Фрагмент 3.12

```

Intent shareIntent = new Intent();
shareIntent.setAction(Intent.ACTION_SEND);
shareIntent.putExtra(Intent.EXTRA_STREAM, FileUtils.getFileUri(file));
shareIntent.setType("text/plain");
shareIntent.setPackage("com.android.bluetooth");
startActivity(Intent.createChooser(shareIntent, "Send to"));

```

Фрагмент 3.13

### 3.2.5. Известяване

За известяванията, е създаден помощен клас NotificationUtils. За да се създаде нотификация, трябва да се извика метода createNotification(фрагмент 3.14) от помощния клас, като му се подадат като аргументи: контекст, заглавие, съержание, иконка, типа и идентификатора на дейността, и екранът, който да бъде пуснат при кликване върху съобщението.

```

public static Notification createNotification(Context context,
                                             String vehicleId, String propertyKey,
                                             String propertyId, ActivityType activityType,
                                             Class aClass, String title,
                                             String content, int smallIcon) {

    Intent resultIntent = new Intent(context, aClass);
    resultIntent.putExtra(RealmTable.ID, vehicleId);
    resultIntent.putExtra(propertyKey, propertyId);
    resultIntent.putExtra(RealmTable.TYPE, activityType.ordinal());
    PendingIntent pendingIntent =
        PendingIntent.getActivity(
            context,
            0,
            resultIntent,
            PendingIntent.FLAG_UPDATE_CURRENT
        );
}

```

```
Notification.Builder builder = new Notification.Builder(context);
builder.setContentTitle(title)
    .setContentText(content)
    .setSmallIcon(smallIcon)
    .setLargeIcon(BitmapFactory.decodeResource(context.getResources(),
                                                R.mipmap.ic_launcher))
    .setSound(Settings.System.DEFAULT_NOTIFICATION_URI)
    .setVibrate(new long[]{1000, 1000})
    .setAutoCancel(true)
    .setContentIntent(pendingIntent);
return builder.build();
}
```

Ф р а г м е н т 3.14

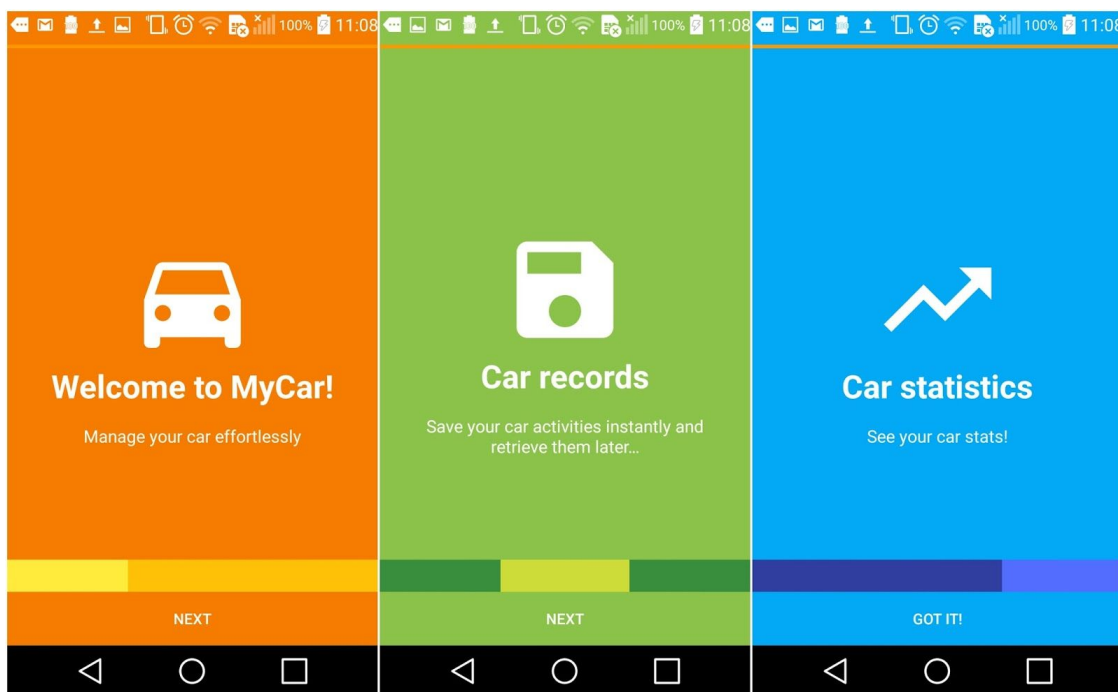
За известяване на определена дата, се използва `AlarmManager`, който да извести приложението, че датата е дошла. Това става чрез `NotificationReceiver` класа, откъдето се извиква метода `triggerNotification`, който известява потребителя, от `NotificationUtils` класа.

## ЧЕТВЪРТА ГЛАВА

## Ръководство на потребителя

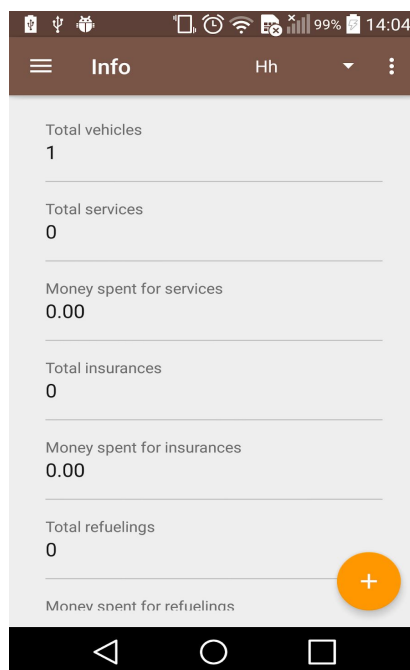
### 4.1. Стартиране на приложението след инсталация

След като приложението бъде инсталирано и стартирано, се появява екранът, показан на фиг.4.1. В екранът, потребителят се запознава с някои от функционалностите на приложението.



Фиг.4.1

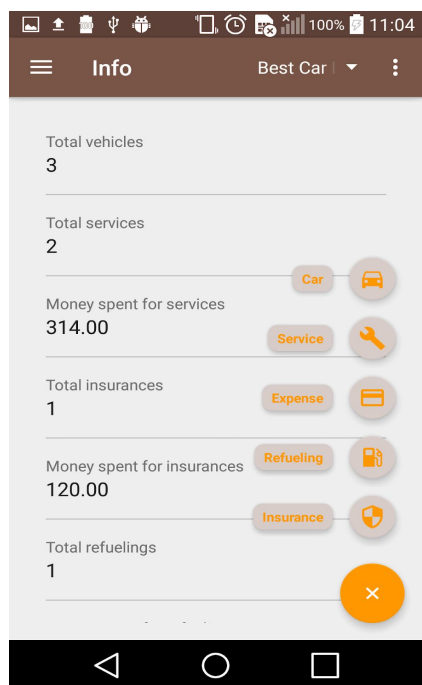
След което, преминаваме към главният екран на приложението, който може да се види на фиг.4.2.



Фиг.4.2

## 4.2. Добавяне на запис

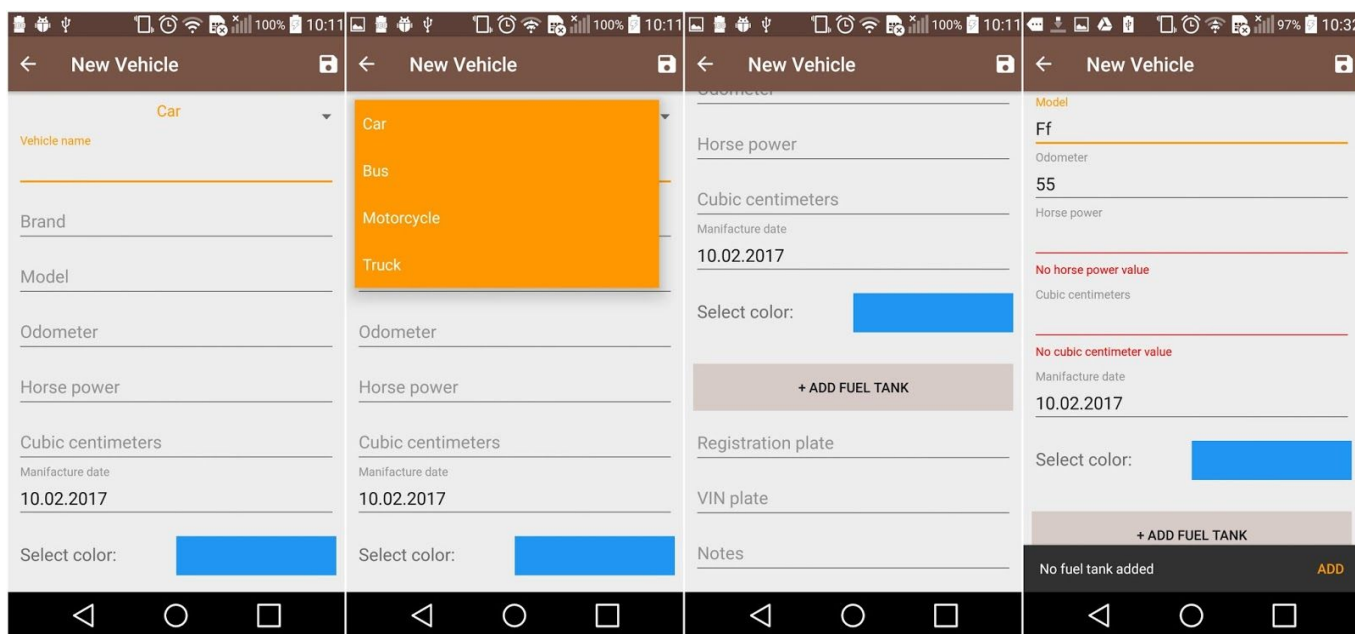
Ако искаме да добавим нов запис, то трябва да се избере какъв да бъде той. Възможностите са: ново превозно средство, сервизна услуга, застраховка, презареждане на гориво и някакъв друг разход. Изборът става от менюто, показано на фиг.4.3.



Фиг.4.3

#### 4.2.1. Превозно средство

При добавяне на нова кола, се стартира нов екран, показан на фиг.4.3. В него се въвеждат данните на автомобила, като всички полета са задължителни, с изключение на полето за бележки(Notes). След натискане на иконката от лентата с инструменти за запаметяване, полетата се валидират и ако нещо не е наред, то се изписва съобщение на съответното поле(виж фиг.4.6).



Фиг.4.3

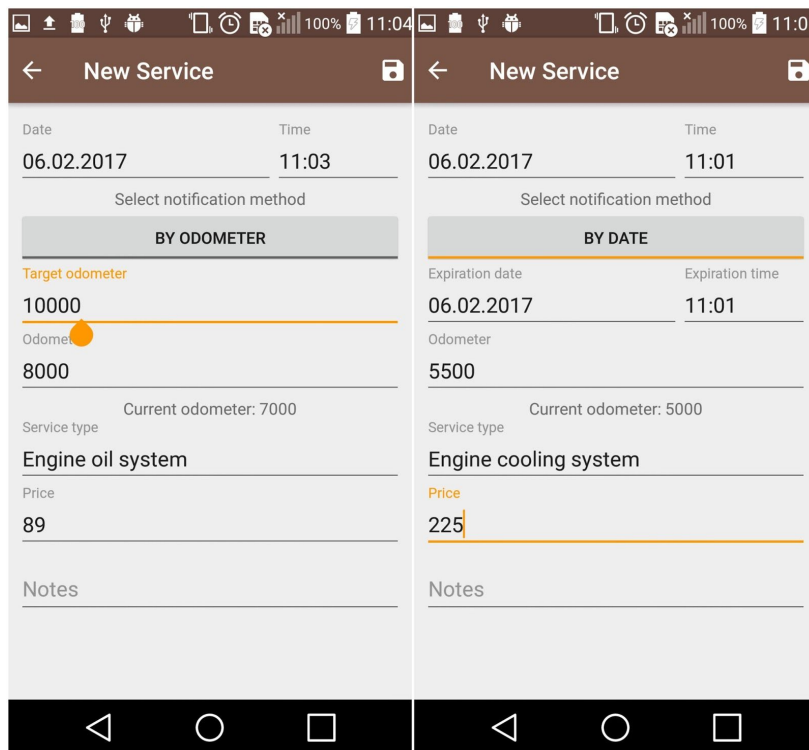
Фиг.4.4

Фиг.4.5

Фиг.4.6

#### 4.2.2. Сервизна услуга

Ако искаме да добавим сервизна услуга към вече съществуваща кола, то се стартира екранът от фиг.4.7 и фиг.4.8. Всички полета са задължителни, само това за бележка(Notes) може да остане празно. На фиг.4.7 известяването става на база въведената стойност на километража, докато на фиг.4.8 е с дата.



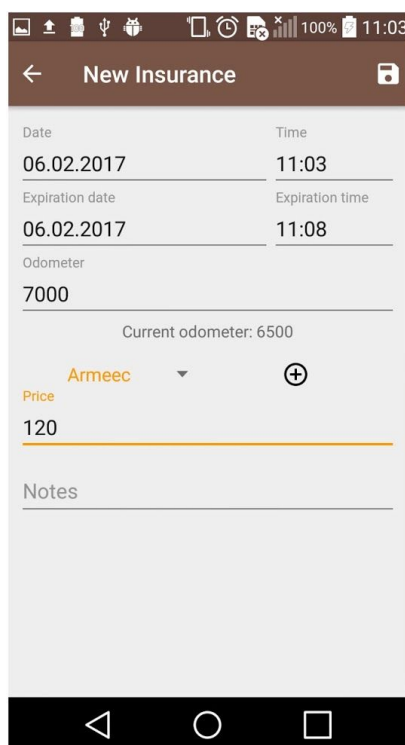
Фиг.4.7

Фиг.4.8

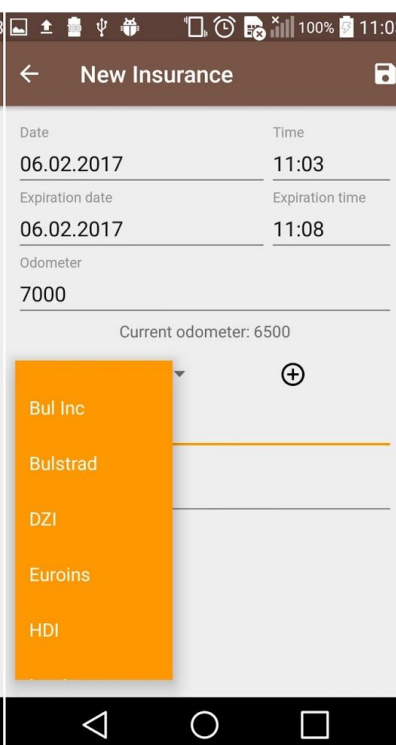
### 4.2.3. Застраховка

При добавяне на нова застраховка, се отваря екранът, показан на фиг.4.9. В него всички полета са задължителни, с изключение на полето за бележка(Notes). Известяването става, при достигане на въведената дата. От падащото меню от фиг.4.10, се избира застрахователна компания. Ако искаме да въведем нова компания, се показва диалоговият прозорец, показан на фиг.4.11.

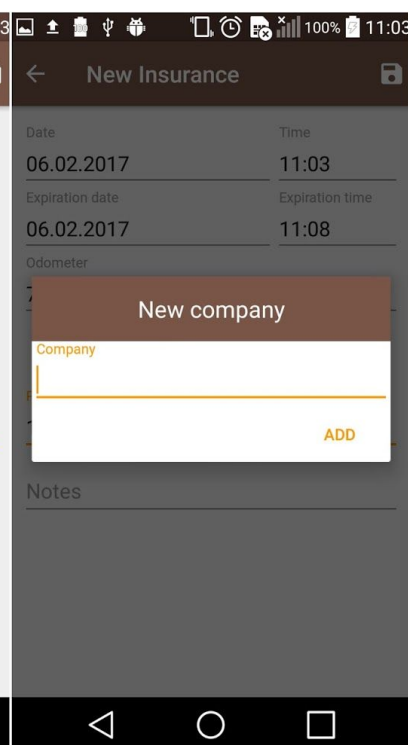




Фиг.4.9



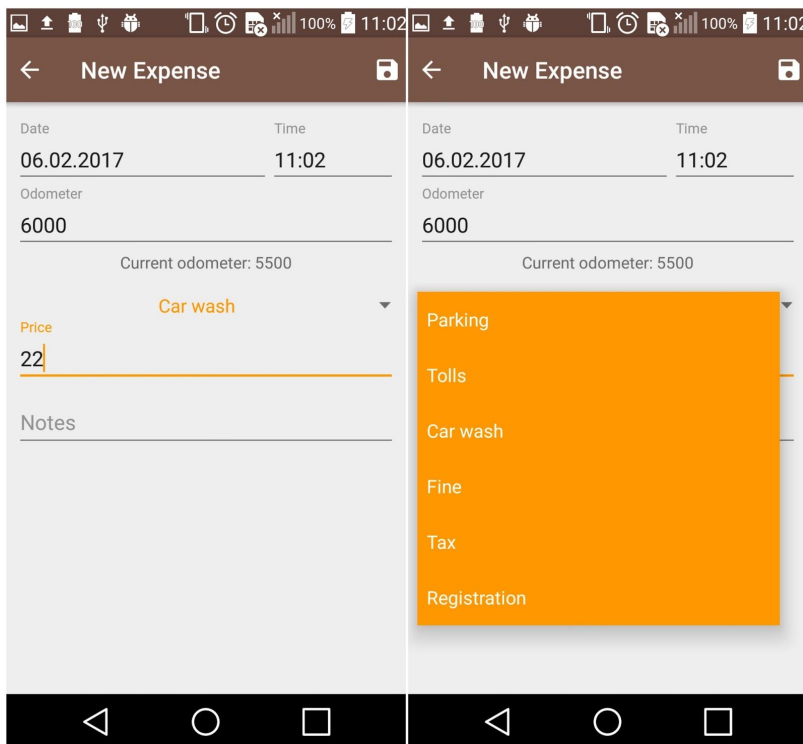
Фиг.4.10



Фиг.4.11

#### 4.2.4. Разход

Ако искаме да добавим нов разход, се отваря нов екран, който е може да се види на фиг.4.12. В него всички полета са задължителни, без това за бележка(Notes). От падащото меню, което може да се види на фиг.4.13, се избира типът на разхода.



Фиг.4.12

Фиг.4.13

#### 4.2.5. Презареждане

Екранът от фиг.4.14, се стартира, когато искаме да добавим ново презареждане. Тук отново имаме валидация, както във всички останали екрани изброени по-горе.


The screenshot shows a mobile application interface for recording a new refueling event. The form is titled 'New Refueling' and contains the following fields and values:

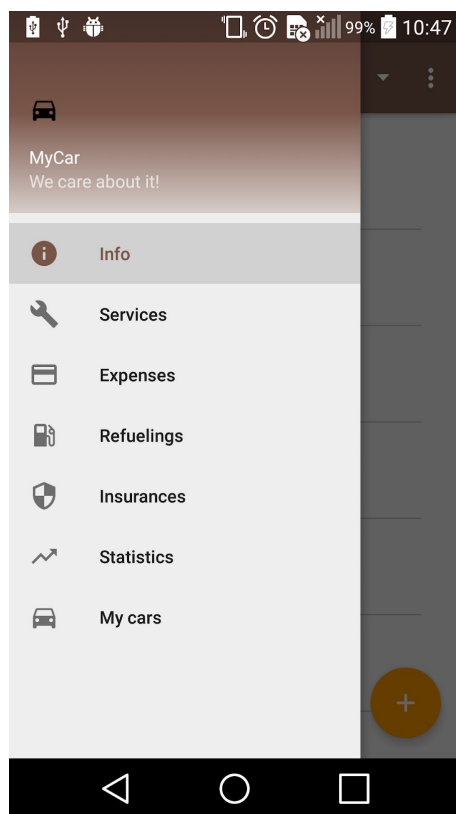
- Date:** 06.02.2017
- Time:** 11:02
- Fuel Type:** Diesel (selected from a dropdown menu)
- Odometer:** 6500
- Current odometer:** 6000 (displayed below the odometer field)
- Full fuel tank:** YES (selected from a dropdown menu)
- Quantity:** 80
- Price:** 166
- Notes:** (empty text area)

The top status bar shows various icons including signal strength, battery level (100%), and time (11:02). The bottom navigation bar features three standard Android navigation icons: back, home, and recent apps.

Фиг.4.14

### 4.3. Преглеждане на записи и статистики.

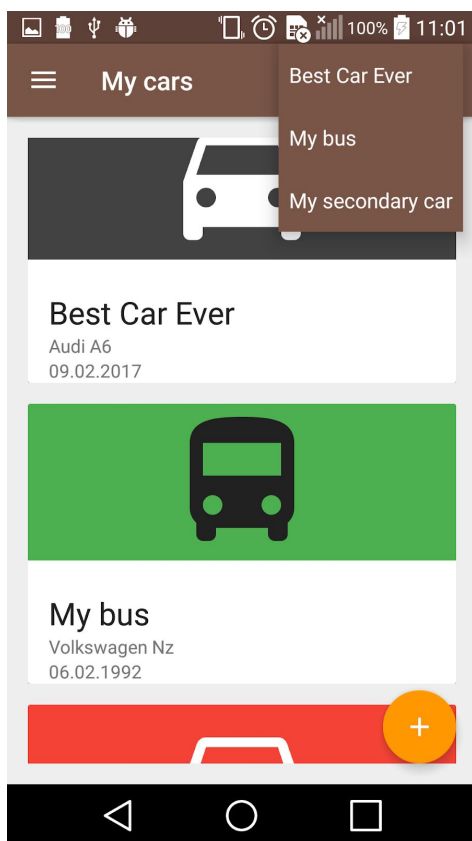
За да се видят вече записаните автомобили или извършените дейности по тях, трябва да отидем в главният екран(този, който се появява при пускане на приложението). Там от менюто, което се появява при плъзгане по екрана или като се натисне бутонът  от лентата с инструменти, може да се прегледат записите и статистиките. Менюто може да се види на фиг.4.15.



Фиг.4.15

#### 4.3.1. Записи

След като бъде избрана опция от менюто на фиг.4.15. се показват всички, дейности по колата, избрана от падащото меню, което може да се види на фиг.4.16.



Фиг.4.16

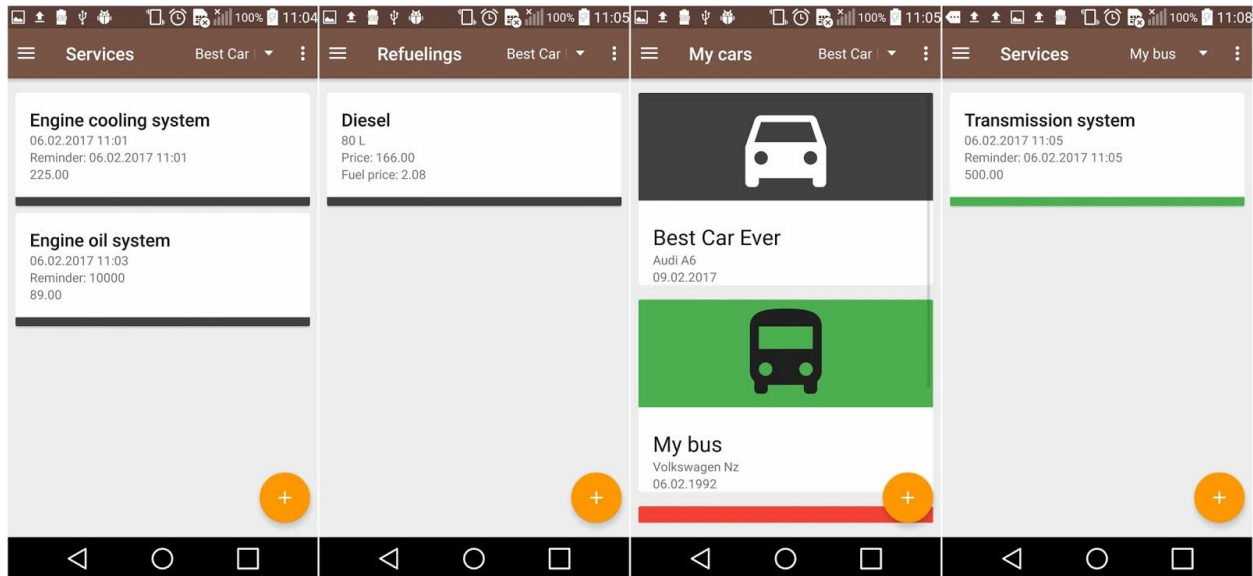
- **Всички записи**

На фиг.4.17 и фиг.4.20, може да се види как се показват всички извършени услуги по колата. Цветната лента, която е най-отдолу на всеки елемент, съответства с цвета на колата.

На фиг.4.18, могат да се видят направените презареждания, а на фиг. 4.19 - превозните средства.

Ако искаме да изтрием някой запис, просто трябва да се плъзне хоризонтално върху съответния запис. При кликуване върху някой от

елементите, се отваря екран, в който може да се види единичен запис, с подробна информация.



Фиг.4.17

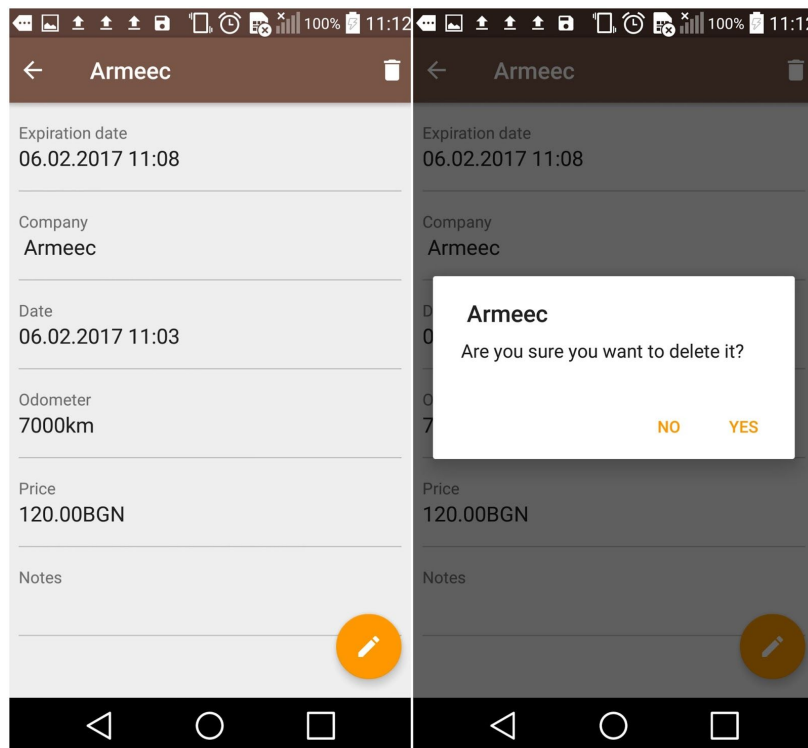
Фиг.4.18

Фиг.4.19

Фиг.4.20

- **Единичен запис**

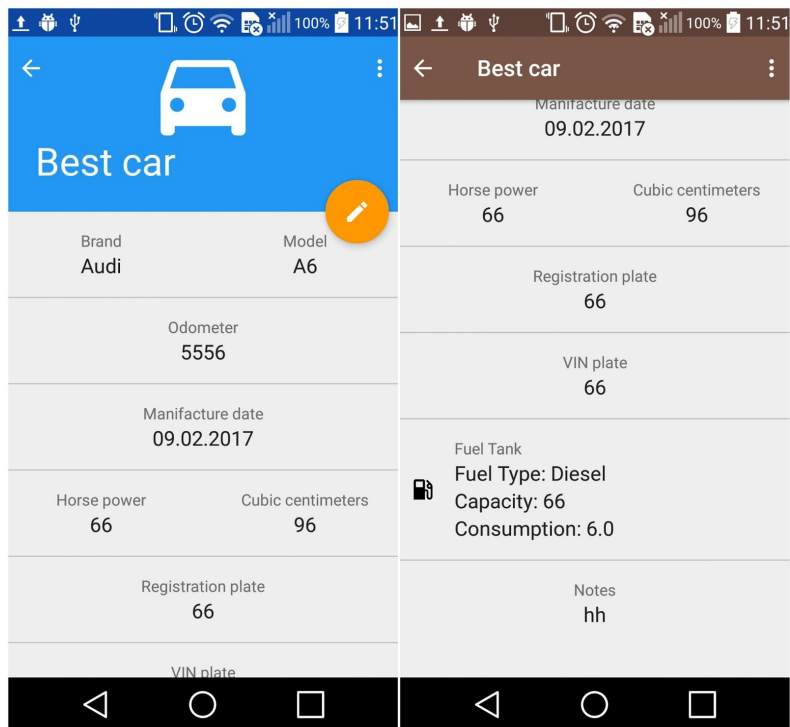
След като се натисне върху запис, се отваря, екранът показан на фиг.4.21, където той може да бъде разгледан по-подробно. При натискане на бутона за редактиране, се отваря екрана за добавяне на запис, само че полетата са попълнени с информацията. Ако искаме да изтрием записа, просто натискаме върху иконката на кошче от лентата с инструменти, след което се появява диалогов прозорец, който иска потвърждение(виж фиг.4.22).



Фиг.4.21

Фиг.4.22

Екранът за преглеждане на запис на автомобил, изглежда по-различно и може да бъде видят на фиг.4.23.



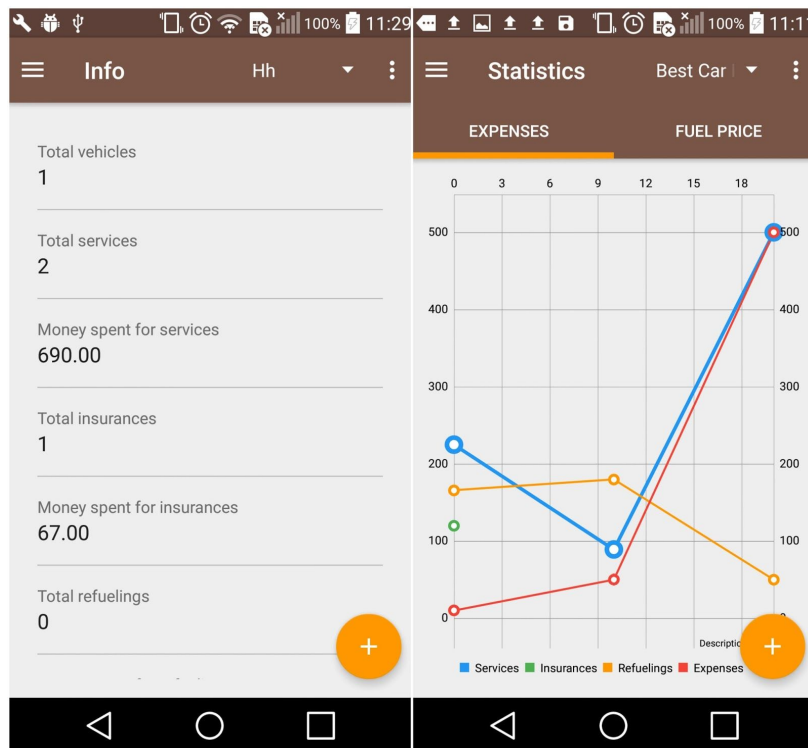
Фиг.4.23

### 4.3.2. Статистики

В приложението могат да се видят статистики от две места. При избор на опцията Info или на Statistics от менюто на фиг.4.15.

- **Info** - може да се видят най-различни данни, например, колко пари са изхарчени за отделните дейностите(виж фиг.4.24).
- **Statistics** - графично представените данни за разходите и за цените на горивата(виж фиг.4.25).



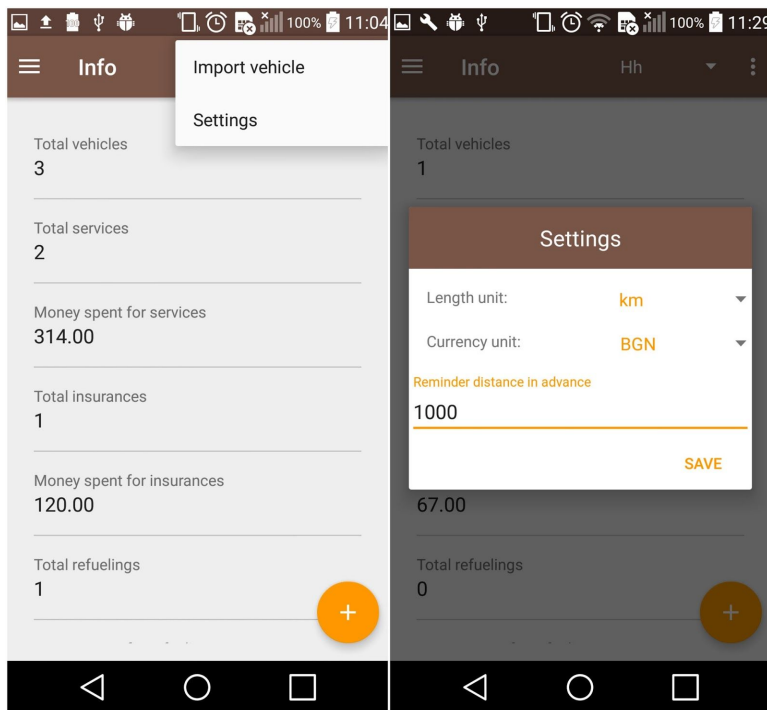


Фиг.4.24

Фиг.4.25

#### 4.4. Настройки

Настройките на приложението могат да бъдат променени като изберем втората опция от менюто на фиг.4.26. След което, се показват диалогов прозорец(виж фиг.4.27), откъдето могат да се променят мерни единици, валута и разстоянието в аванс за известяване.



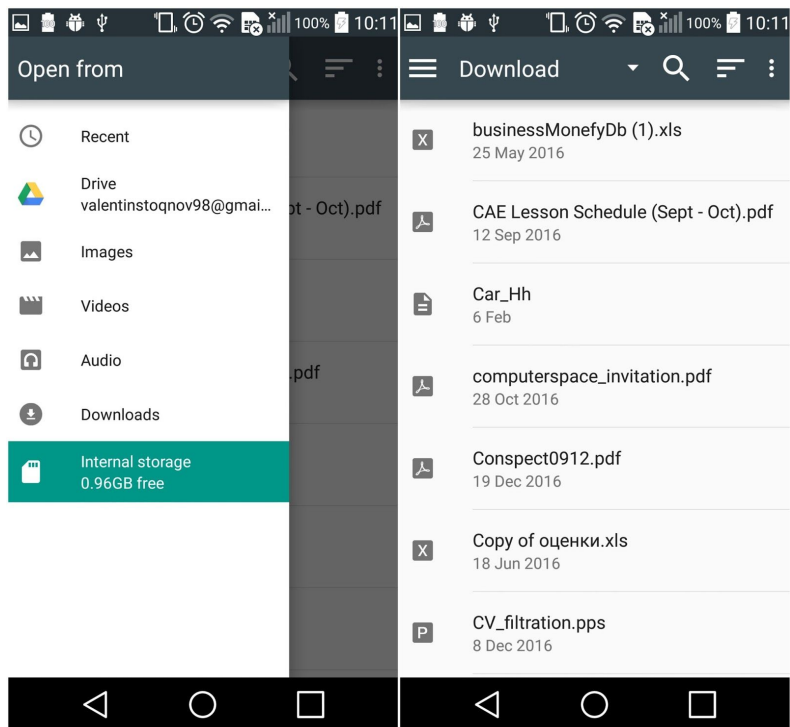
Фиг.4.26

Фиг.4.27

## 4.5. Импорт и експорт

### 4.5.1.Импорт

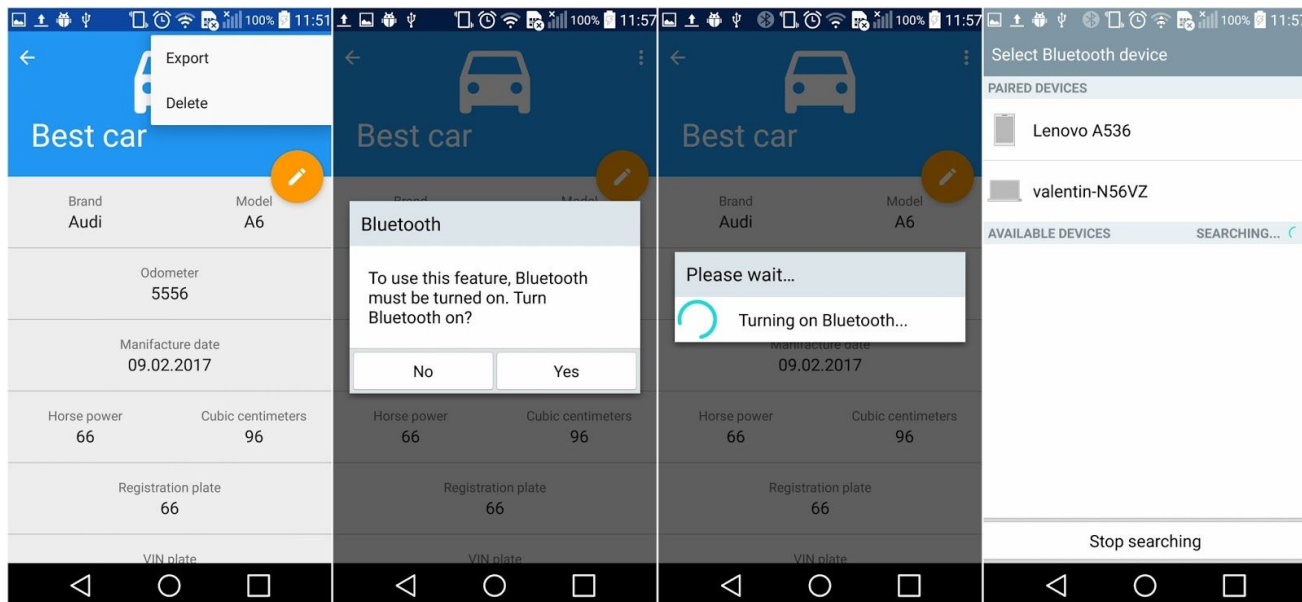
За да импортираме кола, трябва да отидем на главния екран и да изберем първата опция от менюто, показано на фиг.4.26. След което, се отваря екран, от който да се избере файла, от който да се импортира колата. Този екран, може да се види на фиг.4.28. Когато се избере файла, той се прочита и ако е валиден, автомобилът се записва и се извежда съобщение за това.



Фиг.4.28

#### 4.5.2.Експорт

За експорт на превозно средство, трябва да отидем в екранът за преглеждане на записа на съответната кола(фиг.4.23) и да изберем първата опция от менюто на фиг.4.27. След това се появява диалогов прозорец(виж фиг.4.28), който уведомява, че Bluetooth-а ще бъде включен. След което, се появява екран(виж фиг.4.30), от който да бъде избрано устройството, на което да бъде изпратена колата.



Фиг.4.27

Фиг.4.28

Фиг.4.29

Фиг.4.30

## **ЗАКЛЮЧЕНИЕ**

В дипломната работа, успешно е разработено мобилно приложение за Android, което представлява автомобилна сервизна книжка. Приложението покрива всички изисквания, поставени в заданието. Софтуерният продукт е широко приложим за всеки собственик и/или водач на превозно средство.

Бъдещото развитие включва: превеждане на приложението на български; подобряване на потребителският интерфейс, като бъдат добавени различни анимации, които да направят приложението по-интерактивно; по-добра монетизация на приложението.

## ИЗПОЛЗВАНА ЛИТЕРАТУРА

- Най-добри практики за структура от пакети, имена на класове и файлове -  
<https://www.linkedin.com/pulse/android-package-structure-performance-tam-nguyen>
- Документация на Realm базата - <https://realm.io/docs/java/latest/>
- Препоръки за дизайн - Google Material Design Guidelines -  
<https://material.io/guidelines/>
- Официалният сайт за разработчици на Android приложения -  
<https://developer.android.com/guide/index.html>
- Цветова палитра - <https://www.materialpalette.com/>
- Проучване за пазара на операционните системи за мобилните устройства
  - <http://www.idc.com/promo/smartphone-market-share>
  - <https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/>
- Android - [https://en.wikipedia.org/wiki/Android\\_\(operating\\_system\)](https://en.wikipedia.org/wiki/Android_(operating_system))
- Репозитори на проекта - <https://github.com/valentinstoqnov/MyCar>