

**Пловдивски Университет „Паисий Хилендарски”  
Факултет по Математика, Информатика и  
Информационни технологии**



# **ДИПЛОМНА РАБОТА**

**ТЕМА: Мобилен модул за тестова система**

Изготвили:

Александър Руменов Инджов

Фак. № 1701737014

Научен ръководител:

ас. Йордан Тодоров

# СЪДЪРЖАНИЕ

<b>1. УВОД.....</b>	<b>4</b>
1.1. ЦЕЛ .....	4
1.3. ХАРДУЕРНИ ИЗИСКВАНИЯ.....	5
1.4. СОФТУЕРНИ ИЗИСКВАНИЯ .....	5
<b>2. ИЗПОЛЗВАНИ ТЕХНОЛОГИИ .....</b>	<b>6</b>
2.2. АНДРОИД.....	6
2.3. КАКВО Е АНДРОИД?.....	7
2.4. ВЕРСИИ .....	7
2.5. ЗАЩО Е ТОЛКОВА ПОПУЛЯРЕН? .....	7
2.5.1. Безплатен е .....	7
2.6. ПРЕДИМСТВО ПРЕД ДРУГИТЕ ОПЕРАЦИОННИ СИСТЕМИ .....	8
2.7. ПО – ДОСТЪПЕН ЗА ВСЯКАКЪВ ВИД ХОРА .....	8
2.8. RESTFUL .....	9
2.9. RESTFUL WEB SERVICES .....	9
<b>3. АРХИТЕКТУРА.....</b>	<b>11</b>
3.1. ДИАГРАМИ ЗА ПРИЛОЖЕНИЕТО .....	11
3.1.1. Use-Case Диаграма за базови действия за студент .....	11
3.1.2. Use-Case Диаграма за базови действия за преподавател .....	12
3.1.3. Activity Диаграма за повече информация за приложението.....	13
3.2. ДИАГРАМА НА БАЗАТА ДАННИ ЗА СЪРВЪРА.....	15
3.3. ТАБЛИЦА, ОПИСВАЩА ПО-ПОДРОБНО БАЗАТА ДАННИ НА СЪРВЪРА.....	16
<b>4. РАЗРАБОТКА .....</b>	<b>20</b>
4.1. СЪРВЪР.....	21
4.2. КЛИЕНТ .....	32

<b>5. РЪКОВОДСТВО НА ПОТРЕБИТЕЛЯ.....</b>	<b>44</b>
5.1. ИНСТАЛИРАНЕ НА ПРИЛОЖЕНИЕТО .....	44
5.2. ВЛИЗНЕ НА СТУДЕНТ В ПРИЛОЖЕНИЕТО .....	45
5.3. РЕГИСТРИРАНЕ НА СТУДЕНТ ЗА ПРИЛОЖЕНИЕТО .....	47
5.4. РЕГИСТРИРАНЕ НА ПРЕПОДАВАТЕЛ ЗА ПРИЛОЖЕНИЕТО.....	<b>ГРЕШКА!</b>
<b>ПОКАЗАЛЕЦЪТ НЕ Е ДЕФИНИРАН.</b>	
5.5. ВЛИЗАНЕ НА ПРЕПОДАВАТЕЛ В ПРИЛОЖЕНИЕТО .....	51
<b>6. ЗАКЛЮЧЕНИЕ .....</b>	<b>54</b>
<b>7. ИЗПОЛЗВАНА ЛИТЕРАТУРА .....</b>	<b>59</b>
7.1. АНГЛИЙСКА ЛИТЕРАТУРА:.....	55
7.2. БЪЛГАРСКА ЛИТЕРАТУРА: .....	55
7.3. ПОМОЩНИ СТРАНИЦИ И ЛИНКОВЕ: .....	55

## 1. Увод

През последните години, все повече се повишава вниманието към технологизацията на образованието. Това може да се обясни с нарастването на неговата роля за социално-икономическото развитие на обществото. Все по-определено се осъзнава, че от равнището и качеството на образованието зависи състоянието на цялото общество. Развитието на образованието способства за развитието на науката, което на свой ред осигурява разработка и внедряване на нови технологии.

С течение на времето метода за проверка на знанията се е променил, от обикновеното „изпитване на дъската”, той е прераснал в електронен вариант, десктоп приложения и дори мобилни приложения, които спокойно и доста удобно се използват.

### 1.1. Цел

Изискванията за софтуера, посочени в този документ са за апликация – Андроид тестова игра ( Android testing game ) главно за Android OS. Тя е разширение на предишно приложение със заглавие „Мобилна тестова система“.

Тя е средство за проверка на знанията и научения материал по избрана дисциплина. Като за целта в тази разширена версия ( Андроид тестова игра ) ще бъде посветен и собствен учителски панел, за да може тестовете да бъдат оценявани.

### 1.2. Задачи

Андроид тестова игра е предназначена за използване от:

- деца от началното училище
- ученици от средното училище
- ученици от основно училище
- студенти от висшето образование

### 1.3. Хардуерни изисквания

Първоначално за приложението ще е необходимо устройство на което да го инсталирате и използвате. Както се споменава по-долу, по-добре е да имате телефон с по - висока версия андроид операционна система.

Когато приложението не може да се набави от студента и евентуално от преподавателя чрез изтегляне на необходимата версия от Google Play Store, тогава ще му се наложи чрез USB или Bluetooth да се свърже с преносимия си компютър, за да го качи на телефона и съответно да го инсталира и използва.

### 1.4. Софтуерни изисквания

За да се инсталира клиентското приложение, студентът и преподавателя трябва да имат телефон с андроид операционна система, която „съпортва” поне API LEVEL 19 ( така познатата версия kit kat 4.4 ), но е за предпочитане и по - виска API LEVEL заради добавените нови ефекти, които се поддържат в по – високите LEVEL-И.

Също може да се свали от github с git bash, като се използва командата git clone и за линк се използва линка : <https://github.com/sashe944/TestV2-Apk-file.git>

Преподавателите могат да получат достъп до цялостното приложение като използват линка за сваляно отново от github : <https://github.com/sashe944/TestV2.git>

Линк за свалянето на сървърната част от github:

<https://github.com/sashe944/TestV2Server.git>

## 2. Използвани технологии

### 2.1. JAVA като програмен език ?

#### 2.1.1. Що е JAVA ?

Противно на представите на много хора java и javaScript са различни езици. Какви точно са разликите няма да задълбавам, а ще продължа по същество. Java е обектно-ориентиран език създаден от Sun Microsystems.

По синтаксис прилича на C++, но за разлика от него java е платформено независим език. Как се получава тази независимост?. Програмният код, написан на java не се превежда до машинен код, а само се компилира до така наречения *"байткод"* в .class файл.

За четенето и изпълнението на байткода отговаря Java виртуална машина (Java Virtual Machine), която е необходимо да бъде инсталирана. В по-ранните версии на java виртуалната машина черпи допълнително процесорно време за собственото си изпълнение, но в сегашните версии това е вече незабележимо (разликата е под милисекунда).

### 2.2. Андроид

Мобилният свят цъфти. Всъщност, с нарастването на търсенето на смартфони, светът всъщност се е свил в ръката ви. Сега на пазара има разнообразие от смартфони, които се хранят от различни операционни системи. Войната в пазара на смартфони не е само между производителите на хардуер, но и между производителите на операционни системи за мобилни телефони. Google Android, Microsoft и Apple са водещите производители на OS за смартфона.

Въпреки че има твърде много конкуренти за пазара, Android на Google е преодолел всяка друга компания и сега се е превърнал във водещ производител на операционни системи за смартфони. Това е наистина забележително за една компания да постигне това в много кратко време. Ще го намерите наистина интересно, ако знаете какво стои зад популярността на Android.

## 2.3. Какво е Андроид?

Андроид е операционна система, не само за смартфони, но и за таблети, телевизори и часовници наречени още wearables. Базирана е на Linux и е разработена от Open Handset Alliance консорциум от над 80 водещи хардуерни компании като Sony, Samsung и софтуерни компании, част от Google.

## 2.4. Версии

Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich (ICS), JellyBean, KitKat, Lollipop, Marshmallow, Nouget, Oreo – Това са версиите на Андроид, за различните смартфони. Oreo е последната излязла версия и се поддържа от Samsung Galaxy S8/S8 plus.

За последното излезло устройство от Самсунг ( Samsung Galaxy S9/S9 plus) все още е с Андроид Oreo(8.0).

## 2.5. Защо е толкова популярен?

### 2.5.1. Безплатен е

От момента в който излезе, Android OS е безплатен и ще остане безплатен и за в бъдеще. Това решение на Google привлече вниманието на много производители на хардуер из цял свят. Тази популярност на Android OS се дължи предимно на факта, че е безплатна. Това позволява да Google да си партнира с много известни производители на хардуер.

### 2.5.2. С отворен код е

Android OS е също с отворен код. За разлика от други операционни системи, които са защитени от различни процеси за защита, плагиатство и други подобни, Google решава Android да е отворена OS. С това Google позволява на активните програмисти от цял свят да подобрят системата. Все повече нови и нови идеи се имплементират от умни програмисти.

### **2.5.3. Може да се създават собствени приложения**

Ако собствениците на смартфон или таблет с Android OS са заинтересовани от създаването на свои собствени приложения, това не е никакъв проблем, дори и без никакви познания по програмиране. Google предоставя набор от инструменти, които са безплатни и позволяват на потребителите да създават свои приложения с помощта на **Software Development Kit** за Android App Development.

### **2.5.4. Google Маркет**

Google разпространява всички приложения за телефони с Android устройства през своя Google Play (преди Android Market). Тя позволява на потребителите да тестват, използват и споделят различни приложения и да ги препоръчват на другите потребители. В магазина има огромен набор от приложения. Някои от тези приложения са достъпни безплатно, а някои се заплащат. Потребителите на смартфон с Android OS можете да посетят Google Play.

## **2.6. Предимство пред другите Операционни системи**

Причините, които казах по-горе, ви разясняват защо Android е различен от другите. Всъщност не само това е достатъчно, за да бъде най-добрият играч на пазара, който побеждава тежка конкуренция. Android има големи предимства пред другите производители на операционни системи.

## **2.7. По – достъпен за всякакъв вид хора**

Android OS е насочена към обикновения човек, който иска да получи максимални възможности на ниска цена. Дори сега много хора мислят, че iPhone е за богати. Тези обикновени хора, които искат да имат свои собствени смартфони, могат да притежават смартфон, работещ под Android, който да е под техния бюджет.



Освен това iPhone без jailbreaking не приема много SIM, докато Android ще поддържа всяка SIM карта. Това е най-големият плюс за Android.

## 2.8. Restful

### 2.8.1. Какво означава REST?

REST означава **RE**presentational **S**tate **T**ransfer. REST е уеб базирана архитектура която използва HTTP протокол за комуникация на данни. Тя се върти около ресурси, където всеки компонент е ресурс и достъпът до ресурс се осъществява чрез общ интерфейс, използващ стандартни HTTP методи.

В архитектурата REST, един REST сървър просто осигурява достъп до ресурси, а REST клиентът достъпва и ги представя. Тук всеки ресурс се представя като URI-и Глобални ID-та. REST използва различни репрезентации, за да представи ресурсите като Text, JSON и XML. JSON е най-популярния начин за репрезентация на данни за Уеб Сървисис.

### 2.8.2. HTTP Методи

Тези HTTP методи са най – често използвани в REST архитектурата.

- **GET** – Прочитане на ресурс.
- **PUT** – Използва Прехвърля ресурс към сървъра.
- **DELETE** – Унищожава ресурс.
- **POST** – Добавя към съществуващ ресурс на сървъра.

## 2.9. RESTFul Web Services

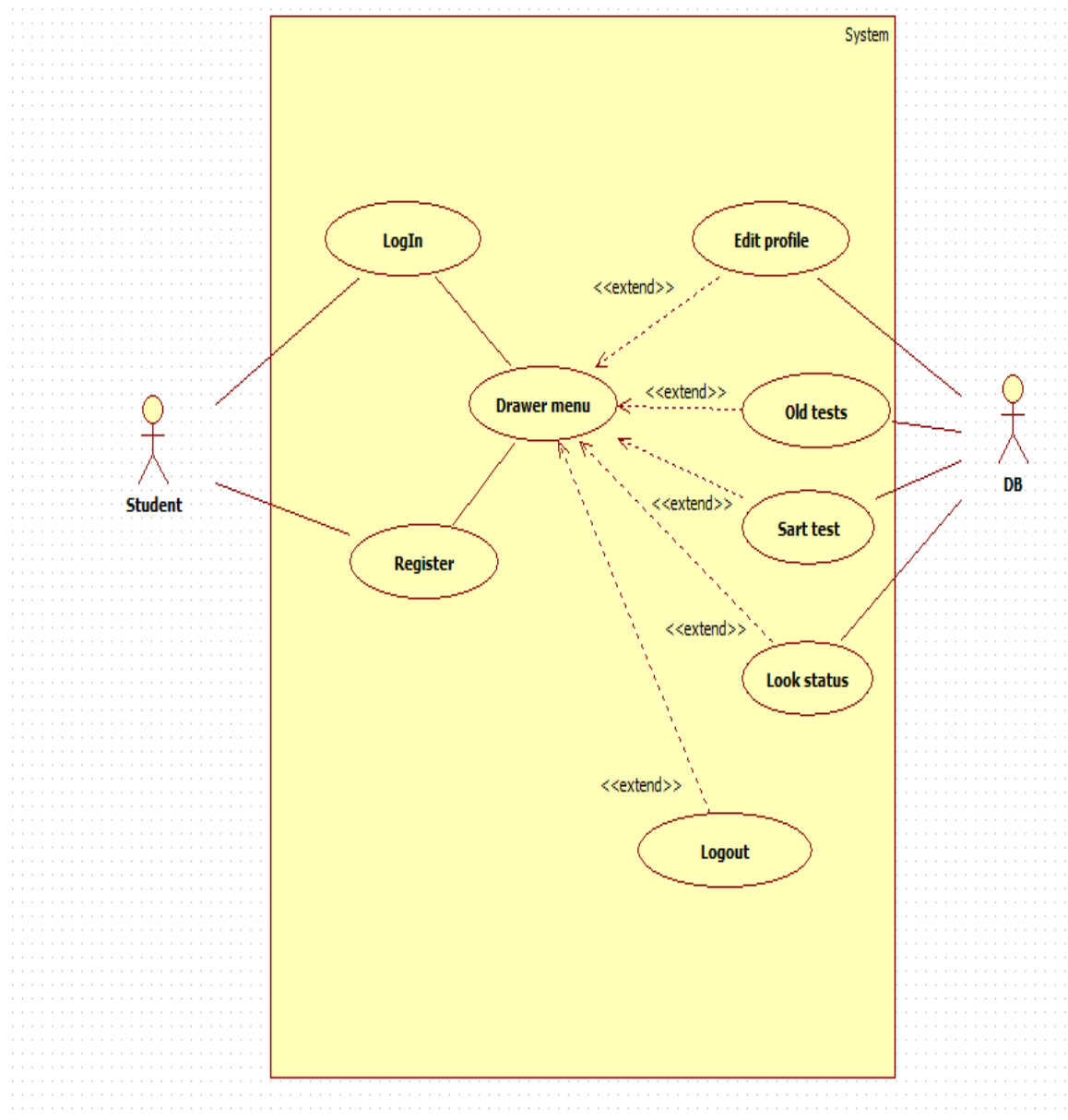
Уеб услугата е колекция от отворени протоколи и стандарти, използвани за обмен на данни между приложения или системи. Софтуерните приложения, написани на различни програмни езици и работещи на различни платформи, могат да използват уеб услуги за обмен на данни чрез компютърни мрежи като интернет по начин, подобен на комуникацията между процесите на един компютър.

Тази оперативна съвместимост (например между Java и Python или Windows и Linux приложения) се дължи на използването на отворени стандарти. Уеб услугите, базирани на REST Architecture, са известни като RESTful Web Services. Тези веб услуги използват HTTP методи за реализиране на концепцията за REST архитектура. Универсалната веб услуга обикновено определя URI (Uniform Resource Identifier), която е услуга, осигуряваща представяне на ресурси като JSON и набор от HTTP методи.

### 3. Архитектура

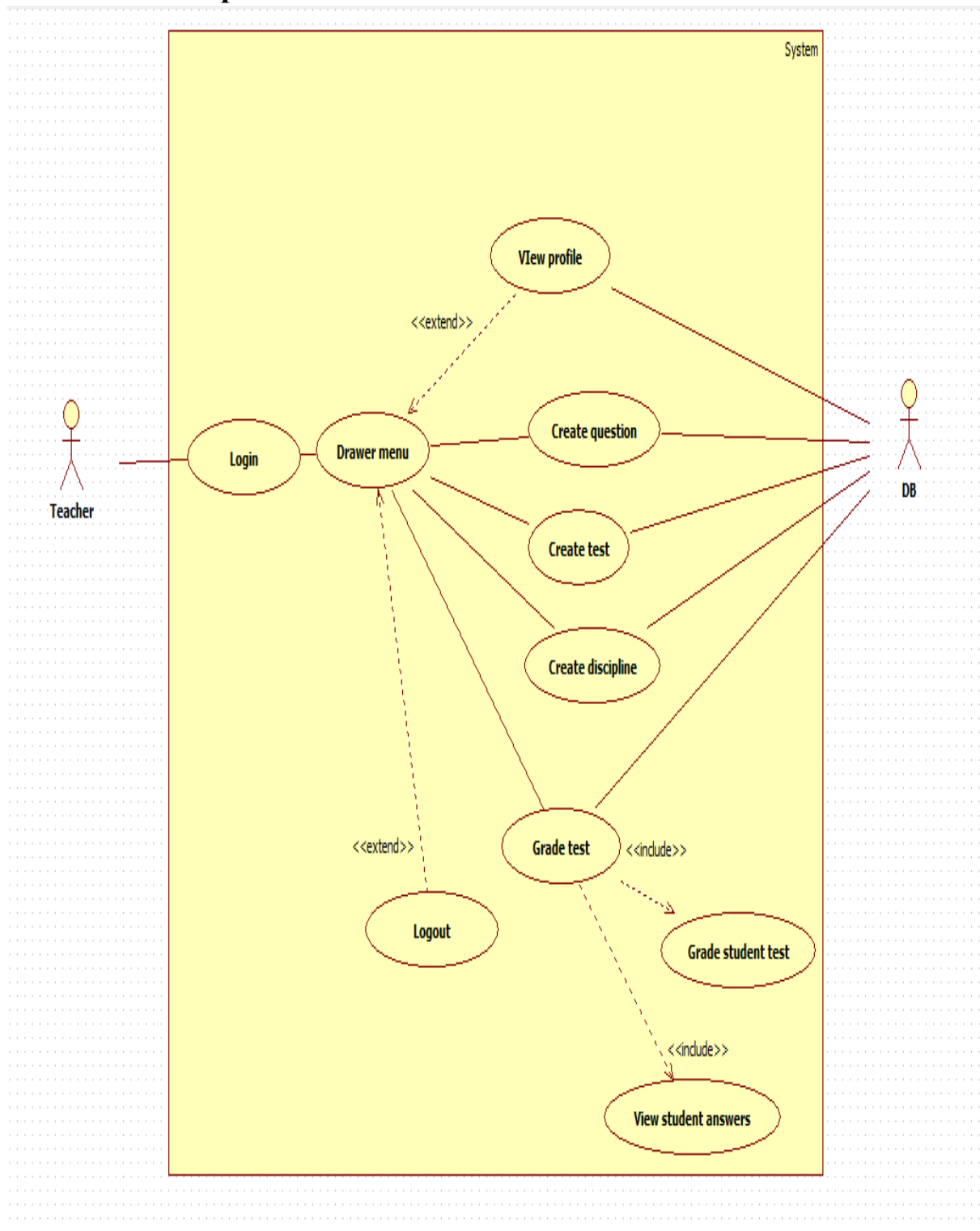
#### 3.1. Диаграми за приложението

##### 3.1.1. Use-Case Диаграма за базови действия за студент



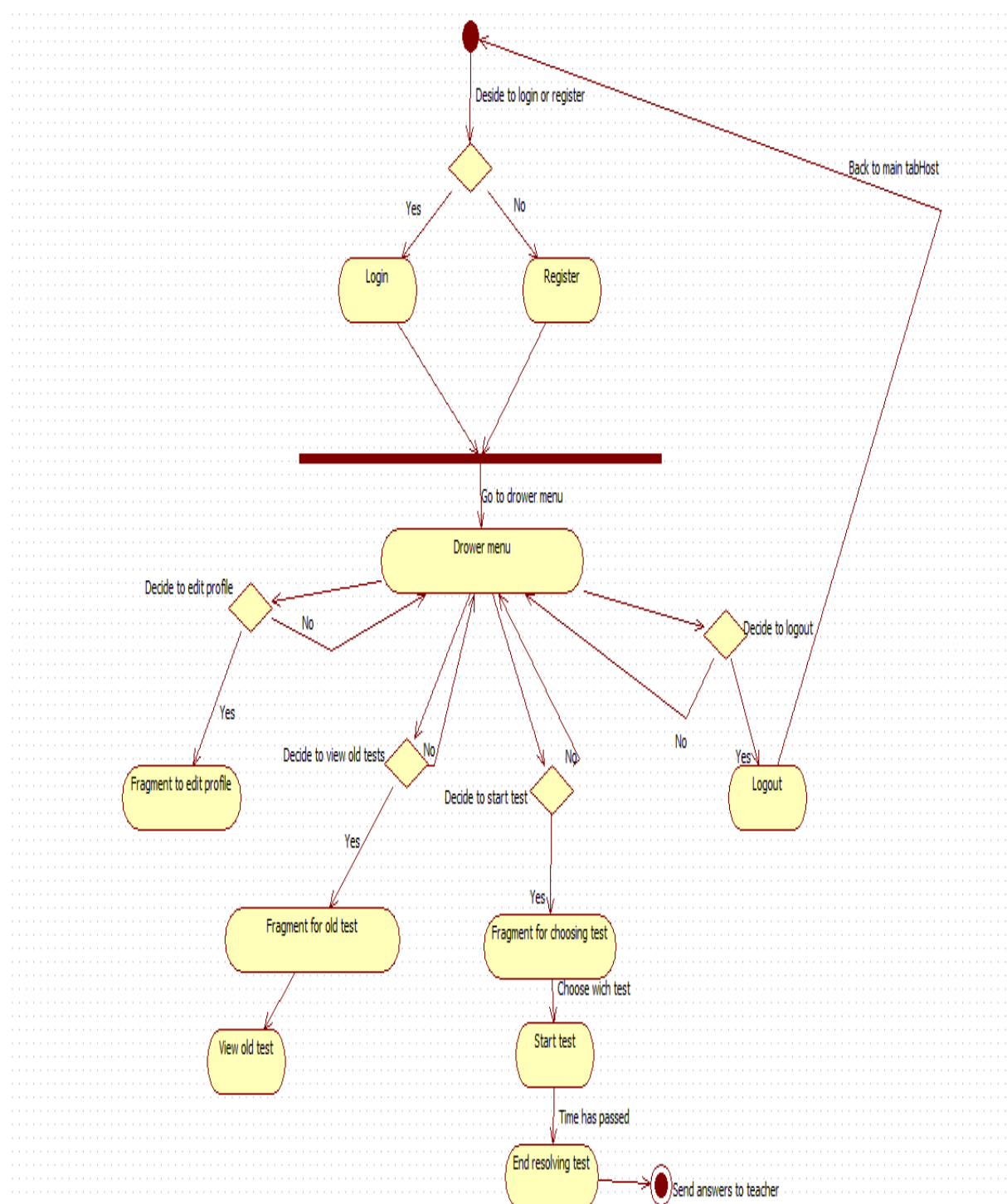
Фигура 1 Упростена Use-Case Диаграма за приложението Тя показва елементарната работа на студента чрез тази use-case диаграма

### 3.1.2. Use-Case Диаграма за базови действия за преподавател



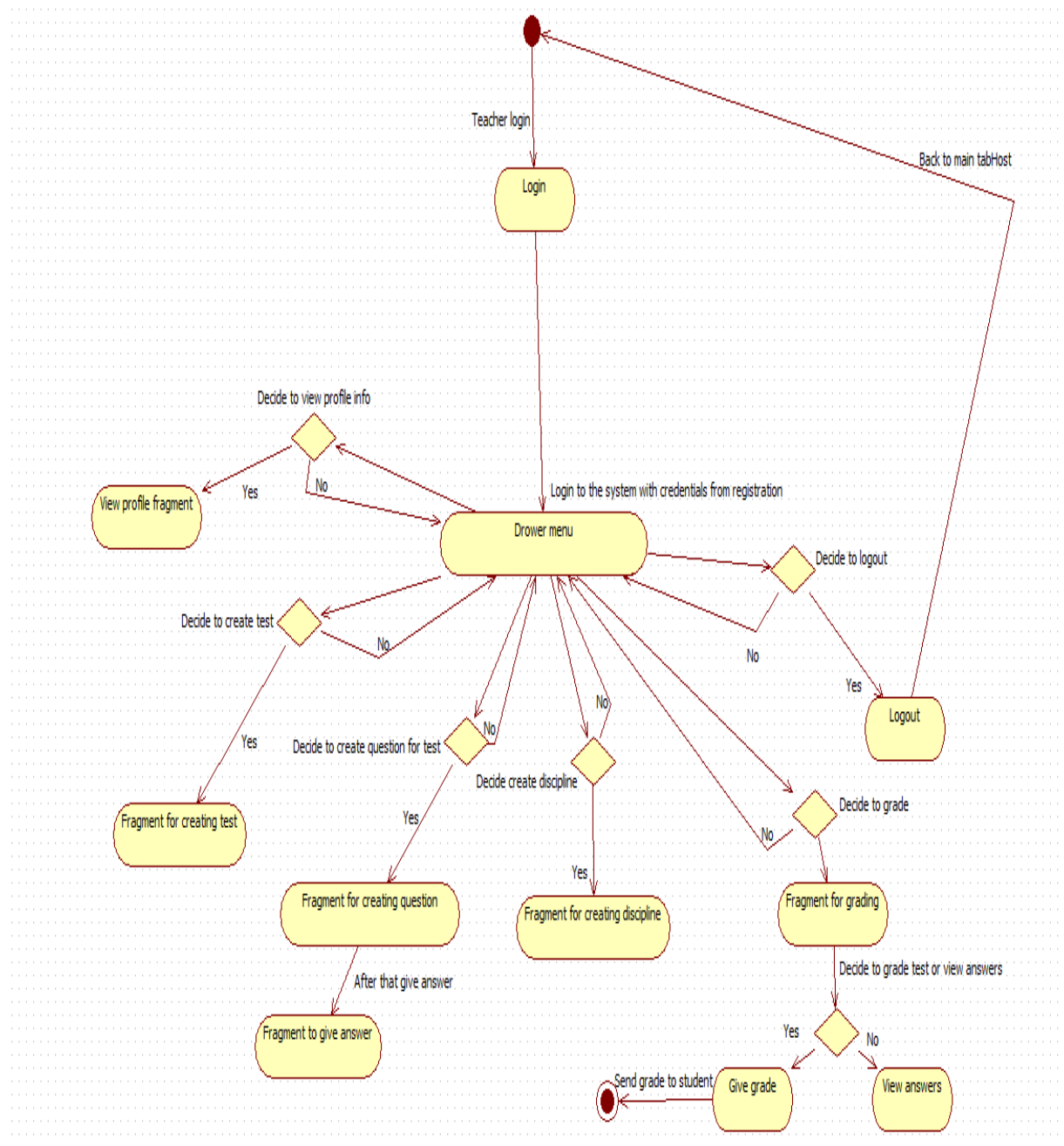
Фигура 2 Упростена Use-Case Диаграма за приложението отнасяща се за преподавател

**Фигура 2.** Activity Диаграма за повече информация за студент



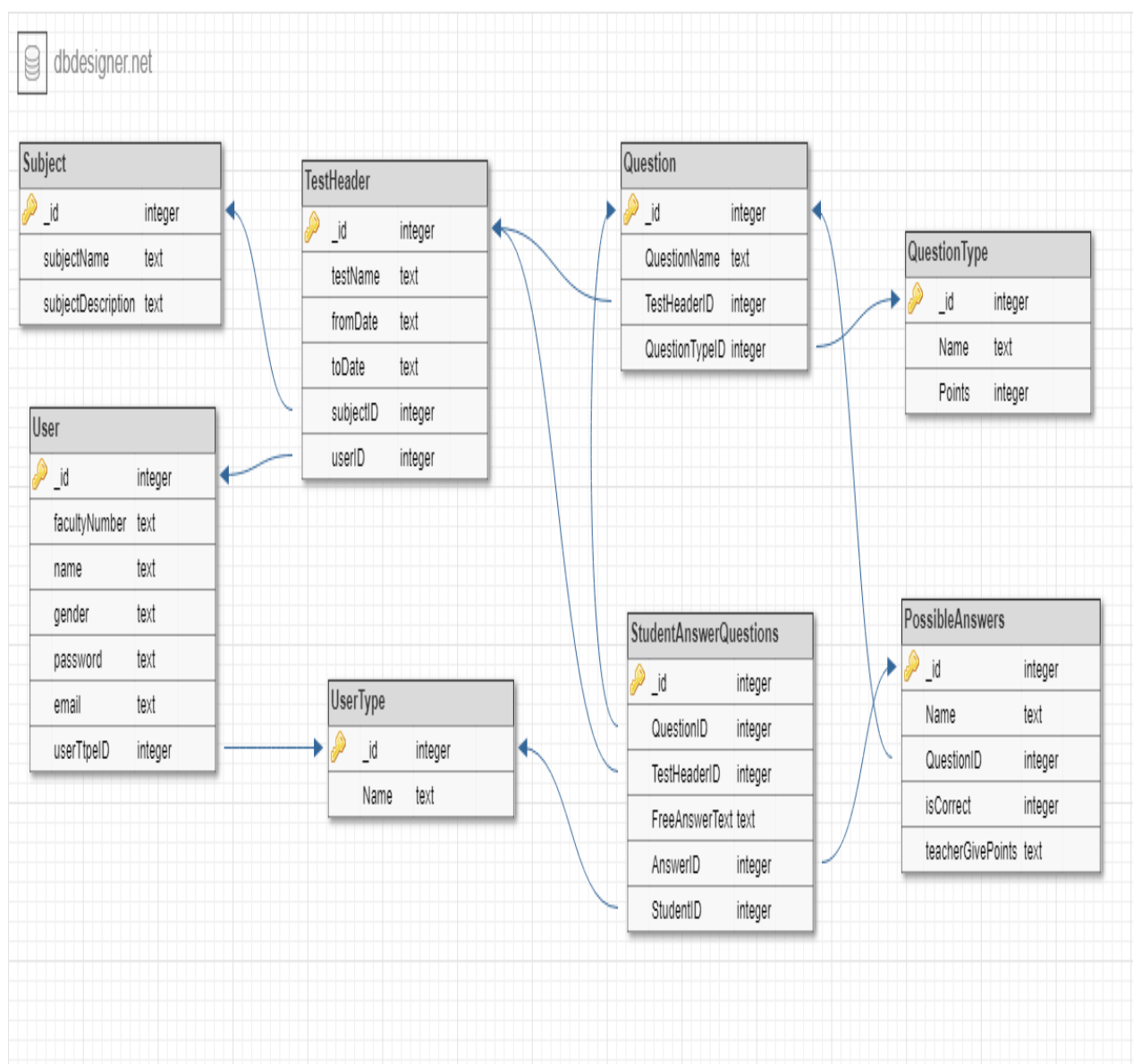
**Фигура 3** Activity Диаграма за студент Тази диаграма уличетворява малко по-подробната работа на студент и студентски панел, през какви стъпки трябва да премине студента за пълноценна работа.

### 3.1.3. Activity Диаграма за повече информация за преподавател



Фигура 4 Тази диаграма уличетворява малко по-подробната работа на преподавател и преподавателски панел, през какви стъпки трябва да премине преподавател за пълноценна работа.

### 3.2. Диаграма на базата данни за сървъра



Фигура 5 Описание на SQL-ите външна база данни за приложението. Същата база данни е описана по-долу за всяка таблица и поле от таблицата.

### 3.3. Таблица, описваща по-подробно базата данни на сървъра

Име на таблицата	Полета в таблиците	Цел на полетата
User	<p><code>_id</code> INTEGER PRIMARY KEY AUTOINCREMENT</p> <p><code>facultyNumber</code> TEXT</p> <p><code>name</code> TEXT</p> <p><code>gender</code> TEXT</p> <p><code>password</code> TEXT</p> <p><code>email</code> TEXT</p> <p><code>userId</code> INTEGER</p>	<p><code>_id</code> за даден User, което <code>_id</code> се увеличава всеки път с 1 за всеки нов запис в таблицата User.</p> <p>Това поле служи за записване на факултетния номер на студент, ако естествено user-а е от тип студент. То се използва от студент</p> <p>Това поле е за записването на името на user-а .</p> <p>Това поле се използва за определянето на пола на User</p> <p>Това поле се използва за запазването на паролата на User.</p> <p>Това поле се използва от преподавателя</p> <p>Това поле служи като външен ключ за таблицата UserType в която се определя дали user-а е студент или преподавател</p>
UserType	<p><code>_id</code> INTEGER PRIMARY KEY</p> <p><code>Name</code> TEXT</p>	<p><code>_id</code> за даден Type на User, с него може лесно да се определи дали даден user е студент или преподавател</p> <p>Това поле се използва, за да може да запомним на кое <code>_id</code> ще е преподавател и на кое ще е студент, в него с текст ги записваме.</p>



Question	<p><b>_id</b> INTEGER PRIMARY KEY AUTOINCREMENT</p> <p>QuestionName TEXT</p> <p>QuestionTypeID INTEGER</p> <p>TestHeaderID INTEGER</p>	<p>_id за даден въпрос за студент. Той отново е AUTOINCREMENT, което означава, че с всеки нов въпрос __id се увеличава с 1</p> <p>Това поле може да се нарече още и Question, целта му е да записва дадения въпрос.</p> <p>Това е поле, което служи за външен ключ за таблицата QuestionType.</p> <p>Това поле е външен ключ към таблицата TestHeader</p>
QuestionType	<p><b>_id</b> INTEGER PRIMARY KEY AUTOINCREMENT</p> <p>Name TEXT</p> <p>Points INTEGER</p>	<p>_id за даден въпрос за студент. Той отново е AUTOINCREMENT, което означава, че с всеки нов въпрос __id се увеличава с 1</p> <p>Това поле репрезентира типовете на въпросите написани в текстов формат</p> <p>Това поле служи за записването на точките за даденият тип на въпроса( 3 типа )</p>

Subject	<p>_id INTEGER PRIMARY KEY AUTOINCREMENT</p> <p>subjectName TEXT</p> <p>subjectDescription TEXT</p>	<p>_id за таблицата както и първичен ключ, то отново е AUTOINCREMENT</p> <p>Това поле се използва за записването на името на предмет(дисциплина) за тест.</p> <p>Това поле се използва за записване на описанието на новосъздадена дисциплина.</p>
PossibleAnswers	<p>_id INTEGER PRIMARY KEY AUTOINCREMENT</p> <p>Name TEXT</p> <p>QuestionID INTEGER</p> <p>isCorrect INTEGER</p>	<p>_id за таблицата както и първичен ключ, то отново е AUTOINCREMENT</p> <p>Това поле е външен ключ към таблицата Question</p> <p>Това поле се използва за проверка(дали отговорът даден от студента е правилен или не е правилен)</p>

TestHeader	<p><b>_id</b> INTEGER PRIMARY KEY AUTOINCREMENT</p> <p>testName TEXT</p> <p>fromDate TEXT</p> <p>toDate TEXT</p> <p>subjectID</p> <p>userID</p>	<p><b>_id</b> за таблицата както и първичен ключ, то отново е AUTOINCREMENT</p> <p>Това поле репрезентира поле за запазване на името на съответният тест</p> <p>Това поле репрезентира поле за запазване на датата за начало на съответният тест</p> <p>Това поле репрезентира поле за запазване на датата за край на съответният тест</p> <p>Това поле репрезентира поле за референцирането на дадената дисциплина за съответният тест. Външно поле за таблицата Subject</p> <p>Това поле репрезентира поле за референцирането на даден студент за съответният тест. Външно поле за таблицата User.</p>
StudentAnswerQuestions	<p><b>_id</b> INTEGER PRIMARY KEY AUTOINCREMENT</p> <p>QuestionID INTEGER</p> <p>AnswerID INTEGER</p> <p>FreeAnswerText TEXT</p> <p>TestHeaderID INTEGER</p> <p>StudentID INTEGER</p>	<p><b>_id</b> за таблицата както и първичен ключ, то отново е AUTOINCREMENT</p> <p>Външно поле към таблицата Question</p> <p>Външно поле към таблицата PossibleAnswers</p> <p>Външно поле към таблицата TestHeader</p> <p>Външно поле към таблицата UserTupe</p>

## 4. Разработка

В тази точка ще се разяснят клиентската и сървърната части от клиент – сървър приложението.

Клиента е написан на Android Studio

Сървъра е написан на Eclipse (JAVA)

Всички файлове за приложението са разположени в 5 пакета(packages)

Пакетът може да бъде дефиниран като група от свързани типове, осигуряващи защита на достъпа и управление на namespace-овете. Използват се в Java, за да се предотвратят конфликтите в именуване, да се контролира достъпът, да се направи по-лесно търсене / локализиране и използване на класове, интерфейси, изброявания(enumerations) и пояснения(annotations).

Единият от пакетите (connection) е по-скоро за улеснение на потребителя. Той съдържа в себе си клас единствено за свързването до SQL-ite база данни. Неговата цел е да показва на потребителя, използващ този сървър дали е пуснат за използване, ако не се е занимавал до сега със сървлети.

```
public class DBConnection {
    Connection DatabaseConnection = null;
    public static void main(String[] args) {

    }
    public boolean DatabaseConnection(){
        try{
            Class.forName("org.sqlite.JDBC");
            DatabaseConnection=DriverManager.getConnection("jdbc:sqlite:/C:/Users/Home/Desktop/TestV2.db");
            return true;
        }
        catch(Exception e)
        {
            e.printStackTrace();
            return false;
        }finally{
            try {
                DatabaseConnection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Фигура 6. Булев метод, който проверява дали връзката с базата данни се усъществува.

Този булев метод се използва по-късно в сървлета за пускането на сървърната част от приложението.

Тук се вижда как се използва JDBC за осъществяването на връзка с SQLite база данни.

Самият Сървър е комбинация от останалите четири пакета( objects, services, server ,servlets), като разбира се не броим допълнителния пети пакет (connection ) състоящ се от един булев клас за проверка в сървъра, както по-горе уточнихме.

#### 4.1. Сървър

Първият пакет необходим за сървъра е наречен server и съдържа servlet за стартиране на сървъра с код:

```
@WebServlet("/TestV2Server")
public class TestV2Server extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public TestV2Server() {
        super();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        response.setContentType("text/html; charset=utf-8");
        DBConnection connection = new DBConnection();
        if(connection.DatabaseConnection()){
            response.sendRedirect("Success.jsp");
        }else{
            response.sendRedirect("Error.jsp");
        }
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doGet(request, response);
    }
}
```

Фигура 7. Стартов Servlet за приложението. С него се стартира сървъра

Той се използва за пускане на server-a, но евентуално може да се стартират по отделно самите разработени сървлети показани по-долу.

Неговата идея е просто да улесни потребителя с едно добре форматирано съобщение, за да не се чуди той излишно стартирал ли е сървъра.

В doGet метода на servlet-a за server-a има инстанция connection, от тип DBConnection за връзка с базата данни, необходима за извличане на данни.

Ако в doGet метода всичко премине успешно, без грешки ще се отвори jsp страницата success със съобщение за успешно съзване:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Successful connection</title>
</head>
<body> <%
    out.println("Database connection established succesfully!");
%>
</body>
</html>
```

Фигура 8. Съобщение за успех

Ако нещо се обърка ще се отвори error.jsp:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Error connection</title>
</head>
<body>
<%
    out.println("Database connection was not established successfully!");
%>
</body>
</html>
```

Фигур 9. Съобщение за грешка

След като вече server-а е пуснат да работи ще се обърне внимани на Влизане(Login) и Регистрация(Register) за потребител(в случая на студент), защото за преподавател Register се извършва от специално за целта сайт.

За да се направи Влизане или Регистрация, първо ще трябва да има създаден за целта обект, който е поставен във втория от четирите пакета(objects) User:

```
public class User {
public long id;
    public String facultyNumber, name, gender, password, userTypeID;
public String toString() {
    return name;
}
```

Фигура 10. Обекта необходим за Регистрация и Влизане в приложението

Обекта User вече е на лице и съдържа цялата информация, която е необходима за студент, следващата стъпка е да се направи FindUserService за базата данни, като той ще се намира в третия(services) от четирите пакета необходими за цялостното приложение.

```
public class FindUserService {
    public User find(String Password,String FacultyNumber){
        User user = new User(); Connection conn = null;Statement stmt = null;
        try{
            Class.forName("org.sqlite.JDBC");
            conn=DriverManager.getConnection("jdbc:sqlite:/C:/Users/Home/Desktop/Test
V2.db");
            conn.setAutoCommit(false);
            stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM User WHERE"
                + " "+Constants.USER_PASSWORD+" = \"" + Password +
                "\" AND "+Constants.USER_FACULTY_NUMBER+"=\"\"
+FacultyNumber + "\"");
            while (rs.next()){
                user.id = rs.getInt("_id");
                user.facultyNumber = rs.getString("FacultyNumber");
                user.name = rs.getString("Name");
                user.gender = rs.getString("Gender");
                user.password = rs.getString("Password");
                user.userId = rs.getString("UserID");
                System.out.println(user.toString());
                return user;} }
            catch(Exception e){
                e.printStackTrace();
            } finally {
                try {
                    stmt.close();
                    conn.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            return
null;
```

Фигура 11. Сървис(FindUserService) за работа с базата данни за потребител. Той се използва за основни операции свързани с базата данни.

Той се използва в UserLoginServlet пригоден за тази цел (Login на потребителя):

```
@WebServlet("/") + Constants.USER_URL_LOGIN)
public class UserLoginServlet extends HttpServlet
{
    private static GsonBuilder gson_builder = new
GsonBuilder().serializeNulls().setDateFormat("MM/dd/yyyy");
    private FindUserService findUserService;
    public UserLoginServlet() {findUserService = new FindUserService();}

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    }
    protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
        String jsonRequest= getBody(request);
        User reqCredentials = new
GsonBuilder().create().fromJson(jsonRequest, User.class);
        String Password = reqCredentials.Password;
        String FacultyNumber = reqCredentials.FacultyNumber;
        User user = findUserService.find(Password, FacultyNumber);
        if (user != null) {
            response.setContentType("application/json;charset=UTF-8");
            Gson gson = gson_builder.create();
            response.getWriter().write(gson.toJson(user));
        }
        else {
            request.setAttribute("error", "Unknown user, please try again");
        }
    }
}
```

Фигура 12. Сървлет отговарящ за Login на студент в програмата

Тук освен, че се вижда как се използва findUserService в него(servlet-a) се забелязва, че се използва метода doPost(), идята зад използването му е вместо приложението да може да се “хаква“ когато се използва doGet() то да е по сигурно и защитено.

Освен това се използва 3<sup>rd</sup> party library(Gson), която се грижи за конструирането на JSON обект от User класа и де-конструирането, когато клиента го потърси ( чрез интернет ).

Кратко допълнение Servlet-ите използвани тук са и част от четвъртия и полседен пакет(servlets) за приложението.

Вижда се отделно как се използва и друг метод наречен getBody(),който служи за тяло на HTTP POST заявката за UserLoginServlet.



```
public static String getBody(HttpServletRequest request) throws IOException {
    String body = null;
    StringBuilder stringBuilder = new StringBuilder();

    BufferedReader bufferedReader = null;
    try {
        InputStream inputStream = request.getInputStream();
        if (inputStream != null) {
            bufferedReader = new BufferedReader(new
InputStreamReader(inputStream, "UTF-8"));
            char[] charBuffer = new char[128];
            int bytesRead = -1;
            while ((bytesRead = bufferedReader.read(charBuffer)) > 0) {
                stringBuilder.append(charBuffer, 0, bytesRead);
            }
            } else {
                stringBuilder.append("");
            }
        } catch (IOException ex) {
            throw ex;
        } finally {
            if (bufferedReader != null) {
                try {
                    bufferedReader.close();
                } catch (IOException ex) {
                    throw ex;
                }
            }
        }

        body = stringBuilder.toString();
        return body;
    }
}
```

**Фигура 13.** Тази фигура показва метода getBody(), а той се използва за самата HTTP POST заявка (body на заявката) .

Забелязва се използването на допълнителен клас „Constants”. В него се намират всички важни думи, които може да се объркат при просто записване като тип String(колони от базата данни, URL за servlet-ите, наименования на таблиците от базата данни). За това се използва и този допълнителен клас, както във всеки servlet така и във всеки service.

```

public class RegisterUserService {
    public User register(String FacultyNumber,String Name,String Password,
String Gender,String UserID){
        Connection connection = null;
        User user = null;
        PreparedStatement statement = null;
        try {
            final String sql = "INSERT INTO User
(FacultyNumber,Name,Password,Gender,UserID) VALUES (?, ?, ?, ?, ?)";
            connection=DriverManager.getConnection("jdbc:sqlite:/C:/Users/Home/Desktop/Test
V2.db");
            statement = connection.prepareStatement(sql);
            statement.setString(1,FacultyNumber );
            statement.setString(2, Name);
            statement.setString(3, Password);
            statement.setString(4, Gender);
            statement.setString(5,UserID);
            statement.executeUpdate();
            FindUserService findUserService = new FindUserService();
            user = findUserService.find(Password,FacultyNumber);
            System.out.println("register user: " + user);
        } catch (SQLException e) {

            e.printStackTrace();
        } finally {
            try {
                statement.close();
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        return user;
    }
}

```

Фигура 14. Тази фигура показва как се регистрира студент във външната база данни посредством RegisterUserService.

След като се разясни как работи Влизането ( поне от сървърната страна на приложението ) е време да се разгледа как работи и Регистрацията.

Самата регистрация работи по доста подобен начин (т.е. и тя използва service(s) и съответно servlet(s)).

Фигурата по-горе показва Service, който се грижи за Регистрация на потребител за приложението.

В този service единственото интересно нещо е, че се използва и друг service, FindUserService, който е и необходим за Влизането в системата . Той се използва с идеята да не се преповтаря писането на код за приложението, иначе всичко друго е по същата идеология.

```

@WebServlet("/") + Constants.USER_URL_REGISTER)
public class UserRegisterServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private static GsonBuilder gson_builder = new
GsonBuilder().serializeNulls().setDateFormat("MM/dd/yyyy");

    private RegisterUserService registerUserService;
    public UserRegisterServlet() {
        super();
        registerUserService = new RegisterUserService();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

    }

    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        String jsonRequest = UserLoginServlet.getBody(request);
        User reqCredentials = new
GsonBuilder().create().fromJson(jsonRequest, User.class);
        User user = new User();
        String FacultyNumber = reqCredentials.facultyNumber;
        String Name = reqCredentials.name;
        String Password = reqCredentials.password;
        String Gender = reqCredentials.gender;
        String UserID = reqCredentials.userID;
        user = registerUserService.register(FacultyNumber, Name, Password,
Gender, UserID);
        if (user != null) {
            response.setContentType("application/json; charset=UTF-8");
            Gson gson = gson_builder.create();
            response.getWriter().write(gson.toJson(user));
        }
        else {
            request.setAttribute("error", "Registration was unsuccessful!");
        }
    }
}

```

Фигура 15. Servlet за регистрирането на потребител в системата.

Тук отново се вижда как се използва service. В UserRegisterServlet се забелязва, че се използва метода doPost(), идята зад използването му е вместо приложението да може да се “хаква“ когато се използва doGet() то да е по сигурно и защитено.

С други думи тук се използва doPost(), за да се цели по-голяма сигурност при използването му.

Освен това се забелязва използването на 3<sup>rd</sup> party library( Gson), която се грижи за конструирането му в JSON от User класа и де-конструирането му, когато клиента го потърси ( чрез интернет ).

Допълнително се вижда как се използва HTTP POST, а в UserLoginServlet за влизане, поради същата идея, кодът на приложението да не става прекалено претрупан.

Освен влизането и регистрацията на студент(както и влизането на преподавател става по същият начин като влизането на студента, а вече регистрацията му става отделно в специален за целта сайт) , той може промени данните си, а за това се използват `service(UpdateUserService)` и `servlet(UserUpdateServlet)` за промяната на данните на студент.

След като направи промяната на данните си студент може да се заеме с решаването на тест създаден от преподавател и след като го реши данните се изпращат обратно на сървър за да могат по-нататък да се вземат от преподавател и да се разгледат резултатите от теста и евентуално да се постави оценка на самият тест.

Цялата тази дейност (записването на отговорите от теста) става в `RegisterStudentAnswerService` за `Service`, а за `Servlet` – `AnswerRegisterServlet`.

Като в `RegisterStudentAnswerService` данните получени от клиентското приложение се записват във `List<StudentAnswer>`.

`StudentAnswer` съответно е нов обект от пакет `objects` за приложението, както `RegisterStudentAnswerService` е част от пакета `services`, а `AnswerRegisterServlet` от `servlets`.

```
public class StudentAnswer {

    public long id, questioId, possibleAnswerId, studentId, testHeaderId;
    public String freeText;

    @Override
    public String toString() {
        return "StudentAnswer [id=" + id + ", questioId=" + questioId + ",
possibleAnswerId=" + possibleAnswerId
                + ", studentId=" + studentId + ", testHeaderId=" +
testHeaderId + ", freeText=" + freeText + "];"
    }
}
```

Фигура 16. Тук се вижда необходимият обект за конструиране и де-конструиране на JSON обект.

След като се разгледаха Обектите за преподавател(пак повтарям регистрирането му става от специален за целта сайт) и за студент ще се обърне внимание и на друг обект от пакета `objects`

Той се казва `Subject`:

```

public class Subject {
    public long id;
    public String name,description;

    @Override
    public String toString() {
        return name;
    }
}

```

Фигура 17. Обект необходим за създаването на Дисциплина.

Той е необходим за създаването на дисциплини ( и евентуално тестове ) от преподавателя за студента.

За създаването на дисциплини се използват съответните service(s)( FindSubjectService,RegisterSubjectService) и servlet(s)(SubjectFindServlet за използването му от клиента като опции за android spinner и SubjectRegisterServlet за регистриране на дисциплина).

Като FindSubjectService има 2 метода в него. Единият find() за откриване на самата дисциплина, а другият е List<Subject> findAllSubjects(), за да се записват всичките регистрирани дисциплини и по-късно в приложението да се избират от android spinner(падащо меню)

Дойде време да се създават тестове, те следват почти същата идеология като останалите. А именно използва се обекта TestHeader ( който между другото се използва като репрезентация на таблица със същото име от базата данни )

```

public class TestHeader {
    public long
id,gradeSingleAnswer,gradeMultipleAnswer,gradeFreeTextAnswer,subjectID,userID;
    public String testName,fromDate,toDate;
    @Override
    public String toString() {

        return testName;
    }
}

```

Фигура 18. Обекта TestHeader, който се използва за целите на приложението.

И също вече познатите Service(s) ( 2 на брой) и Servlet(s) ( 2 на брой ) от съответните пакети services и servlets, като в единият от сървисите ( FindTestService ) притежава 2 метода. Единият find() за откриване на самият тест а другият findAll(), който се използва за изобразяване на всичките тестове регистрирани от преподавателя за студента.

След създаденият тест ще се създадат и въпроси за този тест.

Това става с помощта на обект Question от пакета objects:

```
public class Question {
    public long id,questionTypeId,testHeaderId;
    public String name;
    public List<PossibleAnswer> possibleAnswers;
    public String toString() {
        return name;
    }
}
```

Фигура 19. Код за обекта Question, който се използва за целта на приложението.

Тук най-любопитното нещо, което се вижда е, че се използва List от тип PossibleAnswers. Той( Possible Answers) е друг обект специално направен за отговорите, а те са за трите типа (Въпроси, на базата на които се задават и отговори). За това е създаден List<>, да държи резултатите от базата за PossibleAnswers.

За самото създаване на въпрос се грижат отново Сървиси ( 2 на брой, RegisterQuestionService за регистриране на въпрос в базата данни, FindQuestionService, състоящ се от два метода, **findByQuestionId** за намирането на 1 въпрос от базата данни и **findByTestHeaderId** за намирането на повече от 1 въпрос за съответен тест от записан вече в базата данни) и Сървлети ( и те са 2 на брой ), като по-интересното тук е, че в един от сървлетите(QuestionRegisterServlet) се използва вместо 1 сървис - 2 сървиса.

За регистрирането на въпрос заедно със отговор

```
private RegisterQuestionService registerQuestionService;
private RegisterPossibleAnswersQuestionService registerPossibleAnswerService;
```

Фигура 20. Тази фигура показва, кои два service се използва във (QuestionRegisterServlet).

Остана да се спомене, че за типа на трите типа Въпроси ( на базата на тях се дават и отговорите )

Като тяхната идея е сравнително елементарна, а именно да се вземат данни за тях(въпрос със един верен отговор, въпрос с повече от един верни отговори, въпрос със свободен текст) и да се попълнят отново в съответният android spinner в клиентското приложение

За самите отговори се ползва PossibleAnswer обекта от пакет objects

```
public class PossibleAnswer {
    public long id,questionId,isCorrect, points;
    public String name;
    public String toString() {
        return name;
    }
}
```

Фигура 21. Тази фигура показва класа PossibleAnswer за работа с базата данни. Като той е репрезентация на колона в базата.

Тук отново се използват Services ( за работа с базата ) и Servlets ( за използване от клиентското приложение по-късно).

Service-тата са два на брой, както и Servlets са два на брой и те съответно могат да се намерят в пакети services и servlets

```
public class QuestionType {
    public int id,
        public String name;
        public String toString() {
            return name;
        }
}
```

Фигура 22. На тази фигура е показан обекта QuestionType, като той отново е репрезентация на таблица в базата.

QuestionType е обект, който не е динамичен (т.е. в базата на тази таблица вече има ръчно попълнени данни ) и тук идеята е да се вземат за използване на готово.

За това и има само един Сървис и съответно Сървлет, използващ този сървис.

```
public class FindQuestionTypeService {
    public List<QuestionType> findAll() {
        List<QuestionType> questionTypes = new ArrayList();
        Connection conn = null;
        Statement stmt = null;

        try{
            Class.forName("org.sqlite.JDBC");
            conn=DriverManager.getConnection("jdbc:sqlite:/C:/Users/Home/Desktop/Test
V2.db");

            conn.setAutoCommit(false);
            stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM
QuestionType");

            while (rs.next()){
                QuestionType testType = new QuestionType();
                testType.name = rs.getString("Name");
                testType.id = rs.getInt("_id");
                questionTypes.add(testType);
            }
        } catch(Exception e){
            e.printStackTrace();
        } finally {
            try {
                stmt.close();
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        return questionTypes;
    }
}
```

Фигура 23. Тази фигура показва сървиса за QuestionType, като по-късно той бива използван.

## 4.2. Клиент

Клиента е написан на Android Studio 3.0 специално за Андроид Операционна Сиситема (AndroidOS)

Той отново е разползжен в няколко папки(пакета) за по лесно разграничаване и намиране на необходимите файлове.

Първият пакет е с име Constants, съдържащ един единствен клас Constants:

```
public class Constants {  
    public static final String URL  
    ="http://10.168.160.102:8080/TestV2Server/";  
}
```

Фигура 24. Тази фигура изобразява класа Constants с една статична променлива URL за приложението, като URL служи за връзка със сървъра.

Както в Eclipse трябваше да се използват обекти (User, Teacher за пример) има и втори пакет с име objects, съдържащ всички необходими обекти за приложението

```
public class User {  
    public long id,userTypeID;  
    public String facultyNumber, name, password, gender;  
}
```

Фигура 25. Клас представляващ обекта User за работа в Android Studio.

```
public class Teacher {  
    public long id,teacherUserID;  
    public String name, password, teacherEmail, teacherGender;  
}
```

Фигура 26. Клас представляващ обекта Teacher за работа в Android Studio.

```
public class Subject {  
    public long id;  
    public String name,description;  
    public String toString() {  
        return name;  
    }  
}
```

Фигура 27. Клас представляващ обекта Subject за работа в Android Studio

Ползва се метода toString, за да може да се изобразява информацията от класа правилно без проблем по-късно

```
public class TestHeader {  
    public long  
    id,gradeSingleAnswer,gradeMultipleAnswer,gradeFreeTextAnswer, subjectID,us  
    erID;  
    public String testName,fromDate,toDate;  
    public String toString() {  
        return testName;  
    }  
}
```

Фигура 28. Клас, представляващ обекта TestHeader за работа в Android Studio



```
public class Question implements Parcelable{
    public long id,questionTypeId,testHeaderId;
    public String name;
    public List<PossibleAnswer> possibleAnswers;
    public Question() {
        possibleAnswers = new ArrayList<>();
    }
}
```

Фигура 29. Клас, представляващ обекта Question за работа в Android Studi

```
public class QuestionType {
    public long id,points;
    public String name;
    public String toString() {
        return name;
    }
}
```

Фигура 30. Клас, представляващ обекта QuestionType за работа в Android Studio

```
public class PossibleAnswer {
    public long id,questionId,isCorrect;
    public String name;
    public int points;
    public String toString() {
        return name;
    }
}
```

Фигура 31. Клас, представляващ обекта PossibleAnswers за работа в Android Studio. За началото на проект има специален трети пакет, озаглавен tabHost.

```
public class TabHostActivity extends ActivityGroup {
    TabHost teacherStudentHost;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_tab_host);
        teacherStudentHost = findViewById(studentTeacherTabHost);
        teacherStudentHost.setup(this.getLocalActivityManager());
        TabHost.TabSpec ts1 =
        teacherStudentHost.newTabSpec("Преподавател");
        ts1.setIndicator("Преподавател");
        ts1.setContent(new Intent(this,TeacherLogInActivity.class));
        teacherStudentHost.addTab(ts1);
        TabHost.TabSpec ts2 = teacherStudentHost.newTabSpec("Студент");
        ts2.setIndicator("Студент");
        ts2.setContent(new Intent(this,LogInActivity.class));
        teacherStudentHost.addTab(ts2);
    }
}
```

Фигура 32. Той отговаря за разклонението между Студентски панел и Преподавателски панел.

Той ( tabHost) се състои от две части, Преподавателски панел и Студентски панел и позволява да се избира веднъж индивидуална работа от страна на Студент, а втория път индивидуална работа от страна на Преподавател.

Освен, че се използва Активити за Регистрирането и Влизането на студент/преподавател, клиентското приложение се използват и фрагменти за по-детайлна работа както от страна на студента, така и от страна на преподавателя

След като се разгледат тези три пакета, които са необходими за работата на клиента е време да се обърне внимание на Влизането и Регистрацията на студент в приложението от гледна точка на клиента.

За това идва и четвърти пакет с име student. Login:

```
public class LoginActivity extends AppCompatActivity {
    private static final String TAG = "LoginActivity";
    Button register, login;
    String studentPassword, studentFacultyNumber;
    EditText password, fNumber;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_log_in);
        register = findViewById(R.id.btn_register);
        login = findViewById(R.id.btn_log_in);
        password = findViewById(R.id.et_student_password);
        fNumber = findViewById(R.id.et_fnumber);
        register.setOnClickListener(onClickListener);
        login.setOnClickListener(onClickListener);
    }
    View.OnClickListener onClickListener = new View.OnClickListener() {
        public void onClick(View v) {
            if(v.getId() == R.id.btn_register) {
                Intent regIntent = null;
                regIntent = new Intent(LoginActivity.this,
RegisterActivity.class);
                startActivity(regIntent);
            } else {
                studentFacultyNumber = fNumber.getText().toString();
                studentPassword = password.getText().toString();
                if(!validateFacultyNumber() | !validatePassword()){
                    return;
                }
                else{
                    Intent intent = null;
                    new
LoginAsyncTask(studentFacultyNumber, studentPassword).execute();
                    intent = new Intent(LoginActivity.this,
MenuActivity.class);
                    startActivity(intent);
                }
            }
        }
    };
}
```

Фигура 33. Екран/Activity (за начало) за Влизане на студент в приложението.

Освен това се забелява и валидация, тя не е server side, а client side като за начало, в по-горни версии това ще се оправи и тя ще бъде по-надеждна.

Тук се вижда, че се използват няколко бутона EditText-ове като поленца, и също така валидация за полетата, за да взимат попълнените данни правилно.

Тяхното използване става при натискането на бутона за Влизане с необходимите параметри взети от EditText-овете.

Самите данни от EditText-ове се предават на AsyncTask-a за базата данни, като AsyncTask-a има три метода (onPreExecute onBackground, onPostExecute)

В първият се показва един диалогов прозорец за студента, ако му се наложи да чака.

```
protected void onPreExecute() {  
    super.onPreExecute();  
    Log.d(TAG, "Login in....");  
    dialogLogin.setTitle("Login in, please wait!");  
    dialogLogin.setCanceledOnTouchOutside(false);  
    dialogLogin.show();  
}
```

Фигура 34. Първи от общо трите метода на AsyncTask. Така нареченият onPreExecute метод.

Вторият се връзва с базата данни чрез Post заявка и се взима необходимата информация( ако данните които са дадени за Влизане са коректни тогава няма проблем, но ако не са коректни тогава ще излезе грешка )

```
protected Void doInBackground(Void... voids) {
    Log.d(TAG, "Communicating with server");
    URL url;
    HttpURLConnection urlConnection;
    BufferedReader br;
    try{
        url = new URL( Constants.URL + "TeacherLogInServlet" );
        Log.d(TAG, "url: " + url.toString());
        urlConnection = (HttpURLConnection)
            url.openConnection();
        urlConnection.setRequestMethod("POST");
        Teacher teacher = new Teacher();
        teacher.password = teacherPassword.getText().toString().trim();
        teacher.name = teacherName.getText().toString().trim();
        String creds = new GsonBuilder().create().toJson(teacher);

        byte[] outputInBytes = creds.getBytes("UTF-8");
        OutputStream os = urlConnection.getOutputStream();
        os.write( outputInBytes );
        os.close();

        br = new BufferedReader
            (new InputStreamReader(
                urlConnection.getInputStream()
            ));

        StringBuilder content = new StringBuilder();
        String line = br.readLine();
        while(line != null){
            content.append(line);
            line = br.readLine();
        }
        result = content.toString();
    }catch (MalformedURLException e){
        Log.wtf("WRONG!", e.getMessage());
    }catch (IOException e){
        Log.wtf("WRONG!", e.getMessage());
    }
    return null;
}
```

Фигура 35. Втори от трите метода в AsyncTask. Това е така нареченият метод doInBackground.

В него се извършва цялата необходима работа(взимането на данните от сървъра) с помощта на нова нишка, с иеята главнат нишка на приложението да не се затрупва с работа.

Този AsyncTask е поставен като вътрешен клас в Активитито за Влизане (Login)

```
protected void onPostExecute(Void aVoid) {
    Log.d(TAG, "taking data");
    super.onPostExecute(aVoid);
    dialogLogin.dismiss();
    Log.d(TAG, "result: " + result);
    if (TextUtils.isEmpty(result)) {
        final AlertDialog.Builder adb = new
AlertDialog.Builder(TeacherLoginActivity.this);
        adb.setTitle("Предупреждение");
        adb.setMessage("Използвани са неправилни данни за влизане!");
        adb.setCancelable(false);
        adb.setPositiveButton("Опитай отново", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                dialogInterface.cancel();
            }
        });
        AlertDialog alertDialog = adb.create();
        alertDialog.show();
        Log.d(TAG, "wrong credentials");
    } else {
        Teacher teacher = new GsonBuilder().create().fromJson(result,
Teacher.class);
        App.loggedUserId = teacher.id;
        App.userId = teacher.userId;
        Intent intent = new Intent(TeacherLoginActivity.this,
TeacherMenuActivity.class);
        startActivity(intent);
    }
}
```

Фигура 36. Трети от трите метода в AsyncTask. Така наречения onPostExecute метод.

В този метод също има валидация на базата на userTypeID, която не пуска да се продължи при положение, че са въведени грешни данни за студента.

Всичко това става доста бързо, при наличието на добра и здрава връзка с интернет.

Идеологията за Регистриране(Register) е еднаква с тази на влизането, а иначе казано отново има едно Активити, което има AsyncTask като вътрешен клас с трите гореспоменати метода.

След като се разбра как всъщност работи Влизането и Регистрирането на студент сега ще се обърне внимание на последният пакет наречен teachers. В него има Activity за влизане на преподавател в приложението. Нека сега се обърне внимание и на

Когато студент е влезнал в системата, той може да си редактира профила, да погледне стари тестове, да започне нов тест, да си види статуса от тестовете или да излезе от приложението.

Нека първо да погледнем Редакцията на профил. Тя е идентична със регистрацията на новодошъл студент, само че разполага с полета за промяната на името, промяна на паролата и промяна на пола(Radio buttons).

Също така за промяна на данните на студента се грижи една HTTP POST заявка, с помощта на AsyncTask.

Сега като се видя как студент може да редактира профила си ще се обърне внимание на това как той може да решава тест вече създаден от преподавателя и регистриран в базата данни.

За целта за да вижда информацията за теста се грижи т. нар. Recycle View като за него се използват три отделни дъщерни елемента (item chield layout).

За да се предадат тези три дъщерни елемента се използва adapter с име questionAdapter.

Идеята на този adapter е да работи с тези три дъщерни елемента ( item chield layout ) , а отделно RecyclerView-то само да задели място за брой елементи, които ще се визуализират на базата на 3те типа въпроси регистрирани в базата данни ( таблицата QuestionType )

След това се обръща повече внимание на преподавателя:

```

public class TeacherLoginActivity extends AppCompatActivity {
    private static final String TAG = "TeacherLoginActivity";
    Button teacher_log_in;
    EditText teacherName, teacherPassword;
    String TeacherName, TeacherPassword;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_teacher_log_in);
        teacher_log_in = findViewById(R.id.btn_log_in);
        teacherName = findViewById(R.id.et_teacher_name);
        teacherPassword = findViewById(R.id.et_teacher_password);
        teacher_log_in.setOnClickListener(onClickListener);
    }
    View.OnClickListener onClickListener = new View.OnClickListener() {
        public void onClick(View v) {
            TeacherName = teacherName.getText().toString();
            TeacherPassword = teacherPassword.getText().toString();

            if (!validateTeacherName() | !validatePassword()) {
                return;
            } else {
                new
TeacherLoginAsyncTask(TeacherName, TeacherPassword).execute();
            }
        }
    }
}

```

Фигура 37. Активити, което съдържа кода за Вход на преподавателя в системата.

За помощ отново, както се вижда се извиква AsyncTask, като той притежава същите три метода описани по-горе.

Вече щом има Влязъл преподавател в приложението, той може да избира дали да си погледне профила, дали да създаде дисциплина, да създаде въпрос, да създаде тест, да оцени студентската работа( теста който е решил ) или просто да излезе.

Преди да създаде дисциплина, след като преподавателя е влезнал в системата той ще провери данните за себе си, като това става когато си прегледа профила, а именно за да го направи, той ще използва фрагмент (TeacherViewsHisOwnProfile).

Този фрагмент разполага с информацията за него( под TextView ) и AsyncTask с основните 3 метода onPreExecute(работеш на главната нишка ), doInBackground( стартира своя собствена нишка отделно от главната) и onPostExecute( връщаш резултата в главната нишка )

Тук единственото различно нещо е, че се взима id за HTTP GET заявката, а това става чрез:

```

Uri builtUri = Uri.parse(Constants.URL + "TeacherViewDetailsServlet")
    .buildUpon()
    .appendQueryParameter("id", String.valueOf(App.loggedUserId))
    .build();

```

Фигура 38. Тук се показва как се използва buildUri, за „прилепването“ на id за HTTP GET заявката за даннит.

След това преподавателя ще създаде дисциплина (една от основните задължения на преподавател), а това става със следният код:

```
public class TeacherCreateDisciplineFragment extends Fragment {
    Button ok, back;
    EditText disciplineName, disciplineDescription;
    private static final String TAG = "TeacherCreateDiscipline";
    public TeacherCreateDisciplineFragment() { }
    public View onCreateView(LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState) {
        View disciplineView =
inflater.inflate(R.layout.fragment_teacher_create_discipline, container,
false);
        ok = disciplineView.findViewById(R.id.btnAddDiscipline);
        back = disciplineView.findViewById(R.id.btnCancelDiscipline);
        disciplineName =
disciplineView.findViewById(R.id.disciplineEditText);
        disciplineDescription =
disciplineView.findViewById(R.id.descriptionEditText);
        ok.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                Intent intent = new
Intent(getContext(), TeacherMenuActivity.class);
                String disName = disciplineName.getText().toString();
                String disDescription =
disciplineDescription.getText().toString();
                new
RegisterSubjectAsyncTask(disName, disDescription).execute();
                startActivity(intent);
            }
        });
        back.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new
Intent(getContext(), TeacherMenuActivity.class);
                startActivity(intent);
            }
        });
        return disciplineView;
    }
}
```

Фигура 39. Фрагмент за създаване на дисциплина за тест за студент.

Като цялата работа се извършва отново от AsyncTask като външен клас за фрагмента.

Тук се вижда, че от Activity се преминава на Fragment, това е за по-детайлна работа.

След като е създадена дисциплината, тя трябва да се предаде на android spinner (така нареченото падащо меню) за опция за избор. Това става когато във фрагмент за създаването на теста за студент отново се създаде AsyncTask, но вместо тук да се използва HTTP POST ще се използва HTTP GET, защото в базата не добавяме, а просто вземаме вече добавената информация.



Накрая това става, когато спинър елементите ги подадем на адаптер за попълване в `onPostExecute()` метода от `AsyncTask`-а

```
protected void onPostExecute(Void aVoid) {
    Log.d(TAG, "taking data");
    Log.d(TAG, "result: " + result);
    super.onPostExecute(aVoid);
    dialogLogIn.dismiss();
    Gson gson = new GsonBuilder().create();
    final List<Subject> testSubjects = gson.fromJson(result, new
    TypeToken<List<Subject>>().getType());
    disciplineAdapter = new
    ArrayAdapter<Subject>(getContext(), android.R.layout.simple_list_item_1,
    testSubjects);
    disciplineSpinner.setAdapter(disciplineAdapter);
}
```

Фигура 40. Попълване на адаптер със спинър елементи от тип Subject

След като във фрагмента за създаването на теста има `spinner`( падащо меню със Subject) Ще се подготвят данните за създаване на теста за съответния Subject(дисциплина).

За целта ще трябва да се притежава `id` на преподавателя и `id` на дисциплината

`Id` на преподавателя се взима когато се създаде нов клас `app`, който наследява `Application` клас(той е общ за цялото приложение), а `id` на дисциплината се взима при избиране на съответната дисциплина от `spinner` за целта. Ползва се `listener`-а `setOnItemSelectedListener` за `spinner`-а.

```
public class App extends Application{
    public static long loggedUserId;
    public void onCreate() {
        super.onCreate();
    }
}
```

Фигура 41. `App` класа, който се използва за добиването на логнатият потребител (било то преподавател или студент.

Самото създаване на тест става когато се кликне бутона запиши(save) на приложението.

Тогава се вземат необходимите данни от полетата и се `execute`-ва `AsyncTask`, който очаква обект `TestHeader`.

Вижда се как в `doInBackground` метода от `AsyncTask`-а директно се използва `TestHeader` обекта, вместо `void`, за това се изисква и този обект.

```

protected String doInBackground(TestHeader... testHeaders) {
    Log.d(TAG, "Getting test from the server");
    URL url;
    HttpURLConnection urlConnection;
    BufferedReader br;
    try{
        url = new URL( Constants.URL + "TestRegisterServlet" );
        Log.d(TAG, "url: " + url.toString());
        urlConnection = (HttpURLConnection)
            url.openConnection();
        urlConnection.setRequestMethod("POST");
        String jsonTestHeader = new
GsonBuilder().create().toJson(testHeaders[0]);
        byte[] outputInBytes = jsonTestHeader.getBytes("UTF-8");
        OutputStream os = urlConnection.getOutputStream();
        os.write( outputInBytes );
        os.close();
        br = new BufferedReader
            (new InputStreamReader(
                urlConnection.getInputStream()
            ));
        StringBuilder content = new StringBuilder();
        String line = br.readLine();
        while(line != null){
            content.append(line);
            line = br.readLine();
        }
        return content.toString();
    }catch (MalformedURLException e){
        Log.wtf("WRONG!", e.getMessage());
    }catch (IOException e){
        Log.wtf("WRONG!", e.getMessage());
    }
    return null;
}

```

Фигура 42. Вместо void както до сега се използва направо целият TestHeader обект.

А в onPostExecute се връща резултата от doInBackground под формата на result

```

protected void onPostExecute(String result) {
    super.onPostExecute(result);
    Log.d(TAG, "test header: " + result);
    dialogCreateTest.dismiss();
}

```

Фигура 43. Тази фигура показва отново, че вместо да се използва void тук просто се подава String какъвто е и result, който се връща като резултат от предишния метод.

Сега най-накрая ще се обърне внимание на създаването на тест от гледна точка на клиентското приложение.

Но преди да стане това се създава дисциплина, както се вижда от по-горните абзаци.

След създаване на дисциплината идва ред и на теста, самият тест става отново с фрагмент и AsyncTask, но тук за него се използва и дисциплината, като взимането ѝ става в метода onPostExecute в AsyncTask във фрагмент с име TeacherCreateDisciplineFragment, което е показано в код на Фигура 42, където

създаването на дисциплината беше обяснено по-подробно и на Фигура 43 може да се види onPostExecute метода, описан в абзаца.

Освен AsyncTask-а за взимането на дисциплината( HTTP GET заявка ) се използва и друг AsyncTask за записването на теста. Това става с HTTP POST заявка.

Когато са налични данните за тест, другат стъпка на преподавател е да създаде въпроси и отговори за теста, а това става с помощта на нови два фрагмента ( TeacherCreateQuestionFragment за въпросите и TeacherGiveAnswerForQuestionFragment за отговори на въпроса ).

За TeacherCreateQuestionFragment са използва две HTTP GET заявки под формата на AsyncTask, едният за взимането на id на вече създаденият от преподавателят тест и другият за взимането на типовите въпроси.

Като вече за TeacherGiveAnswerForQuestionFragment след като се взема самият въпрос от TeacherCreateQuestionFragment посредством bundle и се подаде на новият фрагмент след това се използва и съответният AsyncTask за записването на въпрос и отговор заедно в базата за вече създаденият тест.

## 5. Ръководство на потребителя

Приложението е достатъчно добре направено, така че потребителите му( студент или преподавател ) да могат да се ориентират как да работят с него без да има е необходима допълнителна документация как да си решат теста в приложението.

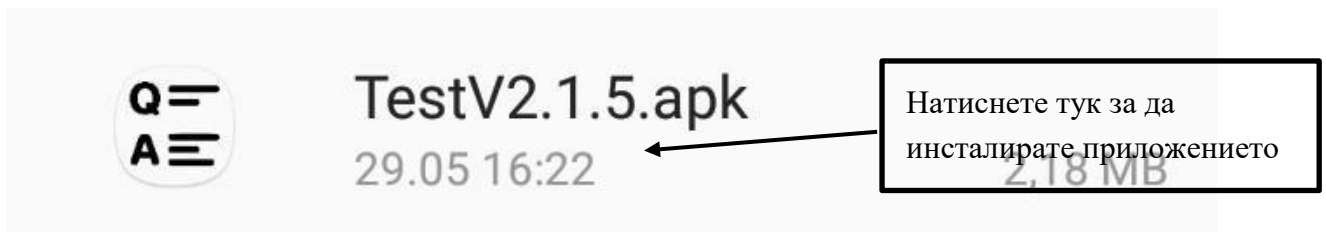
За по-лесна употреба ще бъде добавен един User Guide, който ще бъде представен по-долу в документацията и ще може да се използва от клиентите при евентуален проблем, така също и от разработчиците на Андроид приложения за по-голяма яснота.

### Наръчник за потребителя

За този „Наръчник на потребителя” се предполага, че клиентите вече са се сдобили с apk файла на приложението. Ако все още не са го направили, то те могат да го изтеглят от github на горепосочените линкове(линковете са в точка 2).

#### 5.1. Инсталиране на приложението

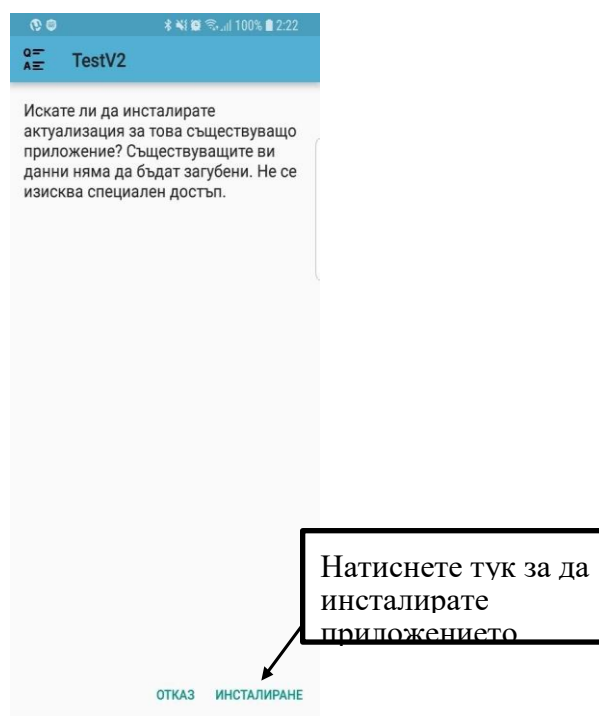
За клиентите ( студент или преподавател) вече щом имат apk файл единственото което трябва да направят е да натиснат върху него и да го инсталират на телефонните си устройства.



Фигура 44. Тази фигура показва на коя от иконите се натиска за да се инсталира приложението на необходимото Android устройство.

След това просто чакате приложението да се инсталира и да бъде готово за използване на телефона.

По-долу ще бъдат предоставени изображения за инсталиране на приложението стъпка по стъпка за по-голямо изяснение и уточнение на хората, използващи това приложение.



Фигура 45. Подробна инсталация на приложението за работа. След като се сдобие с арк файла както беше обяснено по-горе по този начин показан по-горе се инсталира.

За да бъде по-ясно самото инсталиране ще трябва да се концентрираме над това как неговите потребители да достигнат до самата апликация,

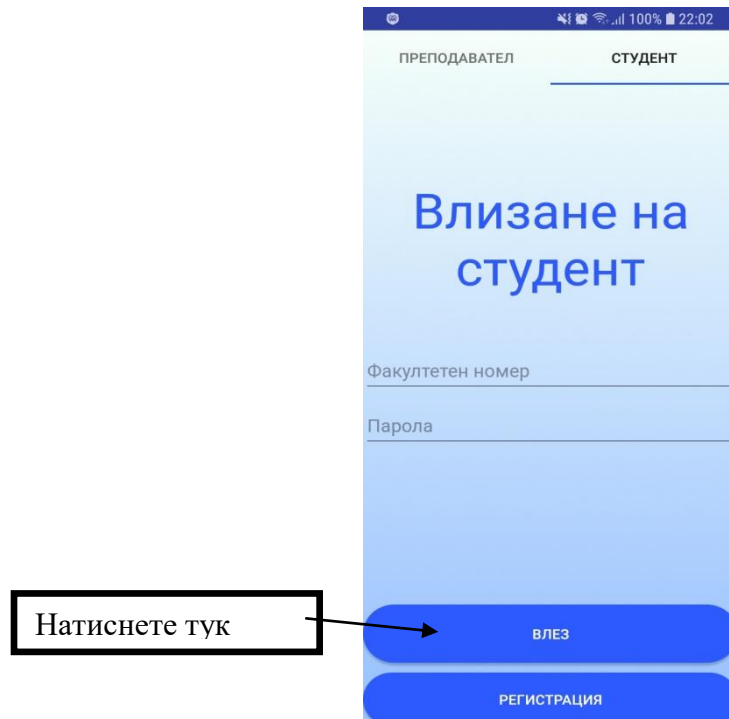
Като потребителите трябва първо да са изтеглили арк файла от github, а след това да навигират до папката в която се намират изтегленият документ и да кликнат върху иконката за инсталиране.

След това те могат да влезнат и да работят свободно с него

## 5.2. Влизне на студент в приложението

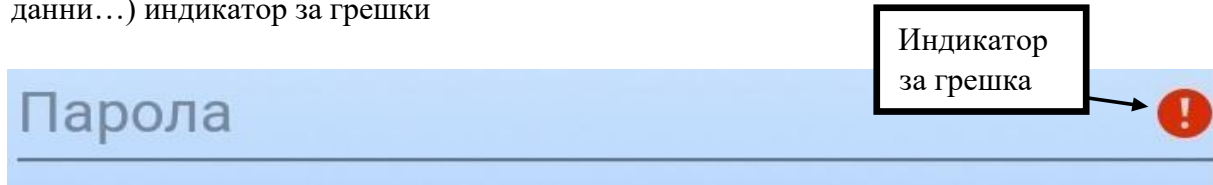
След като апликацията се инсталира на телефона на потребителя той бива посрещнат от един tabHost, разполагащ както с информация за влизане на студент, така и с информация за влизане на преподавател

Фигурата ще бъде показана на долната страница, за по-лесен ориентир



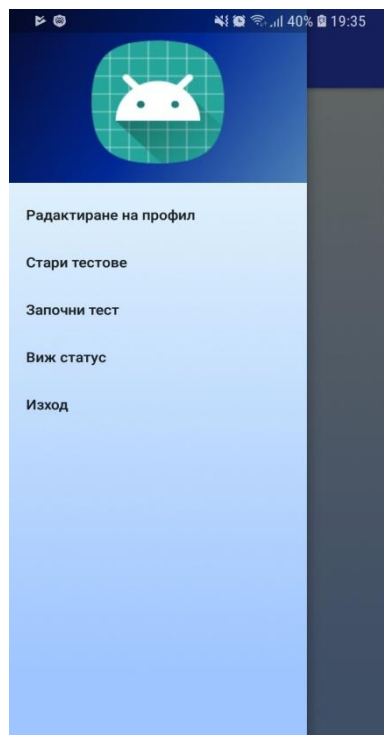
Фигура 46. На тази фигура се показва табхост-а необходим за начало на приложението.

При натискане на бутона „Влез“, ако има грешки ще се покаже (невалидни данни...) индикатор за грешки



Фигура 47. Тук се показва индикатор за грешки.

В противен случай ще влезем успешно в Меню, разполагащо с опциите на студент, като някои от опциите ще се разгледат, а други са „в процес на разработка”



Фигура 48. На тази фигура се вижда така наречения Draw menu и опциите известни за студента, като обаче някои от тях все още не функционират коректно.

Нефункциониращите опции са отбелязани за по-голяма яснота на потребителя, който вече е студент за тази версия на приложението.

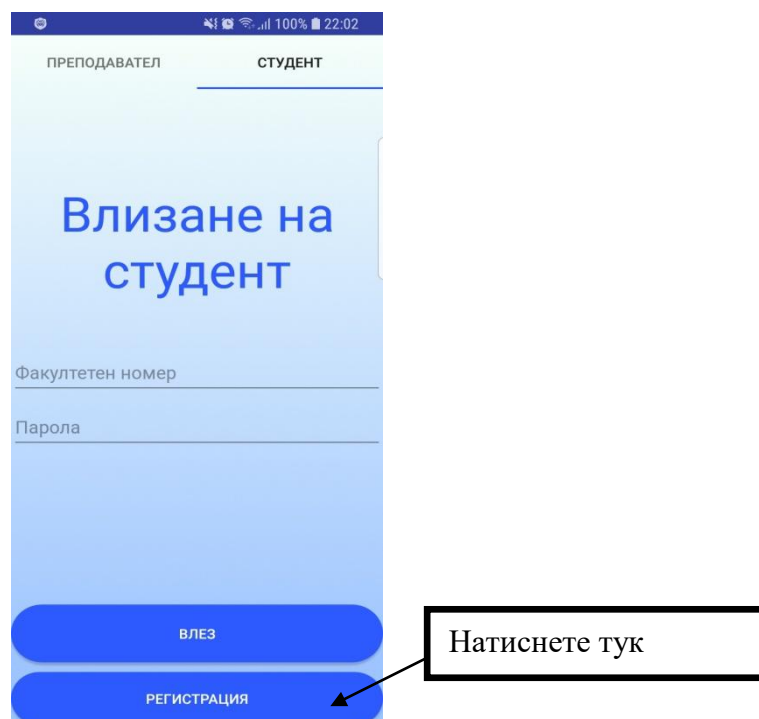
Като те са отбелязани с текст „В процес на разработка” както беше вметнато по-горе, защото дори и в този момент те биват разработвани с цел да се завърши цялостната функционалност на приложението

Ако обаче потребителят( в нашият случай студент) е новодошъл ще трябва да си създаде профил в системата, той ще избере вместо бутон „Влизане” бутон ”Регистрация”

### 5.3. Регистриране на студент за приложението

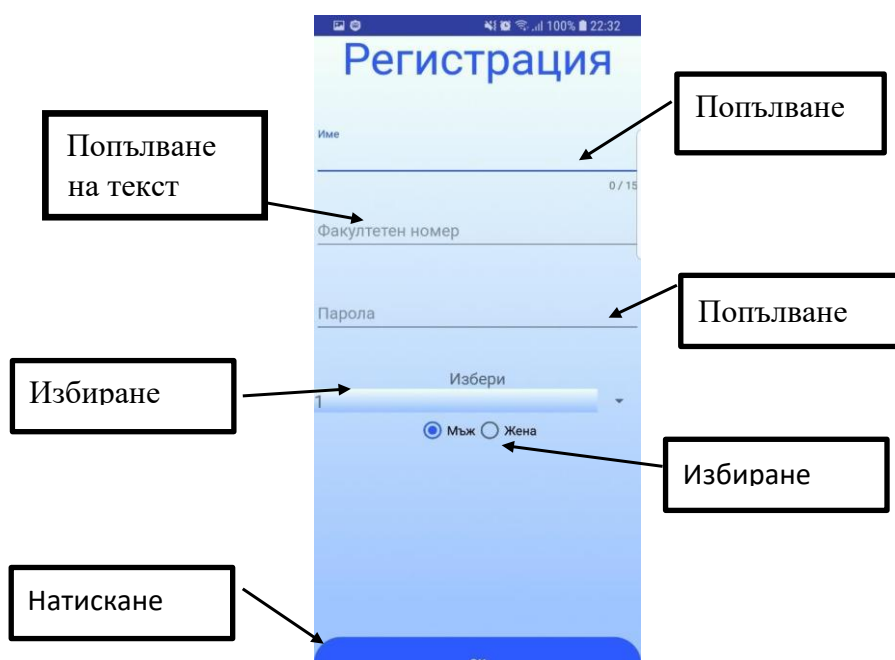
Фигурата за това действие отново ще бъде разположена на долната страница с идея за по –голяма прегледност и видимост.

В нея отново се наблюдава познатият tabHost за началото на приложението, като тук ако студента е по-наблюдателен ще забележи, че този tabHost е разделен на две части обхващашт както студента така и преподавателя, за него ще дойде реч малко по-късно, но сега нека се върнем на темата, а иначе казано „ Регистриране на студент” .



Фигура 49. На тази фигура се показва, освен че може да се влезе и това, че може да се извърши регистрация на студент от притежателя на телефонното устройство.

Когато Потребителят избере да натисне бутона за „Регистрация“, вместо „Влизане“ се показва допълнителна страница, а именно страница за попълване на регистрационния формуляр за студент.



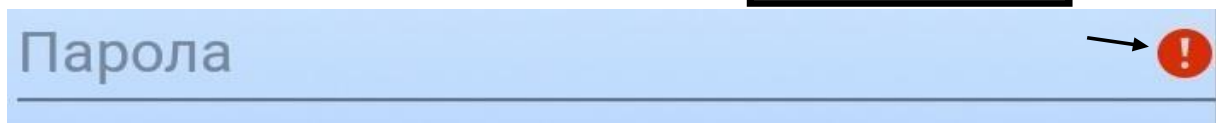
Фигура 50. Тази форма показва така наречената формичка за регистраация на студенти, като всяко едно поле се взима и запазва във външната база за това приложение.



Интересно нещо тук е фактът, че потребителя може да прекрати своята регистрация чрез бутон „Изход“, който за жалост не се вижда тук на фигурата.

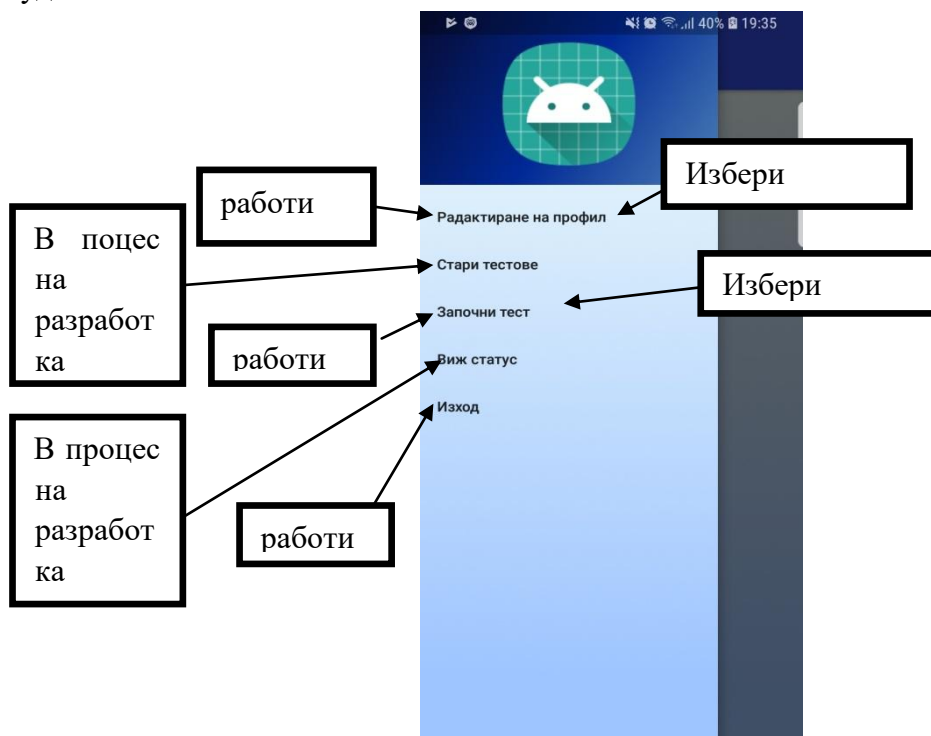
След като въведете желаните от него данни следващата стъпка е да натиснете бутон „ОК“.

Ако обаче не е попълнил полетата с правилните данни, тогава ще му излезе поле с грешка.



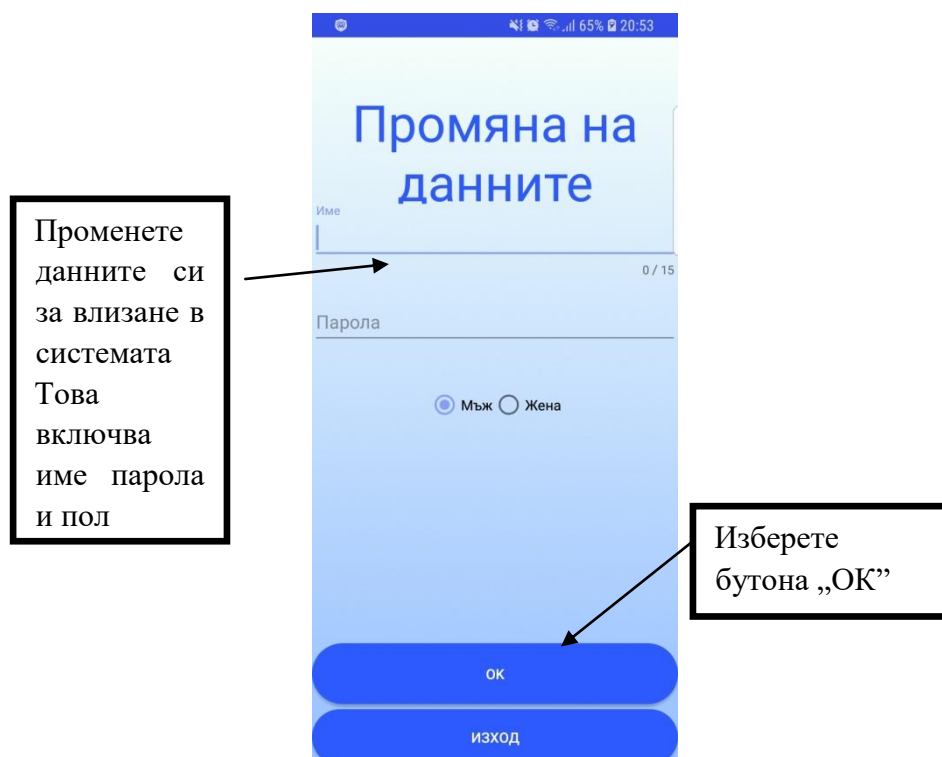
Фигура 51. Тази фигура показва Индикатора за допусната грешка

След като се направи правилна регистрация, отново потребителят, вече студент се озовава на познатото Draw Menu



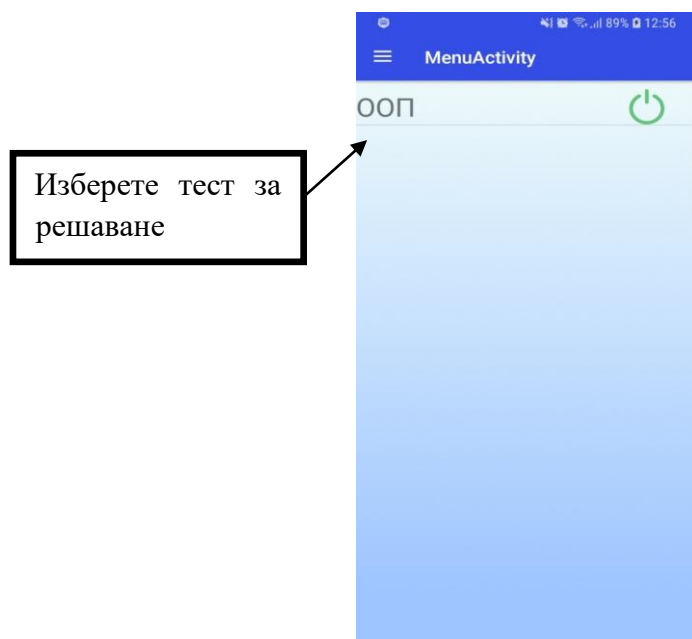
Фигура 52. Draw menu след регистрация на студент. Това Drawer menu е същото като Draw menu-то, което се показва след влизане на студент, по този начин се осигурява достъп до него и по двата начина ( регистрация или вход ).

След като се избере опцията за редактиране на профил ще се покаже екран с форма за редактиране на студентската информация



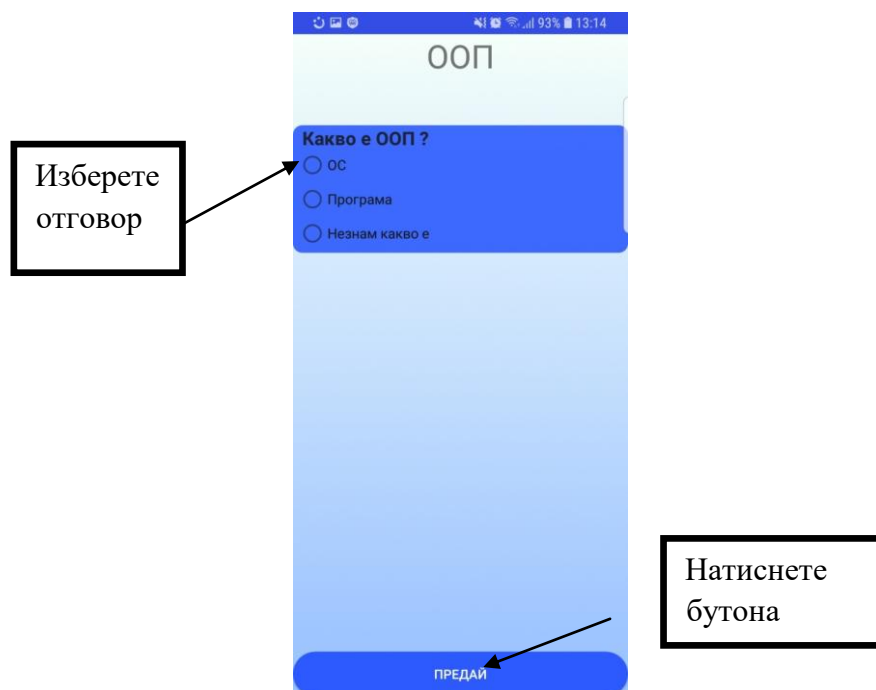
Фигура 53. На тази фигура се вижда формичката която трябва да се попълни от студента за да редактира данни за влизане в приложението

А когато студент избере от менюто опцията за започване на тест ще му се покаже екран с лист с всички възможни тестове за решаване създадени от преподавател.



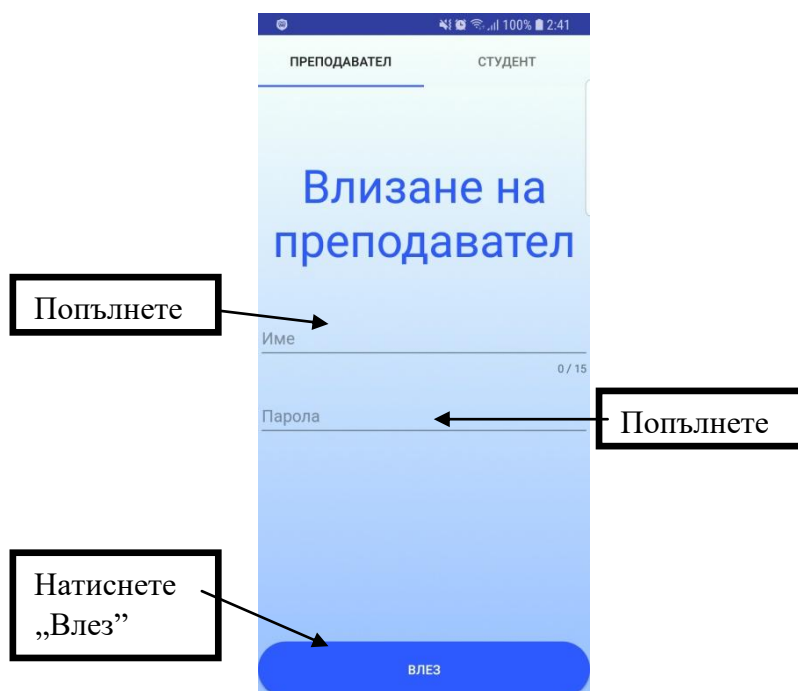
Фигура 54. Тази фигура показва списък със всички възможни тестове от които студента може да избира, кой да решава.

Щом потребителят ( студент ) избере си от менюто тест за решаване той ще започне да отговаря на въпросите на този тест.



Фигура 55. Тази фигура показва как Recycle View изобразява съответните въпроси от избраният тест като след предаването на теста резултатите от отговорите се записват обратно на сървъра.

#### 5.4. Влизане на преподавател в приложението



Фигура 56. Втората част от TabHost-а е екран за влизане на преподавател в системата. Той влиза с име и парола.

След като преподавател вече е регистриран в базата сега спокойно може да влезе в приложението и да работи над неговите задачи.



Фигура 57. Тази фигура показва менюто за преподавател и неговите опции важни за него.

След като влезне с данните си на преподавател, се вижда това меню, в което са показани всички важни действия, които може да извършва.

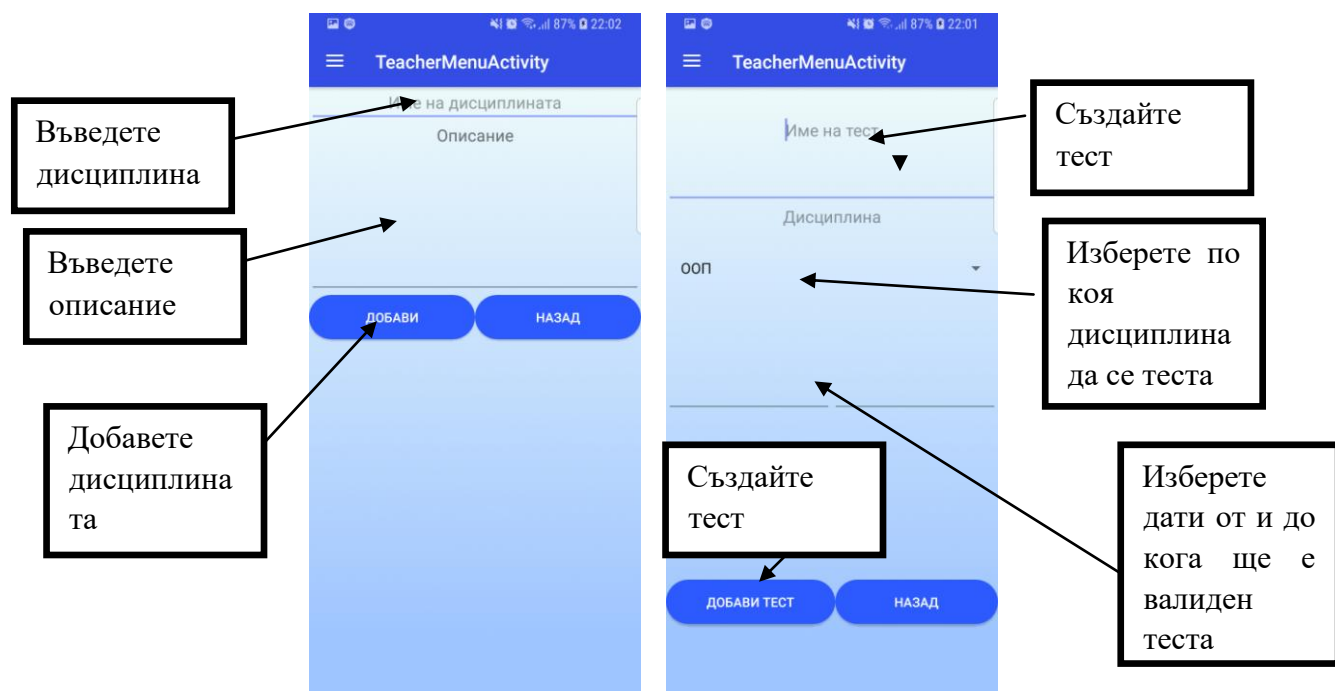
Голяма част от тях са завършени и работят, но има и други които не са, те просто са оставени за нова версия на приложението.

При натискане на Изход от преподавателя, следва изход ( до главният TabHost)

Ако преподавателят реши да създаде дисциплина той ще избере опцията от менюто „създай дисциплина”

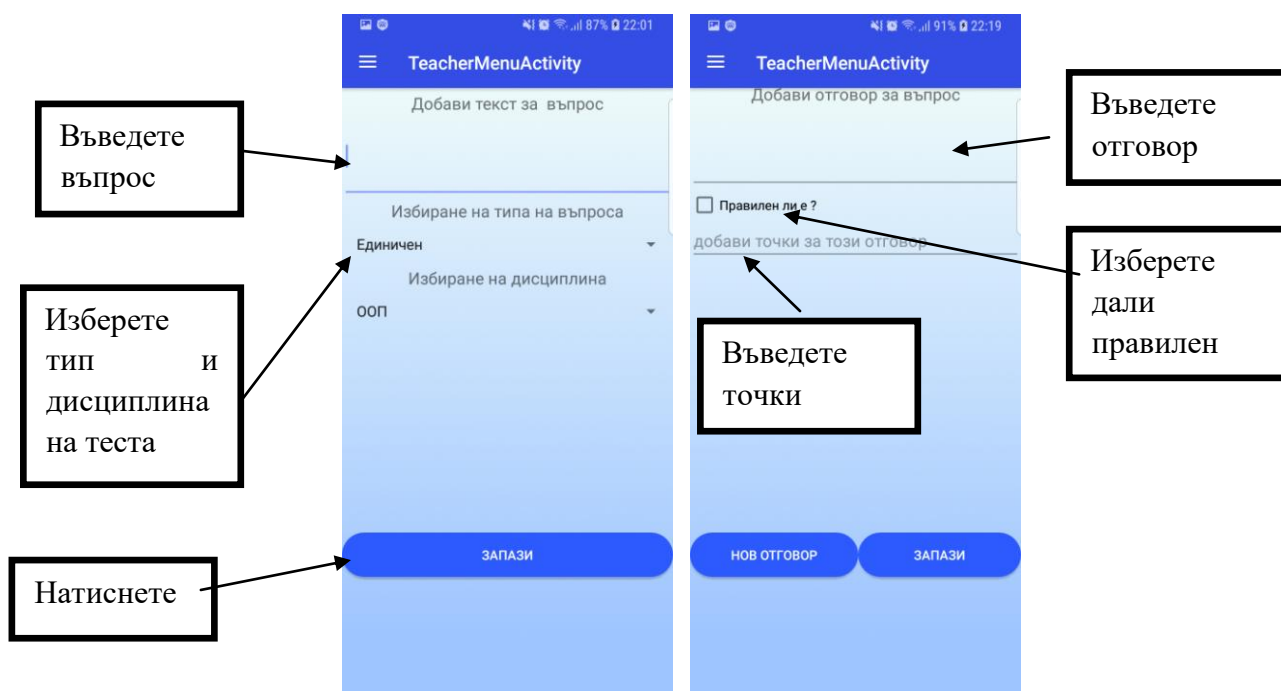
Ако реши да създаде тест той ще избере опцията от менюто „ създай тест”

След като създаде тест следващата му стъпка е да създаде въпроси за този нов тест,а заедно с въпросите ще даде и отговори за теста, като това става когато избере опцията от енюто „Създай въпрос”



Фигура 58. На тази фигура могат да се видят екраните за създаване на дисциплина и респективно създаване на тест на базата на тази дисциплина

Тук са показани екраните за създаване на дисциплина и респективно екрана за създаване на тест за дадената дисциплина от преподавателя, а следващата фигура ще покаже екраните за създаване на въпроси и отговори за съответният тест



Фигура 59. На тази фигура могат да се видят екраните за създаване на въпрос и респективно създаване на отговор на базата на въпроса.

## 6. Заключение

Благодарение на архитектурата на цялостното приложение ( както сървърна така и клиента ) и бъдещо дообработване, то може да стане, една цялостна и работеща система, която да се използва и от други университети.

За сега това приложение е версия 1.0, така че за бъдещо развитие се планира да се завърши цялата функционалност за да работи перфектно и без проблем.

Виждат се няколко „бъга“ в приложението, за това съм си поставил за цел да оправя всички „бъгове“ в системата, било то за дата( бгг който виждате дори в момента на тази версия ).

В процес на разработка са някои от по-малко важните процеси като Виждане на статуса на студент,и разглеждане на стари тестове, но по-важната задача, като създаване на тест от преподавател и решаването на тест от студент, работят безупречно.

Смятам да продължавам да работя над Service-ите и Servlet-ите, както и над AsyncTasks за да довърша липсващите, процеси.

Също така съм си поставил за цел да поработя над визуалния аспект на приложението ( Front-end).

На този етап това са моите планирани цели за бъдещето развитие на клиент-сървър приложението.

## 7. Използвана литература

Документът е предназначен за четене от преподаватели, тестери, и съветници.

Предложена литература за четене:

### 7.1. Английска литература:

THE Java™ Programming Language, Fourth Edition - Ken Arnold ,James Gosling, David Holmes

Android 4 platform SDK techniques for developing smartphone and tablet applications - Satya Komatineni, Dave MacLean

### 7.2. Българска литература:

Въведение в програмирането с JAVA. - Светлин Наков и колектив

### 7.3. Помощни страници и линкове:

Developer.android.com с линк:  
<https://developer.android.com/reference/android/os/AsyncTask.htm>  
за ползването на AsyncTask в android

Mkyong.com линк: <https://www.mkyong.com/java/how-do-convert-java-object-to-from-json-format-gson-api/> за ползването на gson с java.

линк: <https://www.mkyong.com/servlet/a-simple-servlet-example-write-deploy-run/>

За използването на java servlet-и.

Codezheaven.com

линк:

<http://www.coderzheaven.com/2017/01/19/using-gson-in-android-simple-demo/> за ползването на gson с android

Tutorialspoint.com

линк:

<https://www.tutorialspoint.com/servlets/servlets-first-example.htm>

за ползването на Servlet-и в java.

Vogella.com

линк:

<http://www.vogella.com/tutorials/AndroidDrawables/article.html> какво са

drawable ресурси и как се ползват

линк:

<http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.htm>

[1](#) за AsyncTask Example

AngryTools.com линк:

<http://angrytools.com/gradient>

За формирането на градиент за фон на приложението