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

ДИПЛОМНА РАБОТА

ТЕМА: Модул за тестова система

Изготвили:

Александър Руменов Инджов

Фак. № 1701737014

Научен ръководител:

ас. Йордан Тодоров

С Ъ Д Ъ Р Ж А Н И Е

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 Bluethoot 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 захранват 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 преодолял 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 водеши 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 ,Marshmellow, 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 излезнало устроистро 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 апликации

Ако собствениците на смартфон или таблет с Android OS са заинтересовани от създаването на свои собствени апликации, 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 апликации 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

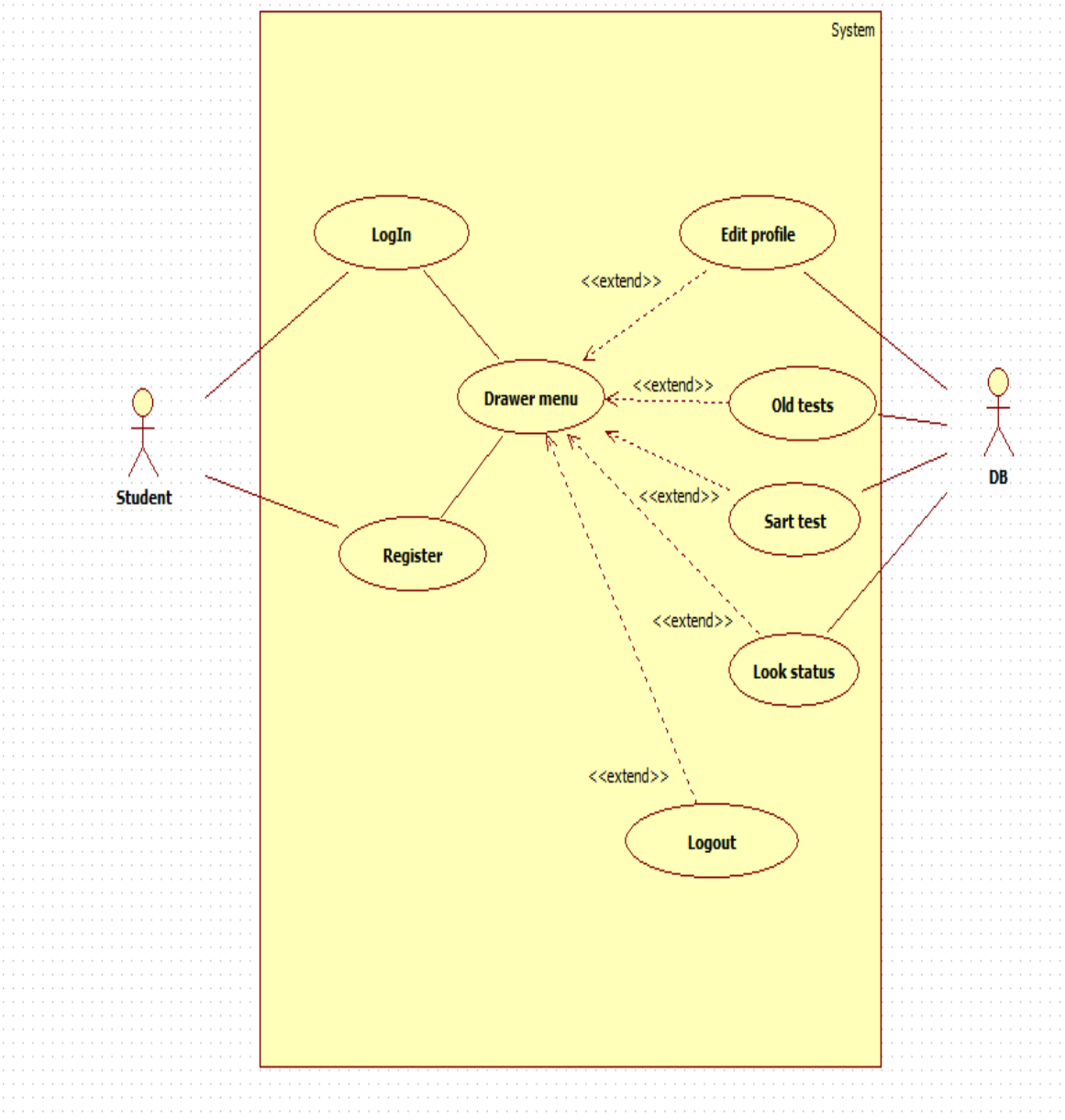
Уеб услугата е колекция от отворени протоколи и стандарти, използвани за обмен на данни между приложения или системи. Софтуерните приложения, написани на различни програмни езици и работещи на различни платформи, могат да използват уеб услуги за обмен на данни чрез компютърни мрежи като интернет по начин, 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 уеб услуги използват HTTP методи за реализиране на концепцията за REST архитектура. Универсалната уеб 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

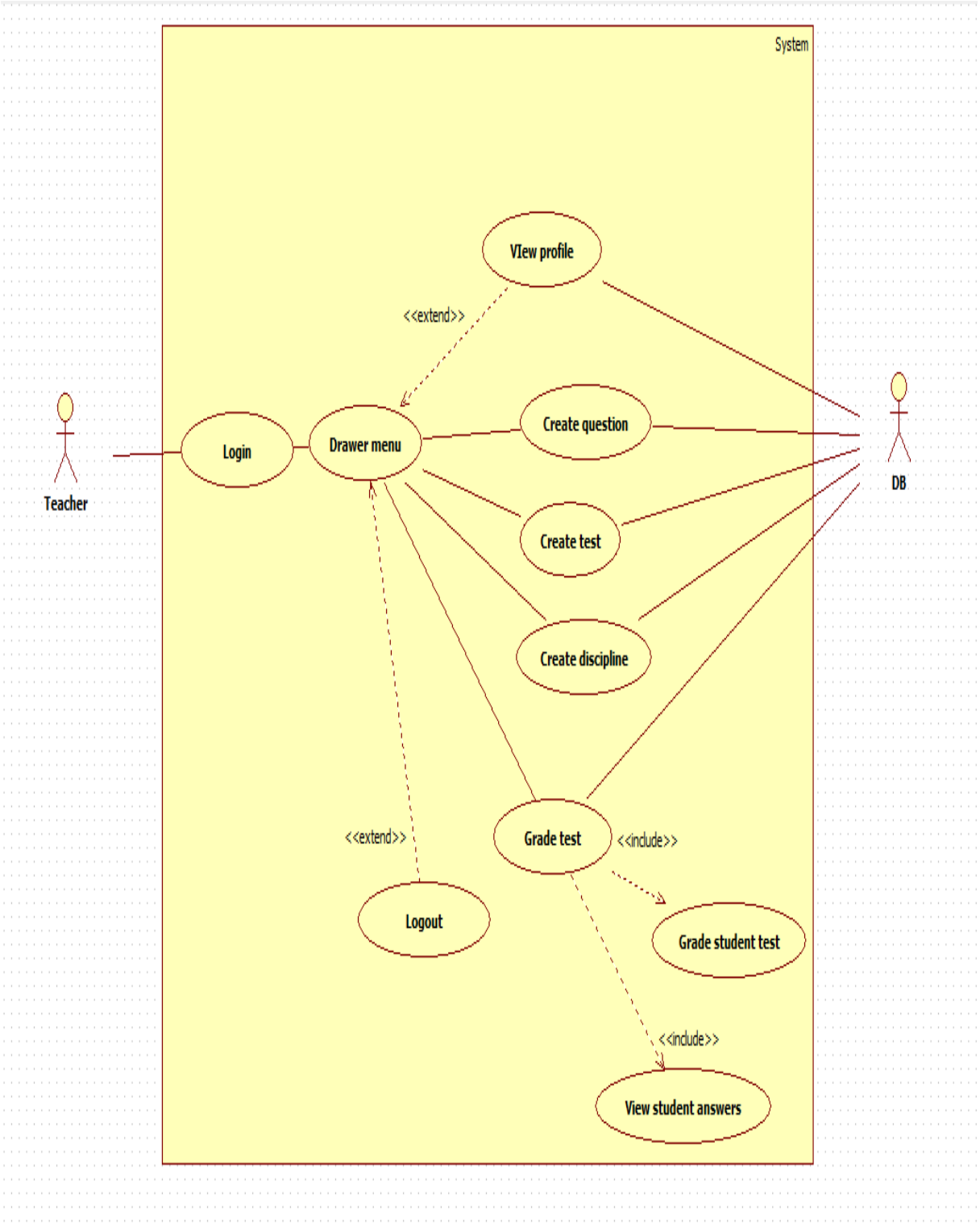
3.1.1. Use-Case Диаграма за базови действия за студент



Фигура 1. Упростена Use-Case Диаграма за приложението

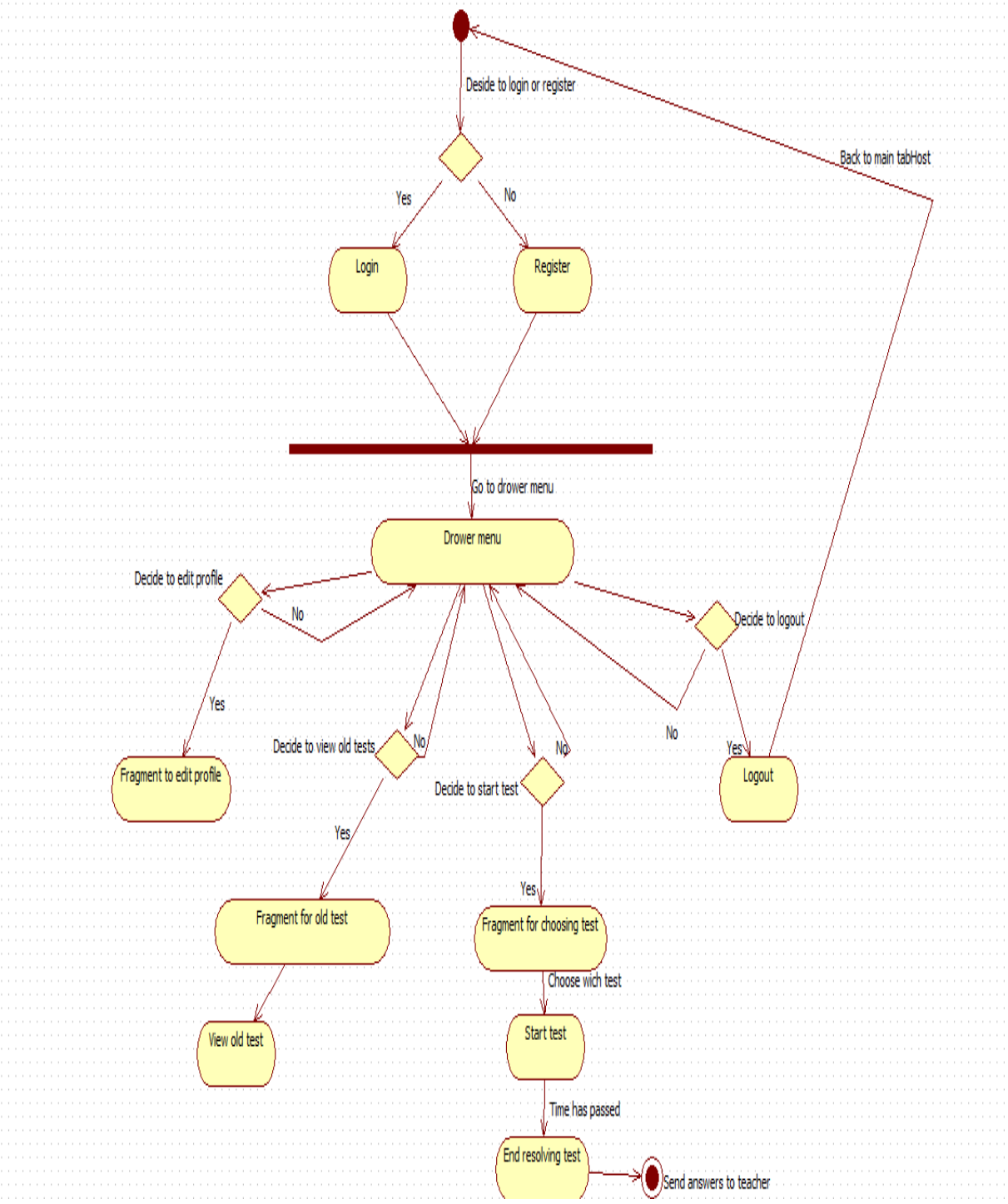
Тя показва елементарната работа на студента чрез тази use-case диаграма

3.1.2. Use-Case Диаграма за базови действия за преподавател



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

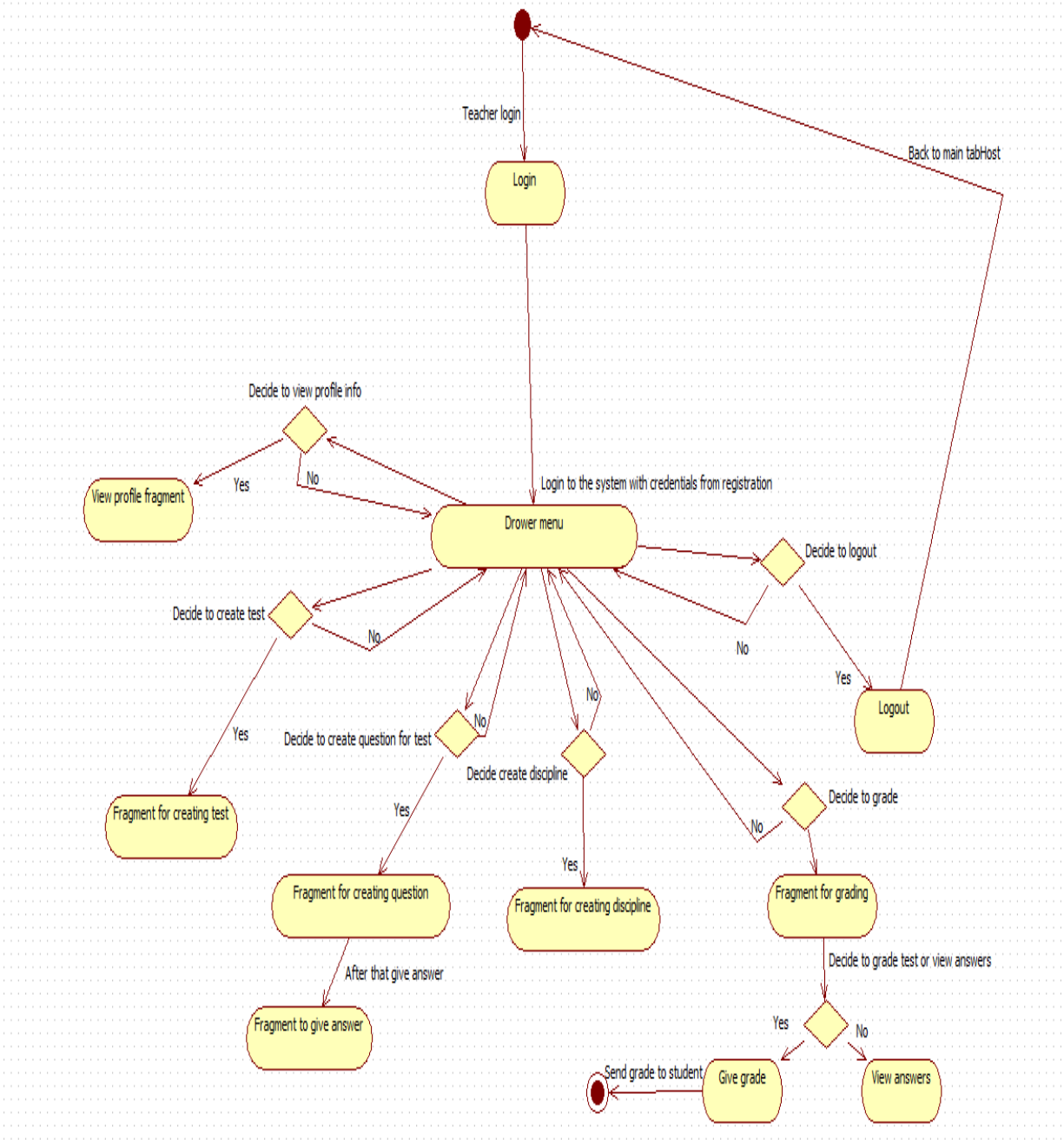
3.1.3. Activity Диаграма за повече информация за студент



Фигура 3. Activity Диаграма за студент

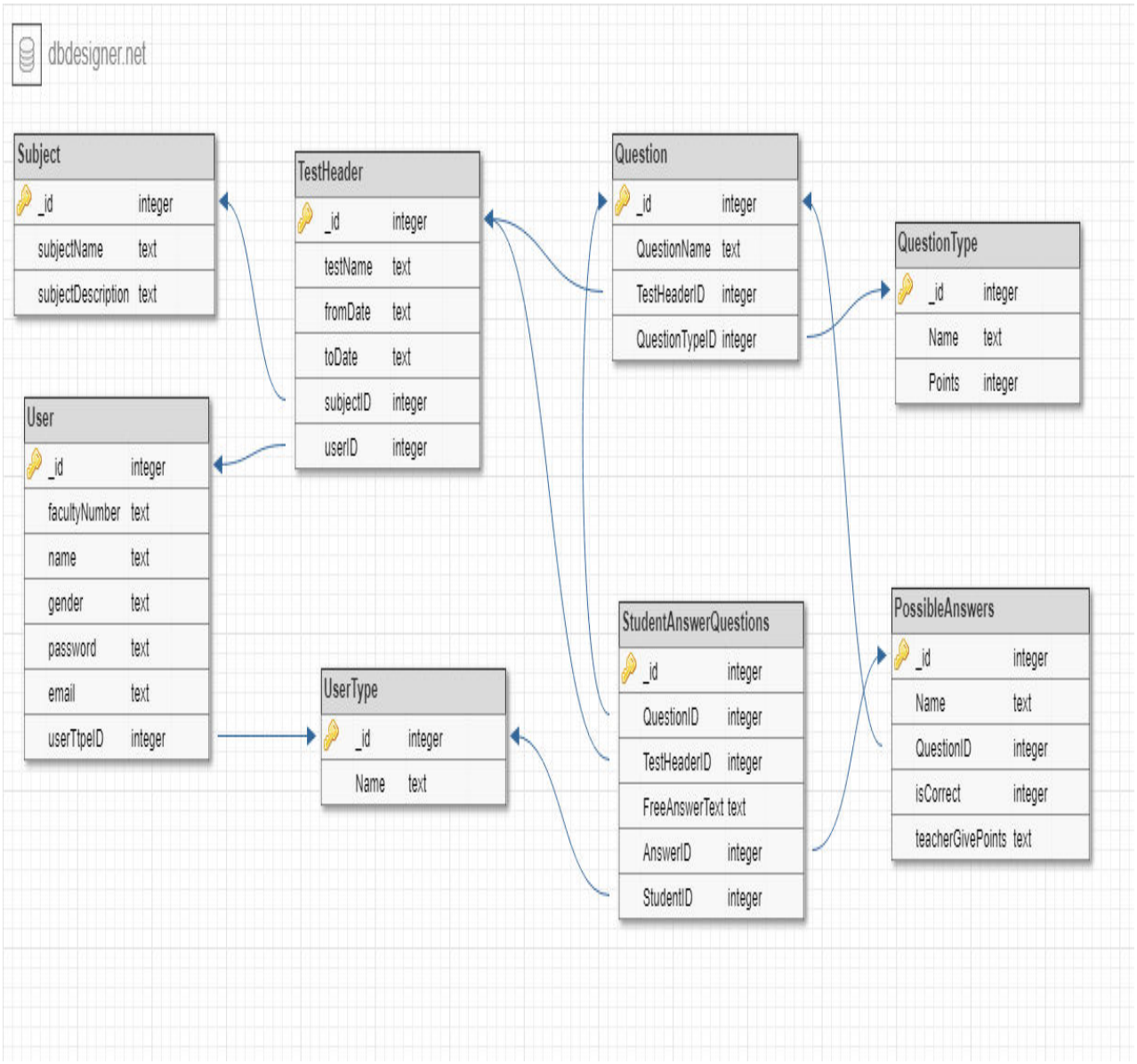
Тази диаграма улицетворява малко по-подробната работа на студент и студентски панел, през какви стъпки трябва да премине студента за пълноценна работа.

3.1.4. Activity Диаграма за повече информация за преподавател



Фигура 4. Тази диаграма улицетворява 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-ite 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	_id INTEGER PRIMARY KEY AUTOINCREMENT	_id за даден User, което _id се увеличава всеки път с 1 за всеки нов запис в таблицата User.
	facultyNumber TEXT	Това поле служи за записване на факултетния номер на студент, ако естествено user-а е от тип студент. То се изплзва от студент
	name TEXT	Това поле е за записването на името на user-а .
	gender TEXT	Това поле се използва за определянето на пола на User
	password TEXT	Това поле се използва за запазването на паролата на User.
	email TEXT	Това поле се използва от преподавателя
	userTypeID INTEGER	Това поле служи като външен ключ за таблицата UserType в която се определя дали user-а е студент или преподавател
UserType	_id INTEGER PRIMARY KEY	_id за даден Type на User, с него може лесно да се определи дали даден user е студент или преподавател
	Name TEXT	Това поле се използва, за да може да запомним на кое _id ще е преподавател и на кое ще е студент, в него с текст ги записваме.

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

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

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

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/Desk
top/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 усъществява.

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 SQL-ite 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 сървлет 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 сървлет за приложението. С него се стартира сървъра

Този сървлет се използва за пускане на сървъра, 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 сървлета има инстанция connection, от тип DBConnection за връзка с базата данни, необходима за извличане на данните от базата данни, които после ще се достъпят от клиента чрез request.

Ако в doGet метода всичко премине усшешно(без грешки) ще се отвори jsp страницата success със съощение за успешно сързване:

```
<%@ 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 succesfuly!");
%>
</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 successfuly!");
 %>
</body>
</html>
```

Фигур 9.Съощение за грешка

След като вече сървъра е пуснат да работи е време да 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

След като обекта вече е на лице и съдържа цялата информация, която е необходима за студент, следващата стъпка е да се направи Сървис за базата данни. Сървисът 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.userTypeID = rs.getString("UserTypeID");
            }
         }
catch(Exception e){
          e.printStackTrace();
         } finally {
            try {
      stmt.close();
      conn.close();
                  } catch (SQLException e) {
                        e.printStackTrace();
                  }
      }
      return user;
```

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

А сървисът се използва в сървлета 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 отговаярщ за Login на студент в програмата

Тук освен, че се вижда как се използва сървиса в сървлета 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 когато се използва doPost се цели по-голяма сигурност.

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

Кратко допълнение Сървлетите използвани са част от четвъртия и полседен пакет(servlets) за приложението.

```
 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 елементи/думи, които може да се объркат при просто записване като стринг. За това се използва и този допълнителен клас, както във всеки сървлет така и във всеки сървис (било то за полета от таблиците или по-дребни неща, които могат да се объркат при просто писане).

```
public class RegisterUserService {
      public User register(String FacultyNumber,String Name,String Password,
String Gender,String UserTypeID){
            Connection connection = null;
            User user = null;
            PreparedStatement statement = null;
            try {
                  final String sql = "INSERT INTO User
(FacultyNumber,Name,Password,Gender,UserTypeID) 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,UserTypeID);
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 полазва как се вкарва студент във външната база данни посредством този сървис.

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

Тя работи по доста подобен начин (т.е. и тя използва сървис и съответно сървлет).

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

В този сървис единственото интересно нещо е, че се използва и друг сървис (FindUserService), 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. Сървлет:

```
@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 UserTypeID = reqCredentials.userTypeID;
            user = registerUserService.register(FacultyNumber, Name, Password,
Gender, UserTypeID);
           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. Сървлет за Регистрирането на потребител в системата.

Тук отново се вижда как се използва сървиса. В сървлета 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 3rd 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 и в Сървлета за влизане, поради същата идея кодът на приложението да не става прекалено претрупан.

Всичко това което беше показано е за Потребител тип Студент, а за

Освен влизането и регистрацията на студент се използват сървиси и сървлети за промяната на данните на студент.

Преподавателя както по-рано беше казано, се създава Регистрацията чрез специален сайт с идентична подредба като Сървъра (connection, objects, server, services, servlets) .

Connection пакета съдържа един клас DBConnecton специално за връзка с базата данни, която е същата база за приложението.

Objects съдържа обекта Teacher:

```
public class Teacher {
	public long id;
	public String name, password, email, gender,usertypeid;
	 public String toString() {
			return name;
		}
}
```

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

След като имаме Обекта Teacher, ще е необходимо също и двата сървиса за Регистриране на Преподавател. Първият е за откриване на Преподавател а втория сървис е вече за реалната Регистрация и попълване на данните в базата, която се използва за приложението.

```
public Teacher find(String Name, String Password){
            Teacher teacher = new Teacher();
       Connection conn = null;
Statement stmt = null;
         try{
          Class.forName("org.sqlite.JDBC");

      conn=DriverManager.getConnection("jdbc:sqlite:/C:/Users/Home/Desktop/Tes
t V2.db");              conn.setAutoCommit(false);              stmt =
conn.createStatement();
             ResultSet rs = stmt.executeQuery("SELECT * FROM User WHERE"
                          + " Name = \"" + Name +
                          "\" AND Password=\"" +Password + "\"");
while (rs.next()){
                    teacher.id = rs.getInt("_id");
teacher.email = rs.getString("Email");
      teacher.name = rs.getString("Name");
      teacher.gender = rs.getString("Gender");
      teacher.password = rs.getString("Password");
      teacher.usertypeid = rs.getString("UserTypeID");
             }            }
catch(Exception e){
          e.printStackTrace();
} finally {
             try {
      stmt.close();
      conn.close();
                    } catch (SQLException e) {
                          e.printStackTrace();
                    }
             }
        System.out.println(teacher.toString());
             return teacher;
      }
```

Фигура 17. Код от сървиса за „Откриване“ на Преподавател.

Този сървис ще се използва в сървлета за Влизане(LogIn), поради простата причина, че сървиса е класа, който има прекия достъп (манипулация) на базта данни.

```
public class RegisterTeacherService {
public Teacher register(String name,String password, String email, String
gender,long usertypeid){
            Connection connection = null;
            Teacher teacher = null;
            PreparedStatement statement = null;
            try {
                  final String sql = "INSERT INTO User (Name, Password, Email,
Gender, UserTypeID) VALUES (?,?,?,?,?)";
                  connection =
DriverManager.getConnection("jdbc:sqlite:/C:/Users/Home/Desktop/TestV2.db");
                        statement = connection.prepareStatement(sql);
    statement.setString(1, name );
statement.setString(2, password);
statement.setString(3, email);     statement.setString(4,
gender);     statement.setLong(5, usertypeid);
statement.executeUpdate();
                        FindTeacherService findTeacherService = new
FindTeacherService();
    teacher =  findTeacherService.find(name, password);
System.out.println("register teacher: " + teacher);
                  } catch (SQLException e) {

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

Фигура 18. Сървис за регистрация на преподавател в базата

След като се разполага вече със сървисите, които са необходими, е ред на сървлета.

Няма да се показва кода му, защото е идентичен на кода за Студент, само че се използва вместо User, Teacher.

По друг начин казано: отново се използва doPost и освен това във форма за регистрация се използва method=”POST”.

Най-съществената разлика е, че тук се използва доста jsp страници за създаването на сайта за регистрирането на преподавател.

За целта по-конкретно се използва Register.jsp, а това jsp се състои от bootstrap в своята head секция.

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Teacher's Panel</title>
      <link rel="stylesheet" href="css/main.css">
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
<!-- jQuery library -->
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></scrip
t
>
<!-- Latest compiled JavaScript -->
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></sc
r ipt> </head>
```

Фигура 19. Тази фигура показва линковете за използване на bootstrat. Тук са използвани CDN(Content Delivery Network), защото е по-бързо, но ако няма достъп до интернет няма да се заредят самите линкове и bootstrap-а.

Форма за попълване от преподавател, която има за цел Регистрацията му в базата данни.

```
<form action="TeacherRegister" method="POST">
    <div class="row">
      <div class="col-25">
        <label for="name">Име</label>
      </div>
      <div class="col-75">
        <input type="text" id="name" name="name" placeholder="Вашето име...">
</div>
```

Фигура 20. Тази фигура показва едното от полетата за формата за попълване, останалите полета са както следва: Парола, Повтори парола, Имейл, Пол, Длъжност

За да се завърши самата форма е показана фигура със съответните бутони необходими за формата (по-долу кат нова формичка)

```
</div>
    <div class="row">
    <div class="col-30">
      <input type="submit" class="btn btn-success" value="Регистрирай">
    </div>
    <div class="col-40">
    <input type="reset" class="btn btn-warning" value="Изчистми">
    </div>
    <div class="col-30">
      <a href="Home.jsp"><input type="button" class="btn btn-danger"
value="Отказ"></a>
    </div>
    </div>
  </form>
</div>

</body>
</html>
```

Фигура 21. Бутоните за регистация, изчистване и отказ от регистриране на пеподавател.

За формата се използва bootstrap за по-прегледна визия и „респонсив“ дизайн, съответно тези две фигури сформират цялостната форма за регистрация, чиито данни се изпращат директно в базата за сървъра.

При натискане на бутона „Отказ“ се вижда как с линк обратно се връща в началната страница на този сайт.

Веднага след самата регистрация преподавателя влиза с креденшали(име и парола)

След това може да види своите идентификационни данни(без парола разбира се).

Това става с помощта на един сървис(ViewDetailsTeacherServie,използващ заявка към базата за взимането на име и емаил на user, тип преподавател) и сървлет(TeacherViewDetails, за предоставянето на данните за извличане чрез AsyncTask от andoid).

След като се разгледаха Обектите за преподавател и за студент ще се обърне внимание и на друг обект от пакета objects Subject:

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

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

Фигура 22. Обект необходим за създаването на Дисциплина.

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

За създаването на дизциплини се използват съответните сървиси(

FindSubjectService,RegisterSubjectService) и сървлети(SubjectFindServlet за използването му от клиента като опции за adroid 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, той следват почти същата идеология като останалите. А именно използва се обекта TestHeader (който между другото се използва като репрезентация на поле 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 23. Обекта TestHeader, който се използва за целите на приложението.

И също вече познатите Сървиси(2 на брой) и Сървлети(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 24. 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 PosibleAnswers. 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 брои, 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 25. Тази фигура показва, кои два сървиса 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 26. Тази фигура показва класа PossibleAnswer за работа с базата данни. Като той е репрезентация на колона в базата.

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

Сървисите са два на брои, както и Сървлетите са два на брои 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 27. 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();
                        }
                  }
                System.out.println(questionTypes.toString());
                  return questionTypes;
```

Фигура 28. Тази фигура показва сървиса за QuestionType, като по-късно той бива използван от сеотвтния сървлет.

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

Както в Eclipse трябаваше 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 30. Клас представляваш обекта User за работа в Android Studio.

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

Фигура 31. Клас представляваш обекта Teacher за работа в Android Studio.

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

Фигура 32. Клас представляваш 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 33. 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 34. Клас, представляващ обекта Question за работа в Android Studio

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

Фигура 35. 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 36. 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 37. Той отговаря за разклонението между Студентски панел и Преподавартелски 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 38. Екран/Активити(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 ЕдитТекст-овете.

Самите данни от EditText-ове се предават на AsyncTask-а за базата данни, като AsyncTask-а 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 39. Първи от общо трите метода на AsyncTask. Така наречения onPoreExecute 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 пробем, 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 40. Втори от трите метода в AsyncTask. Това е така наречения метод doInBackground. В третия просто показания диалога се скрива.

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();
    App.loggedUserId =  new GsonBuilder().create().fromJson(result,
Teacher.class).id;
    Log.d(TAG, "result: " + result);
```

Фигура 41. Трети от трите метода в AsyncTask. Така наречения onPostExecute метод. Всичко това става доста бързо, при наличито на добра и здрава връзка с интернет.

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

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

Влизането(Login) на преподавател в приложението. Тук вместо обекта User за целта ще се използва обекта Teacher, чиито код е показан на **Фигура 36.**

Всичко тук е почти идентично с кода за студент, само че се отнася за преподавател. Вижда се как се използва отново AsyncTask-а с данните въведени от преподавател чрез клавиатурата на телефона си в съответните EditText полета.

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 регистрацият 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.

```
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 42. Активити, което съдържа кода за Вход на преподавател 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 оцновните 3 метода onPreExecute(работеш на главната нишка), doInBackground(стартиращ своя собствена нишка отделно от главната) и onPostExecute(връщашт 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 43. 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 следият кода:

```
public class TeacherCreateDisciplineFragment extends Fragment {
    Button ok;
    Button back;
    EditText disciplineName;
    EditText 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 44.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 45. 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 setOnItemSelected за spiner-а.

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

Фигура 46. App класа, който използваме 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 47. 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 48. 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 AskynTask 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 преподавате 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 TeacherCreateQuestioFragment 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 типовате въпроси.

Като вече за TeacherGiveAnswerForQuestionFragment след като се взема самият въпрос от TeacherCreateQuestioFragment 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 име е необходима допълнителна документация как да си решат теста в приложението.

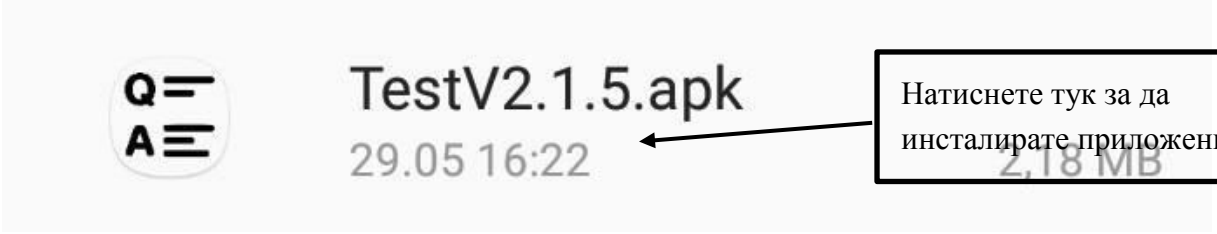
За по-лесна употреба ще бъде добавен един User Guide, който ще бъде представен по-долу в документацията и ще може да се използва от клиентите при евентуален проблем, така също и от разработчиците на Андроид приложения за по-голяма яснота.

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

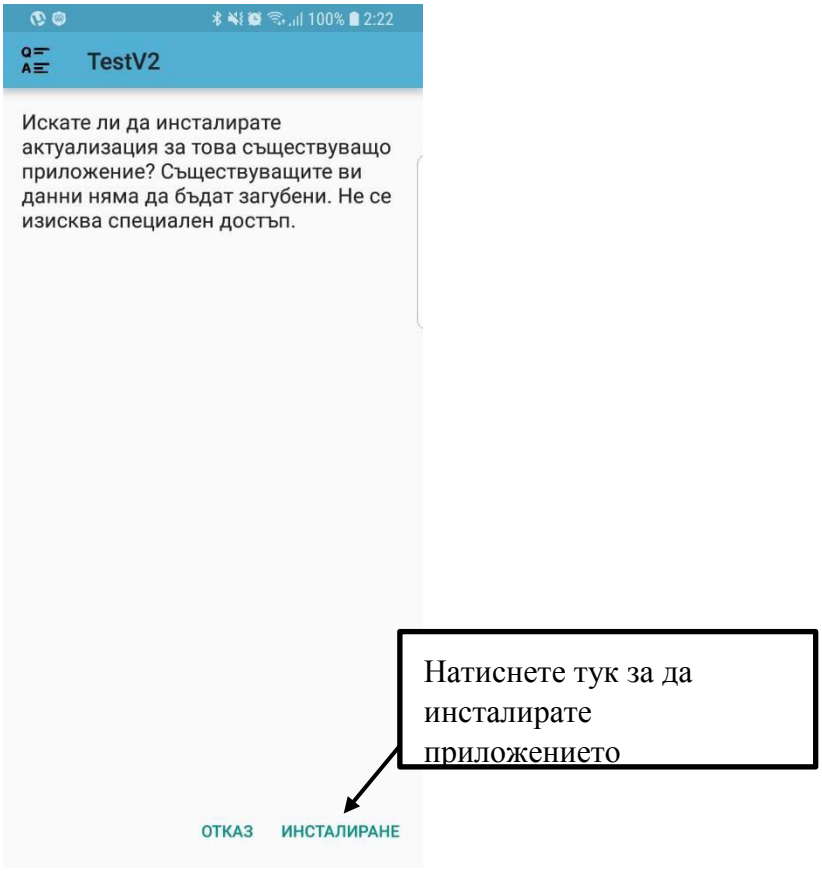
За клиентите (студент или преподавател) вече щом имат apk файл единственото което трябва да направят е да натиснат върху него и да го инсталират на телефонните си устройства.



Фигура 49. Тази фигура показва как се инсталира приложението на необходимото Андроид 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, изпозващи това приложение.



Фигура 50. Подробна инсталация на приложението за работа. След като се сдобиете с apk файла както беше обяснено по-горе по този начин показан по-горе се инсталира.

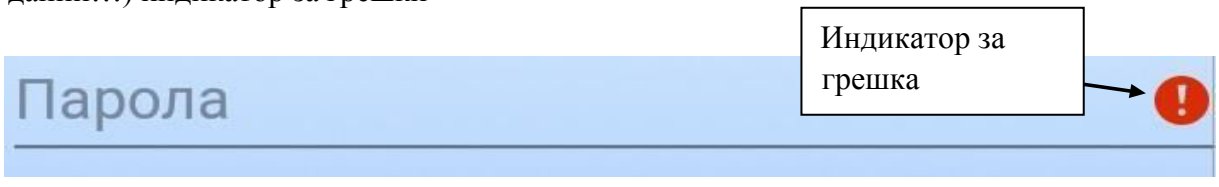
5.2. Влизне на студент в приложението

Фигура 51. На тази фигура се показва табхост-а необходим за начало на приложението.

Този табхост се използва както от студент, така и от преподавател за приложението.

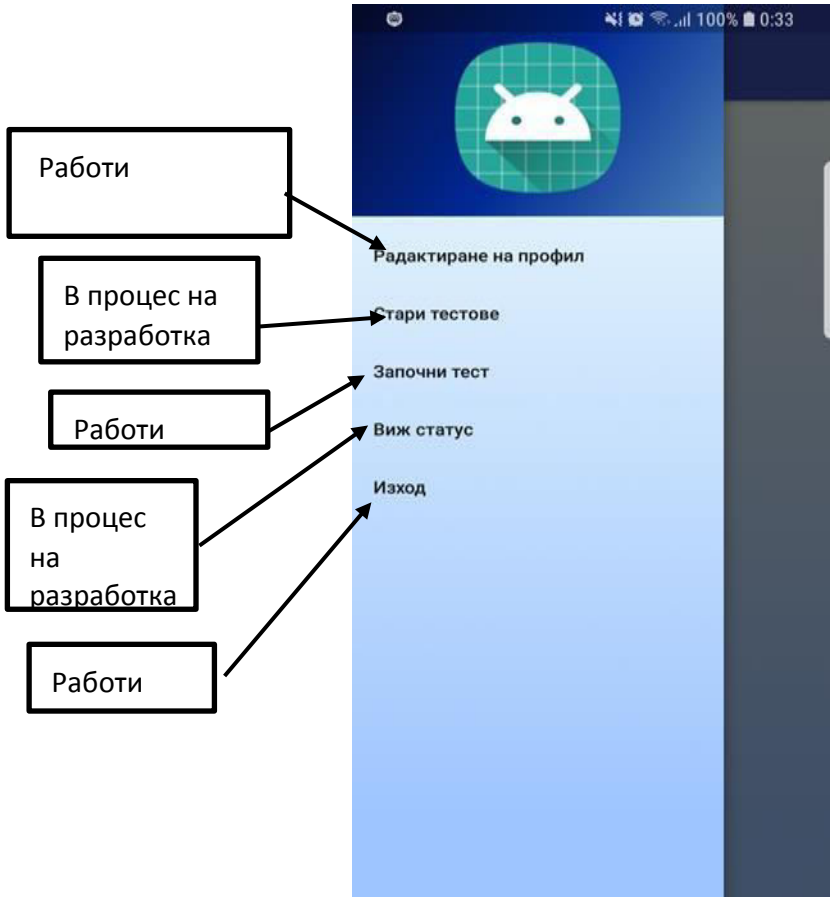
Тук след като апликацията се инсталира на телефона на потребителя той бива посрещнат от един табХост, разполагщ както с информация за влизане на студент, така и с информация за влизане на преподавател

При натискане на бутона ‚Влез‘, ако има грешки ще се покаже (невалидни данни…) индикатор за грешки



Фигура 52. Тук се показва индикатор за грешки.

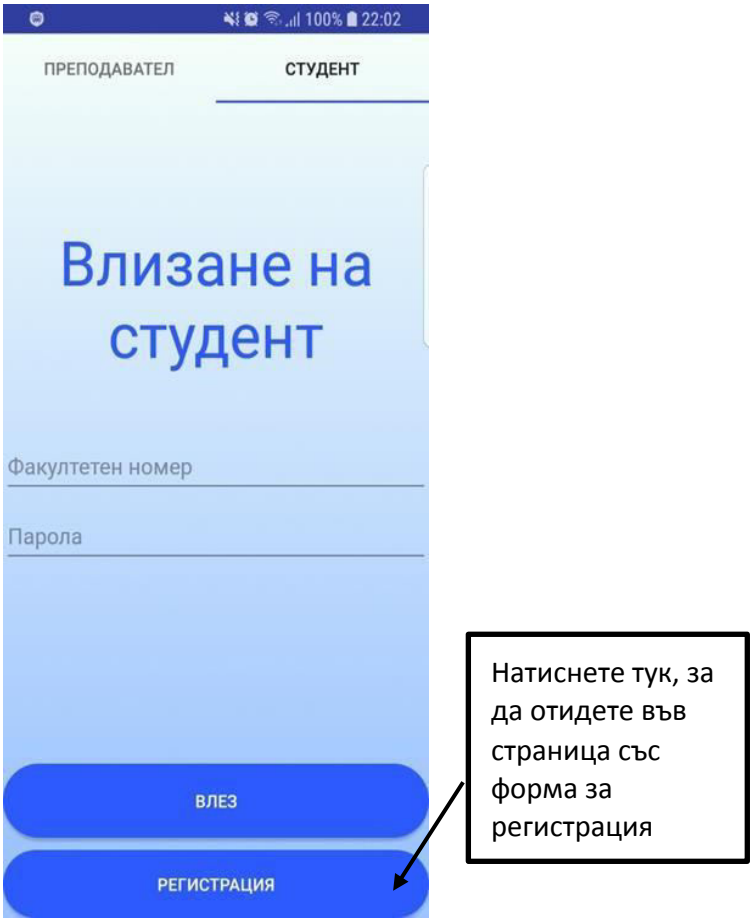
В противен случай ще влезем успешно в Меню, разполагащо с опциите на студент, като някои от опциите ще се разгледат, а други са в процес на допълнителна разработка.



Фигура 53. На тази фигура се вижда така наречения Draw menu и опциите известни за студента, като обаче някои от тях все още не функционират коректно.

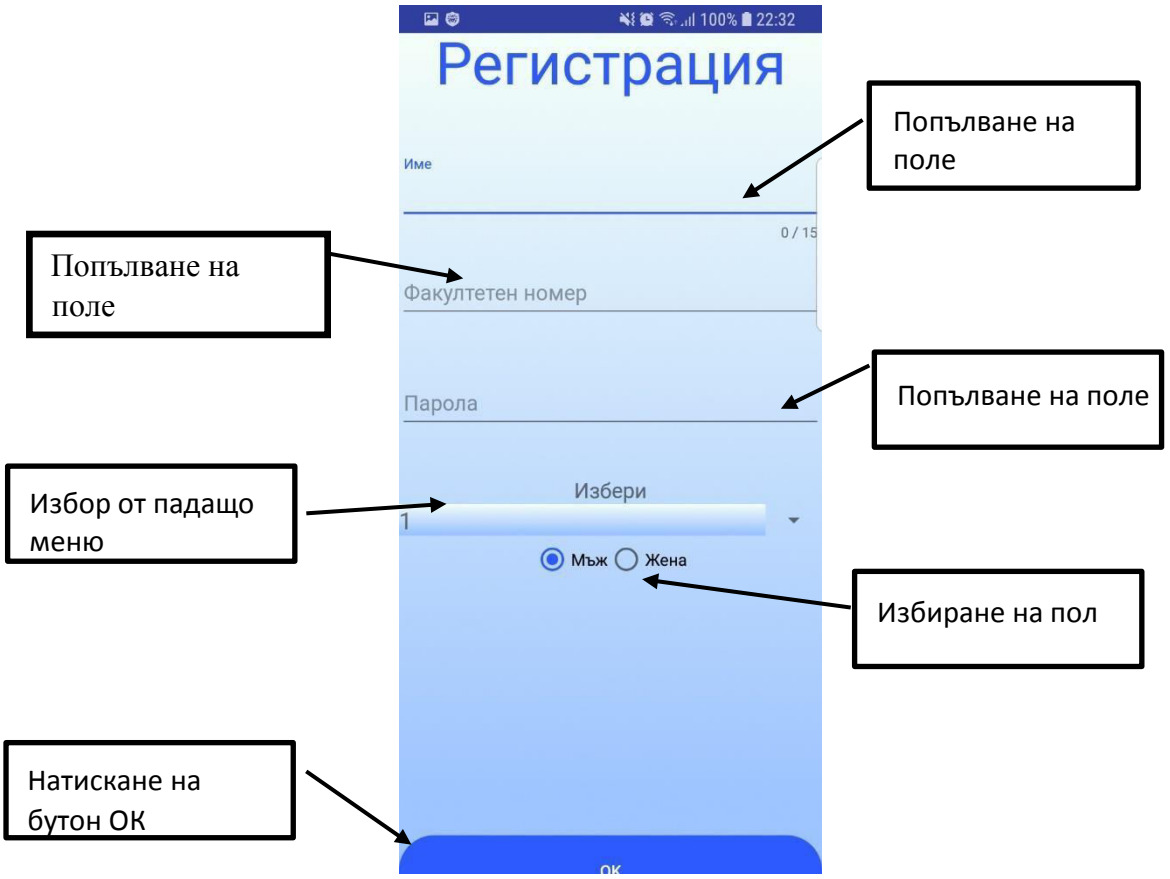
Нефункциониращите опции са отбелязани за по-голяма яснота на студента, за тази версия на приложението.

5.3. Регистриране на студент за приложението



Фигура 54. 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 се избере да се натисне бутона за ‚Регистрация‘, вместо ‚Влизане‘ се показва допълнителна страница, а именно страница за попълване на регистрационният формуляр за студента.

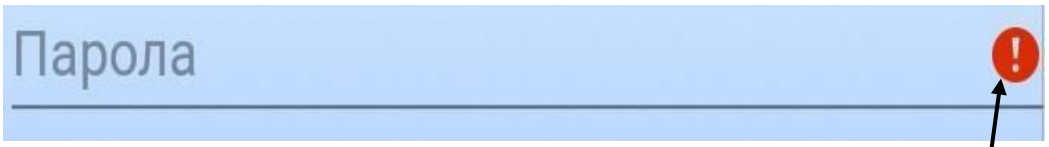


Фигура55. Тази форма показва така наречената формичка за регистраация на студенти, като всяко едно поле се взима и запазва във външната база за това приложение.

Интересно нещо тук е фактът, че студентите могат да прекратят своята регистрация чрез бутон ‚Изход‘, който за жалост не се вижда тук на изображението.

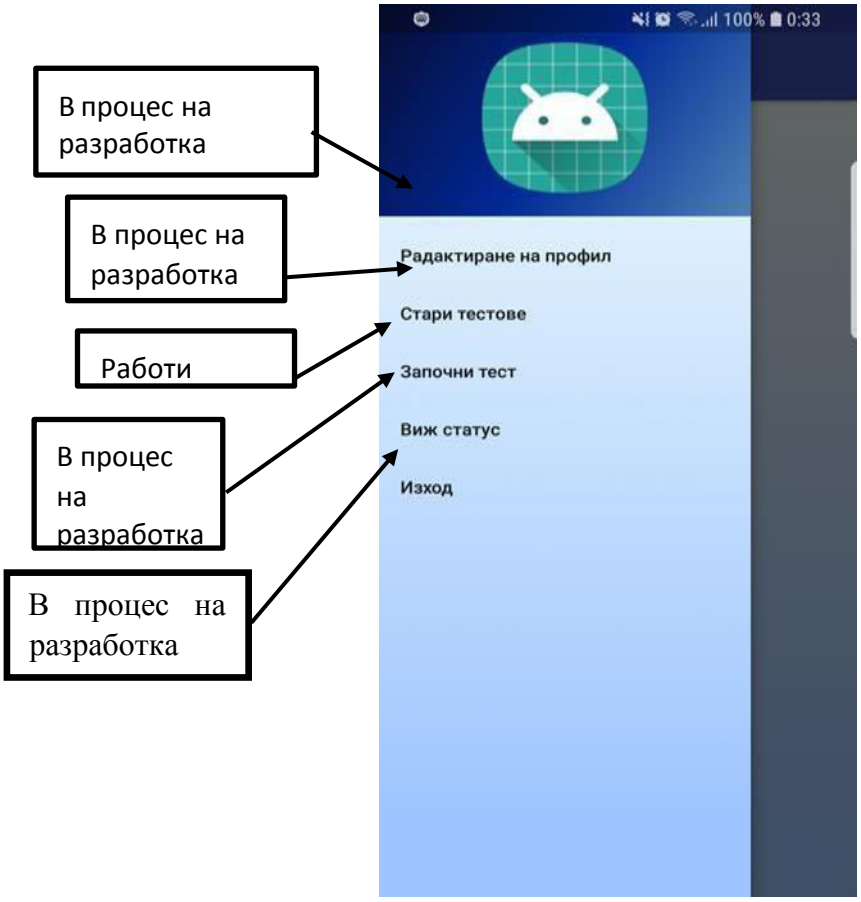
След като въведете желаните от вас данни следващата стъпка е да натиснете бутон ‚ОК‘.

Ако обаче не сте попълнили полетата, тогава ще ви излезе поле с грешка.



Фигура 56· Тази фигура показва, че е взникнала грешка.

След като се направи правилна регистрация, отново студентът се озовава на познатото му Draw Menu



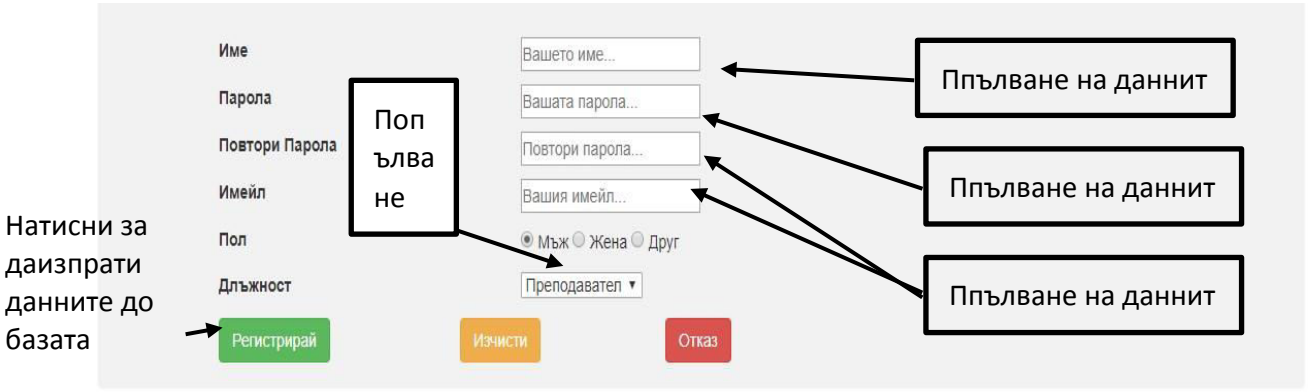
Фигура 57. Draw menu след регистрация на студент. Това Draw menu е същото като Draw menu-то, което се показва след влизане на студент, по този начин се осигурява достъп до него и по двата начина (регистрация или вход).

5.4. Регистриране на преподавател за приложението

Фигура 58. Начална страница за избор на опции от преподавателския сайт

Като начало първо трябва да се избере опцията за регистрация на преподавател в базата данни, за това служи и тази начална страница на сайта, както се вижда от линка по-горе, че се ползват jsp страници написани на java.

И така, като за начало регистрирането на преподавател става когато се избере предложената опция от менюто(за сега е само тази опция)

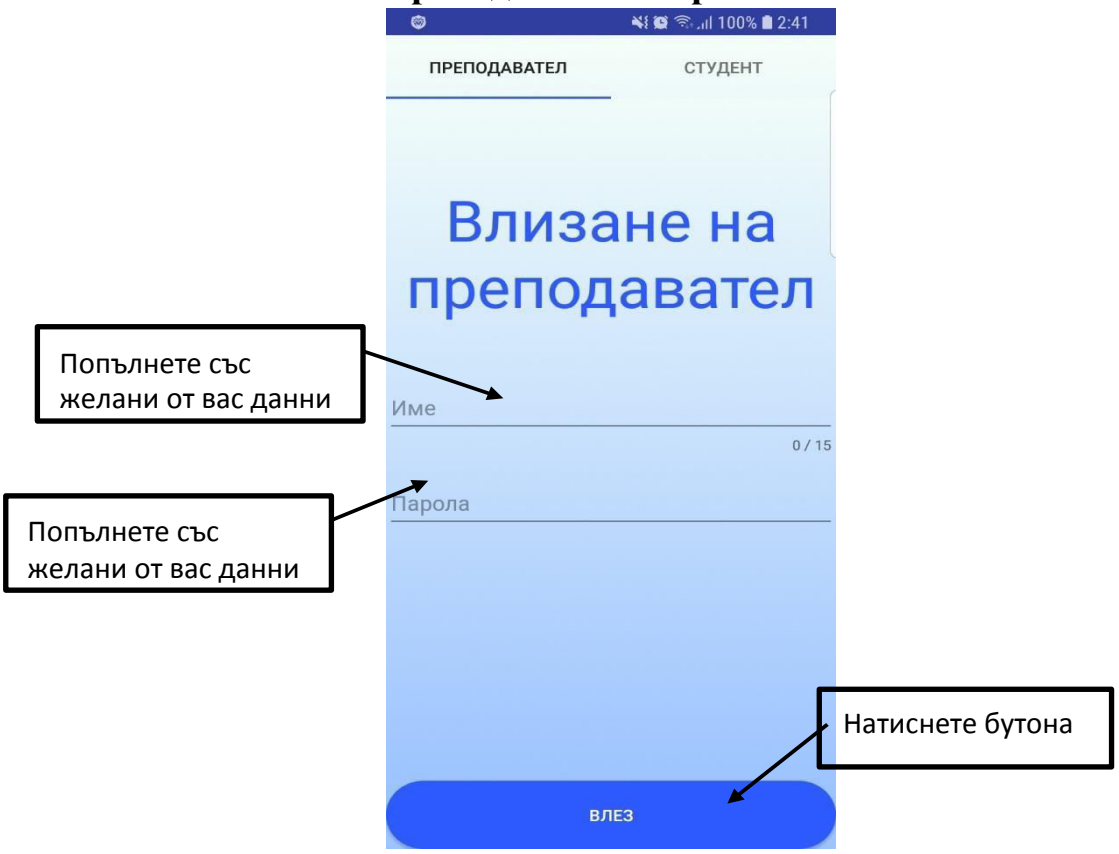


Фигура59. Тази фигура показва регистрационната форма за попълване от преподавател, за да се регистрира в базата данни.

След като преподавателят е кликнал на линка от началната страница за регистриране за него се отваря следната нова страница която съдържа форма с полета за попълване (Име, Парола, Повторете парола, Имейл, Пол, Длъжност)

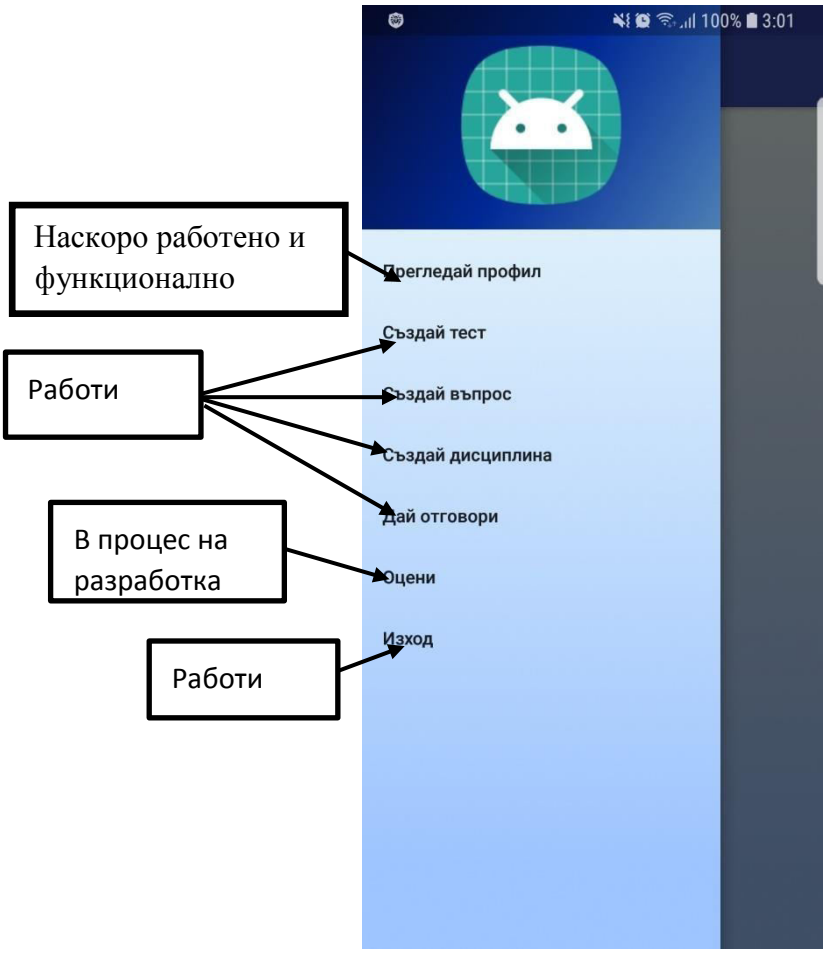
При попълване на желаните от него данни преподавателят вече е регистриран в базата данни и може да се извлича необходимата информация от нея за приложението.

5.5. Влизане на преподавател в приложението



Фигура60. 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 61. 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 деиствия, 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)

6. Заключение

Благодарение на архитектурата на цялосното приложение (както сървъра 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 разбрахте, че това приложение е версия 1.0, така че за бъдещо развитие планираме да довършим цялата функционалност да работи перфектно и без проблем.

Виждат се няколко „бъга“ в приложението, за това ние сме си поставили за цел да оправим всички „ бъгове“ в системата, било то за дата(бъг който виждате дори в момента на тази версия).

Смятаме да продължаваме да работим над сървисите и сървлетите, както и над AsyncTasks занапред за клиентското приложение

Също така да поработим над визуалния аспект на приложението (Front-end).

На този етап това са нашите планирани цели за бъдещето развитие на клиентсървър приложението.

За сега версия 1.0 (това приложение) използва доста проста и елементарна валидация. Тя работи до такава степен, че само да не позволява на преподавател или студент да влизат в него с празни пароли или мейл адреси. Друго яче казано, просто да не прпуснат да си въведат данните. Този тип валидация е доста лесен за разбиране, за това ние сме си поставили за цел в бъдещи версии да променим метода от client side (която се ползва сега) на server side, която е доста по-сигурна от тази.

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 AsyncTack 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.html за AsyncTask Example

AngryTools.com линк:

http://angrytools.com/gradient

За формирането на градиент за фон на приложението