МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационные системы и технологии

Специальность 1-98 01 03 «Программное обеспечение информационной

безопасности мобильных систем»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

Разработка программного средства «Поиск попутчиков»

Выполнил студент Никонович Алексей Владиславович

(Ф.И.О.)

Руководитель проекта асс. Колмаков М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой д.к.т.н., Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты асс. Колмаков М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтроллер асс. Колмаков М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2020

**Реферат**

Пояснительная записка курсового проекта содержит 30 страниц, 12 изображений, 4 источника литературы, 2 приложения.

Основными целями курсового проекта являются: разработка программного средства – «Поиск попутчиков».

Пояснительная записка состоит из введения, пяти разделов, заключения.

Во введении представлена общая информация, дающая представление о предстоящей работе, определены цели.

В первом разделе представлена постановка задачи.

Во втором разделе рассматривается разработка архитектуры проекта.

В третьем разделе рассматривается разработка функциональной модели приложения.

В четвертом разделе представлено руководство пользователя.

В заключении представлены итоги курсового проектирования и задачи, которые были решены в ходе разработки программного средства.

Содержание

[Введение 6](#_Toc515317847)

[1. Постановка задачи. 7](#_Toc515317848)

[2. Архитектура проекта 9](#_Toc515317849)

[3. Схема базы данных 10](#_Toc515317850)

[4. Программная реализация приложения 17](#_Toc515317851)

[4.1 Реализация сервера 17](#_Toc515317852)

[4.2 Реализация мобильного приложения 17](#_Toc515317853)

[4.2.1 REST-клиент 18](#_Toc515317854)

[5. Руководство пользователя 21](#_Toc515317855)

[5.1 Android приложение 21](#_Toc515317856)

[Заключение 27](#_Toc515317857)

[Список использованных источников 28](#_Toc515317858)

[Приложение А 29](#_Toc515317859)

[Приложение B 29](#_Toc515317860)

# Введение

В последнее время мир резко ускорился. Мобильная связь, интернет, социальные сети – за последние 10 лет настолько ускорили процесс передачи информации, что приложения как Web, так и мобильные используются во всех отраслях жизни.

Темой курсового проекта является разработка программного средства «Поиск попутчиков». Данное решение позволяет водителям создавать поездки, а пользователям записываться на них, производить поиск, сортировку и выставлять оценки водителям.

Решение состоит из двух частей: Android-приложения и Web-сервера. Приложение состоит из визуальной части, представленной классическими активностями android, написанными на Java. Сервер спроектирован с помощью Java и Spring Framework и имеет связь с базой данных MySQL.

Решение поможет водителям уменьшить затраты на поездку и найти пассажиров.

Решение поможет пассажирам находить водителей, чтобы добраться до пункта назначения.

Целью курсового проекта является проектирование и реализация программного средства «Поиск попутчиков».

# Постановка задачи.

Spring Framework (или коротко Spring) — универсальный фреймворк с открытым исходным кодом для Java-платформы. Также существует форк для платформы .NET Framework, названный Spring.NET.

Spring стал широко распространённым в Java-сообществе главным образом как альтернатива и замена модели Enterprise JavaBeans. Spring предоставляет бо́льшую свободу Java-разработчикам в проектировании; кроме того, он предоставляет хорошо документированные и лёгкие в использовании средства решения проблем, возникающих при создании приложений корпоративного масштаба.

Между тем, особенности ядра Spring применимы в любом Java-приложении, и существует множество расширений и усовершенствований для построения веб-приложений на Java Enterprise платформе. По этим причинам Spring приобрёл большую популярность и признаётся разработчиками как стратегически важный фреймворк.

MySQL — свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB. Продукт распространяется как под GNU General Public License, так и под собственной коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей. Именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.  
 Java Persistence API (*JPA*) — спецификация API Java EE, предоставляет возможность сохранять в удобном виде Java-объекты в базе данных.  
 Существует несколько реализаций этого интерфейса, одна из самых популярных использует для этого Hibernate. JPA реализует концепцию ORM.

При разработке сервера следует реализовать следующие API:

* REST – это стиль архитектуры программного обеспечения для распределенный систем, таких как WorldWideWeb, который, как правило используется для построения веб-служб.

Клиентом будет выступать мобильное приложение на устройстве с операционной системой Android 6.0 и выше.

# Архитектура проекта

Распределенное приложение состоит из 2-х частей:

Сервер. Данная часть приложения предоставляет REST API и модуль SOAP. Доступ к данным удаленной базы осуществляется с помощью CRUD операций.

Мобильное приложение. Эта часть использует HTTPURLConnection для синхронизации информации локальной базы данных мобильного приложения с удаленной базой данных.

Графическое представление архитектуры распределенного приложения изображено на рисунке 2.1.

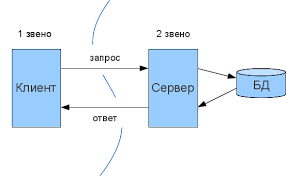
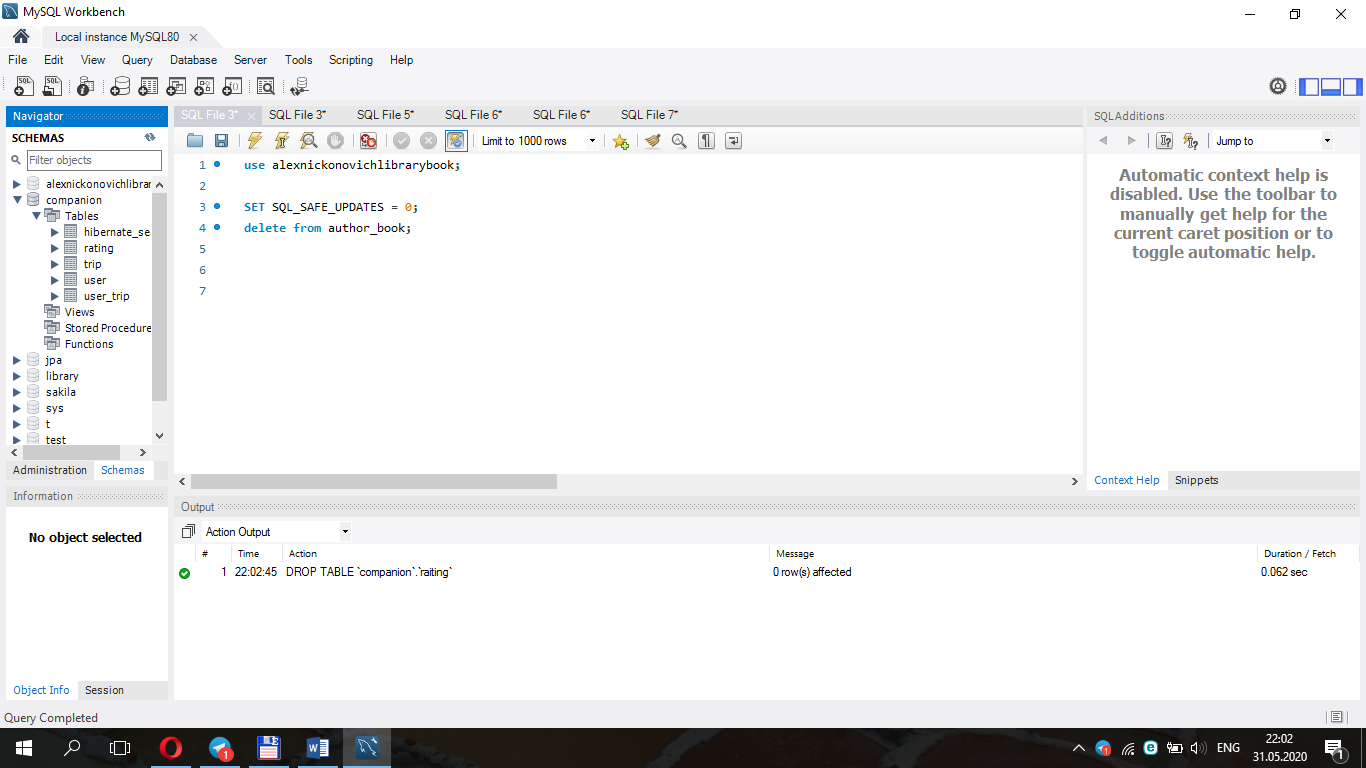


Рисунок. 2.1 – архитектура распределенного приложения

# Схема базы данных

В ходе написания приложения была разработана удаленная база данных «Companion».

В таблице User хранятся данные пользователей. В таблице Trip находится информация о поездках. Таблица Trip показывает пользователей, участвующих в поездке. Таблица Rating хранит отзывы пользователей.



# Программная реализация приложения

## 4.1 Реализация сервера

Программная модель Spring Framework основывается на протоколе [HTTP](https://ru.wikipedia.org/wiki/HTTP) и использует его правила взаимодействия между сервером и браузером. Сервер имеет следующие пути для обработки HTTP-запросов (Приложение А)

* Get – получение курсовых работ или пользователей;
* Delete – для удаления пользователя или работы;
* Post – добавление информации курсовой работе или пользователе;
* Put – обновление информации.

## 4.2 Реализация мобильного приложения

Мобильное приложение реализовано для операционной системы Android 6.0 и выше. Данное приложение было разработано в специальной среде разработки –Android Studio.

Графическая структура классов представлена на рисунке 4.2.1.

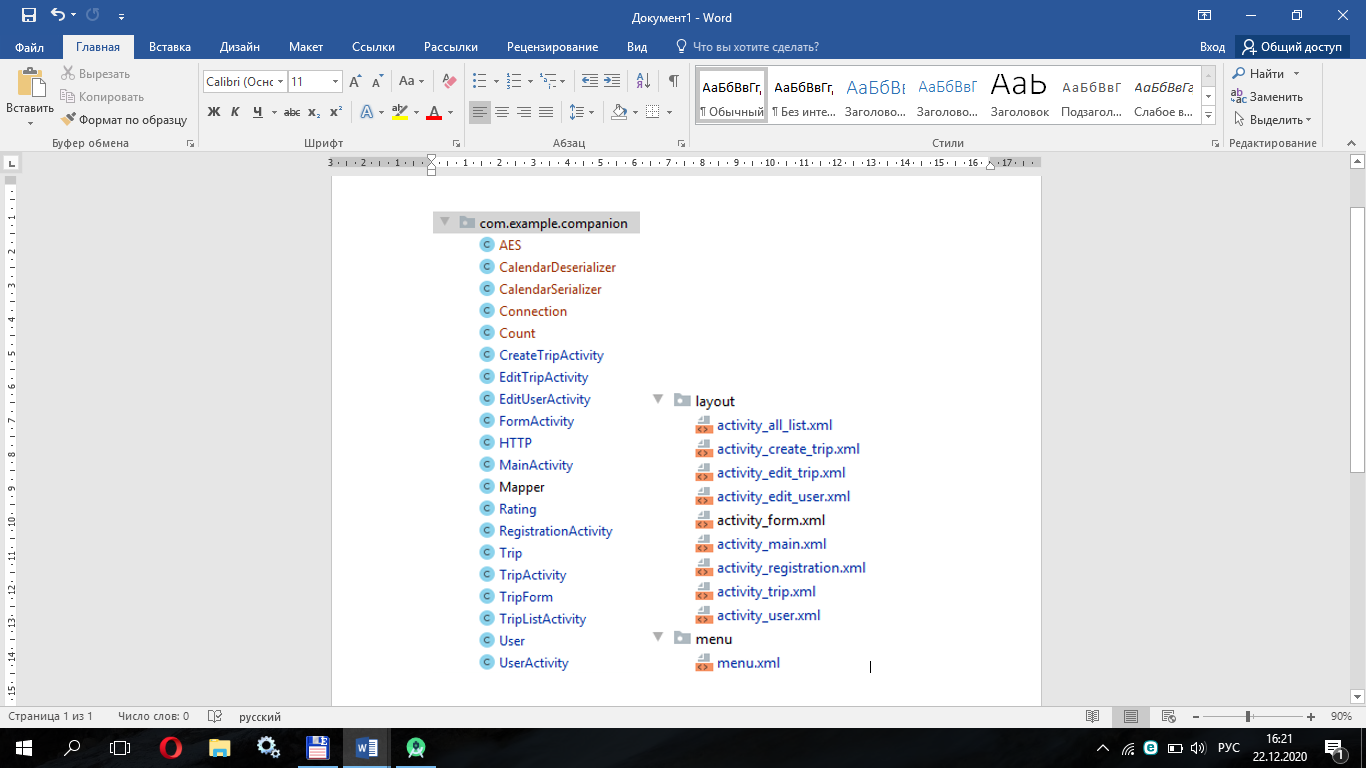


Рисунок 4.2.1 – структура проекта

### Проект в себе содержит следующие классы:

1. AES – шифрование объектов в AES
2. CalendarDeserializer – десериализация даты и времени из JSON;
3. CalendarSerializer – сериализация даты и времени в JSON;
4. Connection – запись пользователя на поездку;
5. Count – количество поездок пользователя
6. CreateTripActivity – активность создания поездки;
7. EditTripActivity – активность редактирования поездки;
8. EditUserActivity – активность редактирования пользователя;
9. FormActivity – активность задания условий поиска;
10. HTTP– работа с HTTP--запросами;
11. MainActivity – загрузочная страница и активность авторизации;
12. Mapper – парсинг данных;
13. Rating – класс оценок пользователей;
14. RegistrationActivity – активность регистрции;
15. Trip – класс поездки;
16. TripActivity – активность отображения поездки;
17. TripForm – класс результатов поиска.
18. TripListActivity – активность отображения поездок;
19. User – класс пользователя;
20. UserActivity – активность отображения информации пользователя;

### **4.2.1 REST-клиент**

Для синхронизации мобильного приложения с сервером применяется архитектурный стиль REST. Основная задача сервера в этой архитектуре предоставить клиентам доступ к ресурсам по их идентификаторам. Под доступом подразумевается как получение информации, так и ее изменение или удаление. В зависимости от реализации сервиса, в ответ на наш запрос придёт ответ в формате JSON. На рисунке 4.2.1.1 представлен пример одного из get-запросов.

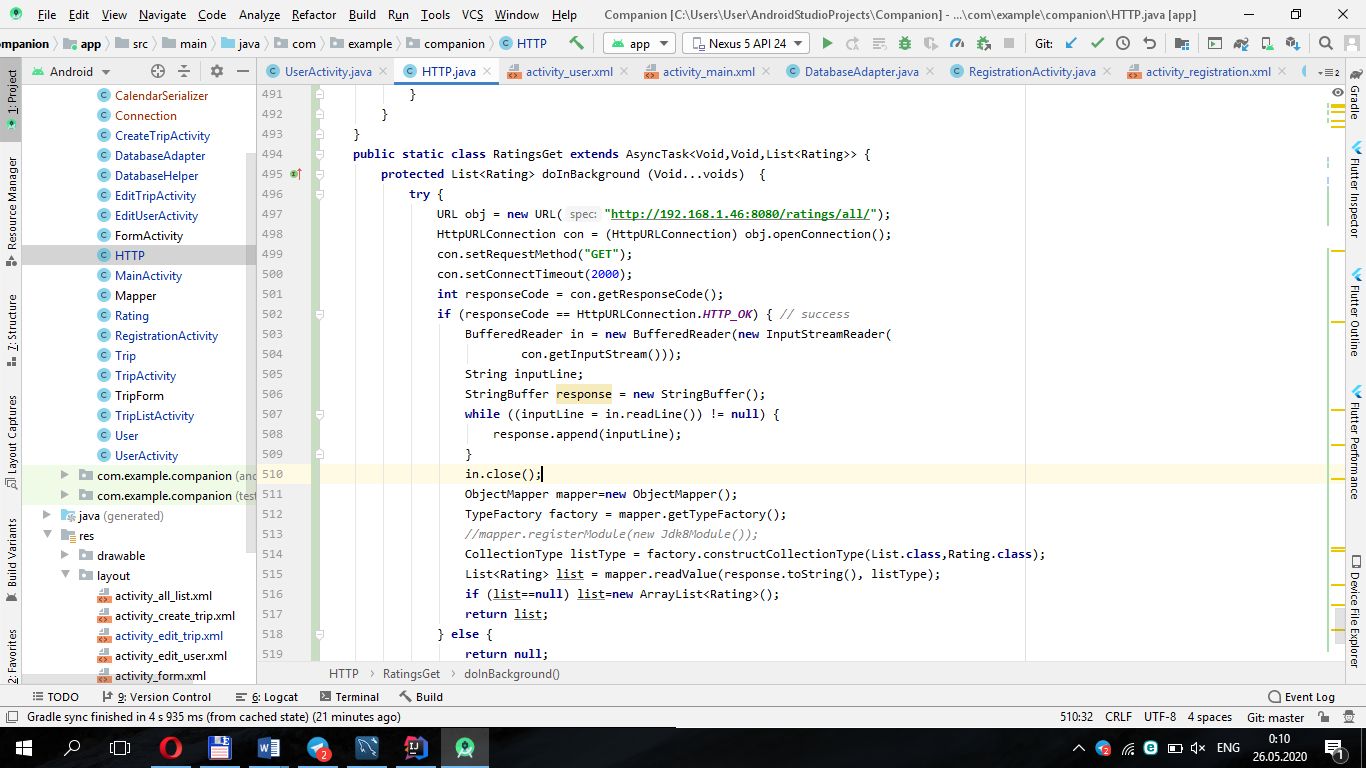


Рисунок 4.2.1.1 – get-запрос

GET – позволяет получить данные, а также конкретный ответ по заданным параметрам. POST – позволяет передать данные на сервер. Например, передача выбранной избранной курсовой работы, для последующей записи в удаленную базу данных.

Таким образом можно сказать, что REST – это взаимодействие «клиент-серверной» архитектуры с помощью HTTP-запросов и HTTP-ответов по протоколу, не подтверждающему состояние.

# Руководство пользователя

## 5.1 Android приложение

Для начала необходимо установить приложение на мобильный устройство с API не меньше 23 (Android 6.0). Запустим наше приложение. Сразу после запуска мы увидим окно авторизации и регистрации (Рисунок 5.1.1). Авторизация происходит с помощью GET-запроса к серверу.

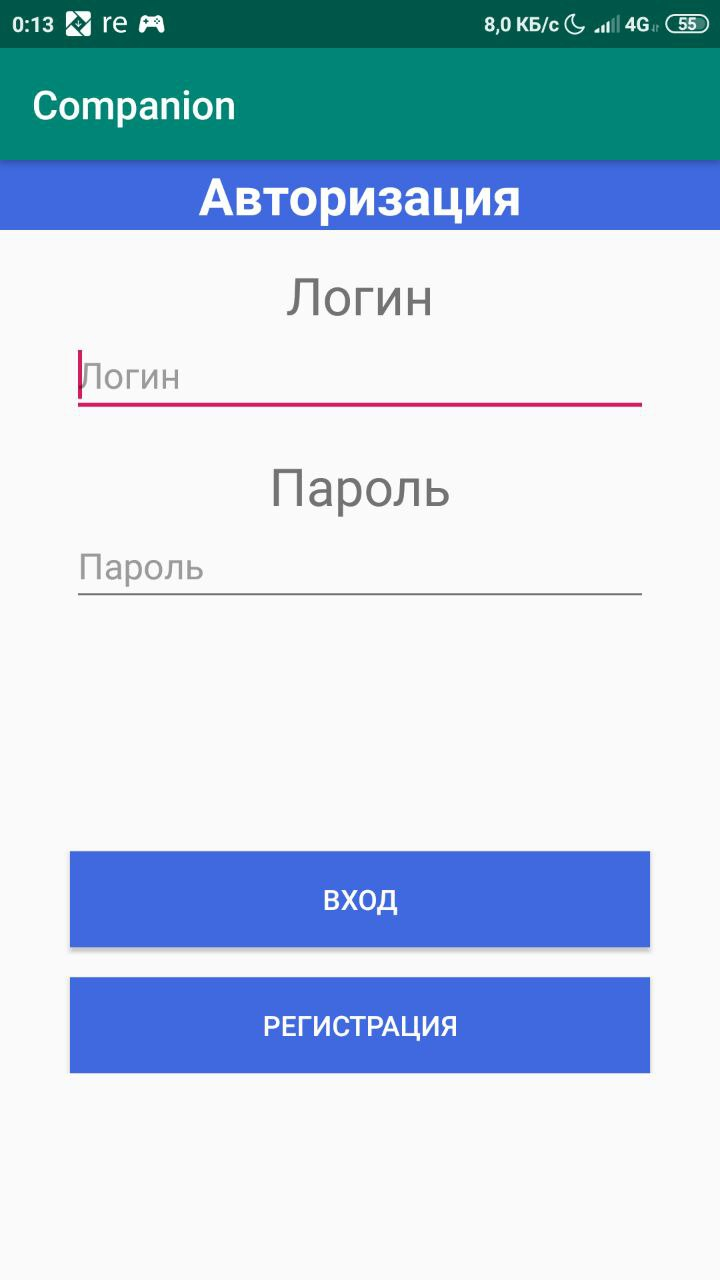


Рисунок 5.1.1 – окно авторизации и регистрации

После ввода данных (имя пользователя и пароль) мы можем видеть все поездки, которые есть на данный момент (Рисунок 5.1.2).

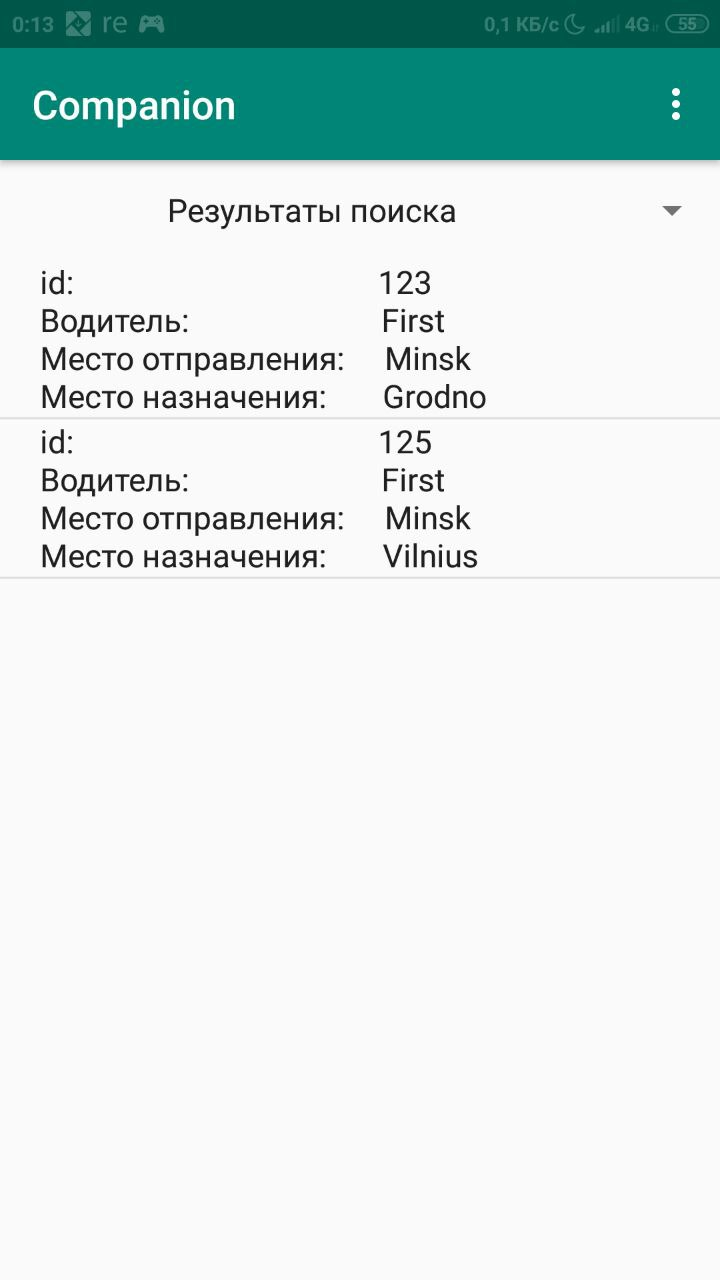
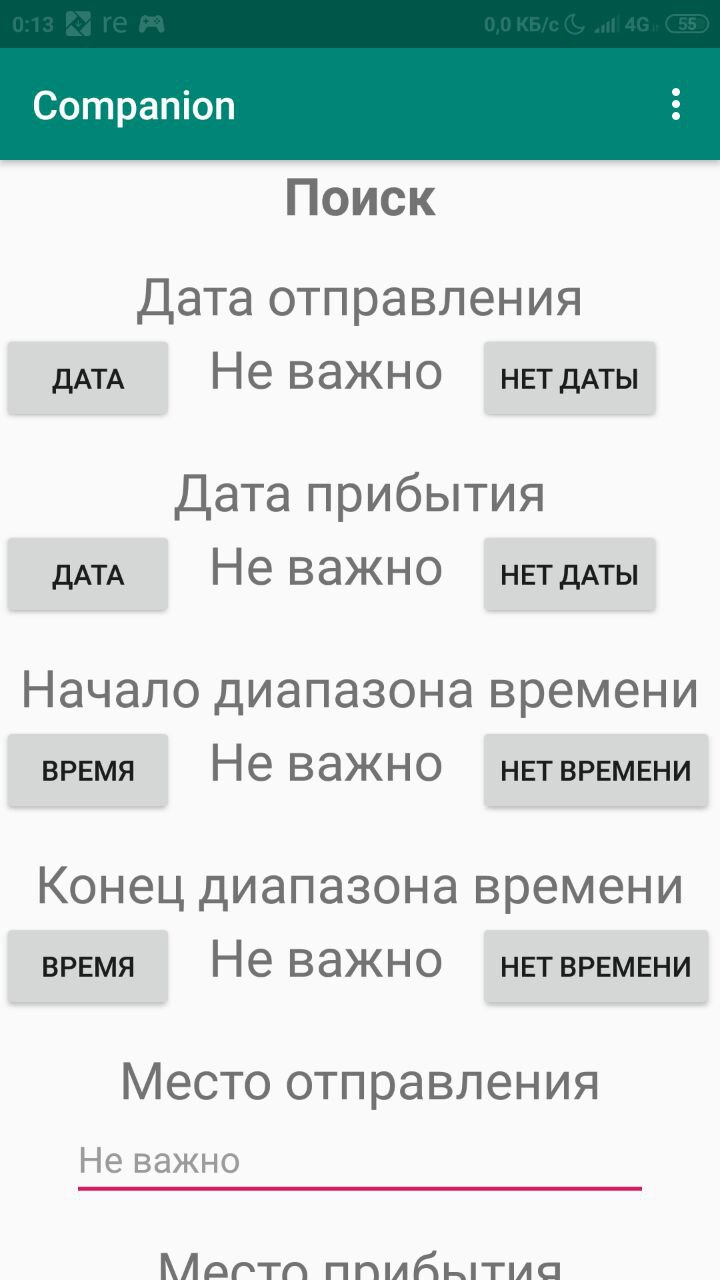


Рисунок 5.1.2 – все поездки

При помощи меню можно перейти на форму поиска (Рисунок 5.1.3).



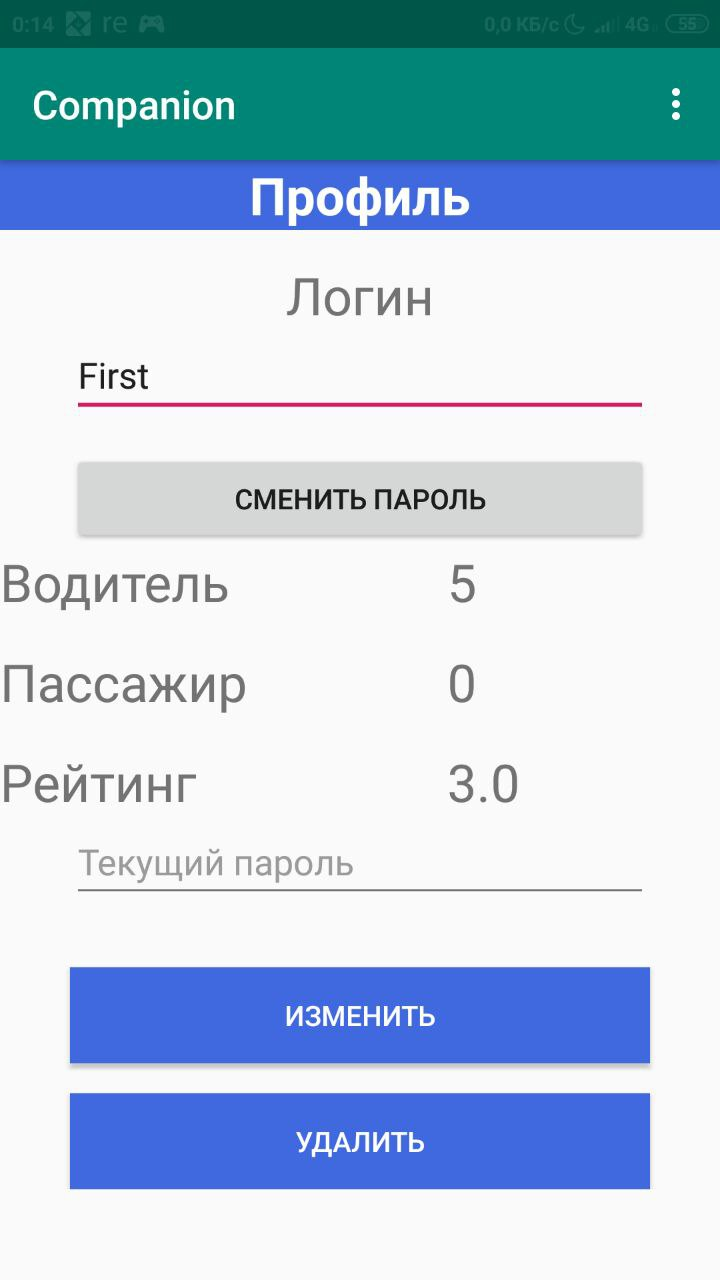
Риснок 5.1.3 – форма поиска

Также через меню можно к перейти к форме создания поездки (Рисунок 5.1.4).



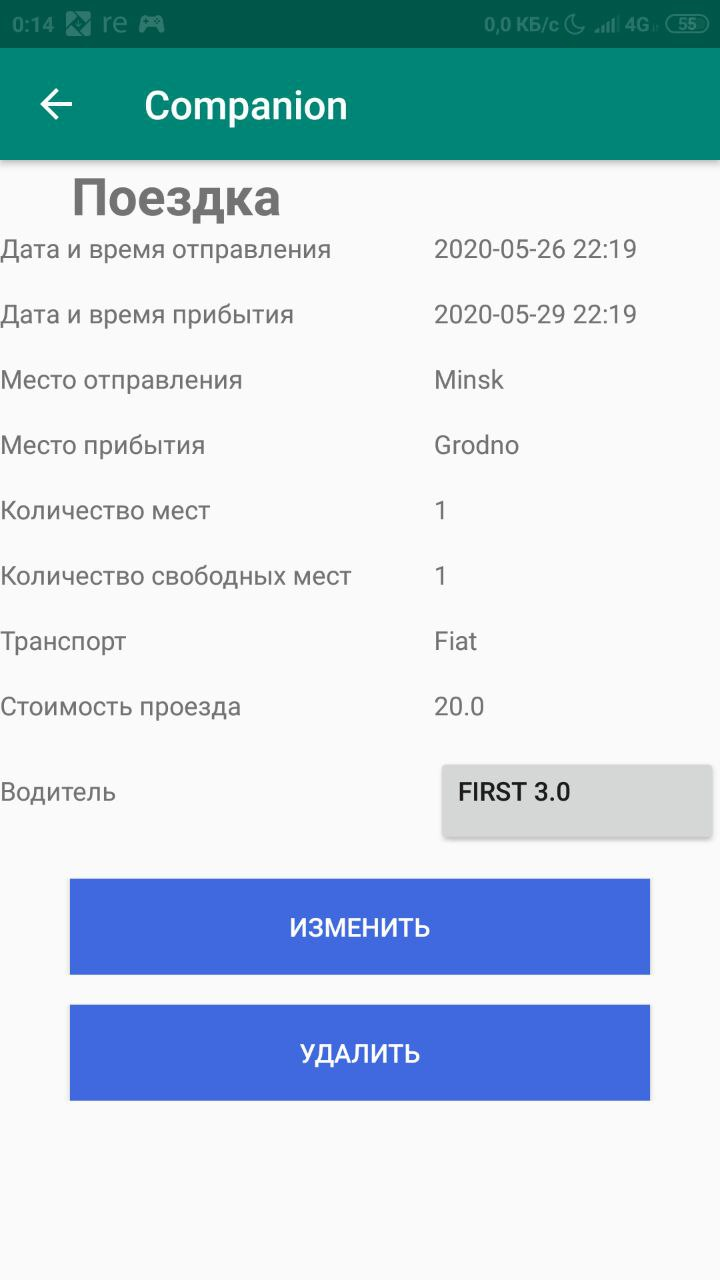
Риснок 5.1.4 – форма создания поездки

Существует также форма редактирования пользователя (Рисунок 5.1.5).



Риснок 5.1.5 – форма редактирования пользователя

При выборе поездки на форме результатов осуществляется переход на страницу с информацией о поездке, также на этой странице можно забронировать место, а водитель может ее удалить (Рисунок 5.1.6).



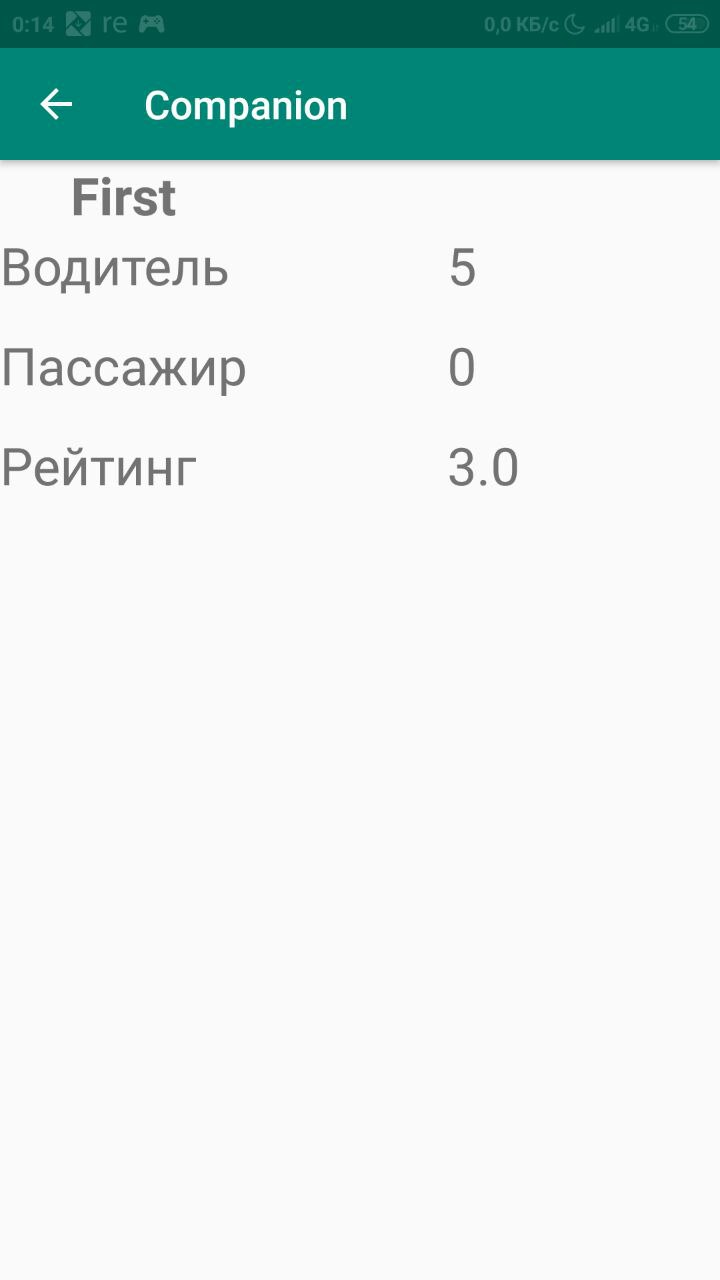
Риснок 5.1.6 – страница информации о поездке

Со страницы поездки водитель может перейти к форме редактирования (Рисунок 5.1.7).



Риснок 5.1.7 – форма редактирования поездки

Также со страницы поездки можно перейти к просмотру информации о водителе (Рисунок 5.1.8).



Риснок 5.1.8 – страница просмотра информации о водителе

# Заключение

1. В итоге разработки курсового проекта было легкорасширяемое android-приложение.
2. В процессе разработки был произведен анализ аналогов, выявление слабых и сильных сторон приложений. Эта информация была использована для улучшения функционала приложения и пользовательского интерфейса.
3. Была создана база данных, которая полностью отвечает поставленным требованиям. На ее основе были разработаны представления разных уровней.

Решая поставленную задачу, пришли к удовлетворительному результату. Таким образом, была достигнута цель, и было создан проект «Поиск попутчиков», который включает в себя android-клиент и сервер.

Программная система имеет простой, интуитивно понятный пользователю интерфейс.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

В результате приложение обладает следующими особенностями:

* Создание и редактирование профиля;
* Создание, редактирование и удаление поездки;
* Запись (отпись) на (от) поездку(и);
* Просмотр профиля другого пользователя ;
* Возможность поставить оценку другому пользователю;

# Список использованных источников

1. Spring Framework -https://spring-projects.ru/projects/spring-framework/;

2. JPA -https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#reference;

3. MySQL -https://dev.mysql.com/doc/;

4. Android documentation - https://developer.android.com/docs.

# Приложение А

**package** com.example.companion;  
  
**import** android.os.AsyncTask;  
**import** com.fasterxml.jackson.databind.ObjectMapper;  
**import** com.fasterxml.jackson.databind.type.CollectionType;  
**import** com.fasterxml.jackson.databind.type.TypeFactory;  
**import** java.io.BufferedReader;  
**import** java.io.InputStreamReader;  
**import** java.io.OutputStream;  
**import** java.net.HttpURLConnection;  
**import** java.net.URL;  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**public class** HTTP {  
 **public static class** UserPost **extends** AsyncTask<Object,Void,Integer> {  
 **protected** Integer doInBackground (Object... object) {  
 **try** {  
 URL obj = **new** URL(**"http://192.168.1.46:8080/users/add/"**);  
 HttpURLConnection con = (HttpURLConnection) obj.openConnection();  
 con.setRequestMethod(**"POST"**);  
 con.setRequestProperty(**"Content-Type"**, **"application/json"**);  
 con.setConnectTimeout(2000);  
 con.setDoOutput(**true**);  
 OutputStream os = con.getOutputStream();  
 ObjectMapper mapper=**new** ObjectMapper();  
 os.write(mapper.writeValueAsString(object[0]).getBytes());  
 os.flush();  
 os.close();  
 **int** responseCode = con.getResponseCode();  
 **return** responseCode;  
 }  
 **catch** (Exception e) {  
 **return** -1;  
 }  
 }  
 }  
 **public static class** UserCountPost **extends** AsyncTask<String,Void,Object> {  
 **protected** Object doInBackground (String... strings) {  
 **try** {  
 URL obj = **new** URL(**"http://192.168.1.46:8080/users/count/"**+strings[0]);  
 HttpURLConnection con = (HttpURLConnection) obj.openConnection();  
 con.setRequestMethod(**"POST"**);  
 con.setConnectTimeout(2000);  
 **int** responseCode = con.getResponseCode();  
 **switch** (responseCode) {  
 **case** HttpURLConnection.***HTTP\_OK***:  
 BufferedReader in = **new** BufferedReader(**new** InputStreamReader(  
 con.getInputStream()));  
 String inputLine;  
 StringBuffer response = **new** StringBuffer();  
 **while** ((inputLine = in.readLine()) != **null**) {  
 response.append(inputLine);  
 }  
 in.close();  
 ObjectMapper mapper=**new** ObjectMapper();  
 Count count = mapper.readValue(response.toString(), Count.**class**);  
 **return** count;  
 **case** HttpURLConnection.***HTTP\_NOT\_FOUND***: {  
 **return new** Count(**null**,**null**);  
 }  
 **default**: **return null**;  
 }  
 }  
 **catch** (Exception e) {  
 **return null**;  
 }  
 }  
 }  
 **public static class** UsersGet **extends** AsyncTask<Void,Void,List<User>> {  
 **protected** List<User> doInBackground (Void...voids) {  
 **try** {  
 URL obj = **new** URL(**"http://192.168.1.46:8080/users/all/"**);  
 HttpURLConnection con = (HttpURLConnection) obj.openConnection();  
 con.setRequestMethod(**"GET"**);  
 con.setConnectTimeout(2000);  
 **int** responseCode = con.getResponseCode();  
 **if** (responseCode == HttpURLConnection.***HTTP\_OK***) {   
 BufferedReader in = **new** BufferedReader(**new** InputStreamReader(  
 con.getInputStream()));  
 String inputLine;  
 StringBuffer response = **new** StringBuffer();  
 **while** ((inputLine = in.readLine()) != **null**) {  
 response.append(inputLine);  
 }  
 in.close();  
 ObjectMapper mapper=**new** ObjectMapper();  
 TypeFactory factory = mapper.getTypeFactory();  
 CollectionType listType = factory.constructCollectionType(List.**class**,User.**class**);  
 List<User> list = mapper.readValue(response.toString(), listType);  
 **return** list;  
 } **else** {  
 **return null**;  
 }  
 }  
 **catch** (Exception e) {  
 **return null**;  
 }  
 }  
 }  
 **public static class** UserGet **extends** AsyncTask<String,Void, Object> {  
  
 **protected** Object doInBackground (String...strings) {  
 **try** {  
 URL obj = **new** URL(**"http://192.168.1.46:8080/users/"**+strings[0]);  
 HttpURLConnection con = (HttpURLConnection) obj.openConnection();  
 con.setRequestMethod(**"GET"**);  
 con.setConnectTimeout(2000);  
 **int** responseCode = con.getResponseCode();  
 **switch** (responseCode) {  
 **case** HttpURLConnection.***HTTP\_OK***:  
 BufferedReader in = **new** BufferedReader(**new** InputStreamReader(  
 con.getInputStream()));  
 String inputLine;  
 StringBuffer response = **new** StringBuffer();  
 **while** ((inputLine = in.readLine()) != **null**) {  
 response.append(inputLine);  
 }  
 in.close();  
 ObjectMapper mapper=**new** ObjectMapper();  
 User user = mapper.readValue(response.toString(), User.**class**);  
 **return** user;  
 **case** HttpURLConnection.***HTTP\_NOT\_FOUND***: {  
 **return null**;  
 }  
 **default**: **return** -1;  
 }  
 }  
 **catch** (Exception e) {  
 **return** -1;  
 }  
 }  
 }  
 **public static class** UserDelete **extends** AsyncTask<String,Void,Boolean> {  
 **protected** Boolean doInBackground(String... strings) {  
 **try** {  
 URL obj = **new** URL(**"http://192.168.1.46:8080/users/delete/"** + strings[0]);  
 HttpURLConnection con = (HttpURLConnection) obj.openConnection();  
 con.setRequestMethod(**"DELETE"**);  
 **int** responseCode = con.getResponseCode();  
 **if** (responseCode == HttpURLConnection.***HTTP\_OK***) {   
 **return true**;  
 } **else** {  
 **return false**;  
 }  
  
 } **catch** (Exception e) {  
 **return false**;  
 }  
 }  
 }  
 **public static class** UserPut **extends** AsyncTask<Object,Void,Integer> {  
 **protected** Integer doInBackground (Object... object) {  
 **try** {  
 URL obj = **new** URL(**"http://192.168.1.46:8080/users/edit/"**+object[0].toString());  
 HttpURLConnection con = (HttpURLConnection) obj.openConnection();  
 con.setRequestMethod(**"PUT"**);  
 con.setRequestProperty(**"Content-Type"**, **"application/json"**);  
 con.setConnectTimeout(2000);  
 con.setDoOutput(**true**);  
 OutputStream os = con.getOutputStream();  
 ObjectMapper mapper=**new** ObjectMapper();  
 os.write(mapper.writeValueAsString(object[1]).getBytes());  
 os.flush();  
 os.close();  
 **int** responseCode = con.getResponseCode();  
 **return** responseCode;  
 }  
 **catch** (Exception e) {  
 **return** -1;  
 }  
 }  
 }  
 **public static class** TripsPost **extends** AsyncTask<Object,Void,List<Trip>> {  
 **protected** List<Trip> doInBackground (Object...object) {  
 **try** {  
 URL obj = **new** URL(**"http://192.168.1.46:8080/trips/all/"**);  
 HttpURLConnection con = (HttpURLConnection) obj.openConnection();  
 con.setRequestMethod(**"POST"**);  
 con.setRequestProperty(**"Content-Type"**, **"application/json"**);  
 con.setConnectTimeout(2000);  
 OutputStream os = con.getOutputStream();  
 ObjectMapper mapper=**new** ObjectMapper();  
 os.write(mapper.writeValueAsString(object[0]).getBytes());  
 os.flush();  
 os.close();  
 **int** responseCode = con.getResponseCode();  
 **if** (responseCode == HttpURLConnection.***HTTP\_OK***) {  
 BufferedReader in = **new** BufferedReader(**new** InputStreamReader(  
 con.getInputStream()));  
 String inputLine;  
 StringBuffer response = **new** StringBuffer();  
 **while** ((inputLine = in.readLine()) != **null**) {  
 response.append(inputLine);  
 }  
 in.close();  
 mapper=**new** ObjectMapper();  
 TypeFactory factory = mapper.getTypeFactory();  
 CollectionType listType = factory.constructCollectionType(List.**class**,Trip.**class**);  
 List<Trip> list = mapper.readValue(response.toString(), listType);  
 **if** (list==**null**) list=**new** ArrayList<Trip>();  
 **return** list;  
 } **else** {  
 **return null**;  
 }  
 }  
 **catch** (Exception e) {  
 **return null**;  
 }  
 }  
 }  
 **public static class** TripsComplietedGet **extends** AsyncTask<String,Void,List<Trip>> {  
  
 **protected** List<Trip> doInBackground (String...strings) {  
 **try** {  
 URL obj = **new** URL(**"http://192.168.1.46:8080/trips/complieted/"**+strings[0]);  
 HttpURLConnection con = (HttpURLConnection) obj.openConnection();  
 con.setRequestMethod(**"GET"**);  
 con.setConnectTimeout(2000);  
 **int** responseCode = con.getResponseCode();  
 **if** (responseCode == HttpURLConnection.***HTTP\_OK***) {  
 BufferedReader in = **new** BufferedReader(**new** InputStreamReader(  
 con.getInputStream()));  
 String inputLine;  
 StringBuffer response = **new** StringBuffer();  
 **while** ((inputLine = in.readLine()) != **null**) {  
 response.append(inputLine);  
 }  
 in.close();  
 ObjectMapper mapper=**new** ObjectMapper();  
 TypeFactory factory = mapper.getTypeFactory();  
 CollectionType listType = factory.constructCollectionType(List.**class**,Trip.**class**);  
 List<Trip> list = mapper.readValue(response.toString(), listType);  
 **if** (list==**null**) list=**new** ArrayList<Trip>();  
 **return** list;  
 } **else** {  
 **return null**;  
 }  
 }  
 **catch** (Exception e) {  
 **return null**;  
 }  
 }  
 }  
 **public static class** TripsFutureGet **extends** AsyncTask<String,Void,List<Trip>> {  
  
 **protected** List<Trip> doInBackground (String...strings) {  
 **try** {  
 URL obj = **new** URL(**"http://192.168.1.46:8080/trips/future/"**+strings[0]);  
 HttpURLConnection con = (HttpURLConnection) obj.openConnection();  
 con.setRequestMethod(**"GET"**);  
 con.setConnectTimeout(2000);  
 **int** responseCode = con.getResponseCode();  
 **if** (responseCode == HttpURLConnection.***HTTP\_OK***) {  
 BufferedReader in = **new** BufferedReader(**new** InputStreamReader(  
 con.getInputStream()));  
 String inputLine;  
 StringBuffer response = **new** StringBuffer();  
 **while** ((inputLine = in.readLine()) != **null**) {  
 response.append(inputLine);  
 }  
 in.close();  
 ObjectMapper mapper=**new** ObjectMapper();  
 TypeFactory factory = mapper.getTypeFactory();  
 CollectionType listType = factory.constructCollectionType(List.**class**,Trip.**class**);  
 List<Trip> list = mapper.readValue(response.toString(), listType);  
 **if** (list==**null**) list=**new** ArrayList<Trip>();  
 **return** list;  
 } **else** {  
 **return null**;  
 }  
 }  
 **catch** (Exception e) {  
 **return null**;  
 }  
 }  
 }  
 **public static class** TripPost **extends** AsyncTask<Object,Void,Integer> {  
 **protected** Integer doInBackground (Object... object) {  
 **try** {  
 URL obj = **new** URL(**"http://192.168.1.46:8080/trips/add/"**);  
 HttpURLConnection con = (HttpURLConnection) obj.openConnection();  
 con.setRequestMethod(**"POST"**);  
 con.setRequestProperty(**"Content-Type"**, **"application/json"**);  
 con.setConnectTimeout(2000);  
 con.setDoOutput(**true**);  
 OutputStream os = con.getOutputStream();  
 ObjectMapper mapper=**new** ObjectMapper();  
 os.write(mapper.writeValueAsString(object[0]).getBytes());  
 os.flush();  
 os.close();  
 **int** responseCode = con.getResponseCode();  
 **return** responseCode;  
 }  
 **catch** (Exception e) {  
 **return** -1;  
 }  
 }  
 }  
  
 **public static class** TripGet **extends** AsyncTask<Integer,Void, Object> {  
  
 **protected** Object doInBackground (Integer...integers) {  
 **try** {  
 URL obj = **new** URL(**"http://192.168.1.46:8080/trips/"**+integers[0]);  
 HttpURLConnection con = (HttpURLConnection) obj.openConnection();  
 con.setRequestMethod(**"GET"**);  
 con.setConnectTimeout(2000);  
 **int** responseCode = con.getResponseCode();  
 **switch** (responseCode) {  
 **case** HttpURLConnection.***HTTP\_OK***:  
 BufferedReader in = **new** BufferedReader(**new** InputStreamReader(  
 con.getInputStream()));  
 String inputLine;  
 StringBuffer response = **new** StringBuffer();  
 **while** ((inputLine = in.readLine()) != **null**) {  
 response.append(inputLine);  
 }  
 in.close();  
 ObjectMapper mapper=**new** ObjectMapper();  
 Trip trip = mapper.readValue(response.toString(), Trip.**class**);  
 **return** trip;  
 **case** HttpURLConnection.***HTTP\_NOT\_FOUND***: {  
 **return null**;  
 }  
 **default**: **return** -1;  
 }  
 }  
 **catch** (Exception e) {  
 **return** -1;  
 }  
 }  
 }  
 **public static class** TripDelete **extends** AsyncTask<Integer,Void,Boolean> {  
 **protected** Boolean doInBackground(Integer... integers) {  
 **try** {  
 URL obj = **new** URL(**"http://192.168.1.46:8080/trips/delete/"** + integers[0]);  
 HttpURLConnection con = (HttpURLConnection) obj.openConnection();  
 con.setRequestMethod(**"DELETE"**);  
 **int** responseCode = con.getResponseCode();  
 **if** (responseCode == HttpURLConnection.***HTTP\_OK***) {  
 **return true**;  
 } **else** {  
 **return false**;  
 }  
  
 } **catch** (Exception e) {  
 **return false**;  
 }  
 }  
 }  
 **public static class** TripPut **extends** AsyncTask<Object,Void,Integer> {  
 **protected** Integer doInBackground (Object... object) {  
 **try** {  
 URL obj = **new** URL(**"http://192.168.1.46:8080/trips/edit/"**+object[0].toString());  
 HttpURLConnection con = (HttpURLConnection) obj.openConnection();  
 con.setRequestMethod(**"PUT"**);  
 con.setRequestProperty(**"Content-Type"**, **"application/json"**);  
 con.setConnectTimeout(2000);  
 con.setDoOutput(**true**);  
 OutputStream os = con.getOutputStream();  
 ObjectMapper mapper=**new** ObjectMapper();  
 os.write(mapper.writeValueAsString(object[1]).getBytes());  
 os.flush();  
 os.close();  
 **int** responseCode = con.getResponseCode();  
 **return** responseCode;  
 }  
 **catch** (Exception e) {  
 **return** -1;  
 }  
 }  
 }  
 **public static class** ConnectionPost **extends** AsyncTask<Object,Void,Integer> {  
 **protected** Integer doInBackground (Object... object) {  
 **try** {  
 URL obj = **new** URL(**"http://192.168.1.46:8080/connections/add/"**);  
 HttpURLConnection con = (HttpURLConnection) obj.openConnection();  
 con.setRequestMethod(**"POST"**);  
 con.setRequestProperty(**"Content-Type"**, **"application/json"**);  
 con.setConnectTimeout(2000);  
 con.setDoOutput(**true**);  
 OutputStream os = con.getOutputStream();  
 ObjectMapper mapper=**new** ObjectMapper();  
 os.write(mapper.writeValueAsString(object[0]).getBytes());  
 os.flush();  
 os.close();  
 **int** responseCode = con.getResponseCode();  
 **return** responseCode;  
 }  
 **catch** (Exception e) {  
 **return** -1;  
 }  
 }  
 }  
 **public static class** ConnectionDelete **extends** AsyncTask<Object,Void,Integer> {  
 **protected** Integer doInBackground (Object... object) {  
 **try** {  
 URL obj = **new** URL(**"http://192.168.1.46:8080/connections/delete/"**);  
 HttpURLConnection con = (HttpURLConnection) obj.openConnection();  
 con.setRequestMethod(**"DELETE"**);  
 con.setRequestProperty(**"Content-Type"**, **"application/json"**);  
 con.setConnectTimeout(2000);  
 con.setDoOutput(**true**);  
 OutputStream os = con.getOutputStream();  
 ObjectMapper mapper=**new** ObjectMapper();  
 os.write(mapper.writeValueAsString(object[0]).getBytes());  
 os.flush();  
 os.close();  
 **int** responseCode = con.getResponseCode();  
 **return** responseCode;  
 }  
 **catch** (Exception e) {  
 **return** -1;  
 }  
 }  
 }  
 **public static class** ConnectionHead **extends** AsyncTask<Object,Void,Integer> {  
 **protected** Integer doInBackground (Object... object) {  
 **try** {  
 URL obj = **new** URL(**"http://192.168.1.46:8080/connections/head/"**);  
 HttpURLConnection con = (HttpURLConnection) obj.openConnection();  
 con.setRequestMethod(**"POST"**);  
 con.setRequestProperty(**"Content-Type"**, **"application/json"**);  
 con.setConnectTimeout(2000);  
 con.setDoOutput(**true**);  
 OutputStream os = con.getOutputStream();  
 ObjectMapper mapper=**new** ObjectMapper();  
 os.write(mapper.writeValueAsString(object[0]).getBytes());  
 os.flush();  
 os.close();  
 **int** responseCode = con.getResponseCode();  
 **return** responseCode;  
 }  
 **catch** (Exception e) {  
 **return** -1;  
 }  
 }  
 }  
 **public static class** RatingPost **extends** AsyncTask<Object,Void,Integer> {  
 **protected** Integer doInBackground (Object... object) {  
 **try** {  
 URL obj = **new** URL(**"http://192.168.1.46:8080/ratings/add/"**);  
 HttpURLConnection con = (HttpURLConnection) obj.openConnection();  
 con.setRequestMethod(**"POST"**);  
 con.setRequestProperty(**"Content-Type"**, **"application/json"**);  
 con.setConnectTimeout(2000);  
 con.setDoOutput(**true**);  
 OutputStream os = con.getOutputStream();  
 ObjectMapper mapper=**new** ObjectMapper();  
 os.write(mapper.writeValueAsString(object[0]).getBytes());  
 os.flush();  
 os.close();  
 **int** responseCode = con.getResponseCode();  
 **return** responseCode;  
 }  
 **catch** (Exception e) {  
 **return** -1;  
 }  
 }  
 }  
 **public static class** RatingsGet **extends** AsyncTask<String,Void, Double> {  
 **protected** Double doInBackground (String...strings) {  
 **try** {  
 URL obj = **new** URL(**"http://192.168.1.46:8080/ratings/all/"**+strings[0]);  
 HttpURLConnection con = (HttpURLConnection) obj.openConnection();  
 con.setRequestMethod(**"GET"**);  
 con.setConnectTimeout(2000);  
 **int** responseCode = con.getResponseCode();  
 **if** (responseCode == HttpURLConnection.***HTTP\_OK***) {  
 BufferedReader in = **new** BufferedReader(**new** InputStreamReader(  
 con.getInputStream()));  
 String inputLine;  
 StringBuffer response = **new** StringBuffer();  
 **while** ((inputLine = in.readLine()) != **null**) {  
 response.append(inputLine);  
 }  
 in.close();  
 ObjectMapper mapper=**new** ObjectMapper();  
 Double aDouble = mapper.readValue(response.toString(), Double.**class**);  
 **return** aDouble;  
 } **else** {  
 **return null**;  
 }  
 }  
 **catch** (Exception e) {  
 **return null**;  
 }  
 }  
 }  
}

# Приложение B

**package** com.example.companion;  
  
**import** java.security.NoSuchAlgorithmException;  
  
**import** javax.crypto.Cipher;  
**import** javax.crypto.KeyGenerator;  
**import** javax.crypto.SecretKey;  
**import** javax.crypto.spec.SecretKeySpec;  
  
**public class** AES {  
 **private static** SecretKey *secretKey*;  
 **static** {  
 **try** {  
 **byte**[] keyBytes=**new byte**[]{35,80,71,-30,-119,-94,94,18,88,29,-4,-117,36,36,67,-2,-9,101,-71,108,-32,-46,-54,-5,78,20,59,56,18,-127,80,59};  
 *secretKey*= **new** SecretKeySpec(keyBytes, **"AES"**);  
 }  
 **catch** (Exception e) {  
  
 }  
 }  
 **public static byte**[] encrypt(String message) {  
 **try** {  
 Cipher cipher = Cipher.*getInstance*(**"AES"**);  
 cipher.init(Cipher.***ENCRYPT\_MODE***, *secretKey*);  
 **return** cipher.doFinal(message.getBytes());  
 }  
 **catch** (Exception e) {  
 **return null**;  
 }  
 }  
}