Cardiff University

School of Computer Science & Informatics

Final Year Project

(CM3203 One Semester Individual Project)

Towards Anonymity In Ridesharing Applications By Using Location Obfuscation Of Origin-Destination Points

- May 2017 -

Author: Nikolay Tsonev

Supervisor : Dr. George Theodorakopoulos

Moderator : Dr. Kirill Sidorov

Abstract

The number of the ridesharing and taxi ordering applications on the market is constantly growing. In the busy world we are living these applications help millions of people commuting on daily basis by saving them time and resources. One of the key elements which makes these applications so successful is their effectiveness. They are reducing the steps required for a someone to find a ride or or share his journey with another passengers. However all this comes with a price - the price of sharing your exact location or home/work address pairs with another users of the application.

This project aims to show how a privacy functionality that can be build inside the ride-sharing application can resolve the issue addressed above. In order this to be achieved the actual GPS coordinates of the user will be slightly altered with technique called location obfuscation. The alteration process can be described simply as addition of random noise to the user current location. The artificial location produced by the this technique is then mapped to the closest nearby establishment an it's address will be the one that the other users can see.

The project will also give a general idea how a more complete ridesharing application can be build. Most of this steps/ideas will be in the future work section of the project.

Acknowledgments

I would like to thank my supervisor, George Theodorakopoulos, for all his help and guidance throughout this project and for the opportunity to learn and explore the Android development that he gave me with this project. Up to now this was a big gap in my experience as computer science student and software developer, but now I feel ready and prepared to work on another Android projects in the near future.

I would also like to give my special thanks to my parents and brother for their constant support throughout the years. They are always source of great inspiration for me.

I would like to further thank to everyone who is taking the time to read this work. I would be grateful if this project has motivated you to learn or explore the any of the topics discussed by it.

Table of Contents

1. Introduction

1.1 Outline

We live in time when the traffic in the big cities is growing at a fast pace with every passing day and the average car occupancy keeps going down. As a result more and more ridesharing applications appear on the market in try to free the big cities from their heavy traffic. If they succeed with their mission they will also lower the greenhouse gas emissions caused by the vehicles in these urban areas. This is not only encouraging governments to support similar projects, but also brings money from investors who support environmental causes. Yet the focus very often stays on the big picture and most of the ridesharing applications don't take into consideration some sides of the location privacy and how important this can be for their users.

This project will attempt to address the issue of the location privacy in the ridesharing category of mobile applications. The main idea of this project will be to show how location privacy feature can be embedded into ridesharing android application and what are the technologies required for that. This kind of privacy will be achieved with the help location obfuscation. There are a lot of methods and articles about the topic and some of them will be discussed in this project. Furthermore the current report will describe how the technique for location obfuscation chosen for the prototype developed in this project works and will also mention other possible ways for achieving this kind of privacy.

1.2. Summary of the project aims

One of the aims of this project will be the development of a prototype of ridesharing application and using it as a base application in which a location privacy feature can be implemented. The ridesharing application will include features like authentication, interaction with Google Maps and communication with API running on a server in the cloud. The application will be able to generate a 'ride bookings' - an objects containing information about a request for a ride - including the location of the pick up and the drop off points. The location obfuscation feature will be embedded as additional feature in one of the screens responsible for the 'ride booking' creation. The technique used for the obfuscation will be adding a random noise to the original locations so that an artificial locations can be created within some range from the original ones. We will also consider extending the feature so that it can give couple of choices for 'fake' locations to the user and he will be able to pick the one he finds the most suitable for his needs.

As a second aim several features and extensions that can be added to the ridesharing application prototype will be discussed in the future work section. This will demonstrate how the prototype application can actually be developed into complete and working application capable of providing ridesharing services.

1.3. Project Scope

The project solution will demonstrate one of the well known techniques for location obfuscation and also provide information about others that can be used. The prototype application is developed for the purposes of illustrating the location obfuscation and will have authentication implemented and working. The authentication will work with the Google Firebase platform which will allow us to monitor the users using their console tool. We can also monitor crash reports showing logs from the application when something goes wrong. The application will have working location and map services with screens using Google map fragments. This screens will be part of the booking creation screens and will allow addresses to be searched.

The prototype application will communicate with an API on a remote server in the cloud. For the HTTP calls the Android project will use the Retrofit library and the API side will use Flask, a micro web framework written in Python. The API service will be deployed on EC2 instance in the Amazon Web Services (AWS) cloud. This instance will communicate with Amazon RDS instance (the relational database instances provided from Amazon) in order to save or retrieve the ‘ride booking’ objects coming from the application.

A ride matching algorithm and possible implementation will be discussed in the future work section. Furthermore the future work section will contain some ideas showing how with the help of the Firebase platform and it’s features for cloud messaging and realtime database a chat functionality can be implemented and added to the application.

1.4 Project Outcomes

The project’s main idea and outcome will be the demonstration of a location obfuscation feature and how such feature can be added to a ridesharing Android application. It will show the code needed for the implementation and will discuss another approaches that may be possible. Furthermore the reader will learn how to implement authentication to an Android application with Google Firebase, how to set up an API using Flask and how to create an environment for his project on the cloud. Finally the reader will learn how to build it’s own ridesharing application based on different ideas and features discussed in the future work section.

1.5 Project Structure

The project starts with this introduction explaining the ideas and the motivation behind the work as well as the actual scope of the work. After that it will prepare the reader for the project by explaining all the background information needed for the development and implementation of the application. This background includes: synthesised information from several researches explaining the location privacy problem with the ridesharing applications, location obfuscation techniques and methods, how the location services work on Android devices and finally some information about the technologies and the frameworks which are going to be used in the project.
After the background information the project will go trough the design and specification showing the design of the location obfuscation algorithm, the database schema and the code structure of the API and the android project. After that the report will look at the implementation of the things discussed in the specification part. Detailed code examples will be given in this section.
A set of test cases for each one of the three main parts of the project will be shown in a Testing and evaluation section. The tests will be conducted so that we can evaluate the code and the prototype and show some results. Finally ideas for the project further development will be given with the clear purpose to demonstrate to the reader some of the things needed for a complete ridesharing application.

2. Background

2.1. Overview

The growth of the mobile users (which are now around 2 billion) have increased the number and the variety of the mobile applications that are being developed. Most of them have their motivation behind the idea to improve parts of our daily life in some ways. Commuting or getting around in a city is something that takes a big part of our time these days and it was expected sooner or later solution for making this part of our life easier to appear. We can even notice the growing traffic of vehicles everywhere. The roads are getting busier and busier. One of the reasons for this is that most of us don’t utilise our assets at their maximum. People very often travel alone in their vehicles. What’s more, a several studies show that the average car occupancy is around 1.5 and tends to fall at 1.2 for people travelling for business. The key for resolving this issue can be found in the sharing economy. This is an economic model in which individuals are able to rent or borrow not utilised assets owned by someone else. In the context of the transportation the non utilised resources are the free/empty seats in the

vehicles. That's why we often categorise the ridesharing applications as part of the sharing economy. The idea for them appeared around 2008 and we started witnessing the first successful ridesharing applications in around 2012. From the day they appeared their user database have been growing exponentially.

These applications take advantage of the location of the driver and the user and their Origin-Destination points of travel. This localisation takes advantage of the Global Positioning System (GPS) which makes use of satellites to determine the user position. Almost every smartphone these days has a GPS chipset. Most of the devices that don't have one use an alternative support technologies for location based services based on cell tower data to get the user location. Often the location gathered from the ridesharing applications is not obfuscated and can reveal the actual position of the users or the places where they live or work in.

2.2. The location privacy concerns that affect the ridesharing applications?

The rideshring services rely very heavily on location data, these apps track where the user is and where he goes. Even further - some of the apps try to expand their location data collection so that they can improve their pick-ups and drop-offs by learning where the user lives and where he works. These pairs of locations are very sensitive data for the user and are serious privacy concern. What's interesting is that they are very often shared with other users of the application so that a match between the users can be created. To put it another way - this information is public within the application users.

The privacy issue arrises when the home/workplace pair (A, B) can be joined with some other data of any public or private dataset containing users and the same kind of home/ workplace map pair. This can potentially identify the user depending on what additional information the second source provides. There are several levels of identification that may appear. For example: The worst case will be an exact match - a record showing a lot of private information and a specific individual. Another case will be a set of users which have the same (A, B) pair. The bigger this set is the better chance we have that the actual user identification is not revealed. Furthermore this location pair (A, B) can be used with completely different anonymised location data set resulting in re-identification on some of the data.

The discussed privacy concerns can be prevented via location obfuscation techniques. And a research shows that the bigger this obfuscation is the better privacy of the original location it creates.

2.3. What is location obfuscation and what are the different techniques of using it?

The concern about privacy in terms of the user's location has led to the discovery and the development of various kinds of location obfuscation techniques. These techniques are used for the creation of non reversible alteration of the user original location in ways that the user position is obscured but still not completely different in terms of information that can be acquired from it. These allows whoever uses the location to offer the service that needed the location in the first place and have similar or the same output as if the original location was used.

Not all of the location obfuscation techniques can be used for hiding a single point. Some of them are effective when we have list of points. In our project we are concerned only about single points - the two points of the the user home/workplace pair (A, B) and we will look only at the most famous and relevant techniques for this scenario.

The most obvious technique for hiding user sensitive geo data is a method which doesn't return any data within a certain distance from given sensitive point. This technique is called "invisible cloaking". Another method used for location obfuscation is "randomising". It works by random noise being added to the original position. The random function can follow different distributions. The most common distributions for location obfuscation are the uniform one and the normal (Gaussian) one. The third most famous obfuscation technique which can be used by our project is the "discretising" method. It uses a rounding technique that lowers the accuracy of the original point.

2.4. Location based services in Android

The location based services in Android work thanks to the Location API which is build in inside the Android operating system. The Location API on other hand talks to system libraries which communicate with the hardware (the GPS chipset for example). Newer versions of Android have the Location API build in the Google Play Services which combines different APIs and aims to make the application interaction with operating system APIs easier.

The Location API have two permissions which can be used for different location accuracy level depending on the requirements. One is ACCESS_COARSE_LOCATION and the other is ACCESS_FINE_LOCATION.

2.5. Software technologies and frameworks used for the project prototype.

The prototype developed in this project will be build for Android API level 23 (Android 6.0). One of the big changes for this version of the Android SDK is that the permissions can be requested during runtime and you can still have a working application with some of the permissions denied although this may result in some not working functionalities of the application.

The application will have authentication with the Google Firebase platform. The Firebase platform provides different tools and APIs to developers which can make the development of specific features more easier than usual. It works as a Back-End as a Service and provides tools like Analytics, Cloud Messaging, Authentication, Realtime Database, plugins for adverts and many more.

The API will be written in Python and will be run on Amazon web services (AWS) instance. AWS is on-demand cloud computing platform.

3. Specification and Design

3.1. Overview

The project discussed in this report aims to present an android application which has a location obfuscation feature. This feature will use and demonstrate the ”randomising” location obfuscation algorithm. The discussed application will imitate a ridesharing application with limited functionalities called “Zaedno”. The application will make use of Google Firebase for authentication. This will control which screens the user can access and will provide information of who is using the application at that moment. Google maps and Google’s location services are used for searching location points on the map and using these locations in the “ride booking” module of the application.

Furthermore there will be an API which will take the “ride booking” objects coming from the instances of the application and will store them in a table inside a database. Both the API and the database will run on servers in the AWS cloud.

The “randomising” technique for the location obfuscation will use a randomly generated values from random function using the uniform distribution.

3.2. Specifications and deviations from the initial plan

The specifications presented in the initial plan proposed the development of a complete ridesharing application which was going to use a location privacy feature. However, after weeks of research in the area of location privacy I noticed that most of the major ridesharing applications doesn't consider the origin-destination points as something they should keep obfuscated and I decided to address this issue in the project and show how a privacy feature addressing this problem can be integrated in a project rather than creating a complete ridesharing application.

The initial plan had the following aims:

- Design simple and aesthetic user interface. (**must** have)

- Develop a system which allows user to create a journey by picking up the initial and the destination points and uploading the journey to a system which distribute it to other users having common origin-destination points in order two similar routes to be matched. (**must** have)

- Add a private feature which allows the user to disguise the pick-up and the destination points by randomly selecting an artificial location within a given range from the original location. (**must** have)

- Add a social network capabilities to the application by allowing each user to add people they like travelling with as "friends". Create a feature allowing private communication between the users. (**should** have)

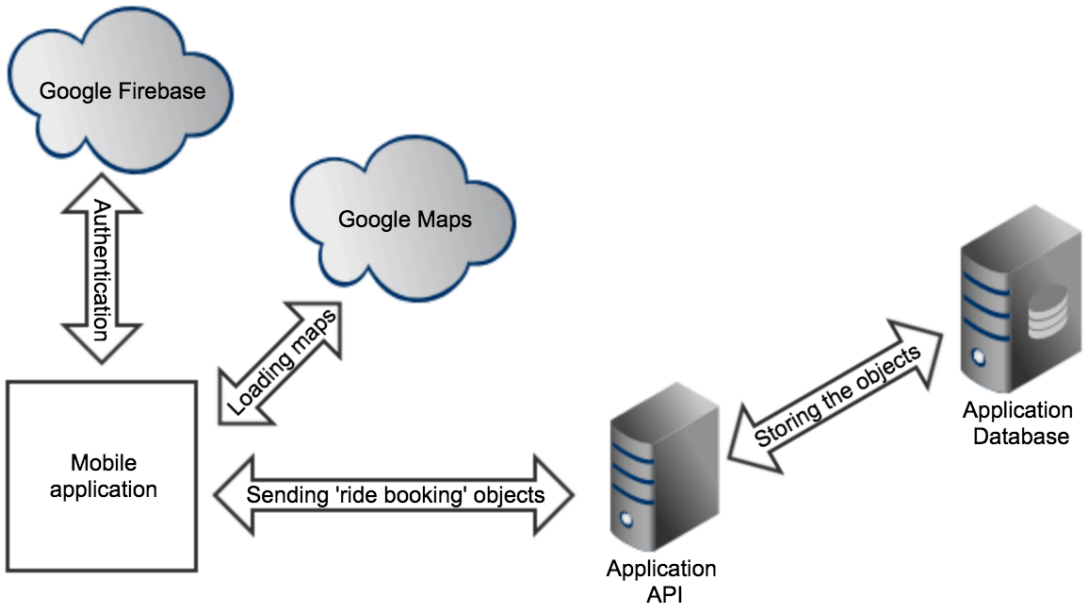
- Add a feature which allows the application to recognise when two users are together. Can be created by QR codes which are scanned by the users. (**could** have)

All **must** have points are required for the project and will be enough for creating a prototype and implementing the location obfuscation feature. The **should** and the **could** have points were desirable if there was enough time and were more specific for the development of a complete ridesharing application.

Considering that the focus of the project changed to a more narrow and specific location privacy issue - the **should** and the **could** have aims are no longer a priority for this project. However I decided to use the authentication feature from the *"Add a social network capabilities to the application"* point which is going to useful when a "ride booking" is submitted from the prototype. The authentication will contain some kind of user identification which will help us to uniquely identify the records stored in our database.

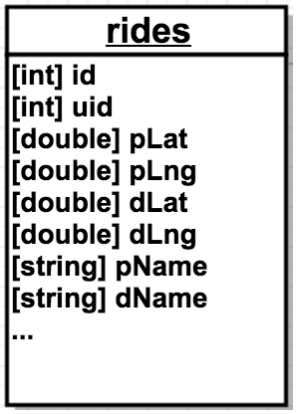
3.3. Application logic design

The application logic design includes 5 main components. An android prototype application, an API, a database, a Google Firebase service and a Google Maps service. The following diagram can illustrate the data flow between the different components.



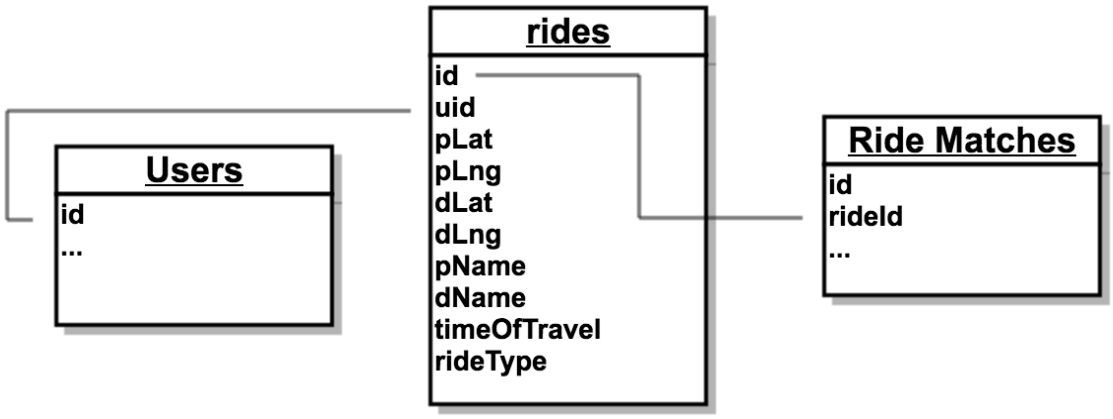
3.4. Database schema

For the purposes of this prototype we will only need one table in the database which will store the “ride booking” objects. (the table schema shown below)



The id will be the unique identifier for the “ride object”. The uid will be the unique identification for the user which made the booking. The pLat, pLng and pName will be the location coordinates and name of the pick up point and the dLat, dLng and dName will be the location coordinates and name for the drop off point.

In a case when more complex prototype is build, for example an application having more functionalities which a ride sharing application may usually have, there can be a lot more tables supporting the additional application functionalities. For example a ‘users’ table can store more information about the users. Similarly there can be ‘ride matches’ table having more information about the status of the ‘ride bookings’. Here is an example how the current basic schema can be extended in this case:



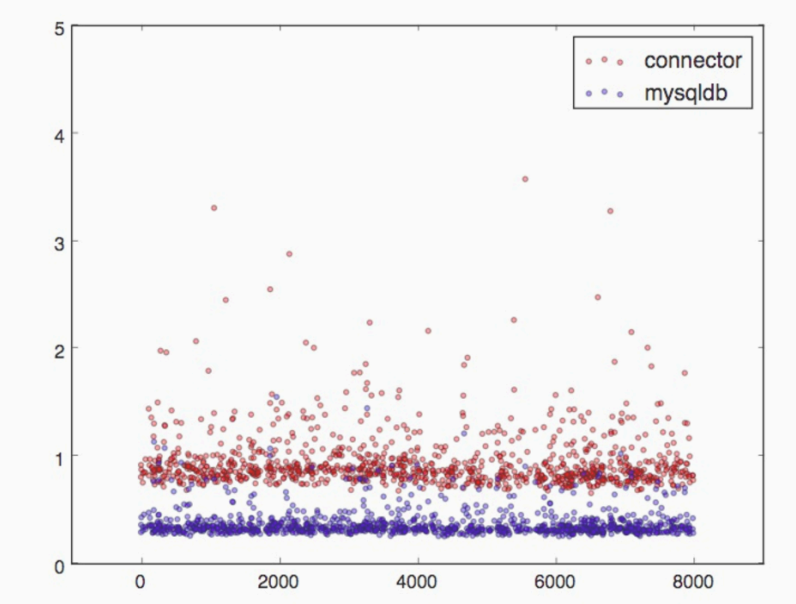
This example is out of the scope of the current prototype and project but can be useful when we talk about the future work.

3.5. API design

The application programming interface (API) will be written in Flask and since the prototype functionalities will only include storage there will be only one route '/ride/new' which will expect POST HTTP requests containing a new 'ride booking' objects coming from an instances of the android application application. The API code will have method called executeQuery() which implement the database connection and the sql query execution. The method will take an sql query string and will return a status code indicating if the record was inserted into the table.

Additionally there will be also a separate file containing the config files needed for the database connection. This needs to be separated because when we upload the API code to a repository we would like to ignore this sensitive information in the .gitignore file.

The API will require a library for MySQL and there are a lot of good libraries for this. Some of them even originating one from another. The two most used and distinguishable ones are MySQLdb which is C based and mysql-connector which is written by Oracle and is python based. After finding a study which compares the performance of the both libraries I picked the MySQLdb as a library for this project. It's much more faster than it's competitor. The following comparison diagram created by the author of the study illustrates how big the difference between the two libraries is:



3.6. Android application design

3.6.1. UI

The application prototype will require several screens so that the information can be communicated and retrieved from the user and the location obfuscation functionality can be shown. These screens will be designed as sketches initially with a sketching application called Sketch. A common user interface methodologies found in several of the top ridesharing applications on the market will be used as ideas for the prototype user interface. The views required for the prototype are:

- **Initial screen** having the logo and the name of the application and two buttons. One used to open the login screen and one used to open the register screen.

- There will be a **login and register screens** having similar buttons and fields.

- A **main screen** for the authenticated user will be created. This screen will have a bottom navigation with 5 buttons (profile, my rides, new ride, chat, friends). This buttons will load different content inside this main screen. Additionally there will be a header having the application name.

- **Several fragments** will be created. Each one having different functionality and being able to be loaded in the main screen. Since our prototype will have limited functionalities there will be some empty fragments. There will be several fragments starting their workflow from the ‘new ride’ fragment. The first fragment which loads when the ‘new ride’ button is clicked will be a view with another button leading to the fragments needed for creating a “ride booking”, a button which redirects the user to his rides - the equivalent of clicking ‘my rides’ from the menu. And there will be a card which can show the next upcoming ride. This will be example of how the application can be extended in the future. When the “new ride booking” button is clicked it will load a fragment with Google maps in it and a search field. This view will be needed for the user to select his pick up point. There will be a button leading to a similar screen for selecting the drop off point. And finally there will be screen for selecting additional data about the ride including time, date and a check button for the privacy feature.

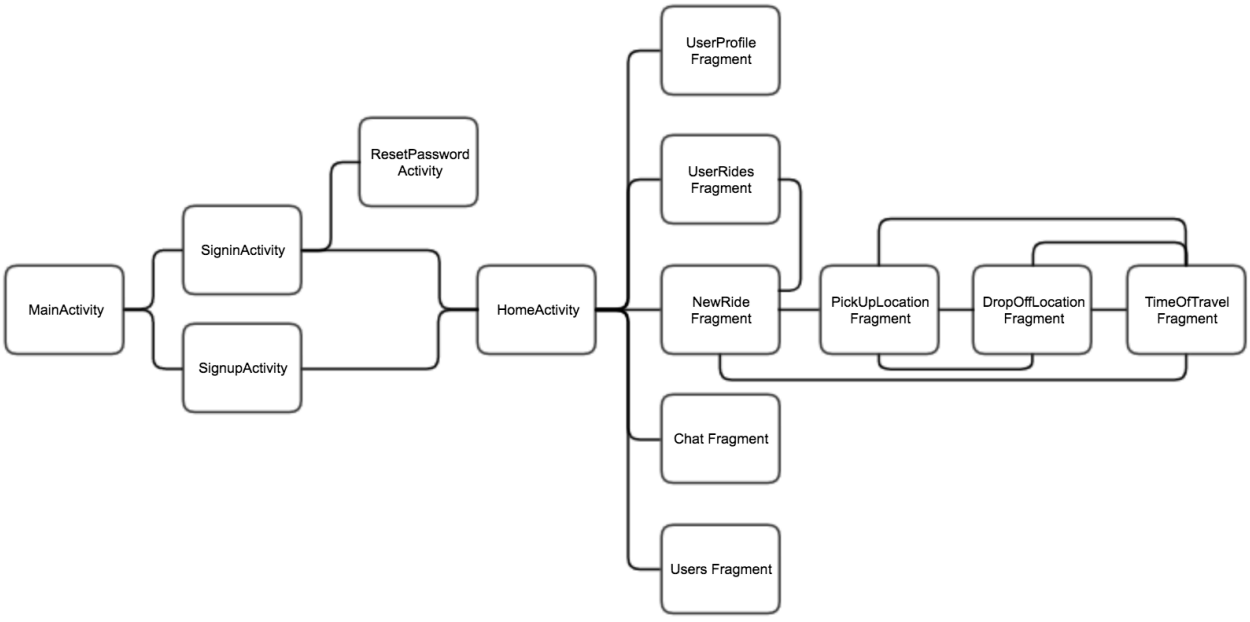
All the sketches will be then written in XML which is the android layout format. This method used by Android will enable us to add functionality behind the views but at the same time the XMLs will separate the user interface of the application from the code that controls the behaviour of the application.

3.6.2. Classes

The prototype application will have several activities and fragments. Each one of them is represented by a class in Java. The activity which loads initially will be called ‘MainActivity’ and it will have two button listeners for the two buttons leading to the ‘signinActivity’ and the ‘signupActivity’. The ‘signinActivity’ and the ‘signupActivity’ will have instance of the FirebaseAuth which will be added in the project via gradle (a build system).

After successful sign in or sign up the user will be redirected to the ‘HomeActivity’. This activity will also have an instance of the FirebaseAuth so that we can keep track who the authenticated user is and redirect it back to the ‘MainActivity’ in case his authentication token is no longer valid. The ‘HomeActivity’ will have the menu and all the button listeners connected with it and will be able to change its content with the different fragments. Several fragment classes will be able to load inside this ‘HomeActivity’ and some of them will have a fragments inside them as well. For example the ‘PickUpLocationFragment’ and the ‘DropOffLocationFrament’ will have a mapFragments as part of them.

The following diagram can show the workflow of the activities and fragments controlling the user interface:



The Retrofit library which is responsible for the HTTP requests to the API will require several classes and one interface in order to work. The interface should represent all the possible API calls together with the object types submitted and the responses objects. For

instance the new ride request will need a class called ‘Ride’ which will represent the model of the new ‘ride booking’ objects. Instance of this class will be converted to JSON via the gson and the Retrofit library and send to the API in case a new ‘ride booking’ is submitted. A ‘NewRideResponse’ class will be also required. This response object will have a message inside it saying if the send ride object was successfully inserted into the database.

3.6.3. Location obfuscation algorithm

The location obfuscation algorithm will be implemented via two methods inside the ‘timeOfTravelFragment’ class. The first method will add a random shift values generated by the second method to each one of the geographic coordinates. The core method of the algorithm will be the one generating the shifted values. It will have the following structure:

```
generateShift(){

        double minShift // the minimum value that we want to be generated
        double maxShift  // the maximum value that we want to be generated
        double shift = minShift + (maxShift - minShift) * randomValue
        // randomValue will be a random number between 0 and 1 selected from the
        // uniform distribution

        // a random boolean will decide whether the method should return negative shift or positive
        // one
        if(randomBoolean){
                return -shift;
        }
         return shift;
}
```

4. Implementation

4.1. Hardware and software used for the project

One of the most important software tools used for the implementation of this project was Android Studio. It is the official integrated development environment used for the development of Android projects. The Android prototype application used a several external frameworks and libraries which were installed and build with **gradle** - the build tool used by Android Studio. These external frameworks are:

- **Firebase Auth** - The external service which we are using for authentication of the users.
- **OkHttp 3** - An HTTP client used in Android for HTTP requests.
- **Retrofit** - An API mapping library (adapter) which uses **OkHttp** to send and receive HTTP requests.
- **gson -** Java library used for conversion of Java objects into their JSON representation.
- **converter-gson** - This library is used together with retrofit when JSON objects converted with **gson** are send or received as HTTP requests.
- **Google play services** - a package which combines several of the google APIs including Location APIs and Google Maps APIs which we are using in this project.

The API also used several additional libraries as dependencies. All of them were inside the requirements.txt file which was installed with the following command on the environment where the API was set up:

```
pip install -r requirements.txt
```

The file had the following dependencies listed inside:

- Flask
- mysqlclient (MySQLdb)

Additionally the project used the following tools and softwares for testing, project management and development:

- Text Editor (Atom)
- Sketching application (Sketch)
- Google Firebase Console
- AWS management console
- SQL client (Sequel Pro)
- FTP/SFTP client (CyberDuck)
- API analysation tool (PostMan)
- Git repository (Bitbucket)
- Task tracking and management software (Trello)

The Android application was tested and deployed on personal Android device having Android version 6.0.1. Additionally the android emulator (emulating Nexus 5X API 23) was used for testing during the development.

4.2. Application code

There are several important parts of the application prototype. This part will try to cover most of them starting with two very important files for the project - the manifest XML file and the gradle script. After that the report will mention the implementation of things like authentication, button clicks, transition between activities and fragments, Google maps search, getting the user location, etc.

The manifest XML file which is one of the most important files of the Android projects. The application developed for this project has the following manifest file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.zaedno">

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

    <!--
         The ACCESS_COARSE/FINE_LOCATION permissions are not required to use
         Google Maps Android API v2, but you must specify either coarse or fine
         location permissions for the 'MyLocation' functionality.
    -->
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

```
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name="com.example.zaedno.MainActivity"
            android:label="@string/title_activity_main"
            android:theme="@style/AppTheme.Fullscreen">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="com.example.zaedno.SignupActivity"
            android:label="@string/title_activity_signup"
            android:parentActivityName="com.example.zaedno.MainActivity"
            android:theme="@style/AppTheme.Fullscreen" />
        <activity
            android:name="com.example.zaedno.ResetPasswordActivity"
            android:label="ResetPassword"
            android:parentActivityName="com.example.zaedno.SigninActivity"
            android:theme="@style/AppTheme.Fullscreen" />
        <activity
            android:name="com.example.zaedno.SigninActivity"
            android:label="@string/title_activity_signin"
            android:parentActivityName="com.example.zaedno.MainActivity"
            android:theme="@style/AppTheme.Fullscreen" />
        <activity android:name="com.example.zaedno.HomeActivity" />
        <!--
             The API key for Google Maps-based APIs is defined as a string resource.
             (See the file "res/values/google_maps_api.xml").
             Note that the API key is linked to the encryption key used to sign the
APK.
             You need a different API key for each encryption key, including the
release key that is used to
             sign the APK for publishing.
             You can define the keys for the debug and release targets in src/debug/
and src/release/.
        -->
        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="@string/google_maps_key" />

        <!--
 ATTENTION: This was auto-generated to add Google Play services to your project for
     App Indexing.  See https://g.co/AppIndexing/AndroidStudio for more information.
        -->
        <meta-data
            android:name="com.google.android.gms.version"
            android:value="@integer/google_play_services_version" />

    </application>

</manifest>
```

This file specifies the permission which the application will use. In the case of “Zaedno” application the Internet permission is used. The application needs the internet permission to access the google play services for location and Google Maps and for accessing the python API. The “ACCESS_FINE_LOCATION” and “ACCESS_COARSE_LOCATION” for accessing the location of the user are also required permissions for this project.

The activities of the application can be found between the ‘application’ tags in the manifest file. The MainActivity is marked as initial (main) activity and opens when the application is started. The SigninActivity and SignUpActivity are the activities needed for the user authentication and the Activity which controls the main functionalities of this project is the HomeActivity.

The second file of big importance for the android projects is the gradle script. It contains information about the build version of the android sdk and the minimum supported SDK version. This file also contains all the project dependencies inside it. We can see the firebase dependency needed for the authentication, the play services dependency needed for the location and google maps, the support design dependency providing some essential design components, the okHttp3 and the retrofit dependency so that we can send and receive HTTP requests.

(the file is shown below)

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 25
    buildToolsVersion "25.0.2"
    defaultConfig {
        applicationId "com.example.leaf"
        minSdkVersion 15
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
}
```

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:25.3.0'
    compile 'com.android.support:cardview-v7:25.3.0'
    compile 'com.android.support:recyclerview-v7:25.3.0'
    compile 'com.android.support:design:25.3.0'
    compile 'com.google.firebase:firebase-auth:10.2.1'
    compile 'com.google.android.gms:play-services-auth:10.2.1'
    compile 'com.google.android.gms:play-services:10.2.1'
    compile 'com.android.support:support-v4:25.3.0'
    compile 'com.squareup.retrofit2:retrofit:2.1.0'
    compile 'com.google.code.gson:gson:2.6.2'
    compile 'com.squareup.retrofit2:converter-gson:2.1.0'
    compile 'com.squareup.okhttp3:okhttp:3.0.1'
    testCompile 'junit:junit:4.12'
}
apply plugin: 'com.google.gms.google-services'
```

One of the first features added to the project was the authentication. As planned it was made with the Google Firebase plugin. One of the most important code which makes the authentication to work is the authentication listener which is listening for authentication changes and if one occurs it redirects the user to the relevant screens. The authentication listener looks like this:

```
mAuthListener = new FirebaseAuth.AuthStateListener() {
    @Override
    public void onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth) {
        FirebaseUser user = firebaseAuth.getCurrentUser();
        if (user != null) {
            Intent intent = new Intent(SigninActivity.this, HomeActivity.class);
            startActivity(intent);
            finish();
        }
    }
};
```

The code snippet above shows a listener which waits for authentication state change. If the new state after the change has a user object it redirects the user to the HomeActivity. This snippet is also a good example of how the intents are made and the redirection works. The

code which handles the logout is similar only the check in the if is different and the intent is reversed:

```
mAuthListener = new FirebaseAuth.AuthStateListener() {
    @Override
    public void onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth) {
        FirebaseUser user = firebaseAuth.getCurrentUser();
        if (user == null) {
            Intent intent = new Intent(HomeActivity.this, MainActivity.class);
            startActivity(intent);
            finish();
        }
    }
};
```

There are parts of the application which we don't want to be accessed from unauthorised users and parts of the application which are in no need to authenticated users. Thats why there is a special logic which restricts the user to some screens depending on his authentication status:

```
//Get Firebase auth instance
auth = FirebaseAuth.getInstance();
FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
if (user == null) {
    Intent intent = new Intent(HomeActivity.this, MainActivity.class);
    startActivity(intent);
    finish();
}
```

The code above checks if the user is authenticated and if he is not - the application will redirect him to the MainActivity from where the sign in and sign up functionalities can be accessed. There is similar logic inside the MainActivity which checks if the user is authenticated and redirects him to the HomeScreen if he is authenticated because.

The next thing which was of big importance for the project is how the activities and the fragments handle the button clicks. There are a lot of buttons around the application and in order to register a click they need a special onClicklListeners. These onClickListeners allow the application to make an appropriate action when a button is clicked by the user.

The code snippet bellow is taken from the UserProfileFragment showing how the logout button works:

```
btn_logout = (Button) view.findViewById(R.id.logout);

btn_logout.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        ((HomeActivity)getActivity()).signOut();
    }
});
```

With the authentication handling implemented, the intents, the button onClickListeners and a little bit of additional knowledge the MainActivity, SigninActivity and the SignupActivity were created.

The next important part of the prototype was the bottom navigation. For the implementation of this navigation the Bottom navigation component provided from the support design library was used. It requires a separate XML file describing the items of the menu.

```
Type to enter text<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item android:id="@+id/user"
        android:title="Profile"
        android:icon="@drawable/user"
        app:showAsAction="always"/>
    <item android:id="@+id/pastRides"
        android:title="My Rides"
        android:icon="@drawable/history"
        app:showAsAction="always"/>
    <item android:id="@+id/newRide"
        android:title="New Ride"
        android:icon="@drawable/new_ride"
        app:showAsAction="always"/>
    <item android:id="@+id/chat"
        android:title="Chat"
        android:icon="@drawable/chat"
        app:showAsAction="always"/>
    <item android:id="@+id/friends"
        android:title="Friends"
        android:icon="@drawable/users"
        app:showAsAction="always"/>
</menu>
```

In the prototype application this file can be found in the \res\menu folder and it's called bottom_navidation.xml

Additionally to work the bottom navigation requires this component to be placed inside the home activity.

```
<android.support.design.widget.BottomNavigationView
    android:id="@+id/bottom_navigation"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    app:itemBackground="@color/blue_light"
    app:itemIconTint="@color/white"
    app:itemTextColor="@color/white"
    app:menu="@menu/bottom_navigation" />
```

The bottom menu updates and fragment changes are controlled by the following code in the HomeActivity.java

```
// bottom navigation
BottomNavigationView bottomNavigationView = (BottomNavigationView)
        findViewById(R.id.bottom_navigation);

bottomNavigationView.setOnNavigationItemSelectedListener
        (new BottomNavigationView.OnNavigationItemSelectedListener() {
            @Override
            public boolean onNavigationItemSelected(@NonNull MenuItem item) {
                Fragment selectedFragment = null;
                switch (item.getItemId()) {
                    case R.id.user:
                        selectedFragment = UserProfileFragment.newInstance();
                        break;
                    case R.id.pastRides:
                        selectedFragment = UserRidesFragment.newInstance();
                        break;
                    case R.id.newRide:
                        selectedFragment = NewRideFragment.newInstance();
                        break;
                    case R.id.chat:
                        selectedFragment = ChatFragment.newInstance();
                        break;
                    case R.id.friends:
                        selectedFragment = FriendsFragment.newInstance();
                        break;
                }
                FragmentTransaction transaction =
getSupportFragmentManager().beginTransaction();
                transaction.replace(R.id.fragment_space, selectedFragment);
                transaction.commit();
                return true;
            }
        });
```

```
FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();
transaction.replace(R.id.fragment_space, NewRideFragment.newInstance());
transaction.commit();
```

The next important part of the prototype were the pickUpLocationFragment and dropOffLocationFragment. They are part of the “ride booking” object creation workflow and help the user to select his addresses of interest. The fragments have several methods providing functionalities like: getting the user current location and showing it on the map, converting location points to address string (street, number of building, etc) by making calls to the Google Maps API, updating the location points and the address string when the map is moved under the pin and also allowing the user to search location by entering address in a search filed. There is also a method which asks for location permission for Android devices running SDK 23 and above. The method looks like that:

```
@Override
public void onRequestPermissionsResult(int requestCode,
                                       String permissions[], int[] grantResults) {
    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_ACCESS_COARSE_LOCATION:
        case MY_PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION: {
            // If request is cancelled, the result arrays are empty.
            if (grantResults.length > 0
                    && grantResults[0] == PackageManager.PERMISSION_GRANTED) {

                setCurrentLocation();

            } else {

                Log.i("PErr", "Permission denied location");
            }
            return;
        }

    }
}
```

Once the permission is acquired the setCurrentLocation function is called. This function updates the location variables and loads the current location on the map fragment. (the function is shown below)

```
private void setCurrentLocation() {
    if (ActivityCompat.checkSelfPermission(getActivity(),
android.Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions(getActivity(),
                new String[]{android.Manifest.permission.ACCESS_COARSE_LOCATION},
                MY_PERMISSIONS_REQUEST_ACCESS_COARSE_LOCATION);
        return;
    }
    if (ActivityCompat.checkSelfPermission(getActivity(),
android.Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {

        ActivityCompat.requestPermissions(getActivity(),
                new String[]{android.Manifest.permission.ACCESS_FINE_LOCATION},
                MY_PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION);
        return;
    }
    mLastLocation =
LocationServices.FusedLocationApi.getLastLocation(mGoogleApiClient);
    if (mLastLocation != null) {

        pickupLoc = new LatLng(mLastLocation.getLatitude(),
mLastLocation.getLongitude());
        CameraUpdate center =
                CameraUpdateFactory.newLatLng(pickupLoc);
        CameraUpdate zoom = CameraUpdateFactory.zoomTo(16);

        mMap.moveCamera(center);
        mMap.animateCamera(zoom);
    }

    mLocationRequest = new LocationRequest();
    mLocationRequest.setInterval(5000); //5 seconds
    mLocationRequest.setFastestInterval(3000); //3 seconds

mLocationRequest.setPriority(LocationRequest.PRIORITY_BALANCED_POWER_ACCURACY);

    LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient,
mLocationRequest, this);
}
```

The following method is used to get latitude and longitude points from address entered in the search field:

```
private void getLocationByAddress(EditText pickup_input) {
    Geocoder geoCoder = new Geocoder(getActivity(), Locale.getDefault());
    try {
        List<Address> addresses =
geoCoder.getFromLocationName(pickup_input.getText().toString(), 1);
        if (addresses.size() > 0) {
            Double lat = (double) (addresses.get(0).getLatitude());
            Double lon = (double) (addresses.get(0).getLongitude());

            pickupLoc = new LatLng(lat, lon);

            updatePickupSearchField();

            CameraUpdate center =
                    CameraUpdateFactory.newLatLng(pickupLoc);
            CameraUpdate zoom = CameraUpdateFactory.zoomTo(16);

            mMap.moveCamera(center);
            mMap.animateCamera(zoom);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

The process performed by this method is called geocoding. The reversed geocoding is not as accurate as the actual geocoding. It may return partial data. For reverse geocoding we use the following function:

```
private void updatePickupSearchField() {
    try {

        Geocoder geo = new Geocoder(getActivity().getApplicationContext(),
Locale.getDefault());
        List<Address> addresses = geo.getFromLocation(pickupLoc.latitude,
pickupLoc.longitude, 1);
        if (addresses.isEmpty()) {
        } else {
            if (addresses.size() > 0) {
                pickup_input.setText("");
                pickup_input.setHint(addresses.get(0).getFeatureName() + ", " +
addresses.get(0).getThoroughfare() + ", " + addresses.get(0).getLocality() + ", "
+ addresses.get(0).getCountryCode());
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

This method of reverse geocoding is needed when the user moves the map around. In that way the application keeps track of the address name of the new location points under the location pin.

The next fragment of interest is the TimeOfTravelFragment. This is the last step of the “ride booking” creation workflow and it has several fields for selecting the date of the travel, the time of the travel and the ride type. The methods taking care of the location obfuscation and the code which sends the “ride booking” object to the database can be also found in this fragment.

There are also two functions assisting the date/time selection fields:

```
private String monthToString(int month){
    switch (month) {
        case 1:  return "January";
        case 2:  return "February";
        case 3:  return "March";
        case 4:  return "April";
        case 5:  return "May";
        case 6:  return "June";
        case 7:  return "July";
        case 8:  return "August";
        case 9:  return "September";
        case 10: return "October";
        case 11: return "November";
        case 12: return "December";
        default: return "Invalid month";
    }
}
```

The code snippet above shows a simple method using switch to convert month number to a month name. This is needed for better representation of the month in the UI.

The next function used for the date/time selection fields is the method button which opens the time picker dialog:

```
btnChangeTime.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        TimePickerDialog timePickerDialog = new TimePickerDialog(getActivity(),
                new TimePickerDialog.OnTimeSetListener() {

                    @Override
                    public void onTimeSet(TimePicker view, int hourOfDay,
                                          int minute) {

                        time.setText(timeToString(hourOfDay, minute));
                        mHour = hourOfDay;
                        mMinute = minute;
                    }
                }, mHour, mMinute, true);

        timePickerDialog.show();
    }
});
```

Probably one of the most important functions part of this fragment is the function which sends the “ride booking” object to the API. The first thing that happens in this is the initialisation on an object of class Ride with all the information collected from the forms. This is how the class ride looks like:

```
package com.example.zaedno;

public class Ride {
    final String uid;
    final Double dLat;
    final Double dLng;
    final Double pLat;
    final Double pLng;
    final String dName;
    final String pName;
    final long time;
    final String rideType;

    Ride(String uid, String dName, String pName, Double dropOffLng_o, Double dropOffLat_o,
Double pickUpLat_o, Double pickUpLng_o, long time, String rideType){
        this.uid = uid;
        this.dName = dName;
        this.pName = pName;
        this.dLat = dropOffLng_o;
        this.dLng = dropOffLat_o;
        this.pLat = pickUpLat_o;
        this.pLng = pickUpLng_o;
        this.time = time;
        this.rideType = rideType;
    }

}
```

This Ride class is needed from gson library which converts it from a Java class to a JSON object which can be send over HTTP request with retrofit.

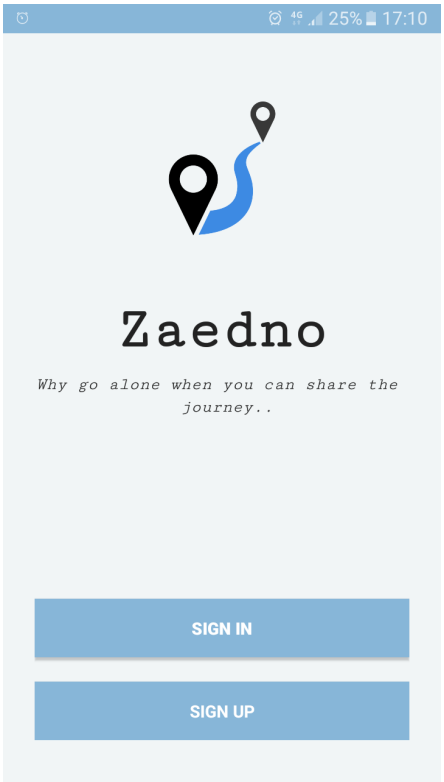
```
btnSubmitRide.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Calendar timeOfTravel = Calendar.getInstance();
        if(btnTomorrow.isChecked()){
            timeOfTravel.add(Calendar.DATE, 1);
        }
        Calendar timeData = Calendar.getInstance();
        timeData.set(timeOfTravel.get(Calendar.YEAR), timeOfTravel.get(Calendar.MONTH),
timeOfTravel.get(Calendar.DAY_OF_MONTH), mHour, mMinute);

        Ride newRide;
        if(privacy.isChecked()){
            newRide = new Ride(((HomeActivity)getActivity()).getUserId(),
dropOffName.getText().toString(), pickUpName.getText().toString(), mDlat_f, mDlng_f, mPlat_f,
mPlng_f, timeData.getTime().getTime(), rideType.getText().toString());
        }
        else{
            newRide = new Ride(((HomeActivity)getActivity()).getUserId(),
dropOffName.getText().toString(), pickUpName.getText().toString(), mDlat, mDlng, mPlat, mPlng,
timeData.getTime().getTime(), rideType.getText().toString());
        }

        Retrofit retrofit = new Retrofit.Builder()
                .baseUrl("http://api_address_here/")
                .client(new OkHttpClient())
                .addConverterFactory(GsonConverterFactory.create())
                .build();

        ZaednoAPI api = retrofit.create(ZaednoAPI.class);

        Call<NewRideResponse> upload = api.newRide(newRide);

        upload.enqueue(new Callback<NewRideResponse>() {
            @Override
            public void onResponse(Call<NewRideResponse> call, Response<NewRideResponse>
response) {
                NewRideResponse res = response.body();
                if(res.getMessage().equals("success")){
                    FragmentTransaction transaction = getFragmentManager().beginTransaction();
                    transaction.replace(R.id.fragment_space, NewRideFragment.newInstance());
                    transaction.commit();
                }
                else{
                    Toast.makeText(getActivity(), "An error occurred. Try again.",
Toast.LENGTH_SHORT).show();
                }
            }

            @Override
            public void onFailure(Call<NewRideResponse> call, Throwable t) {
                Toast.makeText(getActivity(), "An error occurred. Try again.",
Toast.LENGTH_SHORT).show();
            }
        });
    }
});
```

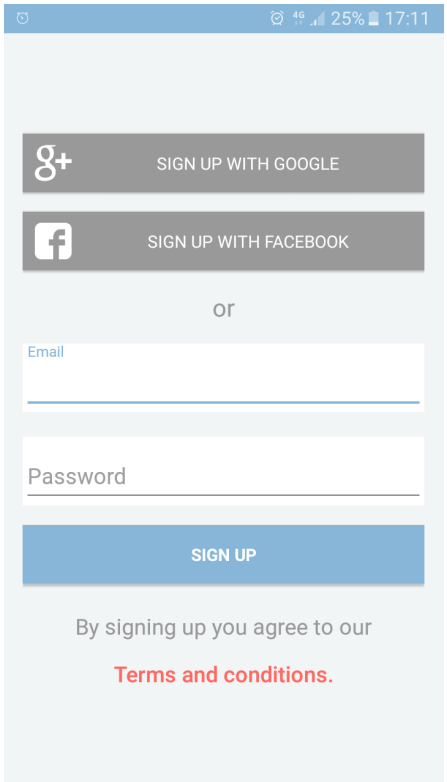
Once when the Ride object is created and has the new ride data a retrofit instance with the API address is created, the http client (OkHTTP) and the converter library (gson). After that the retrofit class generates an implementation of the API interface. Thanks to this interface we can make calls to the actual API. It's important to note that the calls need to be asynchronous because if made from the UI thread an exception will arise. This is buld in from android and ensures that the UI will stay responsive.

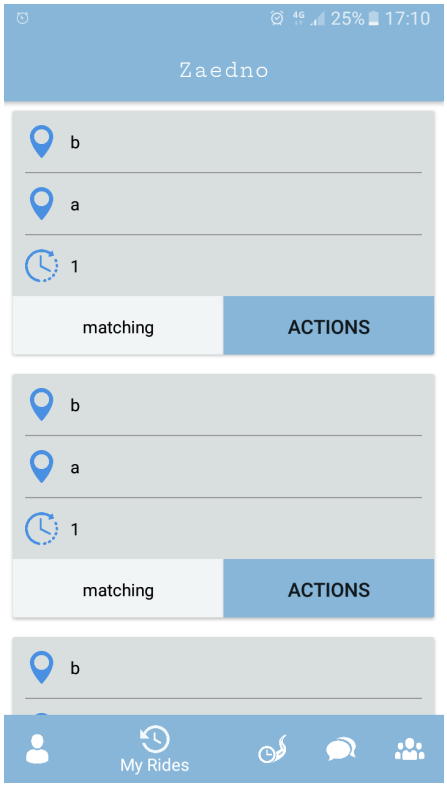
When response comes back from the API the response message is checked and if the the INSERT operation was successful the user is redirected to another fragment. If the message reports for some error a simple text message shows saying that there was an error.

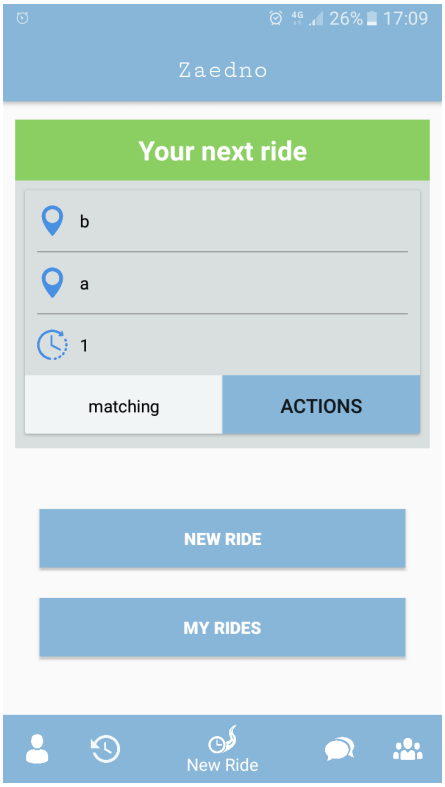
4.3. User Interface

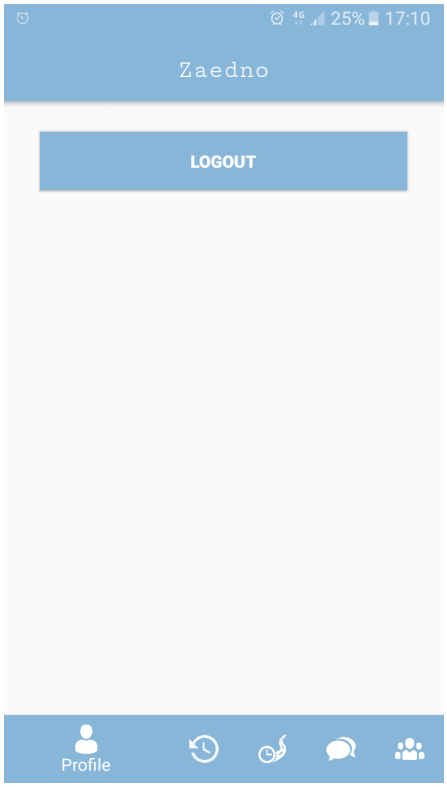
Designing the user interface was the first point and aim from the initial plan and it took me a lot of time to design these screens which lead to the best user experience for the prototype developed in this project. Part of the screens from the user interface are shown bellow:

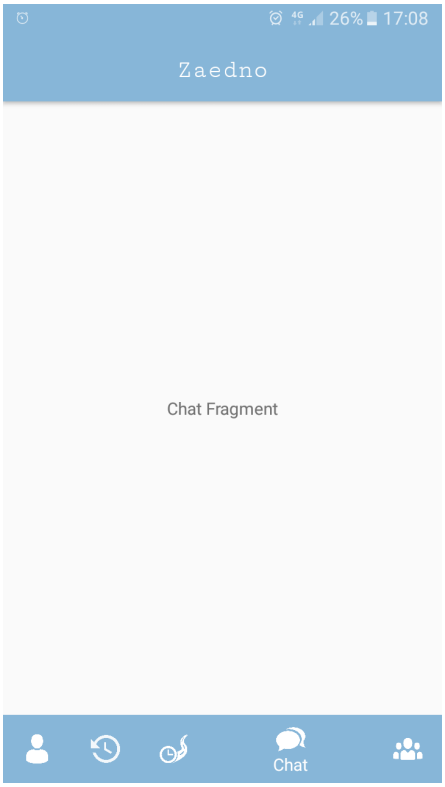


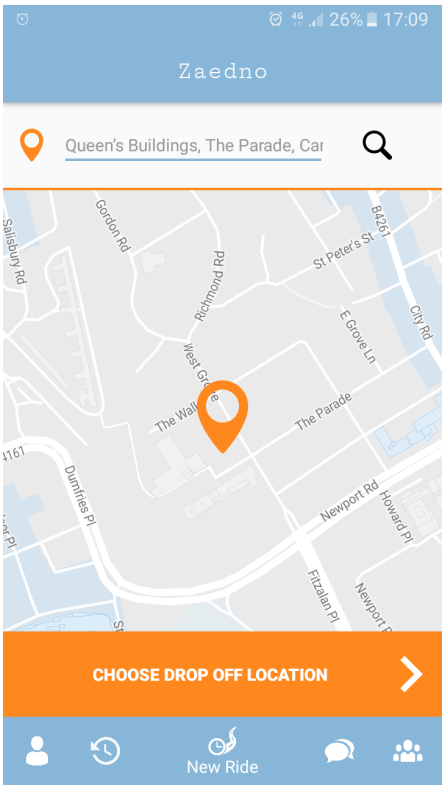


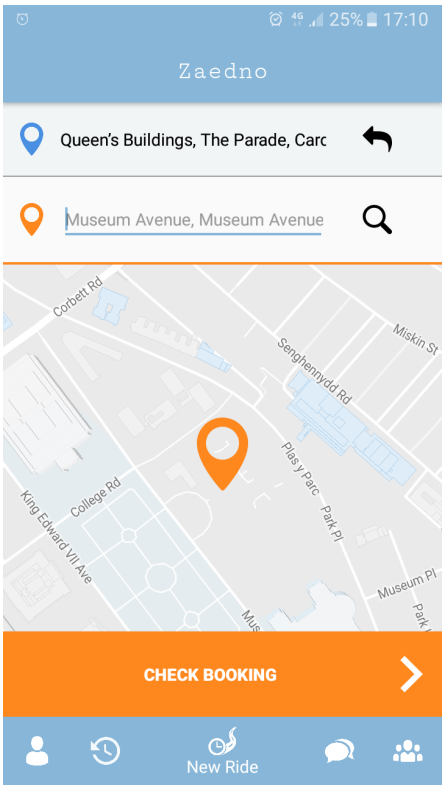

25% 17:10
Zaedno
b
a
1
matching
ACTIONS
b
a
1
matching
ACTIONS
b
My Rides

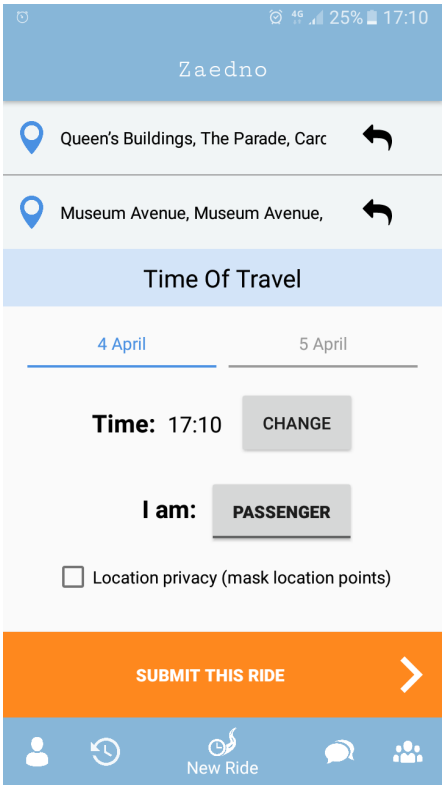

26% 17:09
Zaedno
Your next ride
b
a
1
matching
ACTIONS
NEW RIDE
MY RIDES
New Ride

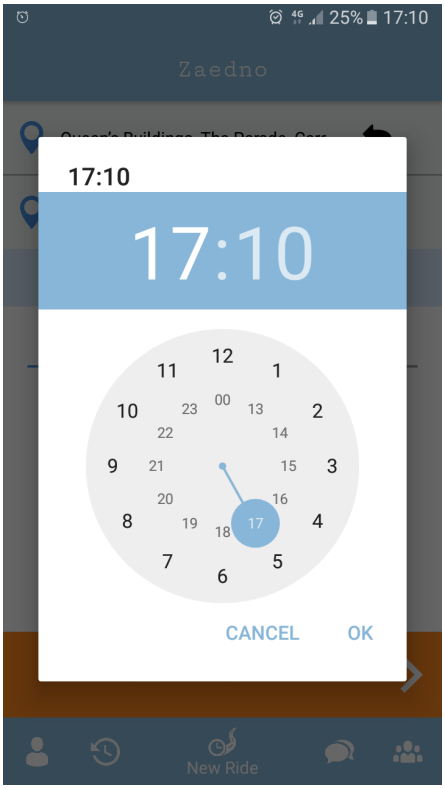

25% 17:10
Zaedno
LOGOUT
Profile


26% 17:08
Zaedno
Chat Fragment
Chat


26% 17:09
Zaedno
Queen's Buildings, The Parade, Car
CHOOSE DROP OFF LOCATION
New Ride


25% 17:10
Zaedno
Queen's Buildings, The Parade, Carc
Museum Avenue, Museum Avenue
CHECK BOOKING
New Ride


25% 17:10
Zaedno
Queen's Buildings, The Parade, Carc
Museum Avenue, Museum Avenue,
Time Of Travel
4 April
5 April
Time: 17:10
CHANGE
I am:
PASSENGER
Location privacy (mask location points)
SUBMIT THIS RIDE
New Ride


25% 17:10
Zaedno
17:10
17:10
CANCEL
OK
New Ride

4.4. Location obfuscation algorithm

The main idea of the project was to demonstrate how a location obfuscation feature can be implement inside a ridesharing application. As mentioned earlier the location obfuscation feature is implemented in the TimeOftravelFragment and it is located in two methods:

```
private void generateFakePoints() {
    mPlat_f = mPlat + generateShift();
    mPlng_f = mPlng + generateShift();

    pickUpName.setText(locToString(mPlat_f, mPlng_f));

    mDlat_f = mDlat + generateShift();
    mDlng_f = mDlng + generateShift();

    dropOffName.setText(locToString(mDlat_f, mDlng_f));

    return;
}

private Double generateShift(){
    Random r = new Random();
    double minShift = 0.0003;
    double maxShift = 0.0015;
    double shift = minShift + (maxShift - minShift) * r.nextDouble();
    if(r.nextBoolean()){
        return -shift;
    }
    return shift;
}
```

In this implementation the minimum and maximum shift points are fixed and can generate offset of maximum 300 meters. This offset won't work everywhere on the Earth because the distance in meters between any two points in longitude varies depending how far from the equator the point is. For more precise calculations a haversine formula can be used.

The random function in this algorithm uses the uniform distribution.

4.5. API

The API handling is handling part of the back end service of the application and is written with Python. The API project is separated on 3 files - api.py, config.py and requirements.txt.

- The requirements file lists all the dependencies which the API is using. Thanks to this file the environment for the project can be easily created on a new instance of the same type.

- The config file contains a dictionary with the important config data for the database connection. The file is separated from the main project (api.py) because we don't want unauthorised people to have access to the database password and login credentials if the project is uploaded to public git repository or posted online. The config file has the following dictionary object inside it:

```
db_config = {
       'host': ‘db_host',
       'username': ‘db_username',

       'password': ‘db_passowrd',

       'database': ‘db_database'
       }
```

The apy.py file imports all the dependencies so that it can use them later in the code.

```
from flask import Flask
from flask import request
from datetime import datetime
import json
import MySQLdb

# importing config data
import config
```

After that it creates the Flask application and calls the run method of the the Flask instance and runs the HTTP server.

```
app = Flask(__name__)

if __name__ == "__main__":
    app.run(host='0.0.0.0')
```

The API contains only one route “/ride/new” which receives POST requests containing “ride booking” objects. The function receiving the request checks if the expected data can be found in the POST request and returns error message if there is something missing in the request. Provided that all the data is available, an SQL insert query is created and passed to a method called executeQuery. This method connects to the database, creates a cursor and in a try/catch block tries to execute the query. If the query executes successfully the changes are committed to the database, if not there is a rollback. After that the database connection is closed and if the insertion was successful the the method returns 1 which says to the function working on the POST request to return successful message because the insertion was indeed successful. If the insertion was not successful the method returns -1 which generates an error message as a response to the request. Here is how the executeQuery method looks:

```
def executeQuery(sql):
    # Open database connection
    db = MySQLdb.connect(config.db_config["host"], config.db_config["username"],
config.db_config["password"], config.db_config["database"])

    # prepare a cursor object using cursor() method
    cursor = db.cursor()
```

```
try:
        # Execute the SQL command
        cursor.execute(sql)

        # Commit your changes in the database
        db.commit()
    except:
        # Rollback in case there is any error
        db.rollback()

    # disconnect from server
    db.close()

    if cursor.rowcount > 0:
        return 1;
    else:
        return -1
```

5. Testing and evaluation

5.1. Testing the Android prototype application

The following tests were performed to ensure that the application prototype is working correctly and that it meets the criteria specified in the Specification and Design point.

No	Description	Expected result	Actual Result	Fix/ Action
1	The application successfully loads when the application icon is clicked.	The application opens and loads one of it's activities (depending on the authorisation state)	The application opened and loaded it's MainActivity.	N/A

2	If application doesn't have authorised user the MainActivity loads when the application is open.	The user opens the application and the MainActivity loads.	As expected the MainActivity is shown on the display.	N/A
3	The user is redirected to the SigninActivity when he clicks the "Sign in" button from the MainActivity.	When the "Sign In" button is clicked the SigninActivity shows on the screen.	As expected the SigninActivity loads.	N/A
4	When the user enters his email and password on the SigninActivity and he clicks the "Sign in" button he is automatically redirected to the HomeActivity if his login credentials are correct.	When correct login credentials are entered in the Sign in screen the user is redirected to the HomeActivity.	As expected the user authenticates and he is redirected to the HomeActivity.	N/A
5	When the user enters wrong email or password on the SigninActivity and he clicks the "Sign in" button an appropriate message for failed authentication is shown.	When wrong login credentials are entered in the Sign in screen a message appears saying that the login credential were wrong.	As expected a message appears containing the text: "Authentication failed, check your email and password or sign up"	N/A
6	The user can successfully navigate between the different fragments from the bottom navigation.	When an icon from the bottom navigation is clicked the fragment inside the HomeActivity is changed.	As expected the user is able to navigate between the fragments by clicking the icons from the bottom navigation menu.	N/A
7	When the user clicks the logout button found in the UserProfileFragment he is unauthenticated and redirected to the MainActivity.	The user is redirected to the MainActivity.	As expected the logout button triggers the unauthentication method and redirects the user to the initial screen.	N/A

8	When the user clicks the “new ride” button in the newRideFragment the PickUpLocationFragment appears and the map shows with the user current location in it.	The Google map should appear showing the user current location.	As expected the PickUpLocationFragment loads and shows the user current location on the Map.	N/A
9	When the user searches for an address by entering the address in the search field - the new address appears below the pin on the map.	The new address shows below the pin. The address text of the new point changes as well.	As expected the location changes and the search field now shows the address of the searched location	N/A
10	When the “Change” button of the time of travel is clicked a Time Picker Dialog appears on the screen allowing the user to set a new time.	The time picker dialog appears on the screen.	As expected the time picker dialog shows on the screen.	N/A
11	When the user changes the time from the time picker dialog - the new time is shown.	The change of time works successfully changing the time displayed inside the fragment once the time picker dialog is closed.	The change of time function works successfully changing the time displayed inside the fragment.	N/A
12	When the user submits the ride from the TimeOfTravelFragment the new ride appears in database table.	The record should appear in the table.	As expected the new ride is stored into the database.	N/A

5.2. Test and evaluation of the API and the DB

The following tests were performed to ensure that the Application programming interface and database work as expected:

No	Description	Expected result	Actual Result	Fix/ Action
1	Successful installation of the needed dependencies from the requirements.txt file on a new environment	A successful installation of the dependencies without any errors or warnings.	The installation was successful and the api.py script run without any dependency errors.	N/A
2	Send a POST request to an unknown route on the API server	404 Error - Not fount	As expected the response of the request is 404 - Not found	N/A
3	Send empty POST request to the ‘\ride\new’ route.	An error message that some data is missing from the POST request.	As expected an error message is returned - “{"message": "Data is missing in the request.”}"	N/A
4	Send POST with some values missing from the required data to the ‘\ride\new’ route.	An error message that some data is missing from the POST request.	As expected an error message is returned - “{"message": "Data is missing in the request.”}"	N/A
5	Send POST request with all data required to the ‘\ride\new’ route.	A message that the request was successfully stored in the database should appear.	As expected a message for successful query appears - “{"message": “success"}"	N/A
6	Send POST request with all data required to the ‘\ride\new’ route but turn off the db server.	An error message that the object was not stored into the database should appear.	As expected an error message appears - “{"message": "There was an error inserting the record."}"	N/A
7	Send a “ride booking” from the application.	The record should appear in the database.	As expected a record appears in the table storing the information send from the application.	N/A

The results from the tests show that the records coming from the mobile application are successfully stored in the database. If there is some error with the data or the database server, appropriate message is returned from the API. This message ensures that the mobile application will react appropriately in both cases: when the data is uploaded successfully and when there was some error preventing the data from being uploaded.

5.3. Testing the location obfuscation feature

The following tests are performed to ensure that the privacy feature works as expected:

No	Description	Expected result	Actual Result	Fix/Action
1	Select addresses in Cardiff and activate the privacy. Check if the obfuscated addresses are within 0.5 km of the original addresses.	The obfuscated locations are below 0.5 km from the original addresses.	As expected the addresses are within the specified range.	N/A
2	Send a ‘ride booking’ object to the API with privacy feature activated. Check if the addresses saved in the database are the obfuscated ones.	When a ‘ride booking’ object is submitted with obfuscated addresses, these addresses are the ones saved in the database.	As expected the obfuscated addresses are saved inside the database table.	N/A
3	Check if the original addresses are returned in the TimeOftravelFragment if the user unchecks the privacy check button.	The textfields showing the pick up and drop off locations change to show the original locations selected by the user.	As expected clicking the checbutton for a second time returns the original addresses.	N/A

6. Conclusions

This project successfully addressed the privacy issue of the origin-destination location points. These pairs of locations, very often used in ridesharing applications can lead to a potential user identification when combined with another public datasets.

A ridesharing application prototype was developed during the project to demonstrate how the location obfuscation can address this privacy issue and to show how the feature can be embedded inside the application. Several relevant to this project location obfuscation methods were discussed in the background section and a solution using the most suitable one was shown.

The project showed that a company don’t need to invest a lot of time and resources for addressing this privacy. The location obfuscation methods discussed in this project are well researched and can provide very good anonymity if embedded in ridesharing application.

The project also showed a key elements needed for the development of an android application using location, google maps, authentication services and communication with API.

7. Future Work

The prototype developed in this project has very limited functionalities needed only to address the specific location privacy issue discussed in the project. We are going to check two ways in which the application and the project can be extended. The first idea for future work involves the location obfuscation algorithm and more precisely how the algorithm can be improved. The second idea is showing several functionalities that can be added to the ridesharing prototype in order to extend the functionalities that it currently has. This can help converting this project into more functional ridesharing application.

7.1. Location obfuscation feature improvements

In this section we will address the location obfuscation algorithm since this was the main focus of the project. The algorithm can be modified so that it works everywhere on Earth by having very precise shift range from the initial locations. This can be achieved with the Haversine formula base equation. Such equation will be able to calculate the distance between two [latitude, longitude] pairs into great-circle distance between them. In this way we can generate points which are always within some range in meters specified in the code. Another way in which the algorithm can be extended is writing a module which will generate several proposition addresses for each original location and a choice will be given to the user. This will allow the user to pick an address which is known to him near his original address. Further the algorithm can use random distribution different from the uniform one. For example a different distribution can have bigger chances of getting the shifted location near the shift range limit than getting the location near the original one.

7.2. The ridesharing prototype application extensions

The prototype application can be extended and turned into more functional ridesharing application with several updates. I have already started to work on additional feature which can show the user rides by taking the objects from the database. The application side of this feature is done and we can see the actual Ride CardViews controlled by the RecyclerView adapter. This cards can be populated with data from the database if the API is extended to return these objects.

During the project I had considered three additional functionalities which can extend the application even further:

One of them is a ride matching algorithm which compares similar active ‘ride bookings’ from the ride table and proposes the closest rides to match a ride created by the user.

Another functionality can be a chat between the users which will allow the users to discuss further details about their travels once a match between them is created. This chat functionality can be easily developed with the Firebase Realtime Database and the Firebase Cloud Messaging since we already have adapted the Google Firebase platform in our project.

Another good idea for extension will be a payment system with QR codes ensuring that the payment is only happening once when the two users are together.

8. Reflection on Learning

Working on this project was clearly a challenge but this is what I was looking for when I was picking the project topic. At the time when I started the work on the project I had no previous experience with Android development. I wanted to focus on learning Android and during the three months in which I had to work on the project i learned a lot about the platform and I managed to prepare myself for eventual job involving Android mobile development. I also learned a lot about location privacy and the different methods used to ensure it - one of which location obfuscation.

I spent time researching and evaluating ridesharing applications in order to find location privacy issues and map these issues with ones found in studies.

During the project I had to make choices about the outcome of the project having in mind the limited time for work that we had. I also had to find the right balance between research, learning and development. There was definitely several parts of the project which required very serious research skills. These parts improved my abilities to read and analyse academic text and also my abilities to extract key elements from very complex studies.

The project definitely improved my project planning skills. Even during the initial plan I had to design the complete architecture of the project in order to set up the aims and the scope of the project. During the development I had to work with various technologies, frameworks and languages and this definitely improved my programming knowledge. I also spend a lot of time sketching and designing the user interface of the application and when I look now what the final version of the application is I think I definitely gained some experience evaluating different UI elements and what user experience they can create.

The project also gave me a great experience with self discipline, time management and planning. And after these three months of hard work i believe these three things go hand in hand.

Writing this report gave me a huge experience in evaluating my work and it’s results. I also improved my writing skills and more precisely my ability to write and think under pressure.

9. References

(1) Shared Economy [Online]. Available: http://www.investopedia.com/terms/s/sharing-economy.asp [Accessed: 25/04/2017]

(2) 2017: The Year The Rideshare Industry Crushed The Taxi - Infographic [Online]. Available: https://rideshareapps.com/2015-rideshare-infographic/ [Accessed: 28/04/2017]

(3) The Ridesharing Revolution: Economic Survey and Synthesis, Robert Hahn and Robert Metcalfe, January 10, 2017 [Online]. Available: https://www.brookings.edu/wp-content/uploads/2017/01/ridesharing-oup-1117-v6-brookings1.pdf [Accessed: 28/04/2017]

(4) Ride-sharing: The rise of innovative transportation services, Emily Nicoll and Sally Armstrong, April 12, 2016 [Online]. Available: https://www.marsdd.com/news-and-insights/ride-sharing-the-rise-of-innovative-transportation-services/ [Accessed: 25/04/2017]

(5) A Short Guide to Writing Your Final Year Project Report Or MSc Dissertation, Cardiff university, February 2011 [Online], Available: https://www.cs.cf.ac.uk/PATS2/wiki/lib/exe/fetch.php?media=project-report.pdf [Accessed: 20/04/2017]

(6) The protection of user location privacy in mobile applications, Shazaib Ahmad, May 2016 [Online]. Available: https://www.cs.cf.ac.uk/PATS2/@archive_file?c=&p=file&p=527&n=final&f=1-CM3203_C1312433.pdf [Accessed: 22/04/2017]

(7) Occupancy rates of passenger vehicles, EEA [Online]. Available: https://www.eea.europa.eu/data-and-maps/indicators/occupancy-rates-of-passenger-vehicles/occupancy-rates-of-passenger-vehicles-1 [Accessed: 25/04/2017]

(8) High Level Summary of Statistics Trend - Car Occupancy [Online]. Available: http://www.gov.scot/Topics/Statistics/Browse/Transport-Travel/TrendCarOccupancy [Accessed: 25/04/2017]

(9) On the Anonymity of Home/Work Location Pairs, Philippe Golle and Kurt Partridge [Online]. Available: https://crypto.stanford.edu/~pgolle/papers/commute.pdf [Accessed: 26/04/2017]

(10) Exploring End User Preferences for Location Obfuscation, Location-Based Services, and the Value of Location, A.J. Bernheim Brush, John Krumm and James Scott [Online]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/ubicomp243-brush.pdf [Accessed: 28/04/2017]

(11) Evaluation of Location Obfuscation Techniques forPrivacy in Location Based Information Systems, Pedro Wightman, Winston Coronell, Daladier Jabba,Miguel Jimeno [Online]. Available: http://www.academia.edu/889879/Evaluation_of_Location_Obfuscation_Techniques_for_Privacy_in_Location_Based_Information_Systems [Accessed: 28/04/2017]

(12) Overview of Google Play Services [Online]. Available: https://developers.google.com/android/guides/overview [Accessed: 10/04/2017]

(13) Requesting Permissions at Run Time [Online]. Available: https://developer.android.com/training/permissions/requesting.html [Accessed: 10/05/2017]

(14) Introduction to Android [Online]. Available: https://developer.android.com/guide/index.html [Accessed: 10/04/2017]

(15) Add Firebase to Your Android Project [Online]. Available: https://firebase.google.com/docs/android/setup [Accessed: 10/04/2017]

(16) Python MySQLdb vs mysql-connector query performance [Online]. Available: http://charlesnagy.info/it/python/python-mysqldb-vs-mysql-connector-query-performance [Accessed: 20/04/2017]

(17) Python MySQL Database Access [Online]. Available: https://www.tutorialspoint.com/python/python_database_access.htm [Accessed: 22/04/2017]

(18) Chapter 3: An Analysis of Android App Permissions [Online]. Available: http://www.pewinternet.org/2015/11/10/an-analysis-of-android-app-permissions/ [Accessed: 15/04/2017]

(19) Calculate distance, bearing and more between Latitude/Longitude points [Online]. Available: http://www.movable-type.co.uk/scripts/latlong.html [Accessed: 01/05/2017]

(20) Haversine formula [Online]. Available: https://en.wikipedia.org/wiki/Haversine_formula [Accessed: 01/05/2017]

(21) Welcome to Flask [Online]. Available: http://flask.pocoo.org/docs/0.12/ [Accessed: 10/04/2017]

(22) MySQLdb User's Guide [Online]. Available: http://mysql-python.sourceforge.net/MySQLdb.html [Accessed: 20/04/2017]