



Nguyen Doan Quoc Thien

# ANDROID MOBLIE APPLICATION FOR TOUR GUIDE SYSTEM

Technology and Communication  
2019

## ACKNOWLEDGEMENT

First of all, I would like to give a big Thanks to Mrs. Pirjo Prosi, my supervisor, for the patience she gave me during the period that I worked on my thesis. Besides, I would like to give big appreciation for all teachers in VAMK, who have taught me during the past journey.

I would like to give a special thanks to Dr. Ghodrat Moghadampour too, as he is the main teacher who guide us in Software Engineering technology. His great and careful work helped me a lot through the hard time when I had to deal with challenge from school

I would like to thank my family for the support they give me during the past years. I would not have been able to finish the thesis without all their mental support. During the up and down period of the thesis writing period, my friends and colleagues have been helping me a lot in term of searching for solution for the errors I encounter as well as teaching me new technology.

Final words, I would like to give to my appreciation for the online community from <https://forums.xamarin.com/> and <https://stackoverflow.com/>, where I get the solution for most of the errors as well as questions I have.

Helsinki, 04.04.2019

Thien Nguyen

VAASAN AMMATTIKORKEAKOULU  
UNIVERSITY OF APPLIED SCIENCES

Technology and Communication

## ABSTRACT

Author	Nguyen Doan Quoc Thien
Title	Android Mobile Application for Tour Guide System
Year	2019
Language	English
Pages	
Name of Supervisor	Pirjo Prosi

---

The main goal of the thesis was to create an android mobile application for users, those who travel to a new destination, to help them to find a local guide to help them discover the new place, new culture.

The application will have two different launch views. The first one is for regular users, who use the application in order to find a tour guide for his/her trip. In that view, a user will be able to register and create an account. All the information of that user will be saved in Azure cloud database. With that account, that user will log in to the application and from then add a new request to find a tour guide for that person's next trip. The application will make sure that the user has to enable the location services on the mobile device for the authorization.

The second view is made for people who would like to become a guide for travelers. Similar to the first view, a user will be able to register, log in to the application. All the data will again be stored on Azure cloud database. Users of this group will be able to view the list of requests, as well as details from the request, and then decide to pick up the request to contact the user from the first group and negotiate about the upcoming trip with them. All views are built on Xamarin with fundamental C# and Azure cloud databases.

The thesis has achieved its main goal, to create a tool in order to help people from different places to communicate with each other and help out in term of traveling, getting to know new people, places and cultures. The thesis has so much potential to be further developed in the future, with a team who have enough knowledge to implement all the features in order to get the application ready to use.

---

Keywords                      Xamarin, C#, Azure

## CONTENTS

1. INTRODUCTION .....	7
1.1 Background .....	7
1.2 Motivations.....	7
1.3 Objectives .....	8
2 RELEVANT TECHNOLOGIES .....	9
2.1 Xamarin .....	9
2.2 C# programming language .....	9
2.3 Visual Studio 2017 .....	9
2.4 Azure Cloud Service .....	10
2.5 Relevant Open source package.....	10
3 APPLICATION DESCRIPTION .....	12
3.1 Function Description .....	12
3.2 Quality Function Deployment .....	12
3.2.1 Must have requirements.....	12
3.2.2 Should have requirements .....	13
3.2.3 Nice to have requirements .....	13
3.3 Use Case Diagram .....	13
3.4 Class Diagram .....	15
3.5 Sequence Diagram.....	16
4 DATABASE AND UI DESIGN.....	20
4.1 Database Design .....	20
4.2 User Interface design.....	21
5 IMPLEMENTATION .....	28
5.1 General structure .....	28
5.2 Implementing shared TravelApp(Portable) project.....	29
5.3 Implementing TravelApp project for Travel User Group .....	31
5.4 Implementing GuidePerson project for Guide User Group.....	39
6 TESTING .....	43
6.1 Signing up and log in for new Traver User .....	43
6.2 Making new request for Travel User Group .....	47

6.3 Sign up and log in for Guide Person Group .....	50
6.4 Managing and accepting requests.....	51
7 CONCLUSION.....	57

### **LIST OF FIGURES:**

Figure 1: Travel User Diagram.....	14
Figure 2: Guide User Diagram .....	15
Figure 3: Class Diagram for Users .....	16
Figure 4: User Log in Sequence Diagram .....	17
Figure 5: User Insert Sequence Diagram.....	18
Figure 6: Guide User Method Sequence Diagram.....	19
Figure 7: ER Diagram .....	21
Figure 8: Log in Design.....	22
Figure 9: Register Design .....	23
Figure 10: Main Tab Design.....	24
Figure 11: Add Request Design.....	25
Figure 12: Tabs Design .....	26
Figure 13: Review Design for Guide User .....	27
Figure 14: Basic Structure .....	28
Figure 15: Structure of a project.....	29
Figure 16: Access Azure Service.....	29
Figure 17: Register method .....	30
Figure 18: Login method .....	30
Figure 19: Get Host method .....	31
Figure 20: RegisterButton of MainActivity.....	31
Figure 21: Register Button function .....	32
Figure 22: Sign in button function.....	32
Figure 23: Layout code design .....	33
Figure 24: Tab Layout Selection .....	34
Figure 25: TabToolBar click .....	34
Figure 26: List of properties .....	35

Figure 27: Insert method .....	35
Figure 28: AndroidManifest Code.....	36
Figure 29: Google Map change and display .....	36
Figure 30: GetRequest and GetProcessed method .....	37
Figure 31: Get Tabs View with custom adapter .....	38
Figure 32: OnListItemClick method.....	39
Figure 33: OnListItemClick on Guide user .....	40
Figure 34: Mark as Pick up method .....	41
Figure 35: Mark as being processed method .....	41
Figure 36: Mark as Processed method.....	42
Figure 37: Successfully register .....	44
Figure 38: Error log in .....	45
Figure 39: Successfully log in main tab .....	47
Figure 40: Add new Request .....	48
Figure 41: New request shown in Pending tab .....	50
Figure 42: Guide User log in successfully .....	51
Figure 43: New request in Waiting tab.....	52
Figure 44: View and process request.....	53
Figure 45: Request is in Processing tab.....	54
Figure 46: View and Accept request .....	55
Figure 47: Request is in Processed Tab.....	56

# 1. INTRODUCTION

## 1.1 Background

As someone who travels often, it is a challenge for me to connect with the local people and culture where I travel. Problems can happen when someone travels to a whole new country where local people have different type of thinking, culture, beliefs, etc. This application come out to help tourist to get quick and better view about where they are about to visit, as well as better connects with local people and reduce the chance that troubles happen along the journey.

Developing a mobile app plays an important role in the technology nowadays, as the number of mobile devices are multiple times more than the number of our population. From that huge market, Android devices get 85.9% of all mobile devices, which show us how big potential that an Android application has to affect our society. /1/

Xamarin is one of the most useful types of API, which offers a cross platform mobile development. Developing an application for android using Xamarin will give us an opportunity for later, if we want to expand the application to IOS operating system.

C# is one of the most popular programming languages at the moment, with approximately 31% of all developers using it regularly. The language creates 17000 jobs each month globally. It is very effective and powerful. /2/

## 1.2 Motivations

The motivation for this thesis comes from personal issues of the author, as I love traveling usually, and most of the time when I travel through Europe, I have to spend quite a lot of time doing research while being busy with work and school. The problem does not only occur to me, but also to many young travelers. This could lead to many other serious problems, as the difference in cultures is huge, and the number of people who travel abroad is enormous and still growing rapidly. Similar to AirBnb, which shows the

problem of housing, this application could show the problem about guiding. The motivation also comes from the personal practice as a junior Android developer.

### **1.3 Objectives**

The main functions of the application should come from the idea of the motivation. There are two separate views. The first view will allow the user to register, add new requests as well as check if their requests are processed yet. The mobile application should connect well with the server using Microsoft Azure mobile services to send and receive all information and display it on the application.

The second view will have a tab that lists all the pending requests from the user of the first group. In addition, this view will include their location that they send using Google API, then the user of this group can check their contact information to negotiate about the upcoming trip. After that, the user will decide to accept the request or not, so that other users will not see that request in the pending tab anymore.



## **2 RELEVANT TECHNOLOGIES**

### **2.1 Xamarin**

Xamarin is the tool that Microsoft develops in order to help developers create cross platform application. Therefore, instead of using two separate programming languages, which are Java to build Android application, and Swift to build IOS application, now Xamarin allows us to build application on both Android and IOS operating system using C# only. By doing that, we do not have to learn two programming languages at the same time, but we can focus on C# only, which is also very popular programming language. /3/

### **2.2 C# programming language**

C# is a general-purpose, imperative, functional, generic, object-oriented programming language and was developed around 2000 by Microsoft. C# is one of the most popular programming languages in the world, with share of 7.45% and still rising. /4/

C# was created by Anders Hejlsberg, and at the moment, the language is being developed by a team which is led by Mads Torgersen. The most recent version is C# 7.3, which was released in 2018. /5/

### **2.3 Visual Studio 2017**

Visual Studio 2017 is an integrated development environment (IDE) which provides development tools, cloud services, libraries that allows developers to create applications for different platforms, from Windows app for desktops, website to mobile application for Android and iOS operating system. Visual Studio 2017 is the newest version of Visual Studio, and it is also the most popular integrated development environment, which the share of 22.82% of the market in 2018. The newest update of the IDE is version 15.9.11, which was released in 2/4/2019. /6/

## 2.4 Azure Cloud Service

Azure is a cloud service which is created by Microsoft in order to help IT developers and programmers to build and manage their work through their cloud data center network. It was firstly announced as Project Red Dog on October 2008, then changed to Windows Azure on February 1, 2010 and finally renamed as Microsoft Azure on March 25, 2014. The service domains in Azure includes: /7/

- Compute
- Storage services
- Database
- Networking
- Developer Tools
- Management and monitoring tools
- Enterprise Integration
- Security and Identity
- Web and Mobile Application

## 2.5 Relevant Open source package

-Xamarin Google Play Service Maps: With the Google Maps Android API, maps which base on Google Maps can be added to the application. The API will handle request that access Maps servers, download data, display map. With API calls, it is also possible to add marker, labels, custom draws to the map as well as change the user's view of a particular map area. This will enhance user experience with the application as well as providing additional functions to the application. /8/

-Xamarin Android Support: Xamarin Support Libraries provide many different tools for building application, such as providing application components, user interface widgets to material design. Xamarin Android Support v4 is compatible with Android 4 (API 14) or higher while Xamarin Android Support v7 is compatible with Android 5(API 21) or higher. /9/

-Azure Mobile Client: this is the library that help programmer to connect their cross-platform application with Azure Mobile Apps service. Azure Mobile Apps service allows programmers to develop an application with many different supports for mobile authentication, offline sync, and push notifications. /10/

## **3 APPLICATION DESCRIPTION**

### **3.1 Function Description**

The purpose of the application is to create an environment for people who travels often to meet locals, who provide the guide service through an android application. The application includes two separate view, which provide different function to different group of users.

- **Travel User Group:** First allows the user to register, then uses that register information to log in. In the main tab, the user will be able to view past request, as well as create a new request which will be sent to Guide User Group for their acceptance. This will require the user to enable the location service on their mobile application, for the authorization.
- **Guide User Group:** This view also allows the user to register and log in. From the main tab, users of this group will see all the pending requests, which they will be able to click in, in order to view the information of that pending request, as well as contact the one who sent the request in order to negotiate about the upcoming trip. After that, the guide person will either accept that request, so that other guide users will not see that request or choose to leave it for the other guide users.

### **3.2 Quality Function Deployment**

#### **3.2.1 Must have requirements**

**For Travel User Group:** The user of this group must be able to register into the application with an account which holds an email and password. After that, the account information can be used to log in to the application. From the application, a new request from user's upcoming trip can be created, stored in the database and sent to Guide User Group.

**For Guide User Group:** The user of this group must be able to register into the application with an account which hold email and password. The account will be used later to

log in to the user into the main tab. After logging in, that user will see a list of requests, which are pending. He can choose to view the details of that request, then click accept it if he agrees with the travel user about the upcoming trip.

### **3.2.2 Should have requirements**

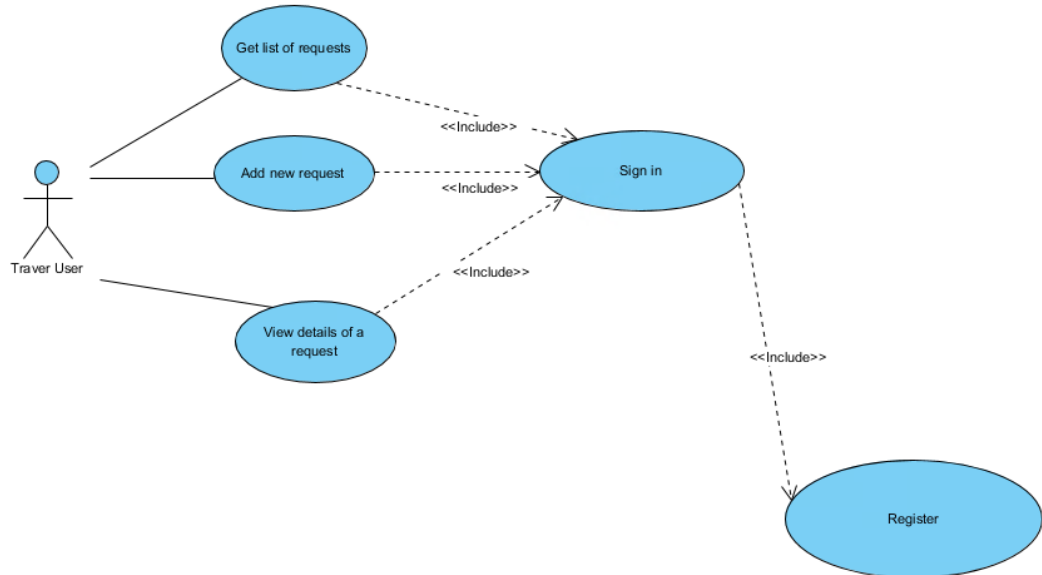
Users from both of the groups should be able to use their social media account to register and log in to the application. The travel user should be requested to enable the location service to identify their location for authorization purpose. A travel user should also be able to review their past requests.

### **3.2.3 Nice to have requirements**

Both views of the application should have a responsive user interface without bug. It would be better if the travel user could get a notification if their request has been accepted. Guide user group should be able to see their past requests that they approved before.

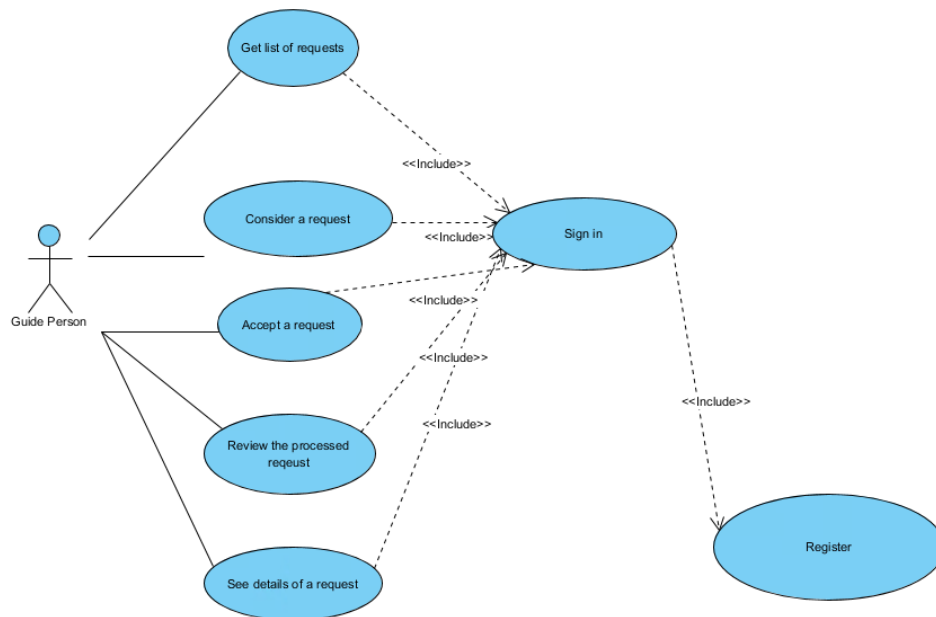
## **3.3 Use Case Diagram**

For the use case diagram of the first view, the user can get a list of requests which were created by other travel users. The user can also create a new request by filling in all the fields in the form and click send button. Lastly, the user can view the details of a request that includes all the information and the map from other user's location. All the features are required by logging in to the application. To be able to log in, the user will need an account by registering.



**Figure 1:** Travel user use case diagram

As shown in the Guide User use case diagram, the user can register for an account, then log in with that account information. After logging in, the user will be able to access to a number of features. First, the user will be able to see a list of requests from different tabs, which are Waiting, Processing and Processed. After that, this user can check all the details of the request's user and put the request's status into Processing. After negotiating with the travel user about the upcoming trip, this user will be able to put the request status to Processed or not, if the user decides to do so, the request will be moved to Processed tab and removed from Processing tab.

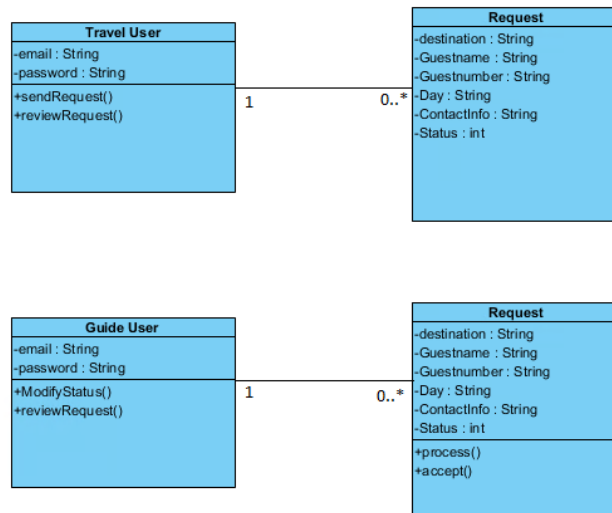


**Figure 2:** Guide User Use Case Diagram

### 3.4 Class Diagram

The travel Application for the first user group includes the signing up and logging in method, which is using the email and password attributes. It also has the sendRequest and reviewRequest method, which can send the new request to the Guide Person group, with the attributes of the requests are: destination, Guestname, Guestnumber, Day, ContacInfo and the status value will be assigned automatically as 0.

The Guide Person Application also includes the signing up and logging in method, which uses the email and password attributes. It contains the reviewRequests, which show the list of the request as well as the details of a request. The request itself will contain a method. For the waiting request, it has the Process method, which allows the user to change its status value from 0 to 1. The Processing request will also have the Accept method, which allows the user to switch its status value from 1 to 2.

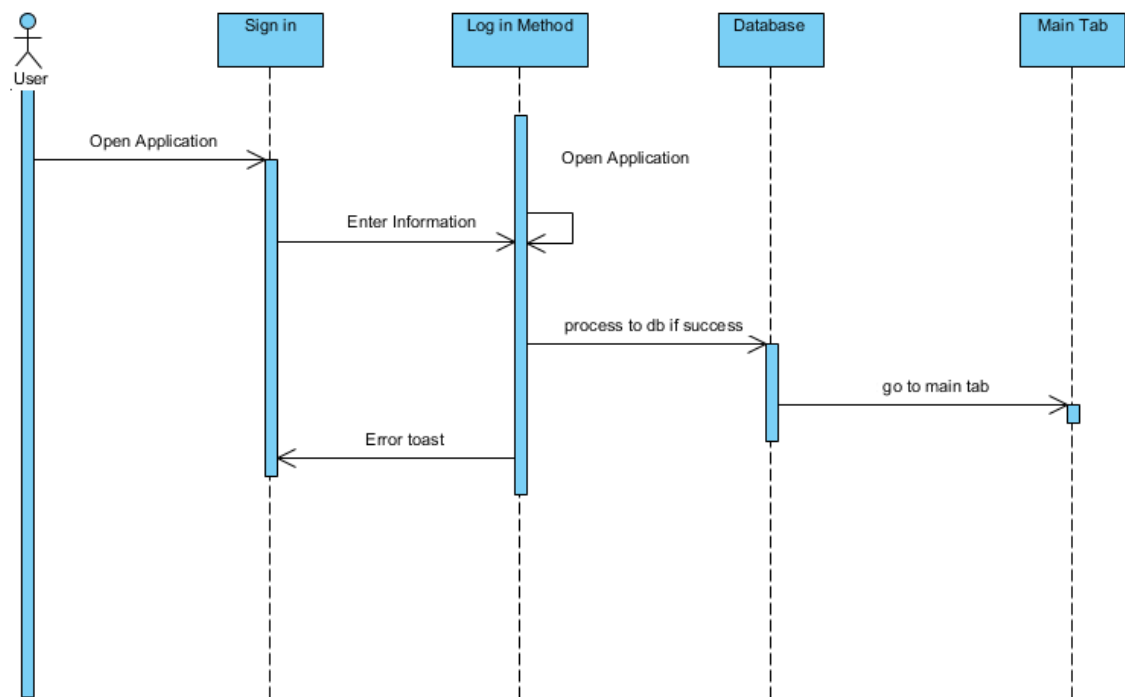


**Figure 3:** Class Diagram for Users

### 3.5 Sequence Diagram

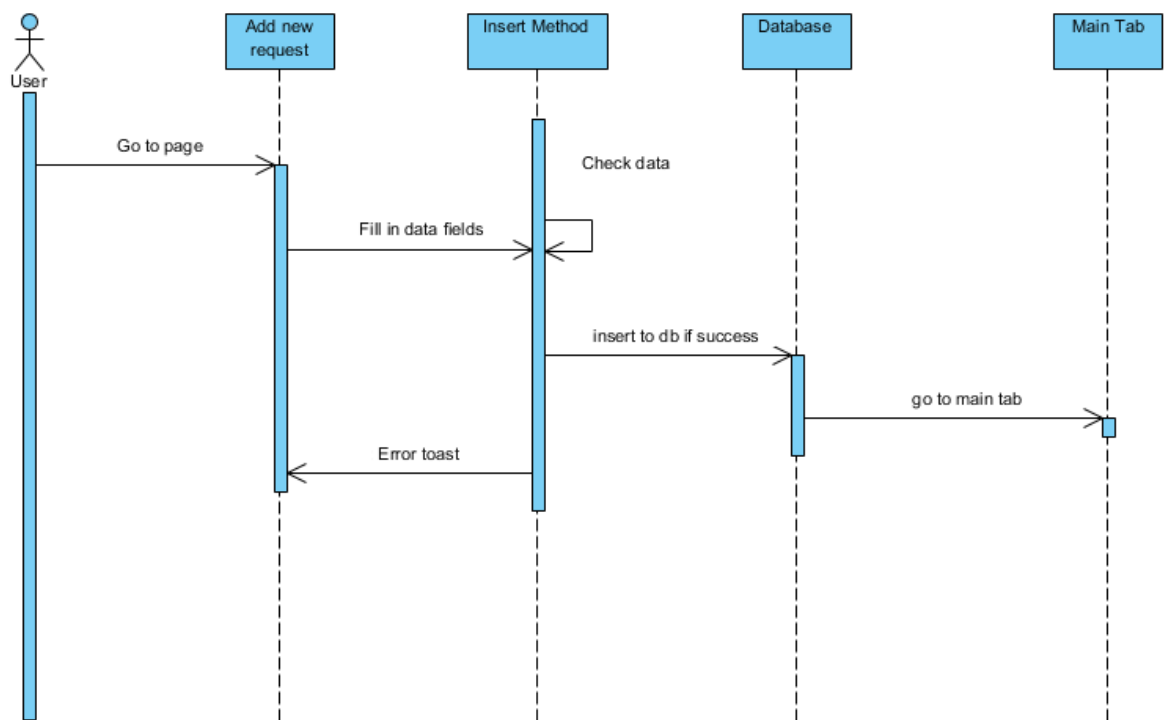
User from both groups will enter the register data, which includes email and password, then hit the register button. If the value is valid, it will be accepted, the data is inserted in the database. Then the user enters that information to log in. If the information is correct with data in the database, the user will be navigated to Main Tab, otherwise the user will see a toast, which asks him to check again the data that he enters.





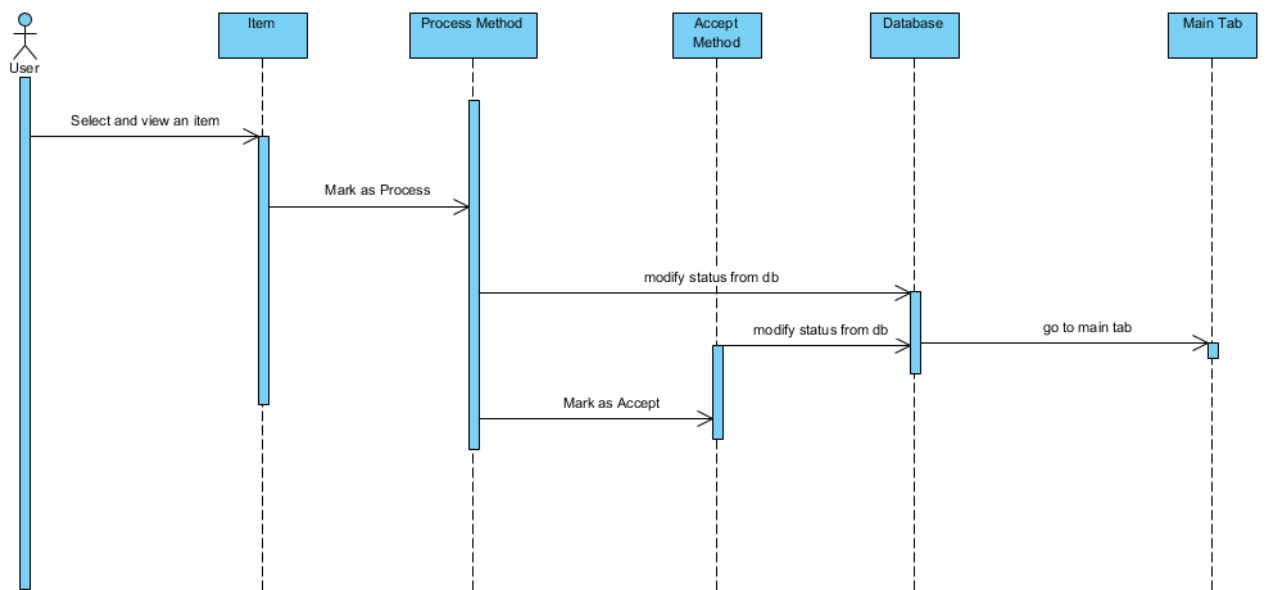
**Figure 4:** User Log in Sequence Diagram

When the Travel user decides to add a new request, the user will be navigated to a view, where he can use the Insert Method. First the user will fill in all the details of the fields as well as allowing the location service. If the data that user input is valid, it will be inserted to the database, otherwise the user will get an error alert to let that user know that the value is invalid.



**Figure 5:** User Insert Sequence Diagram

As shown in Figure 6, the Guide person will be able to check all the details of requests, which include Waiting request and Processing request. For the Waiting request, the user can mark as Processing by clicking on the Process button and calling the Process method. After that, the status value of the request will be changed and updated to the database. Similarly, the user can mark the Processing request to Processed, which will call the Accept method, change the status value of the request to 2 and updated to the database.



**Figure 6:** Guide User Method Sequence Diagram

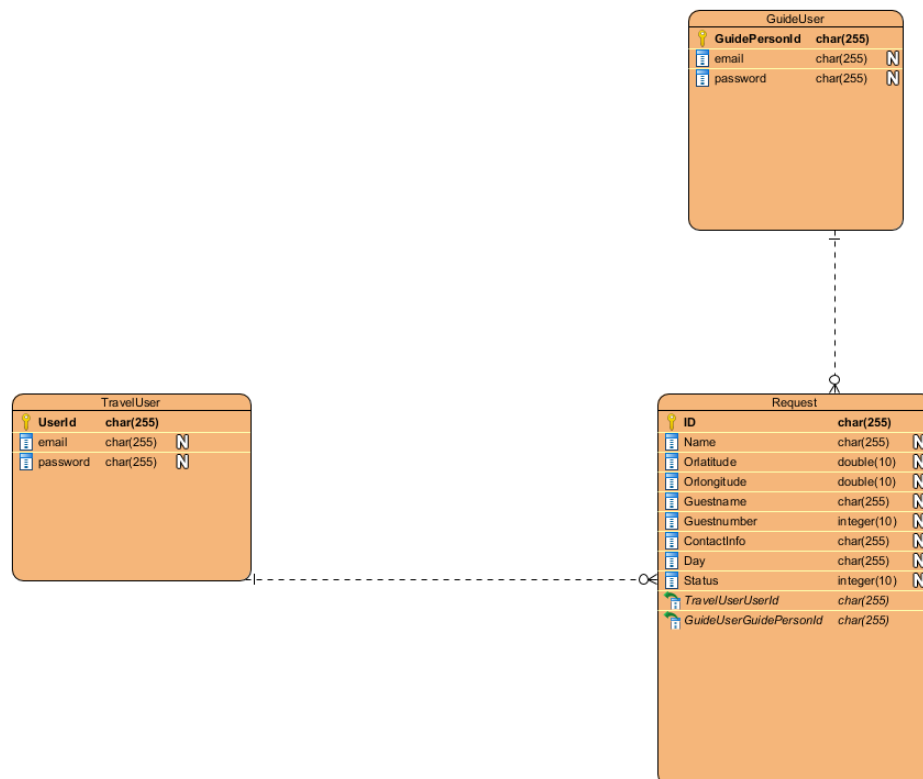
## 4 DATABASE AND UI DESIGN

### 4.1 Database Design

Azure Cloud Database service is used to save all the data of the application. There are three tables for the application, they are

- TravelUser: store user ID, name and password for the travel user group
- GuidePerson: store user ID, name and password for the guide user group
- Request: store request data that User group send and retrieve those data back to GuidePerson group users.

Since SQL Azure databases do not support diagrams in the same way as a normal database does, a third-party tool is used to draw the ER diagram, which is Visual Paradigm.



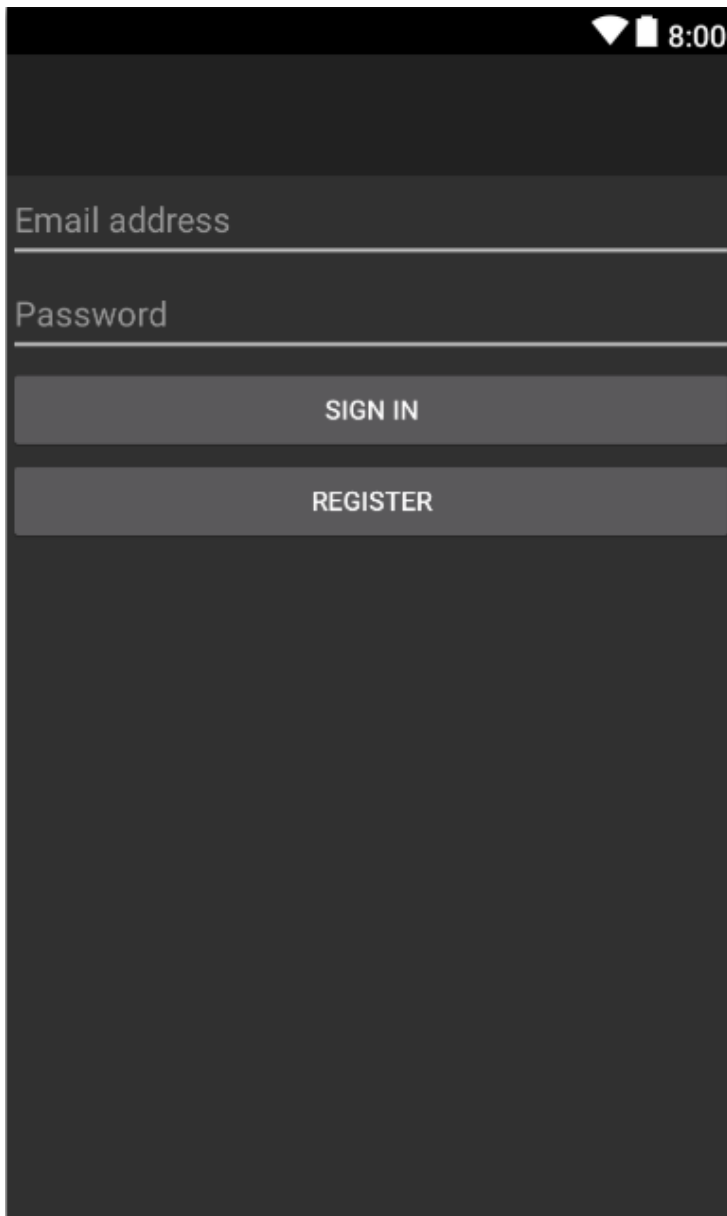
**Figure 7: ER Diagram**

The user table has a one-to-many relationship with PendingRequest table, since one user could create many requests, but one request cannot belong to many users.

The GuidePerson table has a one-to-many relationship with the PendingRequest table as well, it means that one guide person can get many requests, but one request cannot be selected by many Guide users.

**4.2 User Interface design**

The application includes two separate views, one for the travel user and one for the guide person. However, only one page will be designed for both views, in order to focus on the function.



**Figure 8:** Log in Design

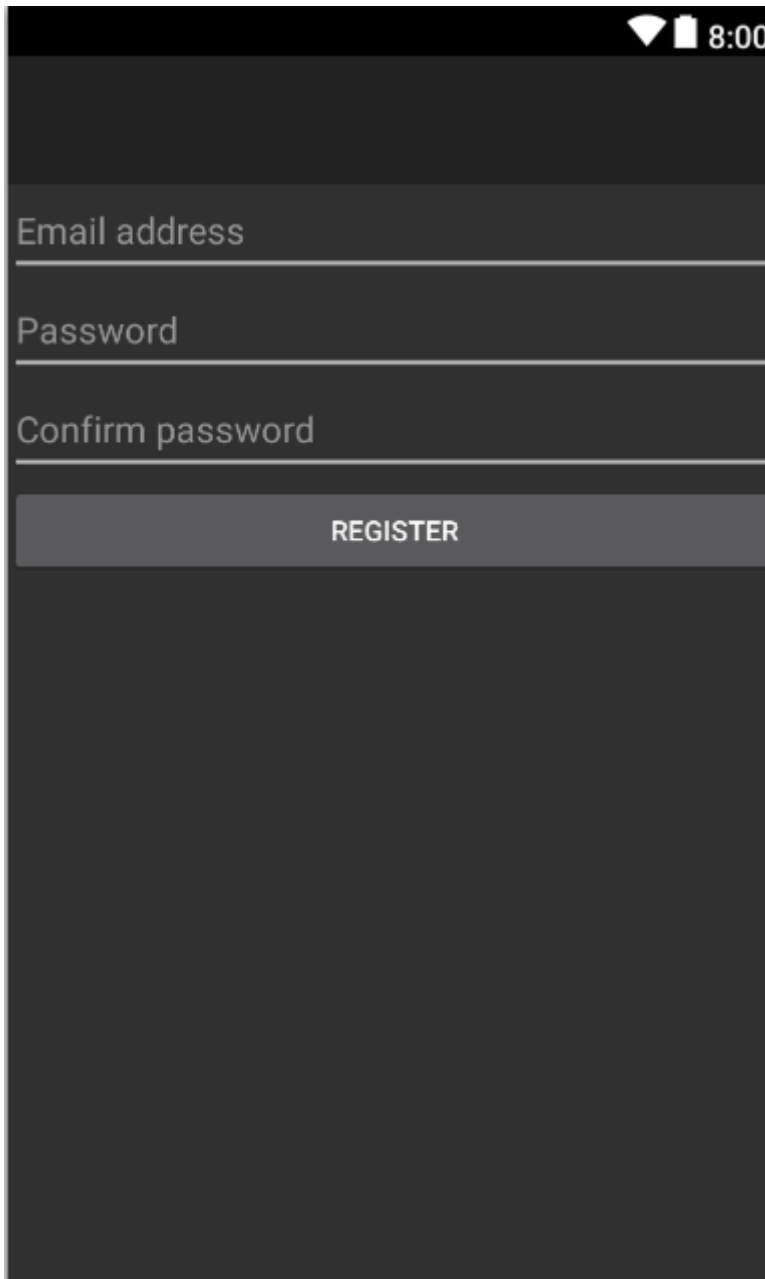
It will include two buttons:

- Sign in
- Register

and two empty boxes, which are:

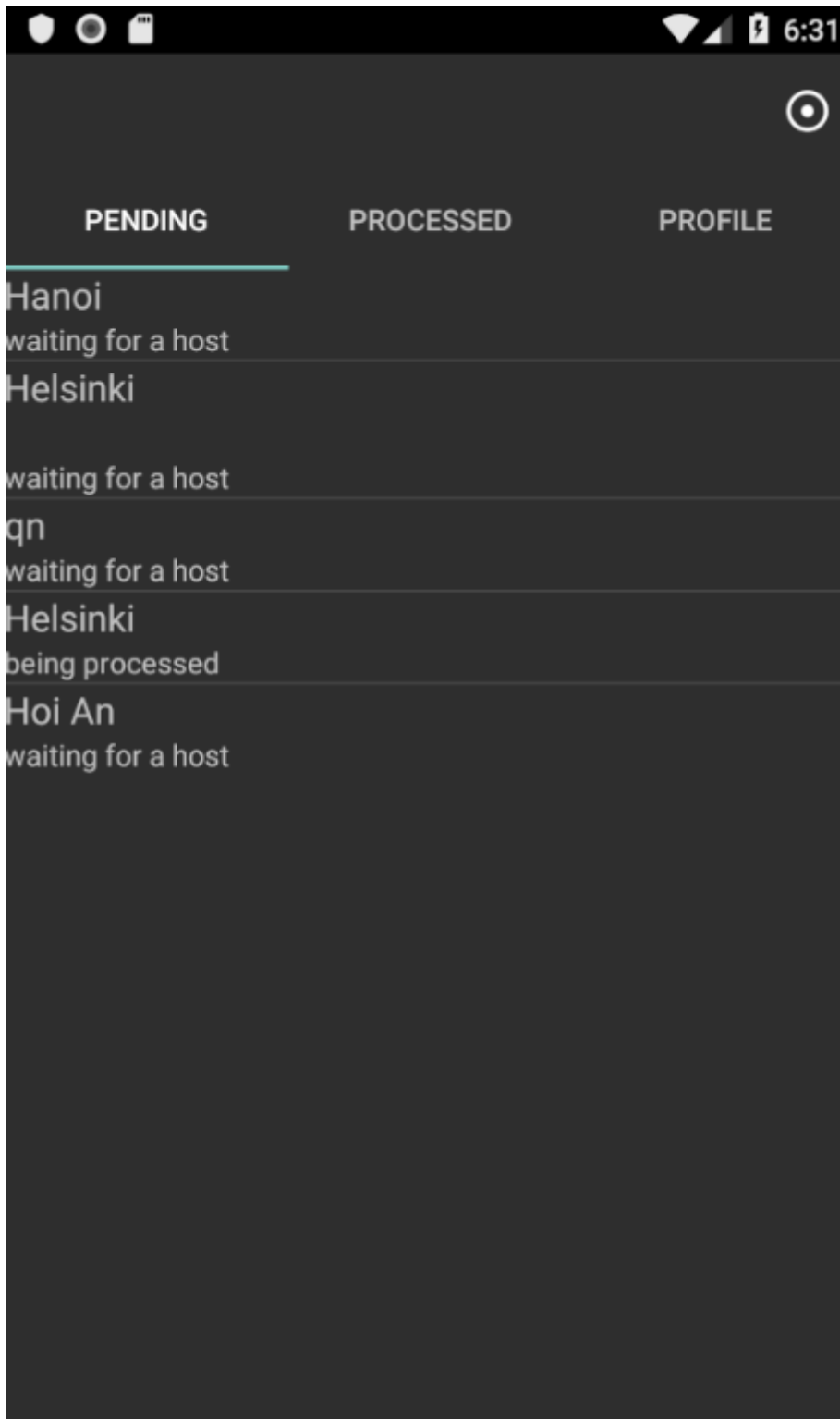
- Email Address
- Password

If the user does not have an account yet, he can click on the register button in order to navigate to the register page:

A screenshot of a mobile application's register page. The screen is dark-themed. At the top, there is a status bar with a Wi-Fi icon, a battery icon, and the time 8:00. Below the status bar, there are three text input fields stacked vertically. The first field is labeled "Email address", the second "Password", and the third "Confirm password". Each field has a light-colored border. Below the input fields is a prominent, wide button with the text "REGISTER" in white capital letters. The bottom portion of the screen is mostly dark and appears to be a continuation of the form or a navigation area.

**Figure 9:** Register Design

After logging in, the user will be navigated to the Main page, where he can view all pending requests, as well as the one which are processed already.



**Figure 10:** Main Tab Design

Figure 10 shows the pending request tab, where all the pending or processing requests are collected. The default tab when the user logs in will be the Pending tab.



When clicking on each request, the detail information of the request will be reviewed

In the right top corner, he will be able to make a new request, which requires the person enables the location service:

Name of Destination

Guest Name

Number of guest

Day

Contact Info

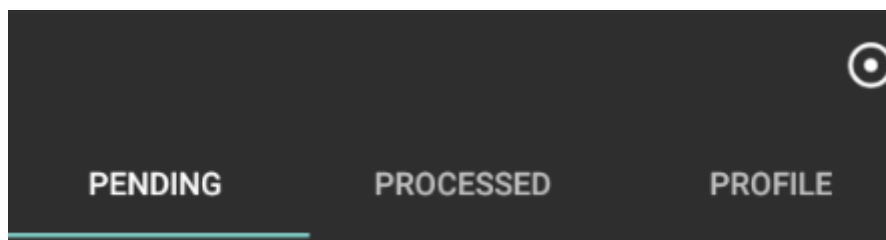
SAVE

Request Origin

Google Map showing Helsinki region with a red location pin. Labels include Helsinki, Laajasalo, Uutela, Fazerila, Gumbosträ, E75, and E18.

**Figure 11:** Add Request Design

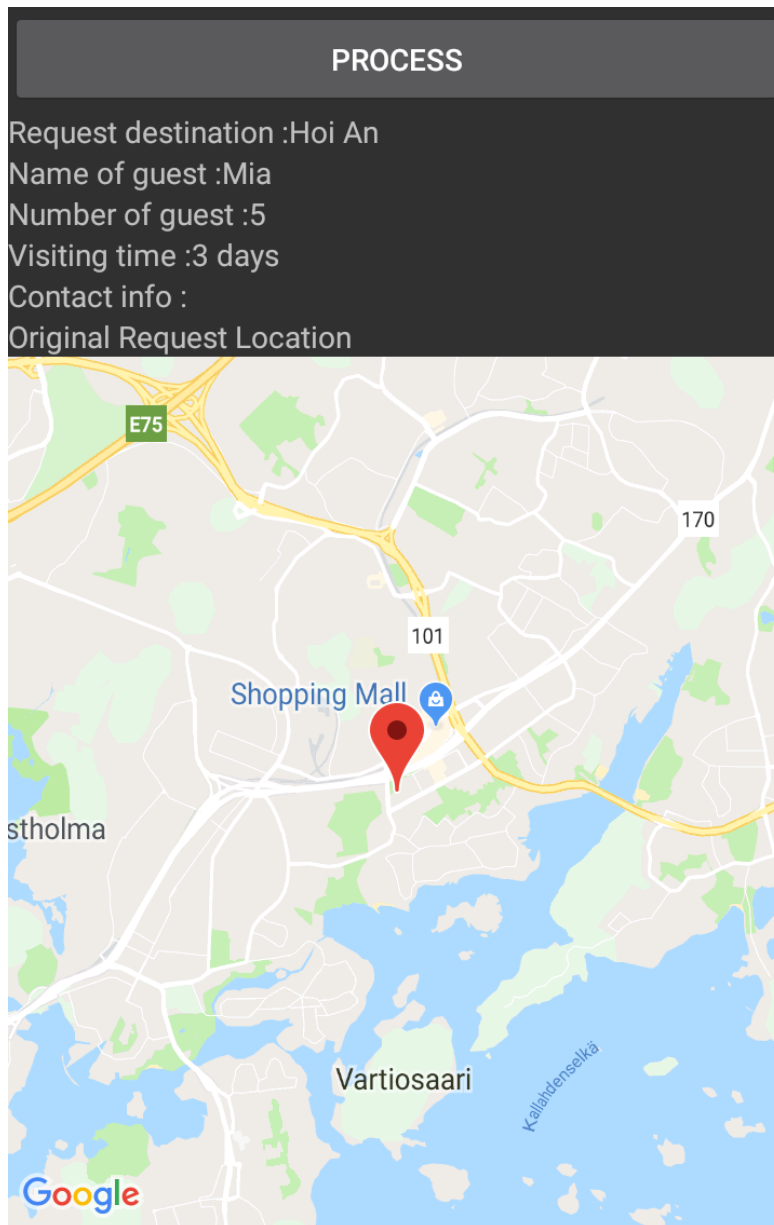
After clicking the Save button, the request will be saved, you can review it in the pending tab. That request will also be stored and retrieved in the Guide User view, when one log in.



**Figure 12:** Tabs Design

The navigation bar is designed as simple as possible, with three different tabs. There is a green line under the tabs which is being selected, in order to prevent the user from being confused by navigating tabs.

Similar to the Travel user view, the Guide user view will have a similar register and log in design. After logging in, the guide user will also be able to see the list of the pending requests. However, instead of viewing only, the person of this group can also choose and put the request to 'Processing' status, which will notify the travel user to know that their request is being taken care of and that person will be contacted soon, by the contact information they provide in the request. Once the negotiation is done between the guide user and travel user, guide user will change the status of the request into 'Processed', which will notify other guide users to know that this request has been accepted by someone else, and also that request will be removed away from the Pending Request Tab.

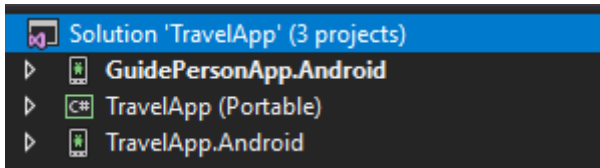


**Figure 13:** Review Design for Guide User

## 5 IMPLEMENTATION

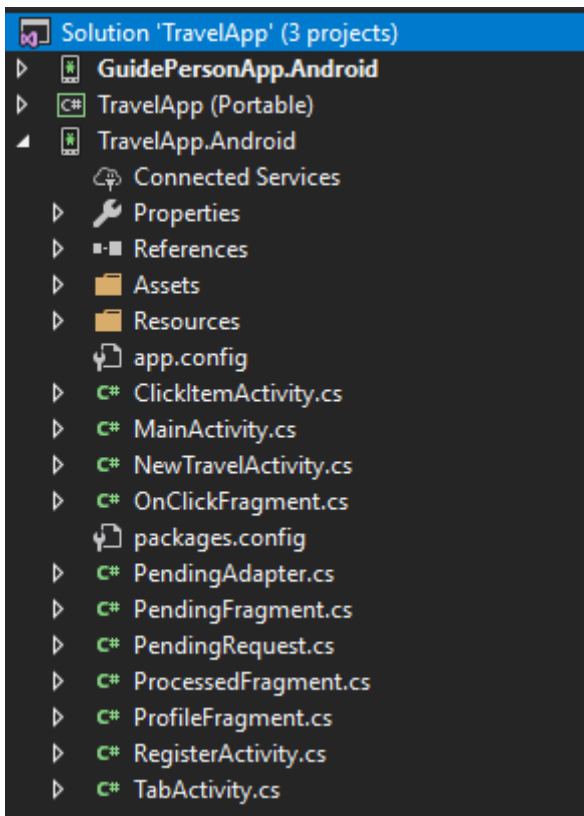
### 5.1 General structure

The figure below shows the structure of the application.



**Figure 14:** Basic Structure

When the application is created, it will include another project called TravelApp(IOS), but since the main target is Android operating system, the IOS project is deleted. After that, a new project called GuidePersonApp is created, whose function handles all the tasks for the Guide user group.



**Figure 15:** Structure of a project

Figure 15 shows the structure of an Android project, which is TravelApp.Android. It includes a list of C# items, such as Activity, Fragment and Adapter. The design files will be stored inside the folder Resources/layout. Meanwhile, all the additional Nuget packages are installed inside References.

## 5.2 Implementing shared TravelApp(Portable) project

First, a public folder is created in this project and it is implemented with some classes, for the purpose of reusing those classes. The first class is AzureHelper, which will connect our project to the resource that is already created on Azure Cloud Service, as well as insert all the data to tables from the resources, which is xamarintravelapp19:

```

1  using Microsoft.WindowsAzure.MobileServices;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace Travel.Model
9  {
10     public class AzureHelper
11     {
12         public static MobileServiceClient MobileService = new MobileServiceClient("https://xamarintravelapp19.azurewebsites.net");
13
14         public static async Task<bool> Insert<T>(T objectToInsert)
15         {
16             try
17             {
18                 await MobileService.GetTable<T>().InsertAsync(objectToInsert);
19                 return true;
20             }
21             catch (Exception)
22             {
23                 return false;
24             }
25         }
26     }
27 }
28

```

**Figure 16:** Access Azure Service

After that, the User class is created, which handles the register and log in methods:

```

public static async Task<bool> Register(string email, string password, string confirmPassword)
{
    bool result = false;

    if (!string.IsNullOrEmpty(password))
    {
        if (password == confirmPassword)
        {
            var user = new User()
            {
                Email = email,
                Password = password
            };

            await AzureHelper.Insert(user);

            result = true;
        }
    }

    return result;
}

```

**Figure 17:** Register method

For the register method, the application will first check if the password is empty or not. If it is not empty, it will continue to check if the Password box and Confirm Password box hold the same data. If everything is correct, the data of the new user, which includes email and password will be inserted into User table.

```

public static async Task<bool> Login(string email, string password)
{
    bool result = false;

    if (string.IsNullOrEmpty(email) || string.IsNullOrEmpty(password))
    {
        result = false;
    }
    else
    {
        var user = (await AzureHelper.MobileService.GetTable<User>().Where(u => u.Email == email).ToListAsync()).FirstOrDefault();

        if (user.Password == password)
        {
            result = true;
        }
        else
        {
            result = false;
        }
    }

    return result;
}

```

**Figure 18:** Login method

The log in method will have the mission to check between the data that user log in and the data which are held in the User table, if they are correct or any box is empty.

Similar to the User class, the GuidePerson class will have register and log in method, however, it will also have an additional method, which is GetHost, with the purpose of checking which Guide user has accepted the request from Travel User, by comparing id:

```
public static async Task<GuidePerson> GetHost(string id)
{
    GuidePerson host = new GuidePerson();
    host = (await AzureHelper.MobileService.GetTable<GuidePerson>().Where(d => d.Id == id).ToListAsync()).FirstOrDefault();
    return host;
}
```

**Figure 19:** Get Host method

### 5.3 Implementing TravelApp project for Travel User Group

For the TravelApp project, there is a list of functions that needed to be implemented. Firstly, the function for the Register Button is implemented, which is inside the MainActivity, when it is clicked:

```
private void RegisterButton_Click(object sender, EventArgs e)
{
    var intent = new Intent(this, typeof(RegisterActivity));
    intent.PutExtra("email", emailEditText.Text);
    StartActivity(intent);
}
```

**Figure 20:** RegisterButton of MainActivity

For this, an intent is used, as intent is a simple message object which is used to communicate from one activity to another. First, the intent is created which has two parameters: Context and Type. The context is the class which stores information about the current application's state, from the place where the intent is created. In this case, the context is set to this, which calls Activity class reference. Context is the sender. Type is the activity destination where the intent is sent to, which is RegisterActivity in this case. The intent will also hold value of emailEditText's Text as "email". Then the activity of the intent is started. It will get navigated to The RegisterActivity, where It retrieves the value of the intent by:

```
string email = Intent.GetStringExtra("email");
```

and assign the value to emailEditText by:

```
emailEditText.Text = email;
```

Now for the function of the Register button of the register page, the follow function is implemented:

```
private async void RegisterButton_Click(object sender, EventArgs e)
{
    var result = await User.Register(emailEditText.Text, passwordEditText.Text, confirmPasswordEditText.Text);
    if (result)
        Toast.MakeText(this, "Success", ToastLength.Long).Show();
    else
        Toast.MakeText(this, "Try again", ToastLength.Long).Show();
}
```

**Figure 21:** Register Button function

Here, async await is used, to keep the application from executing anything before the await command has been executed, which is calling the register method from User class in order to add new user to User table on Azure. If it is successfully executed, It will make a toast as “Success” to let the user know that the data has been added, otherwise It will make a toast to let user know that his register has failed.

```
private async void SigninButton_Click(object sender, EventArgs e)
{
    var email = emailEditText.Text;
    var password = passwordEditText.Text;

    var result = await User.Login(email, password);

    if (result)
    {
        Toast.MakeText(this, "Login successfull", ToastLength.Long).Show();
        Intent intent = new Intent(this, typeof(TabActivity));
        StartActivity(intent);
        Finish();
    }
    else
        Toast.MakeText(this, "Please check again your email address or password", ToastLength.Long).Show();
}
```

**Figure 22:** Sign in button function

Back to the function of SigninButton, the Login function from the User class is called, in order to check if the email and password that the user input are correct as the data on the Azure table. The application will make toasts to let the user know if his login is successful or not. If yes, it will create a new intent, pass this-current class as the context of the intent and TabActivity as Type of destination activity, then end the current activity.



Otherwise, it will make a toast as "Please check again your email address or password" and stay in the current activity.

Once the user successfully accesses the application, he will see a head tab, with 3 separate tabs, as well as a button on top to create a new request. In order to have that, a layout is created as follow:

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:orientation="vertical"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent">
6
7      <android.support.v7.widget.Toolbar
8          android:id="@+id/tabToolbar"
9          android:layout_height="wrap_content"
10         android:layout_width="match_parent"/>
11
12     <android.support.design.widget.TabLayout
13         android:id="@+id/mainTabLayout"
14         android:layout_height="wrap_content"
15         android:layout_width="match_parent">
16         <android.support.design.widget.TabItem
17             android:text="Pending" />
18         <android.support.design.widget.TabItem
19             android:text="Processed" />
20         <android.support.design.widget.TabItem
21             android:text="Profile" />
22     </android.support.design.widget.TabLayout>
23     <FrameLayout
24         android:id="@+id/contentFrame"
25         android:layout_width="match_parent"
26         android:layout_height="wrap_content" />
27 </LinearLayout>

```

**Figure 23:** Layout code design

In order to have three separate tabs, three different fragments are created so that it can be navigated between those tabs. As in Android, only one activity is on the screen at one time, so it is needed to create three fragments, as fragment presents behaviour of user interface in Android, and considered to be sub-activity. Now in the TabActivity, a tab layout is created and switched the tab into three cases, each one with the corresponding tab and fragment: Pending, Processed and Profile.

```
private void TabLayout_TabSelected(object sender, TabLayout.TabSelectedEventArgs e)
{
    switch (e.Tab.Position)
    {
        case 0:
            FragmentNavigate(new PendingFragment());
            break;
        case 1:
            FragmentNavigate(new ProcessedFragment());
            break;
        case 2:
            FragmentNavigate(new ProfileFragment());
            break;
    }
}
```

**Figure 24:** Tab Layout Selection

For the default tab when the user log in, the Pending tab is assigned as default by:

```
FragmentNavigate(new PendingFragment());
```

And to be able to add a new request, a new layout is needed whenever the Add button is clicked, which has the following items:

- destinationEditText
- gusteditText
- numgusteditText
- numdayeditText
- contactinfoeditText
- saveButton
- mapFragment

Also, a NewTravelActivity class is needed, then it calls that Activity from the top button on the tabToolBar:

```
private void TabToolBar_MenuItemClick(object sender, Android.Support.V7.Widget.Toolbar.MenuItemClickEventArgs e)
{
    if (e.Item.ItemId == Resource.Id.action_add)
    {
        StartActivity(typeof(NewTravelActivity));
    }
}
```

**Figure 25:** TabToolBar click

Now a class called PendingRequest is created, where it holds a list of properties:

```
public string Id { get; set; }
public string Name { get; set; }
public double OrLatitude { get; set; }
public double OrLongitude { get; set; }
public double DesLatitude { get; set; }
public double DesLongitude { get; set; }
public string GuestName { get; set; }
public string GuestNumber { get; set; }
public string Day { get; set; }
public string ContactInfo { get; set; }

public int Status { get; set; }

public string GuidePersonId { get; set; }
```

**Figure 26:** List of properties

This is method to insert the new requests into the PendingRequest table on Azure:

```
public static async Task<bool> InsertDestination(PendingRequest pends)
{
    return await AzureHelper.Insert<PendingRequest>(pends);
}
```

**Figure 27:** Insert method

Back to the NewTravelActivity, it will access the data the user inputs and stores it in the PendingRequest table. The most important function of the NewTravelActivity is to access the location of the user. First, the property of the project needs to be modified, and enable a list of permission, in order to allow the mobile device to access its location, they are;

- ACCESS\_FINE\_LOCATION
- ACCESS\_NETWORK\_STATE
- INTERNET
- WRITE\_EXTERNAL STORAGE
- READ\_GSERVICES
- MAPS\_RECEIVE

Also a NuGet package is needed to be installed, which is Xamarin.GooglePlayServices.Map. On the AndroidManifest.xml file, it is necessary to modified and added the API keys as follow:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:installLocation="auto">
  <uses-sdk android:minSdkVersion="15" />
  <uses-feature android:glEsVersion="0x00020000" android:required="true" />
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
  <uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
  <uses-permission android:name="travelapp.android.permission.MAPS_RECEIVE" />
  <permission android:name="travelapp.android.permission.MAPS_RECEIVE" android:protectionLevel="signature" />
  <application android:label="Deliveries" android:theme="@style/Theme.AppCompat">
    <meta-data android:name="com.google.android.maps.v2.API_KEY" android:value="AIzaSyD0xKzpoqbauFbThtamzx9ofVbMYRqFtVM" />
    <meta-data android:name="com.google.android.gms.version" android:value="@integer/google_play_services_version" />
  </application>
</manifest>
```

**Figure 28:** AndroidManifest Code

A map fragment will be required to call `IONMapReadyCallback`, `ILocationListener`, in order to be showed on the tab.

```
public void OnLocationChanged(Location location)
{
    latitude = location.Latitude;
    longitude = location.Longitude;
    mapFragment.GetMapAsync(this);
    destinationmapFragment.GetMapAsync(this);
}

public void OnMapReady(GoogleMap googleMap)
{
    MarkerOptions marker = new MarkerOptions();
    marker.SetPosition(new LatLng(latitude, longitude));
    marker.SetTitle("Your location");
    googleMap.AddMarker(marker);

    googleMap.MoveCamera(CameraUpdateFactory.NewLatLngZoom(new LatLng(latitude, longitude), 10));
}
```

**Figure 29:** Google Map change and display

`OnLocationChanged` will handle the change of the location of the mobile device, so every single time when `GetMapAsync` is executed, this method will get the new coordinator of the location and update the new one. `OnMapReady` put a title of the current location on the map as a marker.

For each of new request, it will carry a status data, with the following value

- 0: waiting for process

- 1: being processed
- 2: processed.

Since a new request is made, it will always carry a status with 0-waiting for process. A method called `GetRequest` is created in `PendingRequest`, in order to call all the requests from the `PendingRequest` table. It will compare the status value of the request, and any request with the status different from 2 will be shown, as they are in the status of waiting for process or being process. Similarly, `GetProcessed` method is created to call all the requests that carry status value as 2, which is processed.

```
public static async Task<List<PendingRequest>> GetRequests()  
{  
    List<PendingRequest> pends = new List<PendingRequest>();  
    pends = await AzureHelper.MobileService.GetTable<PendingRequest>().Where(d => d.Status != 2).ToListAsync();  
    return pends;  
}  
  
public static async Task<List<PendingRequest>> GetProcessed()  
{  
    List<PendingRequest> pends = new List<PendingRequest>();  
    pends = await AzureHelper.MobileService.GetTable<PendingRequest>().Where(d => d.Status == 2).ToListAsync();  
    return pends;  
}
```

**Figure 30:** `GetRequest` and `GetProcessed` method

After that, those methods are in the `PendingFragment` and `ProcessFragment`, in order to get the list of waiting for process, being processed and processed requests. However, since a default adapter is being used, information of the data will not be shown in the way it should be. In order to avoid that case, a custom adapter is made called `PendingAdapter`, where instead of showing the value of the status, it will give a text to each value of the status, then display the text of that status:

```

public override View GetView(int position, View convertView, ViewGroup parent)
{
    var view = convertView;
    PendingAdapterViewHolder holder = null;

    if (view != null)
        holder = view.Tag as PendingAdapterViewHolder;

    if (holder == null)
    {
        holder = new PendingAdapterViewHolder();
        var inflater = context.GetService<Context>.LayoutInflaterService).JavaCast<LayoutInflater>();
        view = inflater.Inflate(Resource.Layout.PendingCell, parent, false);
        holder.Name = view.FindViewById<TextView>(Resource.Id.destinationTextView);
        holder.Status = view.FindViewById<TextView>(Resource.Id.statusTextView);
        view.Tag = holder;
    }

    var pend = pends[position];
    holder.Name.Text = pend.Name;

    switch (pend.Status)
    {
        case 0:
            holder.Status.Text = "waiting for a host";
            break;
        case 1:
            holder.Status.Text = "being processed";
            break;
        case 2:
            holder.Status.Text = "processed";
            break;
    }

    return view;
}

```

**Figure 31:** Get Tabs View with custom adapter

Now in order to view all the details of the item in the list, `OnItemClickListener` method is used. First, a layout of displaying all the information of a request is created and named `Clickitem.xml`. Then with the `OnItemClickListener` method, the intent object will hold value of destination, guest name, number of guests, day, contact information and pass them whenever it starts the `ClickItemActivity` with the intent object

```

public override async void OnListItemClick(ListView l, View v, int position, long id)
{
    base.OnListItemClick(l, v, position, id);
    var pending = await PendingRequest.GetRequests();
    var selectedPends = pending[position];

    var userId = (Activity as TabActivity).userId;

    Intent intent = new Intent(Activity, typeof(ClickItemActivity));
    intent.PutExtra("userId", userId);
    intent.PutExtra("destination", selectedPends.Name);
    intent.PutExtra("pendId", selectedPends.Id);
    intent.PutExtra("guestname", selectedPends.GuestName);
    intent.PutExtra("destination", selectedPends.Name);
    intent.PutExtra("numguest", selectedPends.GuestNumber);
    intent.PutExtra("visittime", selectedPends.Day);
    intent.PutExtra("contactinfo", selectedPends.ContactInfo);

    StartActivity(intent);
}

```

**Figure 32:** OnListItemClick method

#### 5.4 Implementing GuidePerson project for Guide User Group

Similar to the Register and Login function from the TraveApp project for Travel User group, it is needed create and implement required class, layout and methods. After logging in, the user will also be navigated to three tabs, they are Waiting, Processing and Processed. Each of the tabs will contain the list of items, with the corresponding status value as 0, 1, 2. The default tab will be Waiting. These three tabs are implemented as similar as in the Travel User group project. After implementation, there are three fragments, which are onwaitFragment, processingFragment and processedFragment, as well as two layouts, which are pick.xml and processed.xml.

For the onwaitFragment, it shows a list of waiting requests from the PendingRequest table, by calling the GetWaiting method from PendingRequest class and access to all the request that hold the status value of 0.

```

List<PendingRequest> pends;
public override async void OnCreate(Bundle savedInstanceState)
{
    base.OnCreate(savedInstanceState);

    pends = new List<PendingRequest>();
    pends = await PendingRequest.GetWaiting();
    ListAdapter = new ArrayAdapter<Activity, global::Android.Resource.Layout.SimpleListItem1, pends>;
}

public override View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)
{
    return base.OnCreateView(inflater, container, savedInstanceState);
}

public override void OnListItemClick(ListView l, View v, int position, long id)
{
    base.OnListItemClick(l, v, position, id);

    var selectedPends = pends[position];

    var userId = (Activity as tabActivity).userId;
    Intent intent = new Intent(Activity, typeof(pickActivity));
    intent.PutExtra("latitude", selectedPends.OrLatitude);
    intent.PutExtra("longitude", selectedPends.OrLongitude);
    intent.PutExtra("userId", userId);
    intent.PutExtra("pendId", selectedPends.Id);
    intent.PutExtra("guestname", selectedPends.GuestName);
    intent.PutExtra("destination", selectedPends.Name);
    intent.PutExtra("numguest", selectedPends.GuestNumber);
    intent.PutExtra("visittime", selectedPends.Day);
    intent.PutExtra("contactinfo", selectedPends.ContactInfo);

    StartActivity(intent);
}

```

**Figure 33:** OnListItemClick on Guide user

When clicking on an item, It will start the pickActivity, which will get navigated to pick.xml layout. The layout will show all the information of the request, plus a “Pick” button. When clicking the “Pick” button, it will call the MaskAsPick method from the PendingRequest class, which request an object of PendingRequest and guidepersonId. In the method, the status value of the object will be set to 1-being processed, and then update the table PendingRequest, as the status value of the object has been changed.



```

public static async Task<bool> MarkAsPick(PendingRequest pends, string guidePersonId)
{
    try
    {
        pends.Status = 1;
        pends.GuidePersonId = guidePersonId;
        await AzureHelper.MobileService.GetTable<PendingRequest>().UpdateAsync(pends);
        return true;
    }
    catch(Exception ex)
    {
        return false;
    }
}

```

**Figure 34:** Mark as Pick up method

From that, the object has been moved from the Waiting tab to Processing tab, which similarly calls the GetBeingProcessed method from PendingRequest class and access to all the request that hold the status value of 1. This tab could be modified in order to check the guidepersonId, which will check if the guidepersonId from the request is similar to the Id of the user, so that the Waiting tab will only show the request that is waiting from a particular guide person.

```

public static async Task<List<PendingRequest>> GetBeingProcessed(string id)
{
    List<PendingRequest> pends = new List<PendingRequest>();
    pends = await AzureHelper.MobileService.GetTable<PendingRequest>().Where(d => d.Status == 1 && d.GuidePersonId == id).ToListAsync();
    return pends;
}

```

**Figure 35:** Mark as being processed method

After clicking on items from this tab, user can access all the information of the request user and decide to press the Process button or not, by contacting the request user using the contact information in the request and negotiate and discuss about the upcoming trip. If user clicks on the Process button, it will call the MarkAsProcessed method, which will change the status value of the request into 2, which is processed. After that, the list of this tab will be updated, as the data from Azure table has been changed. The request now will be moved to Processed tab.

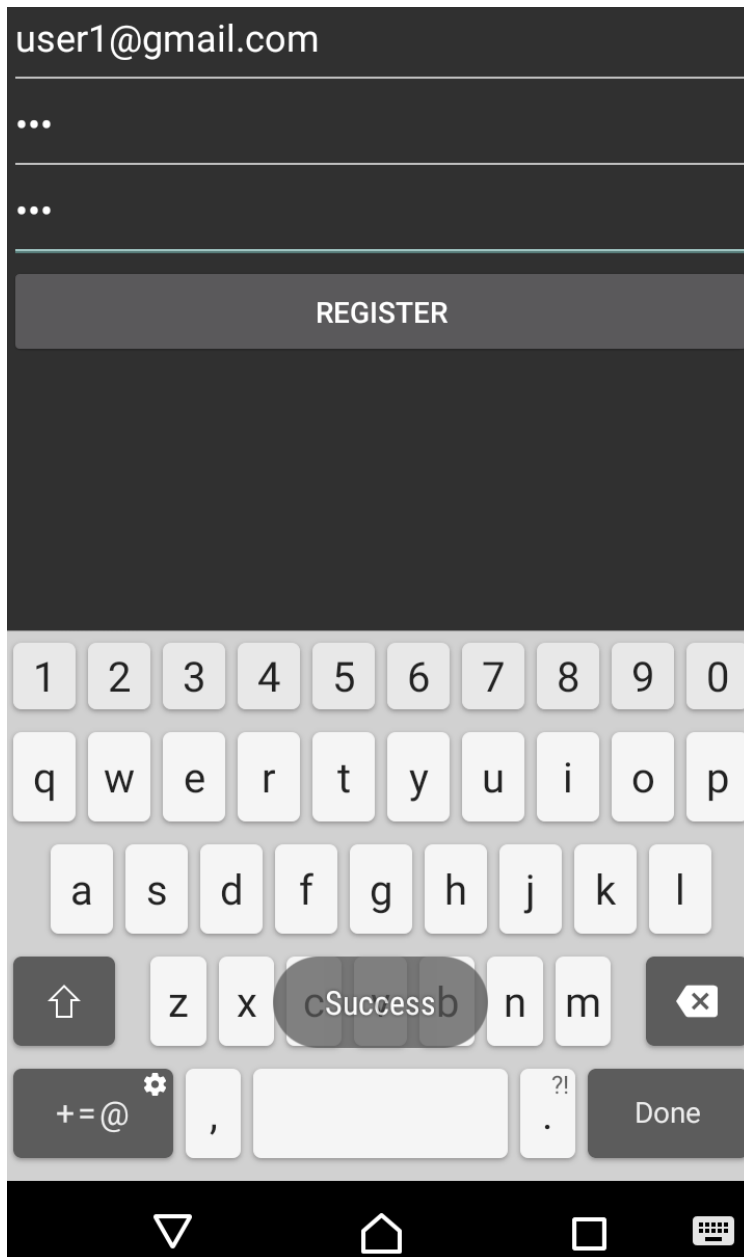
```
public static async Task<bool> MaskAsProcessed(string pendId)
{
    try
    {
        var pends = (await AzureHelper.MobileService.GetTable<PendingRequest>().Where(d => d.Id == pendId).ToListAsync()).FirstOrDefault();
        pends.Status = 2;
        await AzureHelper.MobileService.GetTable<PendingRequest>().UpdateAsync(pends);
        return true;
    }
    catch (Exception ex)
    {
        return false;
    }
}
```

**Figure 36:** Mark as Processed method

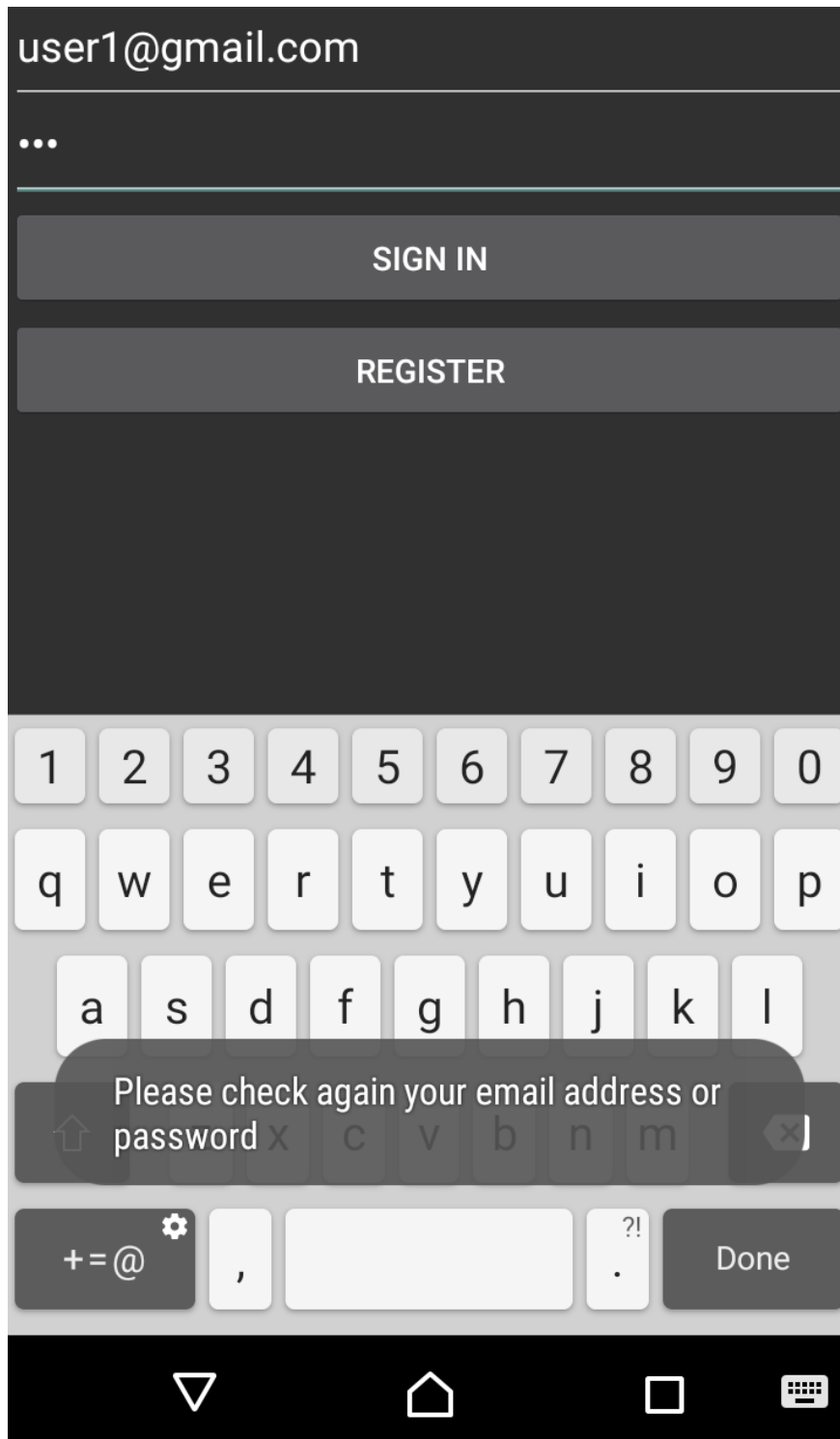
## 6 TESTING

### 6.1 Signing up and log in for new Traver User

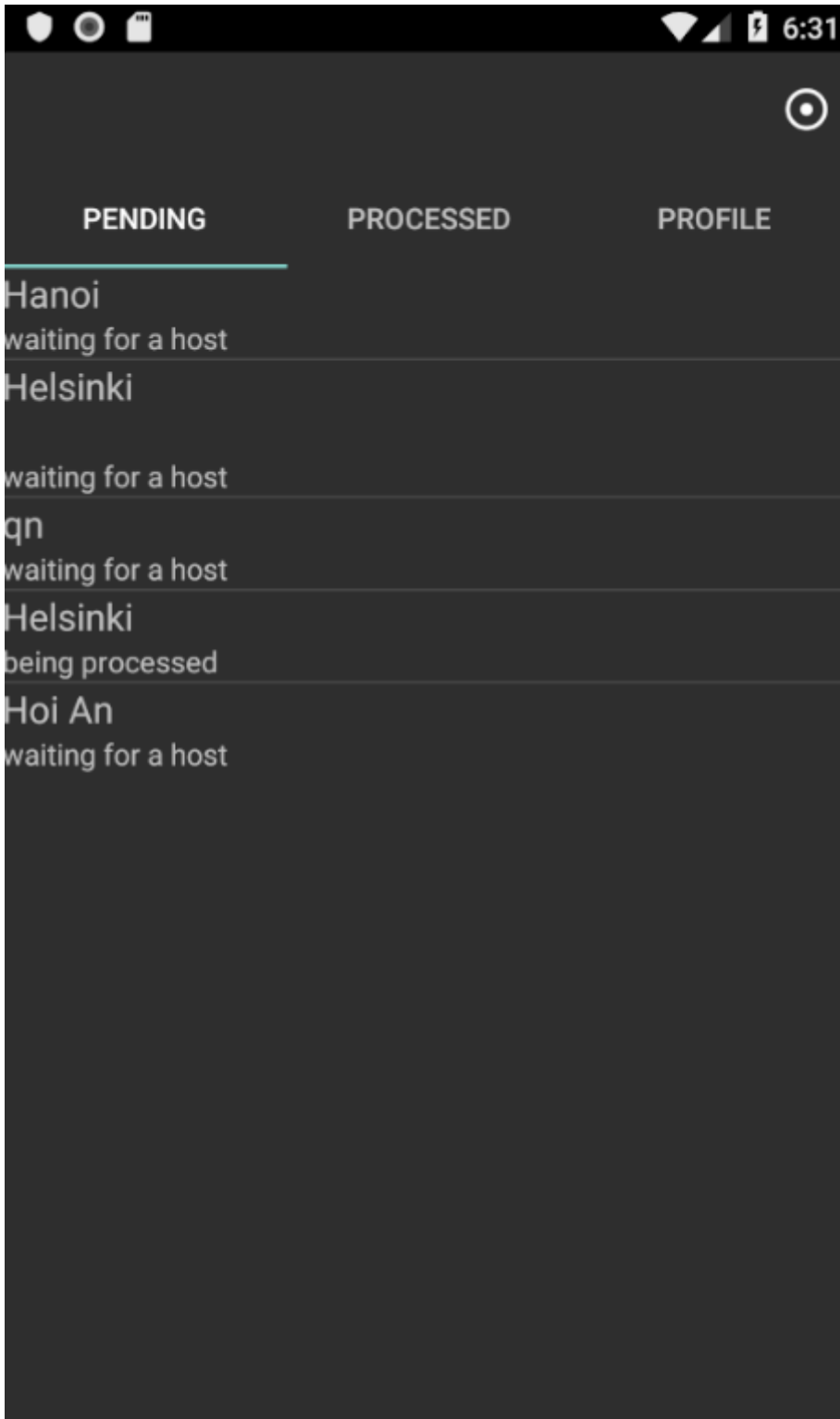
- Testing step:  
Open the application, click in the register, fill in all the fields. Click Register button. After that, use the register information to log in.
- Expected result:  
If the email and password matched the details that user use to register, user will be redirect to main tab. Otherwise, there will be a toast error.
- Actual result



**Figure 37:** Successfully register



**Figure 38:** Error log in



**Figure 39:** Successfully log in main tab

## **6.2 Making new request for Travel User Group**

- **Testing step**  
Click on the add button in the top right corner, fill all the fields for the new request, click add. Then go back and check the Pending tab
- **Expected result**  
After clicking the button, a new page is shown, which allows the user to fill in details of the trip. After clicking the add button and go back, there is a new request in the Pending tab.
- **Actual result**

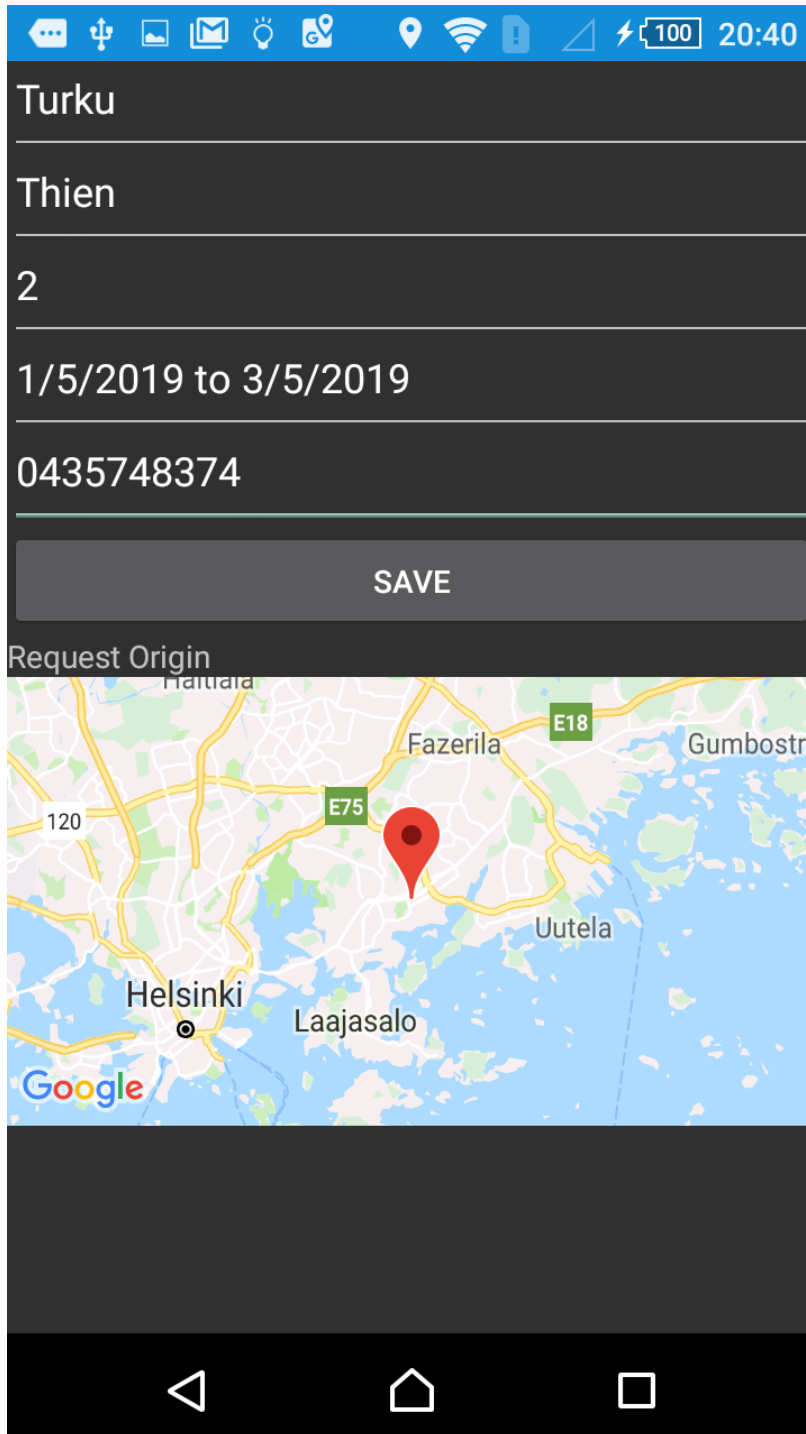
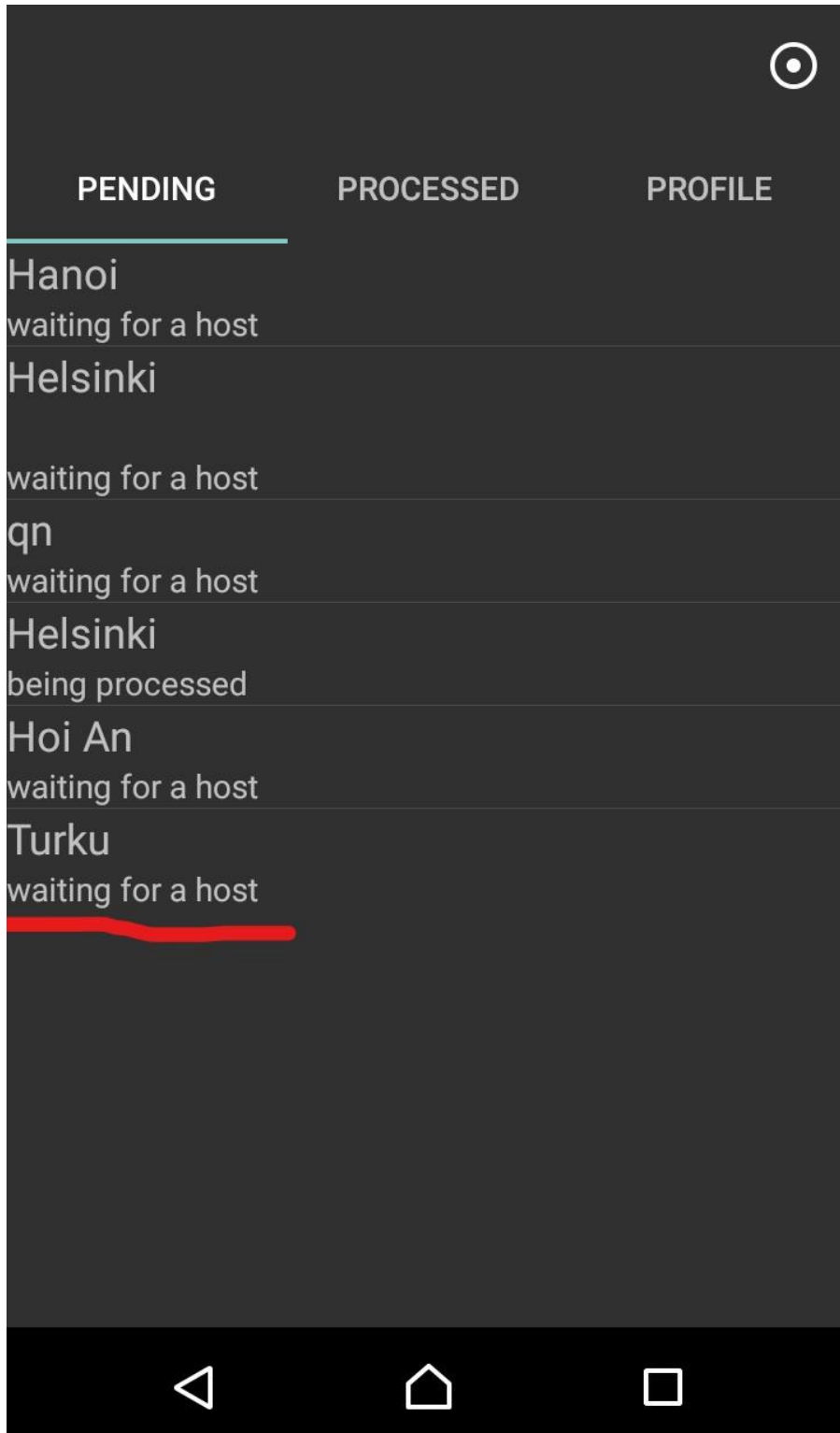


Figure 40: Add new Request

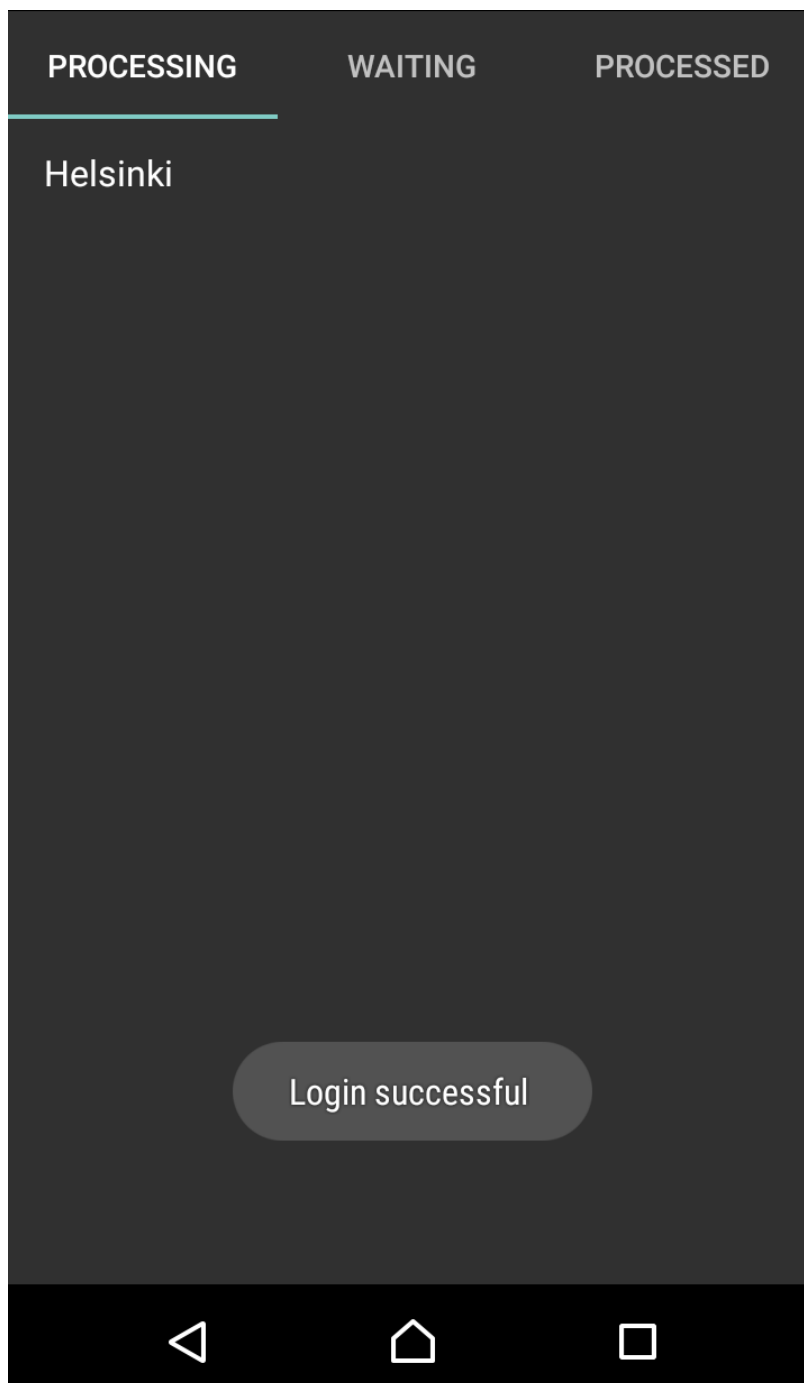




**Figure 41.** New request shown in Pending tab

### **6.3 Sign up and log in for Guide Person Group**

- Testing step
- Open the application, click in the register, fill in all the fields. Click Register button. After that, use the register information to log in.
- Expected result
- If the email and password matched the details that the user used to register, the user will be redirected to the main tab. Otherwise, there will be a toast error.
- Actual result



**Figure 42.** Guide User log in successfully

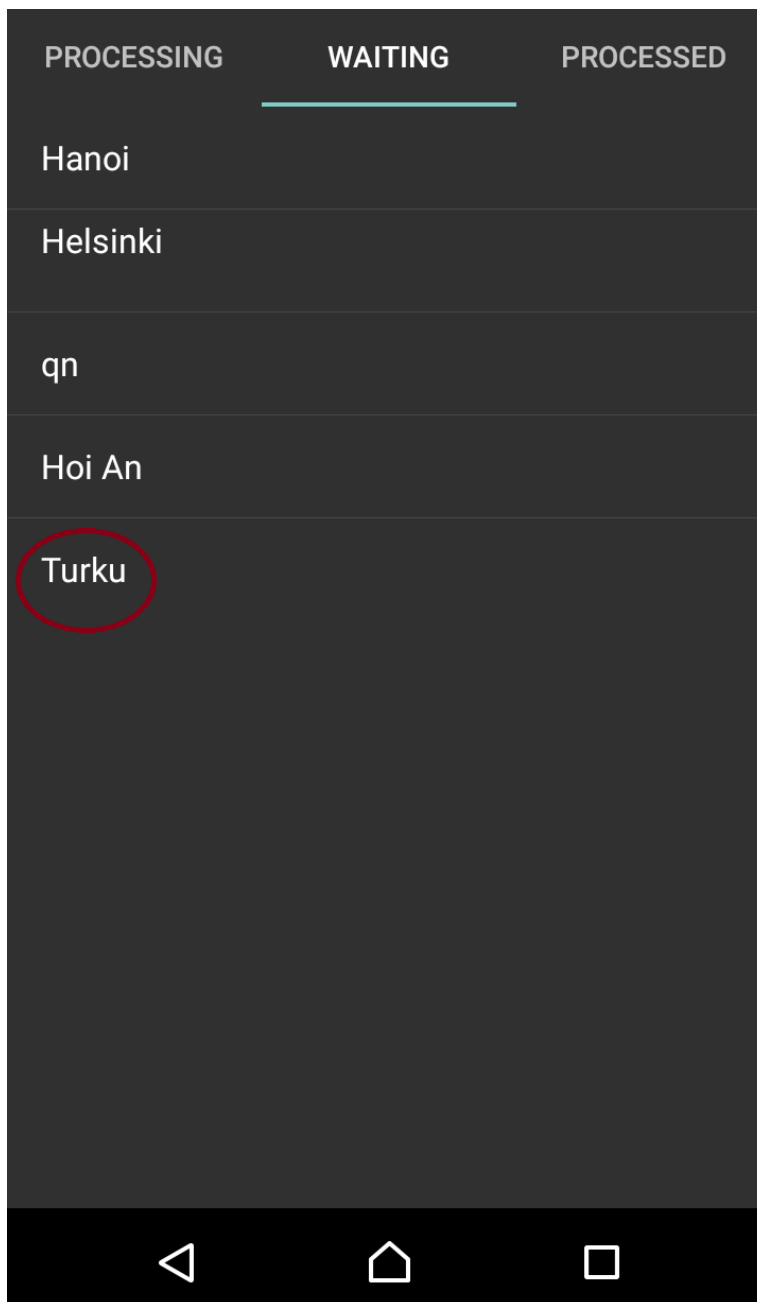
#### **6.4 Managing and accepting requests**

- Testing step

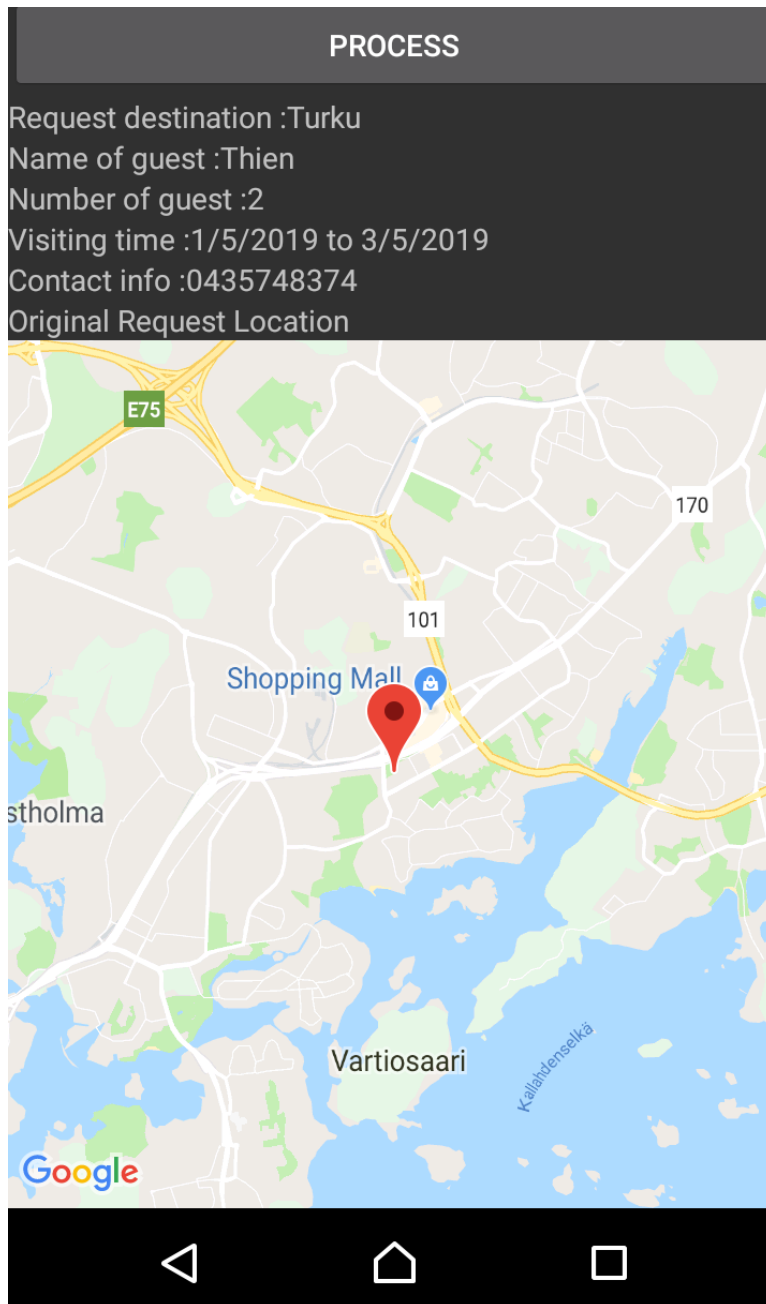
Click on the item in the Waiting list, review one, then click on the Process button to move it to the Processing tab and let the request person know that the re-

quest is being considered and will be contacted soon. If everything is ok, click on the Accept button to mark as the request has been accepted.

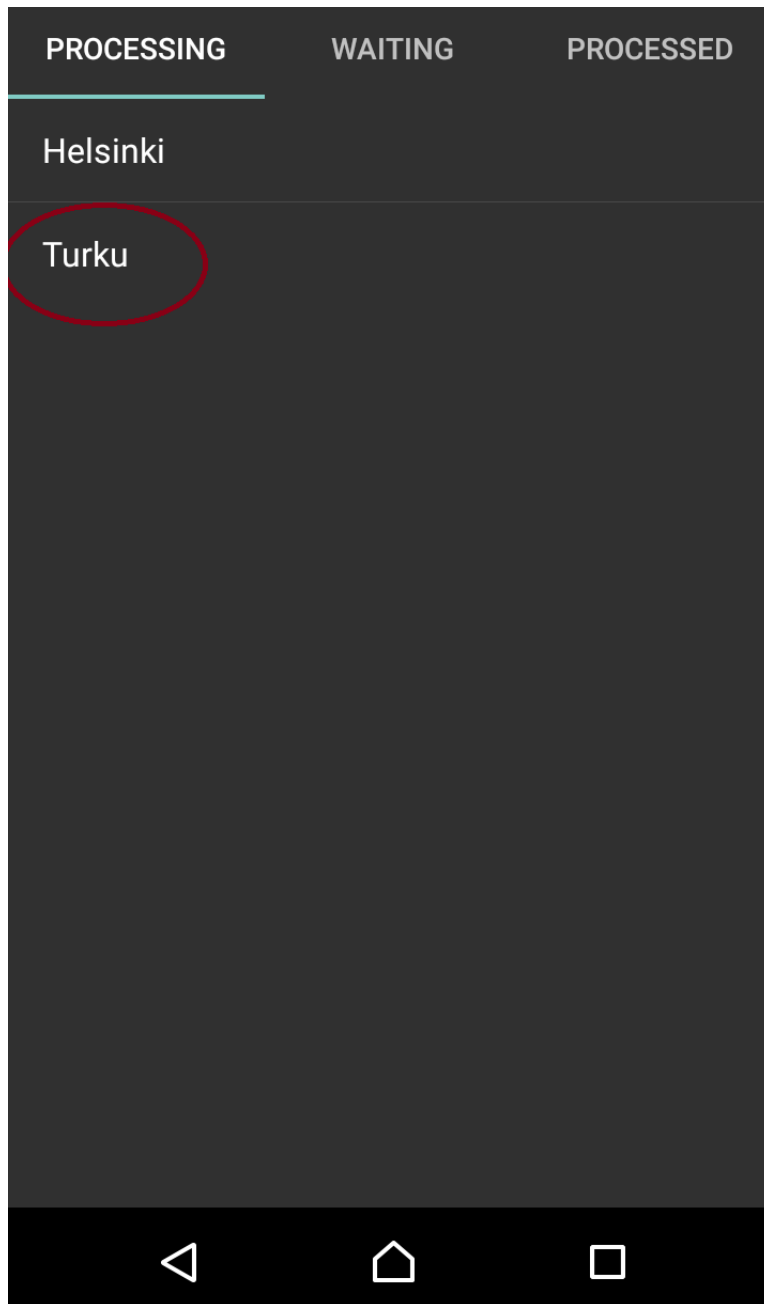
- Expected result  
Everything should work as the testing step. Both user from two group can see the change in the request.
- Actual result



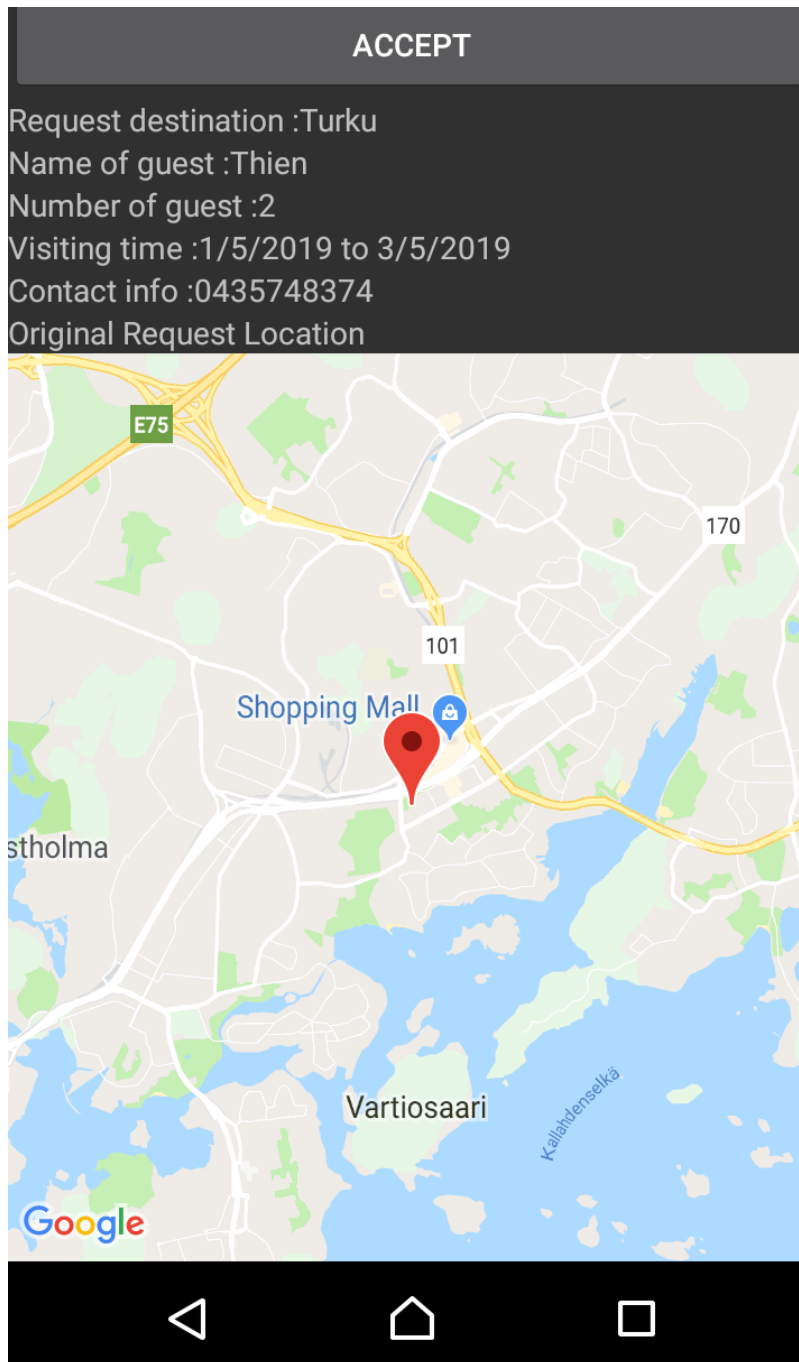
**Figure 43.** New request in Waiting tab



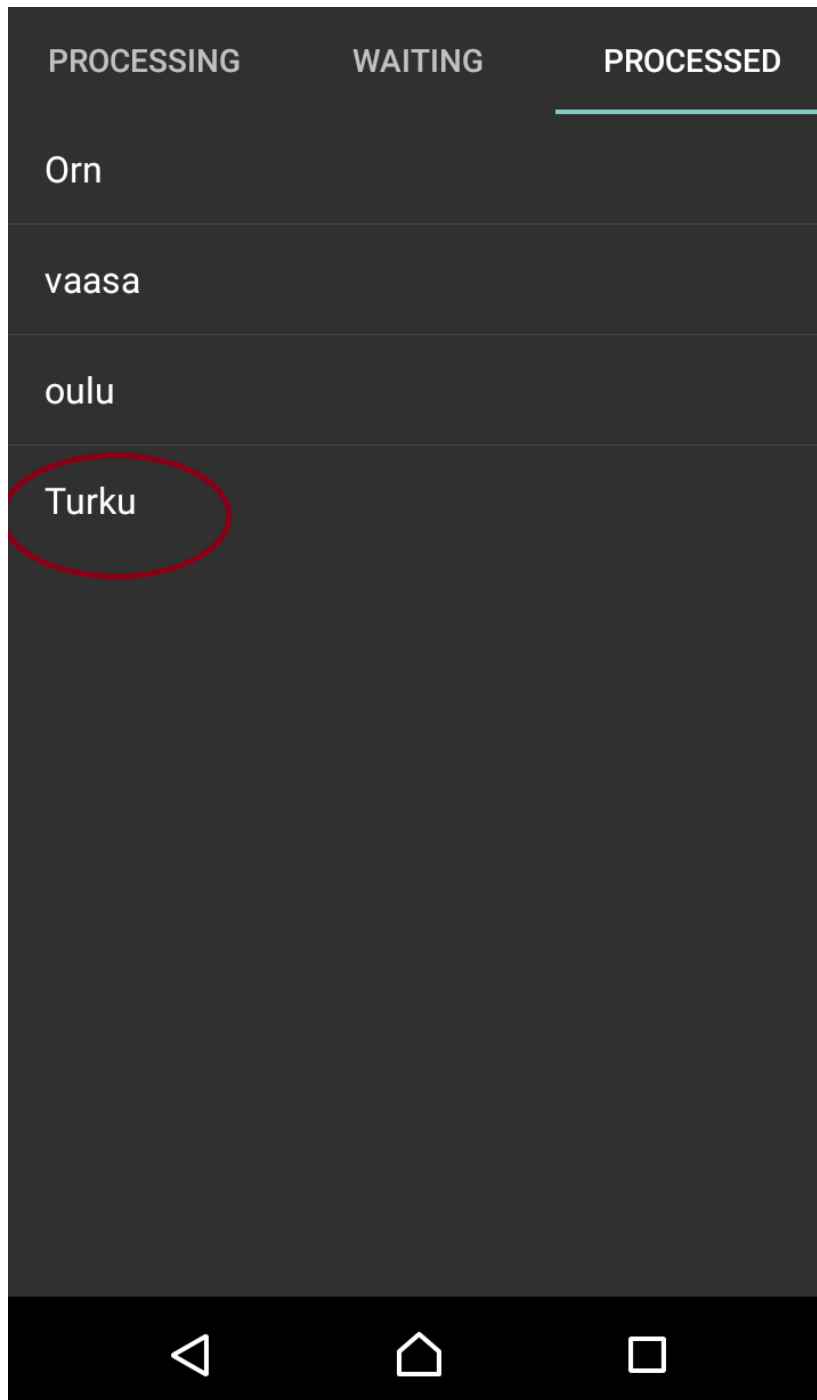
**Figure 44.** View and process request



**Figure 45.** Request is in Processing tab



**Figure 46.** View and Accept request



**Figure 47.** Request is in Processed Tab



## 7 CONCLUSION

The purpose of the work is to create an application, in order to allow travelers and local people meet each other, communicate and learn about new cultures and places. The core function is to let two groups communicate with each other, which is basically completed. However, the application will need a huge improvement with a team of deep knowledgeable people and a long time to fix all the flaws.

A Travel user can create a new request, in order to let guide users to know that there are people who need a guide in their place. On the other hand, Guide users can manage their work, and know that a lot of tourists need their help in order to get to know their places, as well as earn more income by negotiating with the travel users about their incoming trip.

Even though the application seems to be simple, I met a huge challenge since this is the first time that I use a new technology that I do not know from before and build it from 0. The time to learn and read about all necessary technologies is long. But the most difficult part is to start coding and implementing. There are too many errors which are new, and it takes a bit of time in order to wait for the developers of Xamarin find a solution. Also, this technology is not popular, as Android developers use Java mostly while IOS developers use Swift, that makes finding the material on the internet to learn about this technology more difficult. The version of required NuGet packages is not available with the new Visual Studio version as well, so I need to use the old version of the NuGet package, which create a lot of warning and conflicts. Luckily, it does not affect the application much.

### **Future Works:**

The user interface will need a lot of improves, as users nowadays really focus on how the application look like. User should register with more information for the authorization reason, as well as be able to register using social media account. The profile tab should hold the information of the user and contains a method where the user can modify information of himself. The application should work with picture as well. One of the

biggest challenges will be creating chat between travel user can guide user, in order to allow two group to talk with each other on the application.

**REFERENCES:**

/1/ Global market share held by the leading smartphone operating systems in sales to end users from 1st quarter 2009 to 2nd quarter 2018. Accessed 08.04.2019.

<https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/>

/2/ Why Is C# Among The Most Popular Programming Languages in The World? Accessed 08.04.2019.

<https://medium.com/sololearn/why-is-c-among-the-most-popular-programming-languages-in-the-world-ccf26824ffcb>

/3/ What is Xamarin? How does it help in cross-platform mobile app development? Accessed 08.04.2019.

<https://www.thewindowsclub.com/what-is-xamarin-and-cross-platform-mobile-development>

/4/ PYPL PopularitY of Programming Language. Accessed 08.04.2019.

<http://pypl.github.io/PYPL.html>

/5/ C Sharp (programming language). Accessed 08.04.2019.

[https://en.wikipedia.org/wiki/C\\_Sharp\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/C_Sharp_(programming_language))

/6/ Top IDE index. Accessed 08.04.2019.

<https://pypl.github.io/IDE.html>

/7/ What is Azure? – An Introduction to Microsoft Azure Cloud. Accessed 08.04.2019.

<https://www.edureka.co/blog/what-is-azure/>

/8/ Xamarin Google Play Services – Maps. Accessed 08.04.2019.

<https://www.nuget.org/packages/Xamarin.GooglePlayServices.Maps>

/9/ Support Library Features Guide. Accessed 08.04.2019.

<https://developer.android.com/topic/libraries/support-library/features.html>

/10/ Azure Mobile Client SDK. Accessed 08.04.2019.

<https://www.nuget.org/packages/Microsoft.Azure.Mobile.Client/>