National College of Ireland

Software Project (BSHCSDE)

Software Development

Academic Year 2020/2021

Florence Migliorini dos Santos

18138896

X18138896@student.ncirl.ie

Crossing Borders

Technical Report

Contents

Executive Summary

The purpose of this document is to provide an overview of the technical details, explaining in detail the functionality of the application. As the project is in the middle of development, it is not possible to explain in detail how the implementation will be done.

The key focus for the preparation of this documentation is in the requirements part, which I tried to explain in detail.

1.0 Introduction

1.1. Background

I wanted to create an app to connect all forms of public transport in one simple and easy to use app in the hands of the user. This app would eliminate the need for extra cards or tickets as payment, would be completed digitally (Google Pay, Stripe) and a QR code would be used as proof of purchase.

Transport is something that I'm very interested in but sometimes when travelling in other EU cities I find it difficult to know what public transport to use and what area each different ticket covers. This is when I came up with an idea for an all-in-one public transport app that can be used across multiple cities in the EU. This would prevent me from being unsure when travelling in other cities and would also be a very useful tool for me to use everyday getting to work or college. Going all digital and avoiding unnecessary cards and tickets is something I've always wanted for public transport.

I wanted to create a mobile app because of its usefulness in everyday scenarios. Our phones are almost always next to us at any given time, making them a great tool to develop for. Mobile app development is also completely new to me which will provide a great challenge for me and I'm looking forward to seeing how my mobile app development skills develop during the project. I think this project offers me a great opportunity to do extra research and complete tasks that I would otherwise not do as part of my module.

1.2. Aims

Crossing Borders is a mobile application that aims to streamline the user's experience when taking public transport. The app's goal is to help facilitate a quick and comprehensive way to travel. The modes of transport I will be targeting for this app are the Luas, Dart, and Bus.

1. The main objective of this project is to develop a mobile application that can facilitate day-to-day travel in an intuitive way. This is true domestically as well as for international travel. The app will work in the same way when in another supported city in the European Union. This means there is no extra stress when taking transport in an unfamiliar part of Europe.

2. The app will make daily traveling more convenient for users by helping them save time for their routine commutes to work or school. Every step of the user's daily commute is handled through the app. The user can purchase tickets for their preferred method of travel completely digitally using either Google Pay or Stripe. This means the user does not have to carry around any extra cards or a physical ticket to travel. Instead, the user will receive a QR code that lasts for the duration of their trip.

3. Tourism is a large industry across the EU and this app is aiming to make transport as easy as possible for tourists. The route planner calculates multiple ways for the user to get from

their starting point to a designated destination. This will allow tourists to easily figure out what public transport to take, how long the journey will last and what the cost of the trip will be. Purchasing of tickets takes place on the app so there is no need to deal with a convoluted ticket machine UI.

1.3. Technology

The Crossing Borders application is a mobile application that works on Android devices, as it is an open-source language. The app is developed using the android studio IDE with the Java development language. This tool also includes an SDK (Software Development Kit) that helps in the development of certain applications, since it has several packages such as Google APIs.

SDK (Software Development Kit) is a set of tools and libraries provided by a vendor that are required to create software on a specific platform, these tools and libraries that make up the SDK allows for interaction between other software within the platform. These tools include compilation, debugging and other utilities that are normally presented in an integrated development environment (IDE). Many SDKs include code samples, technical support notes, APIs, as well as other documentation to help inform the user about the reference material. SDKs can give access to one or many APIs.

API (Application programming Interface) allows for communication between two systems, where the base system extends the API to provide a set of services to the application. There are different kinds of API architectures, for this project I will be focusing on RESTful applications. A REST API has a Client-Server architecture. The client and server communicate with each other via HTTP (Transfer Protocol) using URIs (Unique Resource Identifiers), CRUD (Create, Read, Update, Delete) and JSON (JavaScript Object Notation) conventions. REST API's store the server responses within the client to reduce redundant server requests, which contributes to a faster and more efficient user experience. APIs are very useful as they're independent from the application that is extending them, this means that APIs can be re-used by a number of applications by using RESTful HTTP actions (mobile app and web app can use the same API).

Vysor an app that lets you display a smartphone screen on a PC in a practical and efficient way.

Firebase is a cloud computing service that allows developers to connect mobile and web applications to cloud services via APIs and SDKs.

Firebase offers a variety of high-quality documentation, including examples, tutorials, and supporting documentation for all of its products. For this application I am using firebases Authentication service.

Postman is to make HTTP requests, as it provides a simple and intuitive interface that facilitates testing and debugging of REST services, as well as the ability to make HTTP requests such as GET, POST, PUT, and DELETE. Furthermore, it can test an API's performance by issuing a series of requests to verify its response performance. Postman allows you to examine the contents of both the request and response to verify that the API call is working as expected. It is a very useful tool for checking HTTP messages, headers and the body of the response to aid in both testing and troubleshooting processes.

SQLite database to ensure that data is persisted for users to access when they use the app.

GitHub It is the most popular platform for hosting and managing projects. I use this platform regularly to upload code from the projects I work on, this version control software allows me to have control over the changes made and safely revert back to previous versions in case of any major issues. (the link)

Dependencies

To develop the entire implementation of the system, it is necessary to use a series of libraries that will help in the development of the system.

Firebase serve as a backend for the app.

Firebase Authentication:

For this application it was important to recognize the user's identity to allow them to have a personal experience with the app. Firebase functionality allows users to register by providing some information that is stored. Figure XX shows the methods used in this project, which are Email/Password and Google. All instructions were followed from google documentation and links already provided in that document. Firebase authentication is a great choice for security reasons too as firebase itself handles the encryption user's passwords using an internal hashing algorithm. This removes the need for a custom hashing/salting method that may have vulnerabilities.

Firebase BoM-> manage the versions.

Zxing -> generate QR code. (link)

GSON -> Convert JSON in JAVA and vice versa.

Volley -> consume API.

Android SDK to run devices and emulators.

Google Mobile Services (GMS) -> APIs that help functionality across the device.

Google API

Since this application needs to display maps and locations on maps, it is necessary to use the Google Maps API. This API allows you to place maps in the application interface. In addition, it allows you to obtain the location of the mobile device and add points on the map, moving along the longitude and latitude. This API is free, but it only allows you to make 2,500 requests per day. If you want to make additional requests every day, you must upgrade to Google Maps for Business API. The use of Google Maps and SDKs requires an API Key, for this application I am using Directions API, GeoLocation API, and GeoCoding API.

Additionally, the Google Sign-In API works by generating a JWT token with the Google Profile data and metadata attached that is validated using a backend Firebase authenticator.

Direction API consists of a service that calculates directions between locations using the available means of transport. This service also allows you to specify a set of waypoints to calculate routes through additional locations, in which case the returned route includes stopovers at each of the provided waypoints. Finally, this API also makes it possible to predict the traffic of the returned path based on averages previously registered.

GeoLocation API consist of a service that can provide the location of the user if they choose to allow it. The user is asked whether they want to provide location information for privacy reasons.

GeoCoding API Provides a service which can translate a certain address into geographic coordinates (latitude and longitude) which are calculated using mathematical equations and the location is then represented by a symbol (or marker) on the map.

Stripe + Google Pay will be offered as an alternative method for payment. Stripe works by creating a source object that stores the user's payment information. Next, Stripe checks if any further action is required such as payment verification from the user's payment source. Once the payment source is

ready to be used a charge request is completed in the backend using the source object information. The *GooglePayLauncher* is used to accept payment which is integrated with Stripe. The API key is a one-of-a-kind identifier that is used to authenticate associated project requests for usage and billing. All the steps and procedures were followed from the documentation provided at this link.

```
dependencies {

    implementation 'androidx.appcompat:appcompat:1.4.0'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.2'
    // Import the BoM for the Firebase platform
    implementation platform('com.google.firebase:firebase-bom:29.3.0')
    implementation 'com.google.firebase:firebase-auth'
    implementation 'com.google.firebase:firebase-firestore'
    implementation 'com.google.android.gms:play-services-maps:18.0.2'
    implementation 'com.google.android.gms:play-services-auth:20.1.0'
    implementation 'com.google.code.gson:gson:2.9.0'
    implementation 'com.android.volley:volley:1.2.1'
    //Zxing
    implementation 'com.google.zxing:core:3.2.1'
    implementation 'com.journeyapps:zxing-android-embedded:4.3.0'
    //Stripe
    implementation 'com.stripe:stripe-android:20.0.1'
    //Stripe-server
    implementation "com.stripe:stripe-java:20.114.0"
    implementation 'com.google.android.gms:play-services-location:19.0.1'
    implementation 'com.google.android.gms:play-services-vision:20.1.2'
    implementation 'com.google.android.gms:play-services-vision-common:19.1.2'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
```

Figure 01: Dependencies.

1.4. Structure

1.4.1 Introduction

Background – In this section I explain my reasoning behind choosing my project idea and how that idea came to be. I also discuss why I decided to use the technologies that I did and how it'll benefit the project. Finally, I discussed why I think this app is important and how it will challenge me.

Aims – *This section addresses the overall aims and objectives of the project and represents what I would like to have accomplished by the end of the development cycle.*

Technology – Here, I discussed the technologies that I used to build the application.

Requirements – This section encompasses all the different requirements/functionality of the proposed project. This includes all the functional and non-functional requirements, which have been split into different headings to address each one individually. Use cases have been designed to illustrate and further explain each requirement and the flow of operations.

Design & Architecture – Here, I list all the screens and display them all in a structure chart to highlight the apps design and how to navigate between the different screens. I have also included a system architecture diagram to show the relationship between the app, database, and required components such as API's.

Implementation – This is where I took some snippets of my core functionalities and elaborated on how they were developed.

Graphical User Interface – I provided screenshots of my GUI and explained the process of how the screens were designed and created using Adobe Xd.

Testing – In this topic, all types of tests performed during the development of the application are documented. Along with an evaluation of the results based on the evidence presented using images and supported documents.

2.0 System

2.1. Requirements

In this section, the requirements analysis will be performed to determine the needs to be satisfied for the application creation. First, the functional requirements that describe the functionality that the system must provide will be discussed. Second, the non-functional requirements that define the emerging properties of the system, such as response time, reliability, etc., will be detailed.

2.1.1. Functional Requirements

This section defines the functionality of the system. In each of the features will be detailed what to do, the criteria for customer satisfaction. The system features are as follows:

Functional Requirement No.	Function Requirement Description
FR1	Actor should be able to Sign In throw a Google account, that gets the auth token and user data
FR2	Actor should be able to Login with Email/Password registered.
FR3	Sign Up, Actor shall have the ability to create an account using email and password.
FR4	Actor shall have the ability to search for a journey using a starting point and a destination.
FR5	Actor shall have the ability to choose how to pay for the journey.
FR6	Actor shall have the ability to add a daily journey as favourites.

2.1.1.1. Use Case Diagram

The use-case diagram, which represents the operations the user can perform on the system.

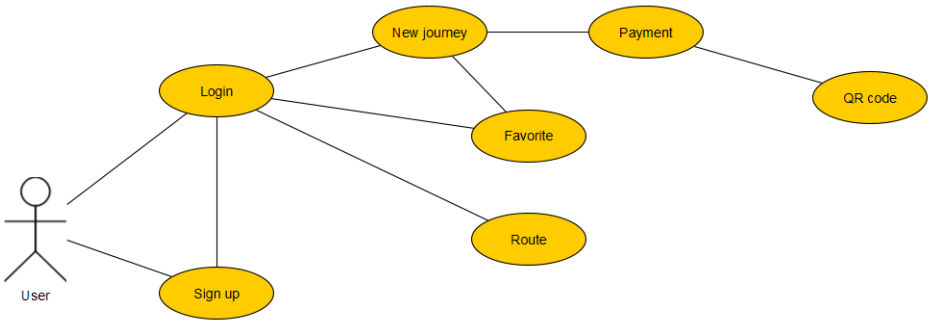


Figure 02: Use Case

2.1.1.2. Requirement 1: Google Sign-In

2.1.1.3. Description & Priority

The Google Sign-In API is being used to identify users and assign them a JWT access token that can be used to access the app's features while the token is still active. After a JWT token has expired the token will need to be refreshed to grant access to the app again. Initially, when a user signs in a GET request is made to retrieve the JWT token and then the ID token must be sent to the firebase backend using a POST request to validate the integrity of the token. If a token is valid a HTTP 200 successful response will be returned, and the user will be redirected to the home screen where they can access the main menu and create a new journey. The JWT token will be stored in local storage and will contain important meta data for the token's validation as well as profile information for the user. This profile information can be used to retrieve the user's name, email, and Google profile picture.

This requirement is a CSF (critical success factor) for the project and acts as a gateway to allow users access to the core functionality of the app. This requirement provides the app with secure authentication using the Google API Sign-In, ensuring that user accounts are secure and validated.

2.1.1.4. Use Case

Scope

The scope of this use case is to verify the actor's google account using a backend firebase server to authenticate the actor's ID token and credentials obtained from the Google Sign-In API.

Technical issues

I have initial worries about implementing the API with the firebase backend to enable the validation of the JWT tokens. I have read the documentation of the Google Sign-In API but I'm still unclear about some of the concepts around the token validation such as the token expiry and refresh and how they will be reflected back to the user.

Dependencies with other requirements

This use case is critical as it's a requirement for all other use cases in the application. Before the actor can access the app, they must be issued a valid and authenticated JWT

token. While a JWT token is active the user will be automatically signed in allowing them access to the other services.

Use Case Diagram

Figure 03: Login use case

Precondition

The actor must have a pre-existing google account to sign in and access the apps services.

Activation

The actor clicks on the google sign in button and selects their google account.

Main flow

1. The actor opens the app
2. The actor clicks on the Google Sign-In Link
3. The actor selects their google account
4. The system retrieves the JWT token using a GET request
5. The system sends the ID token to the firebase backend using a POST request
6. The system validates the ID token using the tokens metadata to ensure the integrity of the token
7. The system sends back a HTTP 200 successful request

8. The actor is redirected to the home screen

Alternate flow

A1: Active JWT token found

1. The actor opens the app
2. The JWT token is found in local storage
3. The use case continues at step 5 of the main flow

Exceptional flow

E1: User does not have a google account

1. The actor opens
2. The actor clicks on the Google Sign-In Link
3. The actor does not have a google account

4. The system prompts the actor to create a Google account before continuing

Termination

The system redirects the actor to the home screen.

Post condition

JWT token is stored in local storage to be retrieved by the system the next time the actor opens the app (as long as the token is still valid).

The actor's Google profile information is stored as JSON data in the JWT token and can be retrieved and displayed by the system to the actor.

The actor must choose what service they would like to choose next for example, view favourites or start a new journey.

Use Case – Requirement 2: User Login

Scope

To allow the actor to login to their account using an email and password combination.

Description & Priority

This use case allows people to create an account without attaching their private Gmail account. Some users may prefer to login using a different email and this gives them the option to do that. To start the user must click the login button and enter their email and password that correspond with their account. Next, the firebase authentication checks that the data entered by the user matches the email and encrypted password in the firebase Realtime database. If successful the system shows a message notifying the user that their login attempt was successful and the user is redirected to the main screen and can now view specific content related to their account such as: favourites, history and can

search for and purchase tickets. If the login attempt is unsuccessful an error message will be displayed, and the user will be prompted to type in their login details again.

Dependencies with other requirements

In order to login to an account, first of all the user must have created an account with the same login details so that the firebase authenticate can confirm the user.

Use Case Diagram

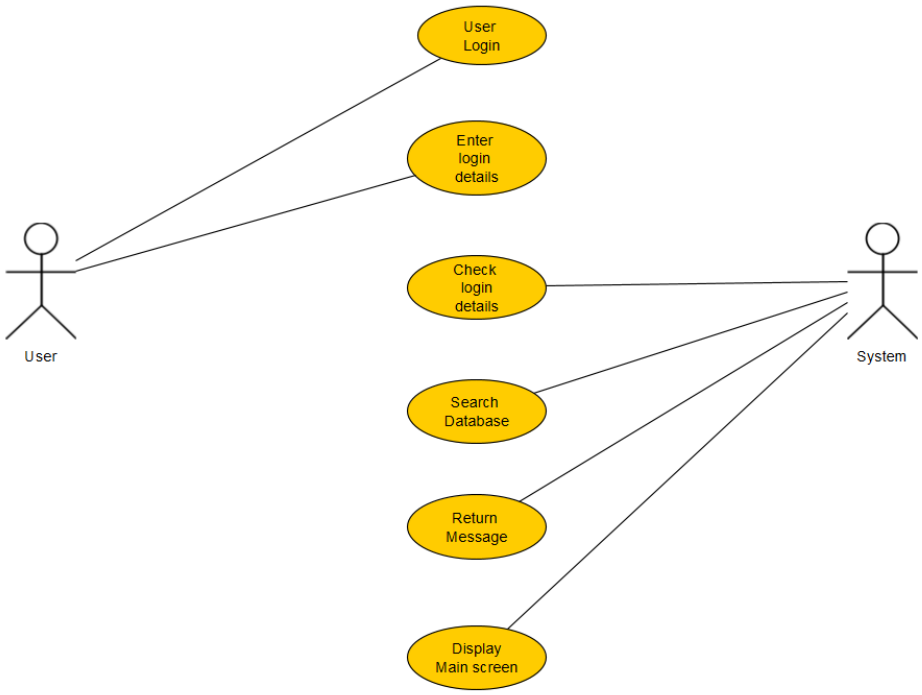


Figure 04: User Login

Precondition

The user must have created an account using the sign up link.

Activation

The user clicks on the login button on the initial screen.

Main flow

1. The user opens the app and clicks on the login button
2. The user enters their email and password and clicks the button to enter their details
3. The system checks that the login details match in the Realtime database
4. System confirms the details entered by the user
5. The system returns a message to the user stating that the login was successful
6. The System redirects the user to the home screen

Exceptional flow

1. The user opens the app and clicks on the login button
2. The user enters their email and password and clicks the button to enter their details
3. The system checks that the login details match in the Realtime database
4. The system could not find the login details in the firebase Realtime database
5. The system prompts the user to re-enter their login details.

Termination

The user is redirected to the home screen and have successfully been logged into their account.

Post condition

The user can now view specific content related to their account such as: favourites, history and can search for and purchase tickets.

Use Case – Requirement 3: User Sign up

Scope

The scope of this use case is to allow the user to register and create an account.

Description & Priority

The user has the option to register and creating an account using an email and password if they do not have a Google account or do not want to use it with the app. To register the data with the firebase authentication the actor must fill in the provided email and password fields. The button press starts an event that uses the data the actor entered into the fields and inserts that data into the firebase Realtime database. The benefit of using the firebase Realtime database to authenticate users is that the passwords are encrypted automatically meaning that the user's data is stored securely and not at risk.

Dependencies with other requirements

There are no prior dependencies for this functionality as this is a starting point for the actor, the only requirement is that the user has not signed up previously using the same email.

Use Case Diagram

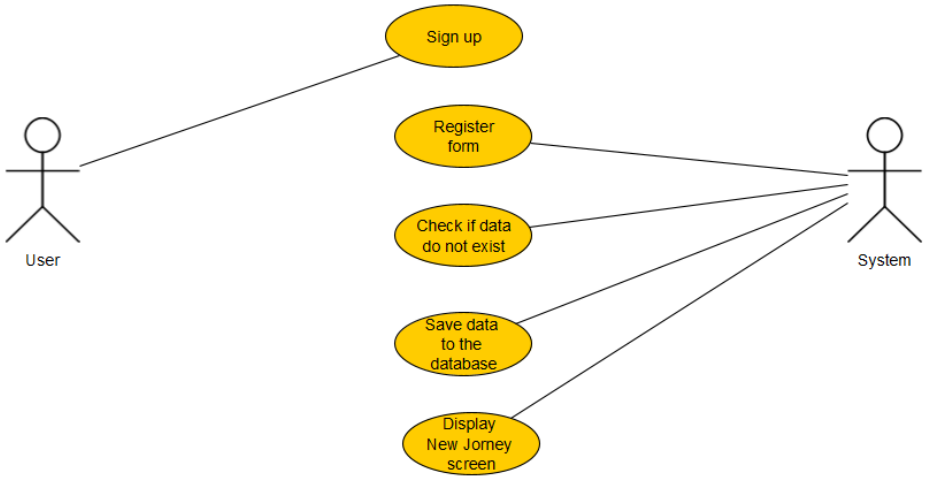


Figure 05: User Sign in

Precondition

The actor cannot have an account registered.

Activation

The actor clicks on the sign-up button and fulfil the form.

Main flow

1. The actor opens the app
2. The actor clicks on the Sign-Up Link
3. The actor fill the fields.
4. The system check on the database if the data does not exist.
5. The system shows a message saying …. Successfully.
6. The system validates the ID token using the tokens metadata to ensure the integrity of the token
7. The system sends back a HTTP 200 successful request
8. The actor is redirected to the home screen

Exceptional flow

E1: Actor uses an email from a previous account to sign up.

1. The actor opens the appp
2. The actor clicks on Sign Up button
3. The actor user an email that has already been used on a previous account

4. The system prompts the actor to use a different email when filling out the sign-up form.

Termination

The system redirects the actor to the home screen.

Post condition

Use Case – Requirement 4: Get Transport Journey

Scope

This use case will allow the user to search for available public transport for a journey based on a start and endpoint.

Description & Priority

Once the actor has logged in the app will get the actors geolocation (if permissions are enabled), the geolocation will then be used as the actor's current location. The actor can choose to keep this starting location or replace it with another location of their choice by entering the address. The actor begins by entering a starting and ending location for the journey and then clicking a button to search for possible routes. The button press starts an event that retrieves the data from the Google Directions API including the available transport for train, bus, and tram, and the time it takes to complete the journey. I have created a custom algorithm to determine the price of the trip by charging €0.20 per kilometre.

Technical issues

Dependencies with other requirements

Firstly, the actor must have signed in using their google account to access this feature. This use case is required in order to develop the favourites and payments use cases.

Use Case Diagram

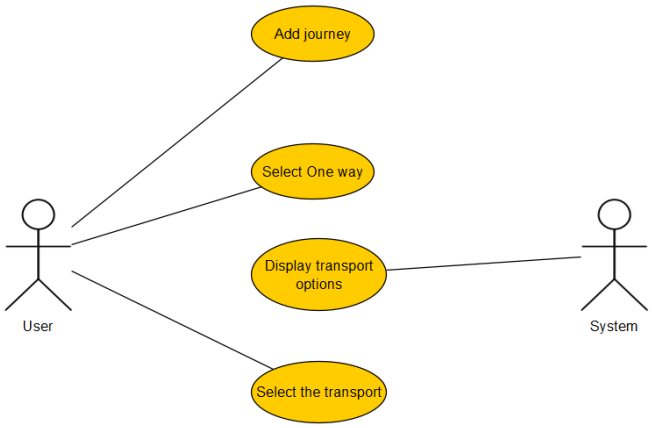


Figure 06: Get transport Journey

Precondition

The actor must have a starting location and destination in mind for their journey

Activation

The actor searches for a new journey using a start location and destination.

Main flow

1. The actor enters in a start location and destination for the journey
2. The actor selects the ticket options they need, for this case the actor will select a single ticket for one passenger.
3. The actor searches for the journey
4. The system makes a request to the Google Directions API using the starting point and destination
5. The system checks to see if the starting point and destination has a transportation method available
6. The ticket price is calculated based on the parameters entered by the user in step 2 as well as the distance of the journey.
7. The system returns the transportation method
8. The actor selects a transportation method returned by the system

Alternate flow

A1: Return Ticket

1. The actor enters in a start location and destination for the journey

2. The actor selects the ticket options they need, for this case the actor will select a return ticket for one passenger.

3. The use case continues at step 3 of the main flow

A2: Multiple passengers

1. The actor enters in a start location and destination for the journey

2. The actor selects the ticket options they need, for this case the actor will select a return ticket for one passenger.

3. The use case continues at step 3 of the main flow

Exceptional flow

E1: User does not have a google account

1. The actor enters in a start location and destination for the journey

2. The actor searches for the journey

3. The system makes a request to the Google Directions API using the starting point and destination, but no public transport options are available for this trip

4. The system outputs an error message to the user

5. The system prompts the actor to enter another journey

6. The actor begins the process again

Termination

The system will present a list of possible transport routes that the actor can take for their journey, the routes will be listed in order of shortest journey duration.

Post condition

The actor chooses one of these routes based on a number of factors such as time, cost, and convenience.

Once the actor has chosen their preferred route, they can choose to add the route to their favourites or simply continue to payment.

Use Case – Requirement 5: Payment

Scope

Allows the actor to pay for a journey including all methods of transportation in one transaction. As a proof of purchase the actor will receive a QR code that will expire after a certain amount of time.

Description & Priority

The user selects their journey and selects the get ticket button which redirects the actor to the payment screen. The system calculates the cost of the trip using an algorithm, taking into account the duration and transportation methods of the journey. The price, starting point and destination are all included on the ticket. If the ticket is acceptable for the actor, they can choose to pay using either Stripe or Google Pay. Google Pay is used as an option for payment in the app and works similarly to the Google Sign-In API. Google Pay sends a payment token with details about the purchase to the app which is then processed by the backend and a payment token is sent to the payment service provider. Stripe will be offered as an alternative method for payment. Stripe works by creating a source object that stores the user's payment information. Next Stripe checks if any further action is required such as payment verification from the user's payment source. Once the payment source is ready to be used a charge request is completed in the backend using the source object information. After payment is confirmed the system sends a QR code to the user as proof of purchase for the journey. Finally, the trip is saved to the users history to act as a receipt or proof of purchase for the trip, this way users can view all their past trips if they desire.

This use case is a CSF for the project and as a result has a very high priority level. A payment feature for an app like this is expected from users and the absence of this feature would put the app at a big disadvantage to competitors and alternative travel options.

Technical issues

Two different payment methods are accepted: Stripe and Google pay. Both are implemented using an API. I will need to follow the documentation to implement these features. Stripe is primarily used for web apps and as a result I may need to make additional changes compared to a web application.

Dependencies with other requirements

Firstly, the actor must have signed in using their google account to access this feature. The requirement to search for a new journey will be required and the add to favourites use case is an optional requirement that will help speed up the process of this use case.

Once a payment has been made a QR code will be presented to the user and the journey will be added to the history screen.

Use Case Diagram

Figure 07: Payment use case

Precondition

The actor selects a route and their preferred method of transportation(s)

Activation

The actor clicks on the get ticket button which redirects the actor to the payment screen.

Main flow

1. The actor clicks get tickets on their selected journey
2. The actor clicks on the Google Pay icon
3. The actor selects a payment method
4. The system gets a payment token from Google pay
5. The system sends the payment token and payment details to the apps backend
6. The system processes the purchase using the payment details in the payment token
7. The system sends the payment token to the payment service provider
8. The system sends confirmation of the purchase to the user
9. The system generates and sends a QR code for the journey
10. The journey is saved to the database and is then fetched using the users history screen

Alternate flow

A1: Pay using Stripe

1. The actor clicks get tickets on their selected journey
2. The actor enters their card details and clicks Pay
3. The system creates a source object with the payment details entered by the actor
4. The system checks if any further action is required by the actor
5. The actor completes the extra action if required
6. The system completes a charge request in the backend
7. The use case continues at step 8 of the main flow

Exceptional flow

E1: Invalid payment option

1. The actor clicks get tickets on their selected journey
2. The actor clicks on the Google Pay icon
3. The actor selects a payment method
4. The system gets a payment token from Google pay
5. The system sends the payment token and payment details to the apps backend
6. The system processes the purchase using the payment details in the payment token
7. The system can't verify the purchase using the payment details provided in the payment token.
8. The system sends an error message to the actor and prompts them to try again
9. The actor starts the payments process over again

Termination

The system confirms the payment and sends the actor the QR code for their journey.

Post condition

The system will generate a QR code that is active for the course of the journey

The actor can use the QR code as proof of purchase or to scan at security gates to gain access to the transportation.

Journey will be added to the actors history.

Use Case – Requirement 6: Add to Favourites

Scope

The scope of this use case is to implement a feature for the actor to save a trip to their favourites, allowing them to save time for their most regularly searched trips.

Description & Priority

The favourites list will act as a quick and convenient way for the actor to access their most frequently used journeys. The actor can search for a journey and add it to their favourites by clicking on the favourite icon located next to the journey information. This button press will serve as the trigger for the event to add the journey to the actors' favourites. The system will then collect the associated information from the journey and the actor (token profile information) and insert them to the favourites table using the database connection that has been established. If all the data is as expected the data will be inserted into the favourites table and the system will send back a confirmation message to the actor. The actor will be able to view the newly added favourite journey in their favourites.

Out of the 5 core functionalities, I believe this requirement has the lowest priority and as a result I will tackle this requirement last.

Technical issues

I will need to ensure that the method of relating the data to the user works as intended and that the search operation performed on the table does not affect the overall performance of the app.

Dependencies with other requirements

Firstly, the actor must have signed in using their google account to access this feature. Once a trip has been added to the actors' favourites, they will be able to quickly create that trip and pay for their ticket without having to search.

In terms of development use cases 1-3 will need to be finished before development can begin on this use case. Sign in is required to generate the foreign key in the favourites database table to connect the user to their favourites. Also, the requirement to search for a new journey will be required.

Use Case Diagram

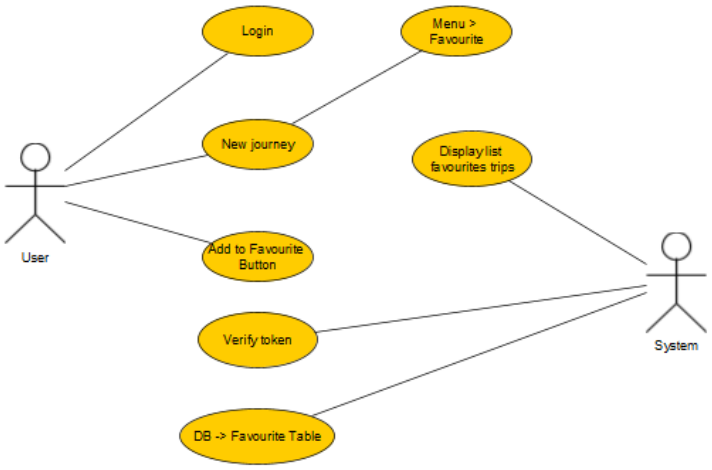


Figure 08: Favourite use case

Precondition

The actor must have a pre-existing google account to sign in and access the apps services.

The actor must search for the journey that they wish to favourite.

Activation

The actor clicks on the favourite icon next to the journey.

Main flow

1. The actor searches for a journey with a start and end point
2. The actor clicks on the icon to favourite the journey
3. The system gathers the associated data into an object
4. The system inserts the object into the database favourites table
5. The system notifies the actor that the journey was added to their favourites
6. The user checks their favourites to verify the journey has been added to their favourites list
7. The system gets the favourites data associated with the actor's email
8. The actor confirms the new journey has been added to their favourites

Alternate flow

A1: Get current location as start point

1. The actor searches for a journey with an end point only
2. The system generates the start point from the user's current geolocation
3. The use case continues at step 2 of the main flow

Exceptional flow

E1: User enters an invalid start or end point

1. The actor searches for a journey

2. The system searches the routes table but can't find any occurrence of the start or end point

3. The system prompts the actor to enter another location as no results were found

Termination

The system adds the trip to the actors' favourite trips.

Post condition

The system will add the trip to the actors' favourite trips list. The trip will be stored in a database table called favourites which will have a one-to-many relationship with the user table to ensure the correct user data is retrieved.

The actor will be able to view their favourite trips with the newly included trip added to the list.

2.1.2. Data Requirements

2.1.2.1 Google API

In order to use the APIs developed by Google, settings were added to the mobile application to integrate these services, as well as the generated access keys. Also based on a small form of the application's digital certificate, known as a SHA-1 fingerprint. The fingerprint is a unique text sequence, generated from the commonly used algorithm, SHA-1 hashing, being unique and exclusive to the device where the application will be developed.

2.1.2.2 Directions API

As an alternative to using the public APIs, I decided to use The Google Directions API. Within provides everything that this application needs including searching for directions using the location and destination, and calculating extra factors like the trips duration, and mode of transport. The objective at this moment is to display a route in the mobile application, for this the API has been implemented in the project. In addition, it is more efficient to use an API that provides everything in a single API, and the search can be done in other cities and countries, using the Google Directions API. This allows me to broaden the scope of the app and no longer be restricted to a single city.

All the steps that were used to apply this API are referenced on the website *developers.google.com.*

The Geolocation and Geocoding APIs are activated in this project. When opening the application, with the user's permission, the geolocation displays in the Edit View which automatically enters the users current location in the corresponding text field for location.

The user can search for an address, by entering the address details, this would be an example of how the geocoding function works.

2.1.2.3 Data Transfer Object (DTO)

They are a very effective pattern for transmitting information between a client and a server, since they allow us to create data structures that are independent of our data model, which allows us to create as many "views" as necessary from a set of tables or data sources. In addition, it allows us to control the format, name and data types with which we transmit the data to adjust to a certain requirement. Finally, if for some reason the data model changes (and with it the entities) the client will not be affected, since it will continue to receive the same DTO. In the application, the DTO is in the model folder. With the use of DTO it became easier to access and map the data.

2.1.3. User Requirements

1. An Intuitive design that is easy to use and pleasant to look at.
2. A simple way to find out the fastest and cheapest way to get from a starting location to a destination.
3. An easy and stress-free way to pay for the transportation required for the journey.
4. An app that can be used for regular daily commutes but also for discovering new transport routes in unfamiliar locations.

2.1.3.1 Non-Functional Requirements

This section presents requirements that do not focus on system behaviour but are qualities or properties that the application must have.

1. Perception requirements

The Style requirements

Application #: 1

Requirement Type: 1st

Description: The system has a simple design

Rationale: The system must have a simple design so that the user feels comfortable using it.

Origin (interested): User

Satisfaction criteria: The user can understand application functionality.

2. Humanity and usability requirements

The. Ease of Use Requirements

Application #: 2

Type of requirement: 2nd

Description: The system should be easy to use for users.

Justification: As the user profile is diverse, the system must be easy to use to allow greatest possible number of users.

Origin (interested): User

Satisfaction criteria: The user can use the application easily and simply.

3. Performance requirements

The. Speed and latency requirements

Application #: 3

Requirement Type: 3rd

Description: The system must be able to respond quickly to user requests.

Justification: The system must respond to user requests as quickly as possible, otherwise the user may get tired and stop using the application.

Origin (interested): User

Satisfaction criteria: The user can quickly access requests.

B. Confidentiality and Availability Requirements

Application #: 3

Requirement Type: 3b

Description: The system will be available most of the time.

Justification: The user must be able to use the system at any time.

Origin (interested): User

Satisfaction criteria: The system must be available and operate flawlessly 99% of the time.

4. Preservation and support requirements

The. Preservation and Support Requirements

Application #: 4

Type of requirement: 4th

Description: The system must have a user manual for the application.

Justification: The user must be able to consult the user manual whenever he wants to know how the application works.

Origin (interested): User

Satisfaction criteria: A user manual will be created to provide information on how to use the app.

2.1.4. Environmental Requirements

N/A

2.1.5. Usability Requirements

To be able to use the application, the user must log in, and the system offers two options for this, the user can register an email and password or log in using the Google account.

Crossing Borders is an easy-to-use app with a practical interface app for all user, furthermore, the design has a simple and effective colour scheme that is visually pleasing.

The system has a simple and intuitive layout. The screens contain the same colour scheme, button styling and font sizes.

The icons are easy to understand and represent their respective functions very well, so there is no need to put explanatory texts.

The user's location is already filled in using geolocation, so the user doesn't have to type.

Navigation

The system offers simple navigation, capable of providing the user with greater ease in locating the elements they are looking for.

Privacy

Each user has a separate account, but no user can have access to other users' information. The system must handle requests according to each user's identifier.

Performance

The system must be able to perform the functions quickly, so that the user does not experience any delay in processing, so as not to cause the user to lose time or patience with the app.

In cases the app being unable to find a route, the system informs the user with an error message, so they are aware that transport options were not found on that particular journey.

2.2. Design & Architecture

2.2.1 Physical architecture

The system will have a client-server architecture, since the client will be the mobile application that connects, through the Internet, to a server to retrieve data from the data source. The drawing of this architecture is shown in the figure 7.

The client-server architecture was chosen since a mobile device has little memory to store public transport data in many cities, so a server is needed to store a large amount of data and process it. In addition, with this architecture, it will not be necessary to replicate the data of each city on different mobile devices, as it will only be stored on the server. Having a server allows for the number of concurrent users to be increased and results in higher scalability.

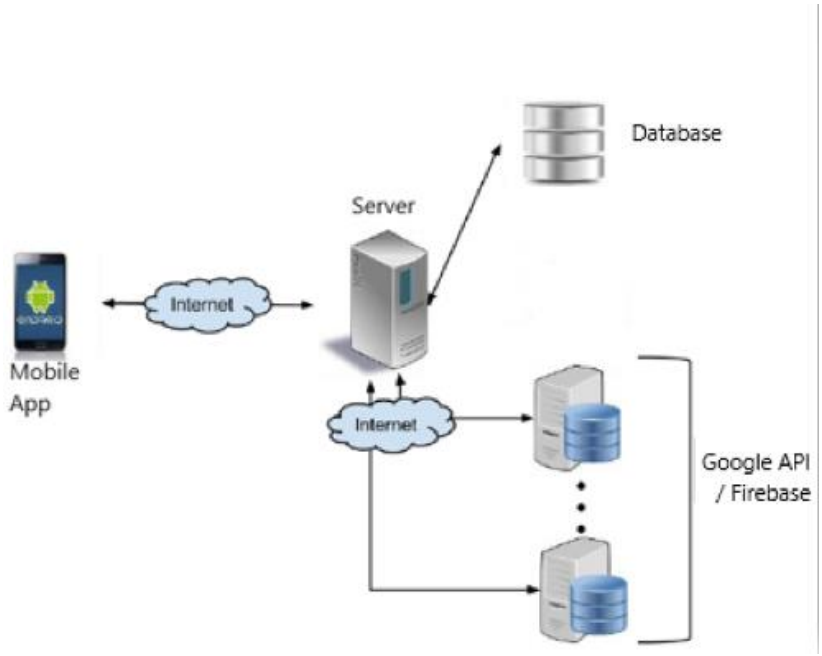


Figure 09: Client-server architecture

In the illustration 7 you can see a mobile application, which contains the complete user interface. This mobile application will allow the user to interact with the application and obtain all the information related to public transport.

For the application to obtain the information, it is necessary to connect to the server through the Internet. This server will be made up of three components: the service layer, which is where the application and the server will exchange requests; the business layer, in which all the necessary operations will be carried out to satisfy the needs of the system's functionalities; and the data layer. This last layer will be in charge of accessing the Crossing Border database and services to obtain and save the data.

The data tier will connect to a database, Google API and Firebase seen in the illustration. The city transport data is obtained from the Directions API (Google API) to display in the application the routes. This API uses a HTTP request to return a result in the form of JSON or XML. The result will return a list of the available transportation methods for the requested trip, this is the primary goal of the project. The instructions used can be viewed from this link

The app also uses a number of API's that require a backend infrastructure. In this application I am using firebase as a backend to help authenticate the tokens obtained from the Google Sign-In API and the Google Pay API.

2.2.2 Navigation map

The structure diagram shown in the figure below showcases the design of the app and clearly demonstrates how the user will be able to navigate from screen to screen while using the app. The menu acts as an easy and intuitive way to navigate between screens and acts similarly to a navbar on a website.

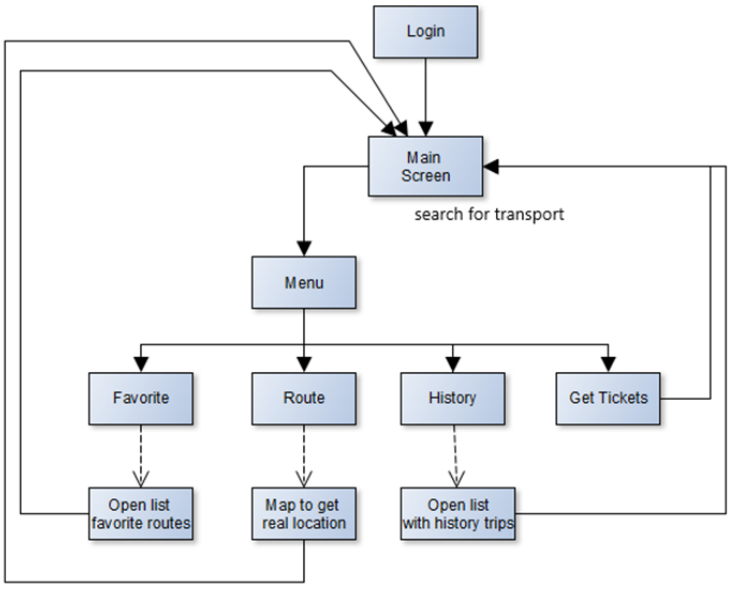


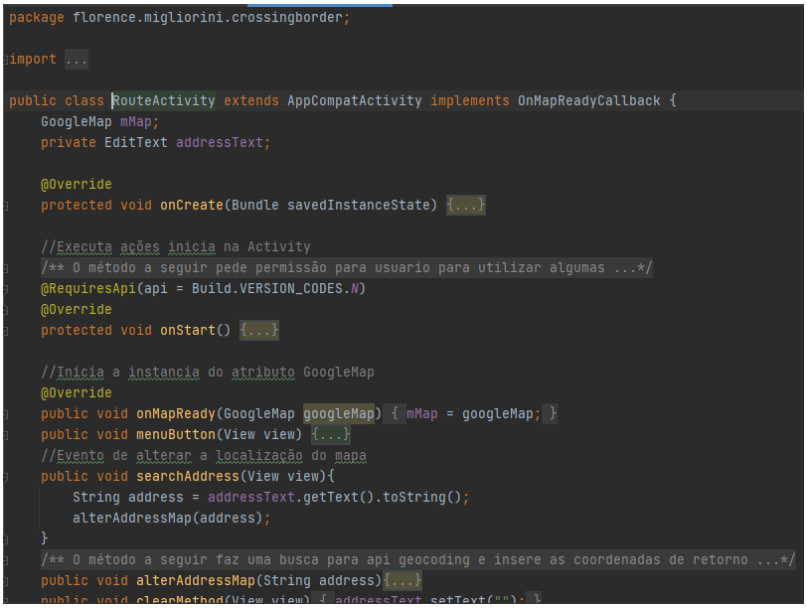
Figure 10: Navigation map

2.3. Implementation

2.3.1 Route - Maps on Android

To provide an application that shows the Realtime location of the user on the map, there is a Google API that allows for working with maps. I'm using the geolocation API and geocoding API.

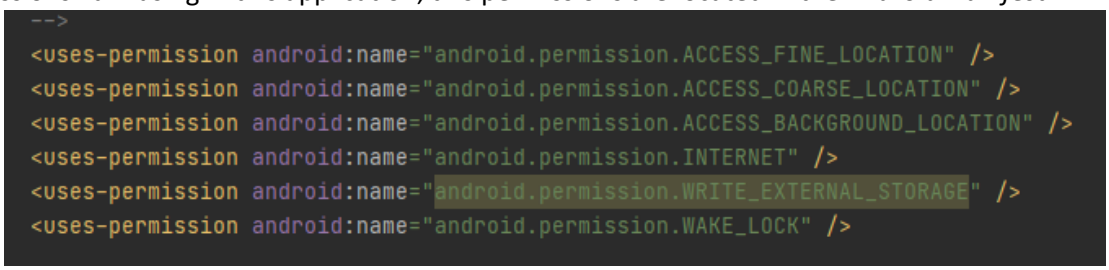
To work with maps, you need to create a layout that contains a fragment in which a map is defined. But in order for this map to be displayed in the application, a Java class must be implemented to display the map view. An implementation example for the map display is shown in figure 11, in which a View class is created and assigned to the shard defined in the layout. All the steps to implement the API you can find on *developers.google.com*.



```
package florence.migliorini.crossingborder;

import ...

public class RouteActivity extends AppCompatActivity implements OnMapReadyCallback {
    GoogleMap mMap;
    private EditText addressText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {...}

    //Executa ações inicia na Activity
    /** O método a seguir pede permissão para usuario para utilizar algumas ...*/
    @RequiresApi(api = Build.VERSION_CODES.N)
    @Override
    protected void onStart() {...}

    //Inicia a instancia do atributo GoogleMap
    @Override
    public void onMapReady(GoogleMap googleMap) { mMap = googleMap; }
    public void menuButton(View view) {...}
    //Evento de alterar a localização do mapa
    public void searchAddress(View view){
        String address = addressText.getText().toString();
        alterAddressMap(address);
    }
    /** O método a seguir faz uma busca para api geocoding e insere as coordenadas de retorno ...*/
    public void alterAddressMap(String address){...}
    public void cleanMethod(View view) { addressText.setText(""); }
```

Figure 11: RouteActivity

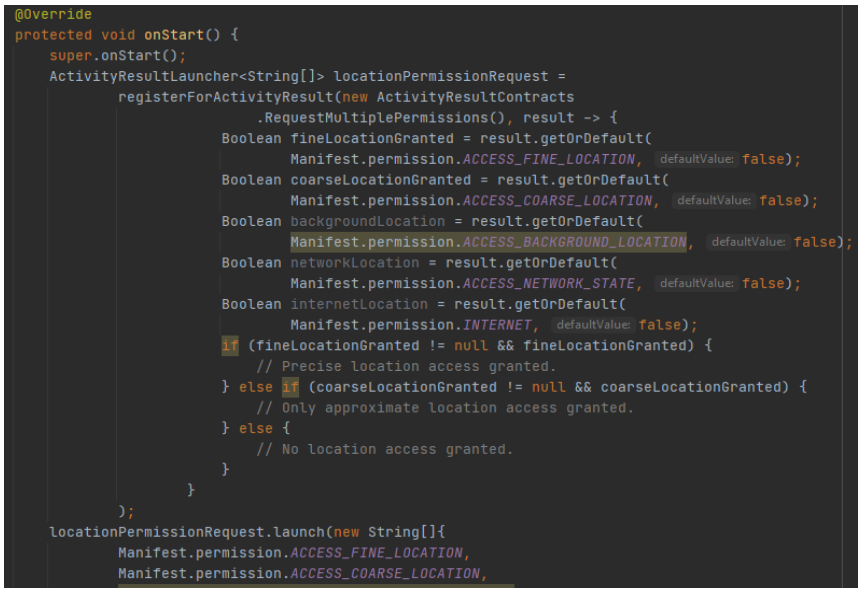
To get the user location, its necessary to get the permissions to use this functionality, as this is going to access restricted data/action from the user. Android has different types of permissions, which can be seen in this link, that I used to gain information on the topic, and in the figure 12 show the permissions I am using in this application, this permissions are located in the *AndroidManifest.xml.*



```
-->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
```

Figure 12: Permission used.

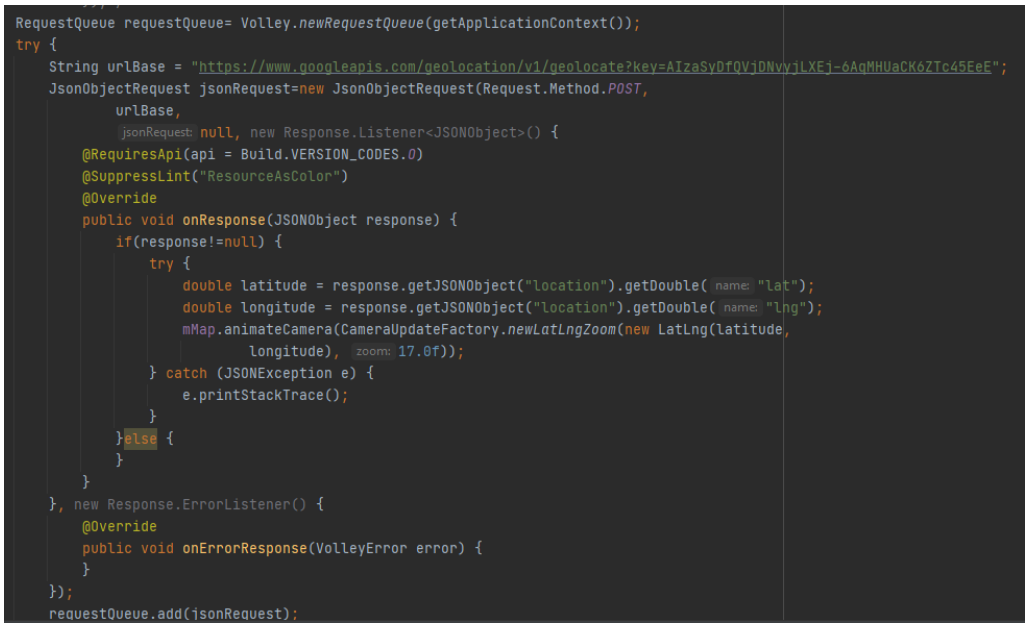
These permissions are called at the onStart() method on the class *RouteActivity.java.*



```
@Override
protected void onStart() {
    super.onStart();
    ActivityResultLauncher<String[]> locationPermissionRequest =
            registerForActivityResult(new ActivityResultContracts
                    .RequestMultiplePermissions(), result -> {
                Boolean fineLocationGranted = result.getOrDefault(
                        Manifest.permission.ACCESS_FINE_LOCATION, defaultValue: false);
                Boolean coarseLocationGranted = result.getOrDefault(
                        Manifest.permission.ACCESS_COARSE_LOCATION, defaultValue: false);
                Boolean backgroundLocation = result.getOrDefault(
                        Manifest.permission.ACCESS_BACKGROUND_LOCATION, defaultValue: false);
                Boolean networkLocation = result.getOrDefault(
                        Manifest.permission.ACCESS_NETWORK_STATE, defaultValue: false);
                Boolean internetLocation = result.getOrDefault(
                        Manifest.permission.INTERNET, defaultValue: false);
                if (fineLocationGranted != null && fineLocationGranted) {
                    // Precise location access granted.
                } else if (coarseLocationGranted != null && coarseLocationGranted) {
                    // Only approximate location access granted.
                } else {
                    // No location access granted.
                }
            }
    );
    locationPermissionRequest.launch(new String[]{
            Manifest.permission.ACCESS_FINE_LOCATION,
            Manifest.permission.ACCESS_COARSE_LOCATION,
```

Figure 13: onStart() method.

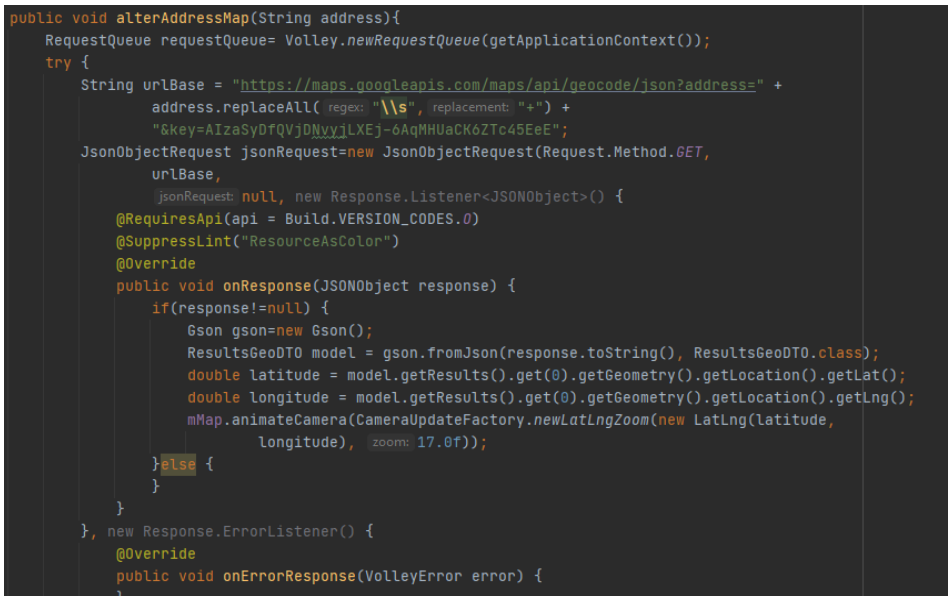
The dependency Volley is used to consume the APIs, figure 14 shows the request. Also, after confirming the user's permissions it then gets the user location.



```
RequestQueue requestQueue= Volley.newRequestQueue(getApplicationContext());
try {
    String urlBase = "https://www.googleapis.com/geolocation/v1/geolocate?key=AIzaSyDfQVjDNvyjLXEj-6AqMHUaCK6ZTc45EeE";
    JsonObjectRequest jsonRequest=new JsonObjectRequest(Request.Method.POST,
            urlBase,
            jsonRequest: null, new Response.Listener<JSONObject>() {
        @RequiresApi(api = Build.VERSION_CODES.O)
        @SuppressLint("ResourceAsColor")
        @Override
        public void onResponse(JSONObject response) {
            if(response!=null) {
                try {
                    double latitude = response.getJSONObject("location").getDouble( name: "lat");
                    double longitude = response.getJSONObject("location").getDouble( name: "lng");
                    mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(new LatLng(latitude,
                            longitude), zoom: 17.0f));
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            }else {
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
        }
    });
    requestQueue.add(jsonRequest);
```

Figure 14: Geolocation.

The *alterAddressMap()* transforms the address through geocoding and change the map address, the figure 15 shows the method.



```
public void alterAddressMap(String address){
    RequestQueue requestQueue= Volley.newRequestQueue(getApplicationContext());
    try {
        String urlBase = "https://maps.googleapis.com/maps/api/geocode/json?address=" +
                address.replaceAll( regex: "\\s", replacement: "+") +
                "&key=AIzaSyDfQVjDNvyjLXEj-6AqMHUaCK6ZTc45EeE";
        JsonObjectRequest jsonRequest=new JsonObjectRequest(Request.Method.GET,
                urlBase,
                jsonRequest: null, new Response.Listener<JSONObject>() {
            @RequiresApi(api = Build.VERSION_CODES.O)
            @SuppressLint("ResourceAsColor")
            @Override
            public void onResponse(JSONObject response) {
                if(response!=null) {
                    Gson gson=new Gson();
                    ResultsGeoDTO model = gson.fromJson(response.toString(), ResultsGeoDTO.class);
                    double latitude = model.getResults().get(0).getGeometry().getLocation().getLat();
                    double longitude = model.getResults().get(0).getGeometry().getLocation().getLng();
                    mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(new LatLng(latitude,
                            longitude), zoom: 17.0f));
                }else {
                }
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
```

Figure 15: alterAddressMap() Method - geocoding.

2.3.2 Favourite Class

The favourites list acts as a quick and convenient way for the actor to access their most frequently used journeys. The actor can search for a trip and add it to their favourites by clicking on the favourite icon located above the location and destination text fields. This button press will trigger the event to add the journey to the actors' favourites. The system will then collect the associated journey and actor information (token profile information) and insert it into the favourites table using the database connection that has been established. If all data is as expected, the data will

be entered into the favourites table. The actor will be able to preview the newly added favourite journey in their favourites.

```
public class FavoriteActivity extends AppCompatActivity {

    private Button btnDelete,btnBack;
    private List<TravelDTO> listFavorites;
    private LinearLayout lnListFavorites;

    //Atributos de elementos selecionado
    private ConstraintLayout favoriteSelected = null;
    private ConstraintLayout filterActive;
    private ConstraintLayout iconFilterActive;
    private TextView textViewActive;

    @SuppressLint({"ResourceAsColor", "WrongViewCast"})
    @RequiresApi(api = Build.VERSION_CODES.O)
    @Override
    protected void onCreate(Bundle savedInstanceState) {...}

    @RequiresApi(api = Build.VERSION_CODES.O)
    public void filterAll(View view){...}
    @RequiresApi(api = Build.VERSION_CODES.O)
    public void filterBus(View view){...}
    @RequiresApi(api = Build.VERSION_CODES.O)
    public void filterTrain(View view){...}
    @RequiresApi(api = Build.VERSION_CODES.O)
    public void filterLuas(View view){...}
    @SuppressLint("ResourceType")
    public void clearLastFilter(){...}
```

Figure 16: Favourite Activity class.

2.3.3 DbFavorite

To access the SQLite database within the Android platform, I use an access API, which comes in the SDK package.

Two classes will be used to create the database via the application, and both can be seen in the code in Figure XX.

By extending its SQLiteOpenHelper class, Android Studio forces the developer to implement two methods that are of paramount importance for the correct implementation of the database creation:

onCreate() method is called when the application creates the database for the first time. In this method, all the creation guidelines and the initial population of the bank must be included.

onUpgrade() method is the method responsible for updating the database with some structural information that has been changed. It is always called when an update is needed, so as not to have any kind of data inconsistency between the existing database on the device and the updated version that the application will use.

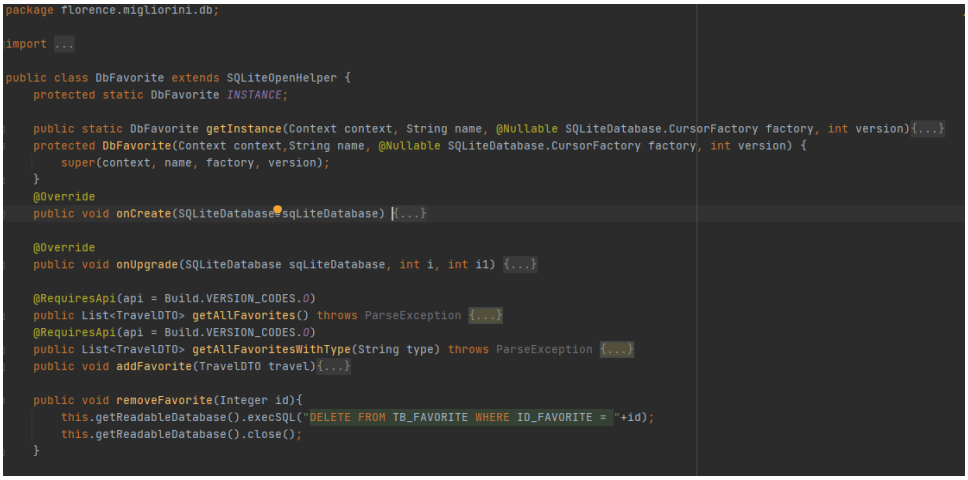


Figure 17: DbFavorite

The code in Figure 18 contains the name of the table and database creation code, that is located in the onCreated() method.

```
@Override
public void onCreate(SQLiteDatabase sqLiteDatabase) {
    String CREATE_TABLE="CREATE TABLE TB_FAVORITE (" +
            "ID_FAVORITE INTEGER PRIMARY KEY AUTOINCREMENT," +
            "DS_LOCATION VARCHAR(100)," +
            "CD_TRANSPORT INTEGER," +
            "DS_DISTINY VARCHAR(100)," +
            "DT_TIME VARCHAR(100)," +
            "NUM_VALUE INTEGER," +
            "DT_DURATION VARCHAR(100)," +
            "DS_TITLE_TICKET VARCHAR(100)," +
            "DT_HOUR_DEPARTURE VARCHAR(100)," +
            "DT_HOUR_TRAVEL VARCHAR(100)," +
            "NUM_PASSENGERS INTEGER)";
    sqLiteDatabase.execSQL(CREATE_TABLE);
}
```

Figure 18: onCreate() method.

2.3.4 Login Class

The user has two option to do the Login to access the main screen for this application. If the user does not have an account, they can register in the system by clicking in the Sign-Up button, that will bring the actor to fill out a form by providing an email and password for their new account. Completing the form and being authorised into the system, will grant access to main screen. The Figure 19 shows the method that configures the login with firebase auth.

The Google Sign-In API will be used to identify users and assign them a JWT access token that can be used to access the app's features while the token is still active. After a JWT token has expired the token will need to be refreshed to grant access to the app again. Initially, when a user signs in a GET request is made to retrieve the JWT token and then the ID token must be sent to the firebase backend using a POST request to validate the integrity of the token. If a token is valid a HTTP 200 successful response will be returned, and the user will be redirected to the home screen where they can access the main menu and create a new journey. The JWT token will be stored in local storage and will contain important meta data for the token's validation as well as profile information for the user. This profile information can be used to retrieve the user's name, email, and Google profile picture.

Clicking the login with google button initiates an intent previously created by google and obtained in the signIn() method.

```
public void createAccountAction(View view){
    alertMail.setText("");
    alertPassword.setText("");
    if(validateMail()){
        if(validatePassword()){
            registerWithFirebaseAuth(etEmail.getText().toString(),
                    etPassword.getText().toString());
        }
    }
}
//Registra uma nova conta com firebase auth
public void registerWithFirebaseAuth(String email, String password){
    mAuth.createUserWithEmailAndPassword(email, password)
            .addOnCompleteListener( activity: RegisterActivity.this, new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    if (task.isSuccessful()) {
                        FirebaseUser user = mAuth.getCurrentUser();
                        SQLiteMan.getInstance(getApplicationContext(), databaseTag: "database")
                                .setUserConnected(etEmail.getText().toString());
                        startHome();
                    } else {
                        alertPassword.setText("Your password must be in a format of at least 4 character
                                "containing letters, numbers and special characters (.,_)");
                    }
                }
            });
}
```

Figure 19: Register Activity

```
package florence.migliorini.crossingborder;

import ...

public class LoginActivity extends AppCompatActivity {

    private FirebaseAuth mAuth;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
        Button bLogin = findViewById(R.id.button);
        Button bSignup = findViewById(R.id.button3);
        Button bInfo = findViewById(R.id.button4);
        EditText etEmail = findViewById(R.id.editTextTextEmailAddress2);
        EditText etPasword = findViewById(R.id.editTextTextPassword2);

        // Firebase Auth
        mAuth = FirebaseAuth.getInstance();
        bSignup.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String sEmail = etEmail.getText().toString();
                String sPassword = etPasword.getText().toString();
                mAuth.createUserWithEmailAndPassword(sEmail, sPassword)
                        .addOnCompleteListener( activity: LoginActivity.this, new OnCompleteListener<AuthResult>() {
                            @Override
```

Figure 19: Login Activity

```
//Configura o botão de login com firebase auth
private void configLoginWithFirebaseAuth(){
    final LoginActivity activity= this;
    bLogin.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            errorMsgTextView.setText("");
            mAuth.signInWithEmailAndPassword(etEmail.getText().toString(), etPasword.getText().toString())
                    .addOnCompleteListener(activity, new OnCompleteListener<AuthResult>() {
                        @Override
                        public void onComplete(@NonNull Task<AuthResult> task) {
                            if (task.isSuccessful()) {
                                FirebaseUser user = mAuth.getCurrentUser();
                                SQLiteMan.getInstance(getApplicationContext(), databaseTag: "database")
                                        .setUserConnected(etEmail.getText().toString());
                                startHome();
                            } else {
                                errorMsgTextView.setText("Confira o email e a senha e tente novamente");
                            }
                        }
                    });
        }
    });
```

Figure 20: configLoginWithFirebaseAuth() method.

2.3.5 History Class

The history list in this app is a convenient way for the actor to keep track of trips and payments made. All trips in which the payment is made are saved in the history. The system will then collect the associated journey and actor information (token profile information) and insert it into the history table using the database connection that has been established.

The contructListFavorites() method is used in this class as in the FavoriteActivity class to build the dynamic list of history. What is being accomplished in this method is the building of the layout and bringing all the attributes from TravelDTO class and transport icons from the castingCdTypeTransport() Method as well.

```
}
//Constroi a lista de histórico dinamicamente
@RequiresApi(api = Build.VERSION_CODES.O)
public void constructListFavorites(List<TravelDTO> list){...}
//Deleta um item do histórico.
@RequiresApi(api = Build.VERSION_CODES.O)
public void deleteHistory(View view){
    if(historicSelected!=null){
        @SuppressLint("ResourceType") TextView tx = (TextView) historicSelected.findViewById(historicSelected.getId()+5);
        SQLiteMan.getInstance(getApplicationContext(), databaseTag: "database").removeHistoricById(Integer.parseInt( s: tx.getText()+""));
        lnListHistoric.removeView(historicSelected);
    }
}
public void menuButton(View view) {...}
public void backButton(View view) {
    Intent intent = new Intent( packageContext: HistoryActivity.this, HomeActivity.class);
    startActivity(intent);
    finish();
}
public Integer castingCdTypeTransport(Integer cd,String color){
    if(color == "default"){
        switch (cd){
            case 1:
                return R.drawable.ic_baseline_train_24;
            case 2:
                return R.drawable.ic_baseline_directions_bus_24;
            case 3:
                return R.drawable.ic_baseline_subway_24;
        }
    }else if(color == "white"){
        switch (cd){
            case 1:
                return R.drawable.ic_baseline_train_white_24;
            case 2:
                return R.drawable.ic_baseline_directions_bus_white_24;
            case 3:
                return R.drawable.ic_baseline_subway_white_24;
        }
    }
    return null;
}
}
```

Figure 21: History Activity class.

2.3.6 DbHistory

The DbHistory Activity works in the same way that DbFavorite.

```
@Override
public void onCreate(SQLiteDatabase sqLiteDatabase) {
    String CREATE_TABLE="CREATE TABLE TB_HISTORY (" +
            "ID_HISTORY INTEGER PRIMARY KEY AUTOINCREMENT," +
            "DS_LOCATION VARCHAR(100)," +
            "CD_TRANSPORT INTEGER," +
            "DS_DISTINY VARCHAR(100)," +
            "DT_TIME VARCHAR(100)," +
            "USER_EMAIL VARCHAR(100))";
    sqLiteDatabase.execSQL(CREATE_TABLE);
}
```

Figure 22: TB_History

```
public void addHistory(TravelDTO travel,String email){
    ContentValues values = new ContentValues();
    values.put("DS_LOCATION",travel.getLocation());
    values.put("CD_TRANSPORT",travel.getCdTransport());
    values.put("DS_DISTINY",travel.getDestiny());
    values.put("DT_TIME",travel.getDtInitial().toString());
    values.put("USER_EMAIL",email);
    this.getReadableDatabase().insert( table: "TB_HISTORY", nullColumnHack: null,values);
    this.getReadableDatabase().close();
}
```

Figure 23: addHistory() method.

```
@RequiresApi(api = Build.VERSION_CODES.O)
public List<TravelDTO> getHistory(String email){
    List<TravelDTO> list = new ArrayList<>();
    Cursor cursor = this.getReadableDatabase().rawQuery( sql: "SELECT * FROM TB_HISTORY " +
            "WHERE USER_EMAIL = '"+email+"'", selectionArgs: null);
    TravelDTO travel = null;
    if(cursor.moveToFirst()) {
        do {
            travel = new TravelDTO(
                    cursor.getInt( i: 0),
                    cursor.getString( i: 1),
                    cursor.getString( i: 3),
                    LocalDate.parse(cursor.getString( i: 4)),
                    cursor.getInt( i: 2),
                     duration: null,
                     value: null,
                     dtDuration: null, dsTitleTicket: null, dtHourDeparture: null, dtHourTravel: null, numPassengers: null
            );
            list.add(travel);
        } while (cursor.moveToNext());
    }
    this.getReadableDatabase().close();
    return list;
}
```

Figure 24: getHistory() method.

2.3.7 DbLogin

In this Activity, the User table is created, where the application saves all emails from users who log in to the app using sign up or a Google account, making it possible to identify who saved each trip in the History and Favourite tables. shown in Figure 25.

```
@Override
public void onCreate(SQLiteDatabase sqLiteDatabase) {
    String CREATE_TABLE="CREATE TABLE USER (EMAIL VARCHAR(100))";
    sqLiteDatabase.execSQL(CREATE_TABLE);
}

@Override
public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i1) {
    sqLiteDatabase.execSQL("DROP TABLE IF EXISTS USER");
    onCreate(sqLiteDatabase);
}

public Boolean login(String email){
    Cursor cursor = this.getReadableDatabase().query( table: "USER",new String[]{"EMAIL"},
            selection: "EMAIL = ?",new String[]{email}, groupBy: null, having: null, orderBy: null, limit: null);
    cursor.moveToFirst();
    Boolean bol = ((cursor.getCount()>0));
    this.getReadableDatabase().close();
    return bol;
}
```

Figure 25: DbLogin

```
public String getUserConnected(){
    Cursor cursor = this.getReadableDatabase().rawQuery( sql: "SELECT * FROM USER", selectionArgs: null);
    String email = null;
    if(cursor.getCount()>0){
        cursor.moveToFirst();
        email = cursor.getString( i: 0);
    }
    this.getReadableDatabase().close();
    return email;
}
public void setUserConnected(String email){
    ContentValues values = new ContentValues();
    values.put("EMAIL",email);
    this.getReadableDatabase().insert( table: "USER", nullColumnHack: null,values);
    this.getReadableDatabase().close();
}
public void logoutDb(String email){
    this.getReadableDatabase().delete( table: "USER", whereClause: "EMAIL = ?",new String[]{email});
}
```

Figure 26: DbLogin

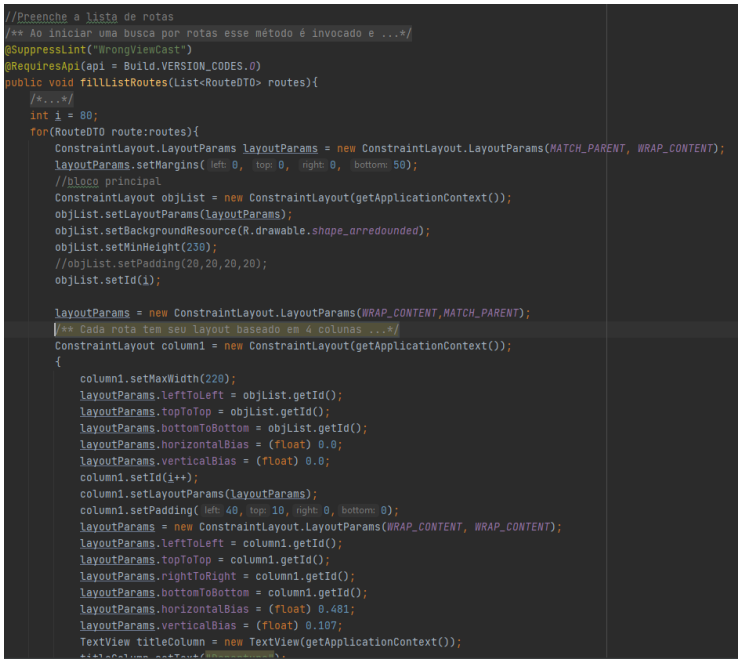
2.3.8 Home Class – Get ticket

The Home Activity is the important class for this application, where the actor can add information about the destination and the system returns the available routes.

When starting a route search, the fillListRoutes() method is invoked and receives a list of possible routes with modes of transport, and fills the linear layout with the corresponding information.

When creating an element dynamically, an id is required and, in this list, the ids start with base 80 and are added 1 by 1.

Each route has its layout based on 4 columns and each scope in Figure 27 corresponds to a column with its respective elements.



```
//Preenche a lista de rotas
/** Ao iniciar uma busca por rotas esse método é invocado e ...*/
@SuppressLint("WrongViewCast")
@RequiresApi(api = Build.VERSION_CODES.O)
public void fillListRoutes(List<RouteDTO> routes){
    /*...*/
    int i = 80;
    for(RouteDTO route:routes){
        ConstraintLayout.LayoutParams layoutParams = new ConstraintLayout.LayoutParams(MATCH_PARENT, WRAP_CONTENT);
        layoutParams.setMargins( left: 0,  top: 0,  right: 0,  bottom: 50);
        //bloco principal
        ConstraintLayout objList = new ConstraintLayout(getApplicationContext());
        objList.setLayoutParams(layoutParams);
        objList.setBackgroundResource(R.drawable.shape_arredounded);
        objList.setMinHeight(230);
        //objList.setPadding(20,20,20,20);
        objList.setId(i);

        layoutParams = new ConstraintLayout.LayoutParams(WRAP_CONTENT,MATCH_PARENT);
        /** Cada rota tem seu layout baseado em 4 colunas ...*/
        ConstraintLayout column1 = new ConstraintLayout(getApplicationContext());
        {
            column1.setMaxWidth(220);
            layoutParams.leftToLeft = objList.getId();
            layoutParams.topToTop = objList.getId();
            layoutParams.bottomToBottom = objList.getId();
            layoutParams.horizontalBias = (float) 0.0;
            layoutParams.verticalBias = (float) 0.0;
            column1.setId(i++);
            column1.setLayoutParams(layoutParams);
            column1.setPadding( left: 40,  top: 10,  right: 0,  bottom: 0);
            layoutParams = new ConstraintLayout.LayoutParams(WRAP_CONTENT, WRAP_CONTENT);
            layoutParams.leftToLeft = column1.getId();
            layoutParams.topToTop = column1.getId();
            layoutParams.rightToRight = column1.getId();
            layoutParams.bottomToBottom = column1.getId();
            layoutParams.horizontalBias = (float) 0.481;
            layoutParams.verticalBias = (float) 0.107;
            TextView titleColumn = new TextView(getApplicationContext());
```

Figure 27: fillListRoutes() Method

Value calculation based on €0.20 per KM, Figure 28.

```
TextView priceText = new TextView(getApplicationContext());
layoutParams = new ConstraintLayout.LayoutParams(WRAP_CONTENT, WRAP_CONTENT);
layoutParams.leftToLeft = column4.getId();
layoutParams.topToBottom = titleText.getId();
layoutParams.rightToRight = column4.getId();
layoutParams.bottomToBottom = column4.getId();
layoutParams.verticalBias = (float) 0.125;
priceText.setId(i++);
priceText.setTextSize(18);
priceText.setTextColor(Color.rgb( red: 15,  green: 48,  blue: 99));
priceText.setLayoutParams(layoutParams);
String price;
Integer priceInt = 100;
String numberPassangers="1";
//Calculo do valor baseado em 0.10 por KM
if(route.getLegs().get(0).getDistance()!=null){
    price = route.getLegs().get(0).getDistance().getText().split( regex: " ")[0];
    DecimalFormat formatPrice = new DecimalFormat( pattern: "#.00");
    numberPassangers = spinner.getSelectedItem().toString();
    numberPassangers = numberPassangers.split( regex: " ")[0];
    price = String.valueOf(formatPrice.format( number: 0.20*Double.parseDouble(price)));
    RadioButton rbOneWay = findViewById(R.id.rbOneWay);
    if(!rbOneWay.isChecked()){
        price = String.valueOf(Double.parseDouble(price)*2);
    }
    price = String.valueOf(Double.parseDouble(price)*Integer.parseInt(numberPassangers));
    priceInt = Integer.parseInt(String.valueOf(price)
            .replaceAll( regex: "\\.", replacement: ""));
    priceText.setText("€ "+Double.parseDouble(price));
}
column4.addView(priceText);

ConstraintLayout btnGetTicket = new ConstraintLayout(getApplicationContext());
layoutParams = new ConstraintLayout.LayoutParams(MATCH_PARENT, WRAP_CONTENT);
layoutParams.leftToLeft = column4.getId();
layoutParams.topToBottom = priceText.getId();
layoutParams.rightToRight = column4.getId();
layoutParams.bottomToBottom = column4.getId();
btnGetTicket.setBackgroundResource(R.drawable.shape_arredounded_fifteen_blue);
btnGetTicket.setPadding( left: 20,  top: 20,  right: 20,  bottom: 20);
btnGetTicket.setLayoutParams(layoutParams);
Integer finalPriceInt = priceInt;
```

Figure 28: Price based on €0.20 per km

Actions to query possible routes.

The method executes 3 requests in Google Directions API (the link)

1. Search routes for train type

2. Search routes for bus type

3. Search routes for tram type

To consume the API, the RequestQueue class from the Volley library is used (the link).

For JSON manipulation I chose to use a DTO design, that is being used every time needed because it is more transparent about what is being fetched and received instead of knowing the name of all keys in JSON Object. For casting JSON to DTO (Java class) the GSON class was used (the link). Figure 29 shows the searchAction() method.

```
public void searchAction(View view){
    //Pede as coordenadas de um endereço especifico.
    getCoordenation(etDestination.getText().toString(), type: 2);
    getCoordenation(etLocation.getText().toString(), type: 1);
    new Timer().schedule(() -> {
                //Montagem do URL BASE para todas as demais requisições.

                String dateFormat="";
                //Chave da api
                String API_KEY = "AIzaSyDfQVjDNvyjLXEj-6AqMHUaCK6ZTc45EeE";
                String urlBase = "https://maps.googleapis.com/maps/api/directions/json?";
                urlBase+="key="+API_KEY;
                urlBase+="&alternatives=true";
                Calendar calendar = Calendar.getInstance();
                if(daySelected!=0){
                    calendar.set(yearSelected,monthSelected,daySelected);
                }
                if(hour!=0){
                    calendar.set(yearSelected,monthSelected,daySelected,hour,minute);
                }
                dateFormat = String.valueOf(calendar.getTimeInMillis()/1000);
                if(dateFormat!=""){
                    urlBase+="&departure_time="+dateFormat;
                }
                urlBase+="&destination="+destiny;
                urlBase+="&origin="+origin;
                urlBase+="&mode=transit";
                //urlBase+="&transit_mode=tram|train|bus";
                RequestQueue requestQueue= Volley.newRequestQueue(getApplicationContext());
                try {
                    JsonObjectRequest jsonRequestTrain=new JsonObjectRequest(Request.Method.GET,
                            url: urlBase+"&transit_mode=train",
                            jsonRequest: null, new Response.Listener<JSONObject>() {
                        @RequiresApi(api = Build.VERSION_CODES.O)
                        @SuppressLint("ResourceAsColor")
                        @Override
                        public void onResponse(JSONObject response) {
                            if(response!=null) {
                                Gson gson=new Gson();
                                DirectionsMainDTO modelMaps = gson.fromJson(response.toString(),DirectionsMainDTO.class);
                                if(modelMaps.getRoutes().size()>0){
                                    typeTransportSelected = 1;
                                    listRoutesTrain = modelMaps.getRoutes();
```

Figure 29: searchAction() method.

Volley library, JSON manipulation with DTO design are also used for the filter methods Figure 30, getCoordenation(), Figure 31, and Figure 32 for initialLocalization().This helped to make the code less redundant as the code didn't have to be repeated.

The following methods (filterTrain(), filterBus(), filterLuas(), clearLastFilter()) perform the actions of filtering the routes by a specific transport type and the action of clearing the list of routes before filling the new one. Also, the logic of changing colors and filter icons is related to transport types. Figure 30.

```
@RequiresApi(api = Build.VERSION_CODES.O)
public void filterTrain(View view){...}
@RequiresApi(api = Build.VERSION_CODES.O)
public void filterBus(View view){...}
@RequiresApi(api = Build.VERSION_CODES.O)
public void filterLuas(View view){
    if(filterActive!=null){
        clearLastFilter();
        view.setBackgroundResource(R.drawable.btn_madison);
        ConstraintLayout icon = findViewById(R.id.btnLuasTypeHome);
        TextView txBtn = findViewById(R.id.tx8tnLuasTypeHome);
        icon.setBackgroundResource(R.drawable.ic_baseline_subway_white_24);
        txBtn.setTextColor(Color.rgb( red: 255, green: 250, blue: 250));
        filterActive = (ConstraintLayout) view;
        textViewActive = txBtn;
        iconFilterActive = icon;
        listRoutes.removeAllViews();
        typeTransportSelected = 3;
        try{
            fillListRoutes(listRoutesLuas);
        }catch (Exception e){
            e.printStackTrace();
        }
    }
}
@SuppressLint("ResourceType")
public void clearLastFilter(){
    filterActive.setBackgroundResource(R.drawable.shape_arredounded);
    if(typeTransportSelected == 1){
        iconFilterActive.setBackgroundResource(R.drawable.ic_baseline_train_24);
    }else if(typeTransportSelected==2){
        iconFilterActive.setBackgroundResource(R.drawable.ic_baseline_directions_bus_24);
    }else if(typeTransportSelected==3){
        iconFilterActive.setBackgroundResource(R.drawable.ic_baseline_subway_24);
    }
    textViewActive.setTextColor(Color.rgb( red: 56, green: 56, blue: 56));
}
```

Figure 30: filter and clear methods.

The following method queries the coordinates of an address sent using the google geocoding API, the link.

The request to the API is also done with Volley, the link.

```
public synchronized CoordenationDTO getCoordenation(String address,int type){
    RequestQueue requestQueue= Volley.newRequestQueue(getApplicationContext());
    try {
        String urlBase = "https://maps.googleapis.com/maps/api/geocode/json?address=" +
                address.replaceAll( regex: "\\s", replacement: "+")
                .replaceAll( regex: "(R\\.)", replacement: "Rua")+
                "&key=AIzaSyDfQVjDNvyjLXEj-6AqMHUaCK6ZTc45EeE";
        JsonObjectRequest jsonRequest=new JsonObjectRequest(Request.Method.GET,
                urlBase,
                jsonRequest: null, new Response.Listener<JSONObject>() {
            @RequiresApi(api = Build.VERSION_CODES.O)
            @SuppressLint("ResourceAsColor")
            @Override
            public void onResponse(JSONObject response) {
                if(response!=null) {
                    Gson gson=new Gson();
                    ResultsGeoDTO model = gson.fromJson(response.toString(), ResultsGeoDTO.class);
                    if(model.getResults().size()>0){
                        double lat = model.getResults().get(0).getGeometry().getLocation().getLat();
                        double lng = model.getResults().get(0).getGeometry().getLocation().getLng();
                        if(type==1){
                            origin = lat+","+lng;
                        }else if(type==2){
                            destiny = lat+","+lng;
                        }
                    }
                }else {
                    return;
                }
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
            }
        });
        requestQueue.add(jsonRequest);
        return null;
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
```

Figure 31: getCoordenation() method.

The following method initializes the current location to a not-so-precise radius. Using the geolocation API, the link

Also a request to the geocoding API, the link

```
public void initialLocalization(){
    RequestQueue requestQueue= Volley.newRequestQueue(getApplicationContext());
    try {
        String urlBase = "https://www.googleapis.com/geolocation/v1/geolocate?key=AIzaSyDfQVjDNvyjLXEj-6AqMHUaCK6ZTc45EeE";
        JsonObjectRequest jsonRequest=new JsonObjectRequest(Request.Method.POST,
                urlBase,
                jsonRequest: null, new Response.Listener<JSONObject>() {
            @RequiresApi(api = Build.VERSION_CODES.O)
            @SuppressLint("ResourceAsColor")
            @Override
            public void onResponse(JSONObject response) {
                if(response!=null) {
                    try {
                        double latitude = response.getJSONObject("location").getDouble( name: "lat");
                        double longitude = response.getJSONObject("location").getDouble( name: "lng");
                        RequestQueue requestQueue= Volley.newRequestQueue(getApplicationContext());
                        try {
                            String urlBase = "https://maps.googleapis.com/maps/api/geocode/json?latlng=" +
                                    latitude +","+ longitude +
                                    "&key=AIzaSyDfQVjDNvyjLXEj-6AqMHUaCK6ZTc45EeE";
                            JsonObjectRequest jsonRequest=new JsonObjectRequest(Request.Method.GET,
                                    urlBase,
                                    jsonRequest: null, new Response.Listener<JSONObject>() {
                                @RequiresApi(api = Build.VERSION_CODES.O)
                                @SuppressLint("ResourceAsColor")
                                @Override
                                public void onResponse(JSONObject response) {
                                    if(response!=null) {
                                        try{
                                            String road =  response.getJSONArray( name: "results").getJSONObject( index: 0)
                                                    .getString( name: "formatted_address").toString().split( regex: ",")[0].toString();
                                            etLocation.setText(road);
                                        }catch (JSONException e){
                                            e.printStackTrace();
                                        }
                                    }else {
                                        return;
                                    }
                                }
                            }, new Response.ErrorListener() {
                                @Override
```

Figure 32: initialLocalization() method.

The home screen allows the user to choose the number of passengers for their trip, the user can choose between the selection shown below (1-3 passengers) Figure 33. An array list was created that returns the list of the number of passengers for the users selection.

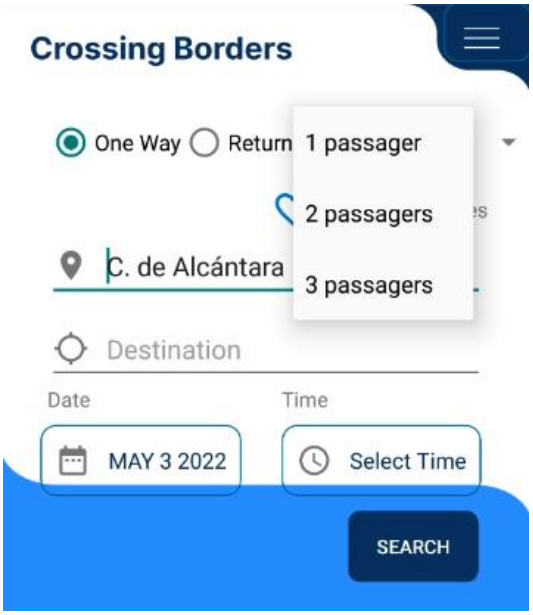


Figure 33: Spinner passengers list

In the mobile application, this service is called immediately after the user requests the execution of this use case, and the list returned is displayed in a Spinner, enabling the selection of passengers, as shown in Figure 33. Figure 35 shows the part of the code, Figure 34 shows the array located in *strings.xml* file.

```xml
<!-- Spinner Passagers -->
<string-array name="passagers">
    <item>1 passager</item>
    <item>2 passagers</item>
    <item>3 passagers</item>
</string-array>
```

Figure 34: string.xml file

```
spinner = (Spinner) findViewById(R.id.sp_passagers);
ArrayAdapter<String> adapter = new ArrayAdapter<~>( context: HomeActivity.this,
        android.R.layout.simple_list_item_1, getResources().getStringArray(R.array.passagers));
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinner.setAdapter(adapter);
```

Figure 35: Spinner code.

2.3.9 Payment Class

Stripe supports many types of electronic payments, for example, it is possible to easily integrate wallets like Apple Pay, Google Pay, and others into the app, in the same way, Stripe supports credit and debit cards.

Stripe provides a native UI to securely accept payments in Android and iOS apps. Likewise, the SDK comes with a predefined UI for accepting payments in Google Pay, which is used in this app.

The first step with Stripe is to create an account, which is necessary to obtain a personal access key.

All the steps to configure in the link.

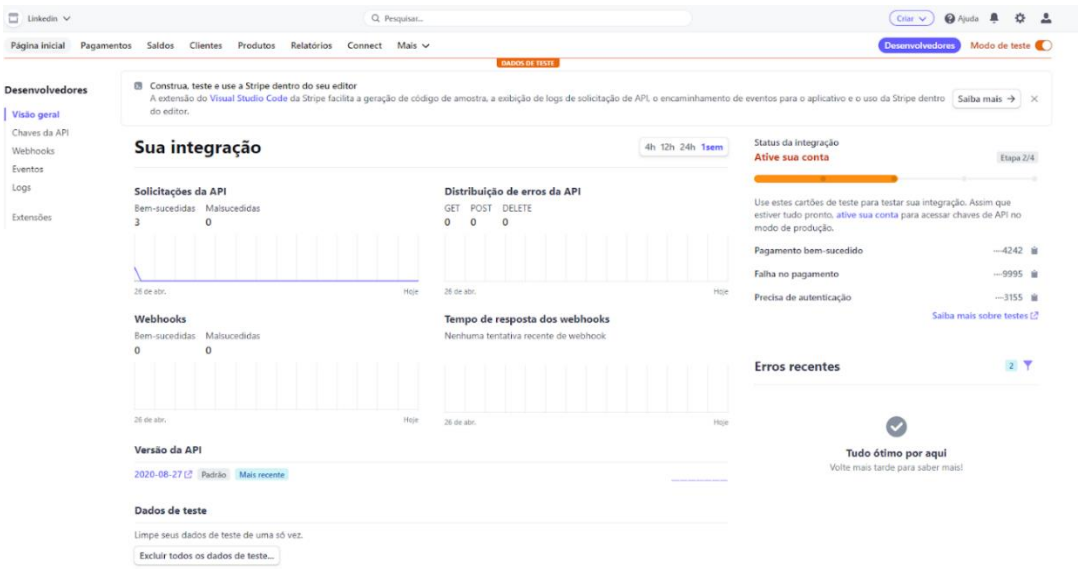


Figure 36: Stripe

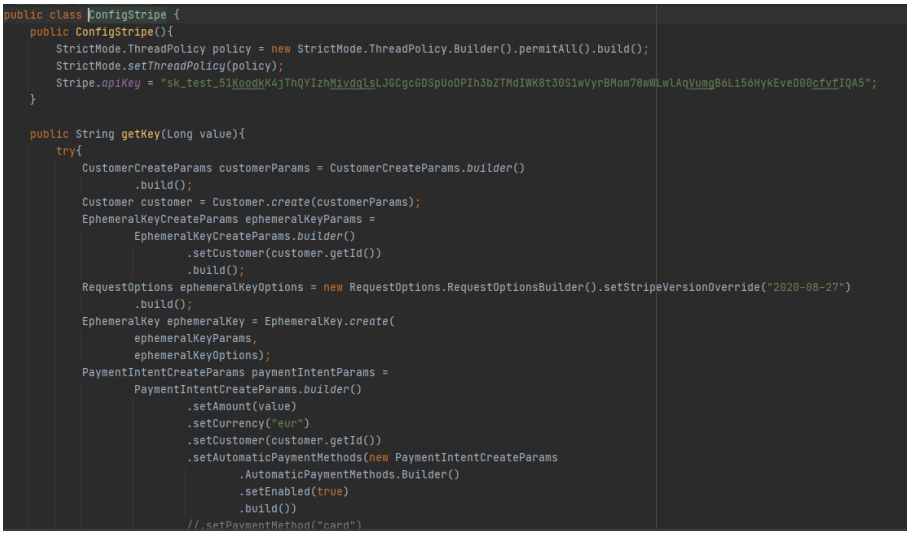


Figure 37: Stripe API key configuration.

The payment method with google pay was used based on Stripe documentation, link. Figure 38 shows the configuration for the payment button with Google Pay.

```
public void configPaymentWithGooglePlay(){
    final GooglePayLauncher googlePayLauncher = new GooglePayLauncher(
            activity: this,
            new GooglePayLauncher.Config(
                    GooglePayEnvironment.Test,
                    merchantCountryCode: "US",
                    merchantName: "Widget Store"
            ),
            this::onGooglePayReady,
            this::onGooglePayResult
    );
    btnGooglePlayPay.setOnClickListener(
            v -> googlePayLauncher.presentForPaymentIntent(strpe.getPaymentIntentClientSecret())
    );
}
private void onGooglePayReady(boolean isReady) {
    // implemented below
}

private void onGooglePayResult(GooglePayLauncher.Result result) {
    // implemented below
}
```

Figure 38: Google Pay

The following method configures and creates an intent for the Stripe payment method, card payment. Link

```
private void fetchPaymentIntent() {
    paymentIntentClientSecret = strpe.getPaymentIntentClientSecret();
    customerConfig = new PaymentSheet.CustomerConfiguration(
            strpe.getCustomer(),
            strpe.getEphemeralKey()
    );
    PaymentConfiguration.init(getApplicationContext(),strpe.getPublishableKey());
    PaymentSheet.Configuration configuration = new PaymentSheet.Configuration( merchantDisplayName: "Example, Inc.");

    paymentSheet.presentWithPaymentIntent(paymentIntentClientSecret, configuration);
}
```

Figure 39: fetchPaymentIntent() method.

The following method performs actions for Stripe's return. If the payment is confirmed, the method executes an intent for the Activity responsible for generating the QR Code by passing the necessary information. Also, it's then added to purchase history and favourites (if selected) using a local SQLite database. Link.

```
@RequiresApi(api = Build.VERSION_CODES.O)
void onPaymentSheetResult(final PaymentSheetResult paymentSheetResult) {
    if (paymentSheetResult instanceof PaymentSheetResult.Canceled) {
        Log.d( tag: "Stripe", msg: "Canceled");
    } else if (paymentSheetResult instanceof PaymentSheetResult.Failed) {
        showAlert( title: "Erro", message: "Ocorreu um erro inesperado ao tentar executar o pagamento");
        Log.e( tag: "Stripe",  msg: "Got error: ", ((PaymentSheetResult.Failed) paymentSheetResult).getError());
    } else if (paymentSheetResult instanceof PaymentSheetResult.Completed) {
        Intent intent = new Intent( packageContext: PaymentActivity.this,PlanActivity.class);
        intent.putExtra( name: "titleTicket",titleTicket.getText());
        intent.putExtra( name: "locationName",locationName.getText());
        intent.putExtra( name: "locationTime",locationTime.getText());
        intent.putExtra( name: "destinationName",destinationName.getText());
        intent.putExtra( name: "destinationTime",destinationTime.getText());
        intent.putExtra( name: "TimeTicket",TimeTicket.getText().toString());
        intent.putExtra( name: "dateTicketPlan",dateTimePay);
        intent.putExtra( name: "numberPersons",numberPersons.getText().toString());
        intent.putExtra( name: "imgTypeTransport",typeTransport);
        TravelDTO travel = new TravelDTO( id: null,locationName.getText().toString(),destinationName.getText().toString()
                , LocalDate.now(),typeTransport,dateTimePay,valueTicket,
                TimeTicket.getText().toString(),titleTicket.getText().toString(),
                locationTime.getText().toString(),destinationTime.getText().toString()
                ,Integer.parseInt(numberPersons.getText().toString()));
        if(favoriteSelection == 1){
            SQLiteMan.getInstance(getApplicationContext(), databaseTag: "database").addFavorite(travel);
        }
        SQLiteMan.getInstance(getApplicationContext(), databaseTag: "database")
                .addHistoric(travel);
        startActivity(intent);
        finish();
    }
}
```

Figure 40: onPaymentSheetResult() method..

2.3.10 SQLiteMan Class

The singleton pattern is an object creation pattern that ensures that there is only one instance of a class and provides a global access point to it. For this application, I believe it is critical to have only one instance. For example, in the project, a database management class (SQLiteMan class - centralizing class for the database) is used to encapsulate SQLite's read and write operations. As SQLite doesn't support multithreading very well. If two threads are computing operations to the database at the same time, the error "The database is locked" will be reported. By using Singleton, I can have more control over the access to properties and methods of a class, and I also reduce unnecessary memory consumption by using many unnecessary instances of a class.

```
package florence.migliorini.db;

import ...

public class SQLiteMan {
    /** Classe centralizadora para o banco SQLite ...*/
    protected static SQLiteMan INSTANCE;
    public static SQLiteMan getInstance(Context context,String databaseTag){...}
    //Construtor do singleton que inicia todas tabelas e databases
    protected SQLiteMan(Context context, String databaseTag){...}
    private DbLogin dbLogin;
    private DbFavorite dbFavorite;
    private DbHistory dbHistoric;

    public void setUserConnected(String email) { dbLogin.setUserConnected(email); }
    public void logoutUser() { dbLogin.logoutDb(getUserConnected()); }
    public String getUserConnected(){
        String email = dbLogin.getUserConnected();
        return email;
    }

    @RequiresApi(api = Build.VERSION_CODES.O)
    public List<TravelDTO> getListFavorites() throws ParseException {...}
    @RequiresApi(api = Build.VERSION_CODES.O)
    public List<TravelDTO> getListFavoritesWithTransportType(String type){...}

    public void removeFavoriteById(Integer id) { dbFavorite.removeFavorite(id,getUserConnected()); }

    public void addFavorite(TravelDTO travel) { dbFavorite.addFavorite(travel,getUserConnected()); }
```

Figure 41: SQLiteMan class.

2.4. Graphical User Interface (GUI)

Screen design

In order for a user to access all the functions specified above, an Android application was created for the user to interact with. A series of screens have been designed to create an intuitive and easy to use app with the primary goal to retrieve data from all public transport in Dublin based on the user's search. The app also allows for the payment of the journey in a very convenient way and the allocation of a ticket in the form of a QR code.

The screens used in the app are:

Splash Screen -> When the Application is opened and is launching, the first start-up screen is Splash Screen with the logo, this gives the app a professional look while the app is getting the data leaving the user with a good impression of the application.

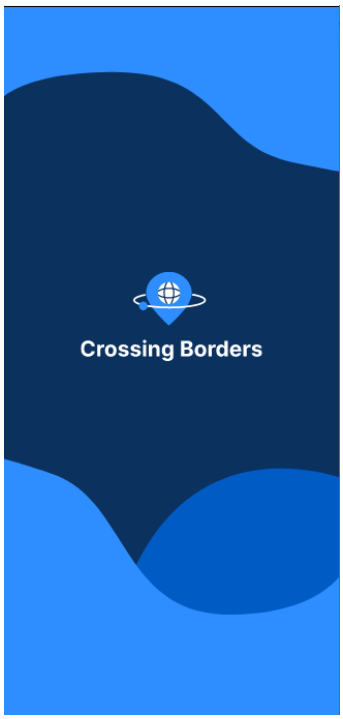


Figure 42: Splash Screen

Login Screen -> The user has the option to register by filling in their email and password for their registered account or to link their Google account. The google account is authenticated by validating the JWT token using a firebase backend.

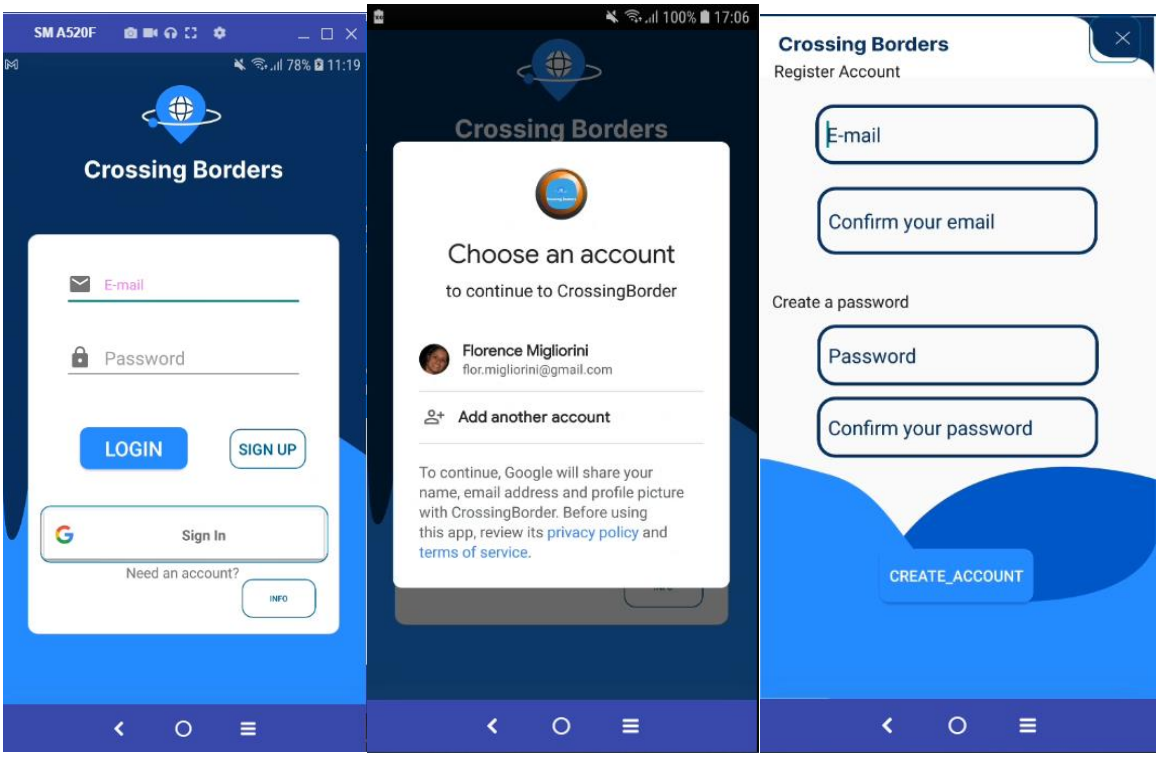


Figure 43: Login Screen

Main screen -> this screen is the most important of all since it is where the user enters their desired destination and then the available transport in the city appears. This screen will be accessed whenever the user wishes to get a transport option for their journey and to buy the associated ticket. The user can click on their preferred transport method and the results will be filtered accordingly.

This screen will appear after the user logs in, the geolocation will determine the city that the user is in, for this reason it is necessary to have GPS and WIFI activated. The app asks permission to allow to get the location. (Figure 44).

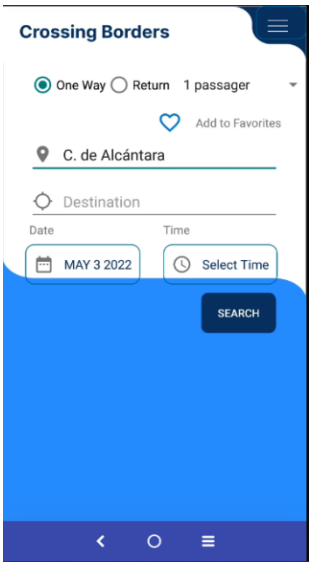


Figure 44: Main Screen

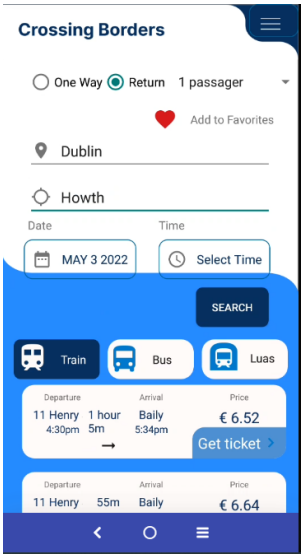


Figure 45: Home Screen displaying options

If for some reason the journey option has no route, the system displays a message, as shown in Figure 46.

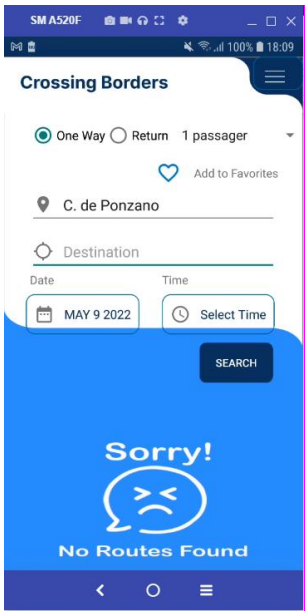


Figure 46: No routes Found.

When the user chooses the best option for their journey, the system displays the payment method.

Payment Screen -> This screen displays the ticket with the options chosen by the user. At this point, the user can choose to pay with their card details using stripe or pay with Google Pay using a payment option available in their digital wallet. After the payment is confirmed, the user will be re-directed to a confirmation screen to let the user know their payment was successful, Figure 49, then the system displays the QR Code. (Figure 48).

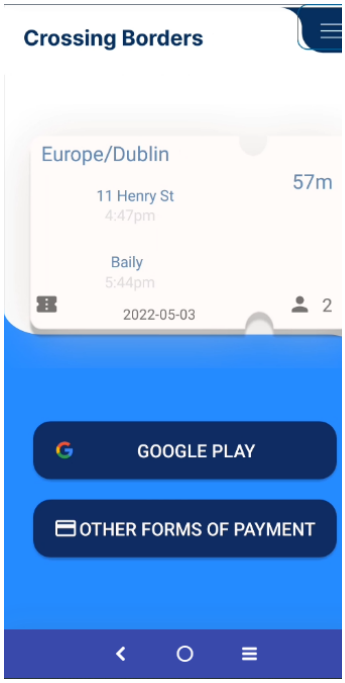


Figure 47: Payment Screen

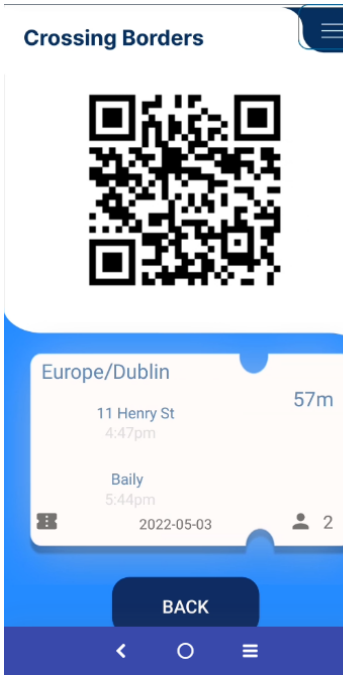


Figure 48: QR Code Screen

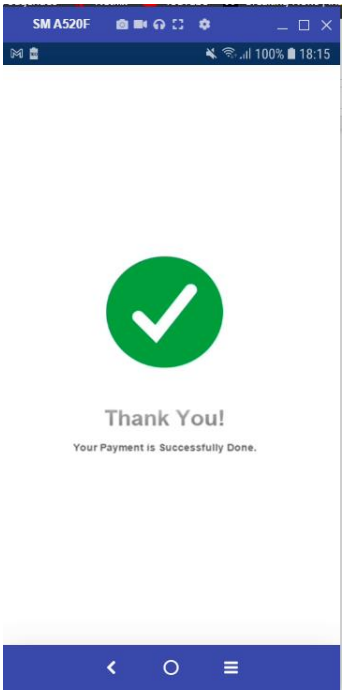


Figure 49: Confirm payment

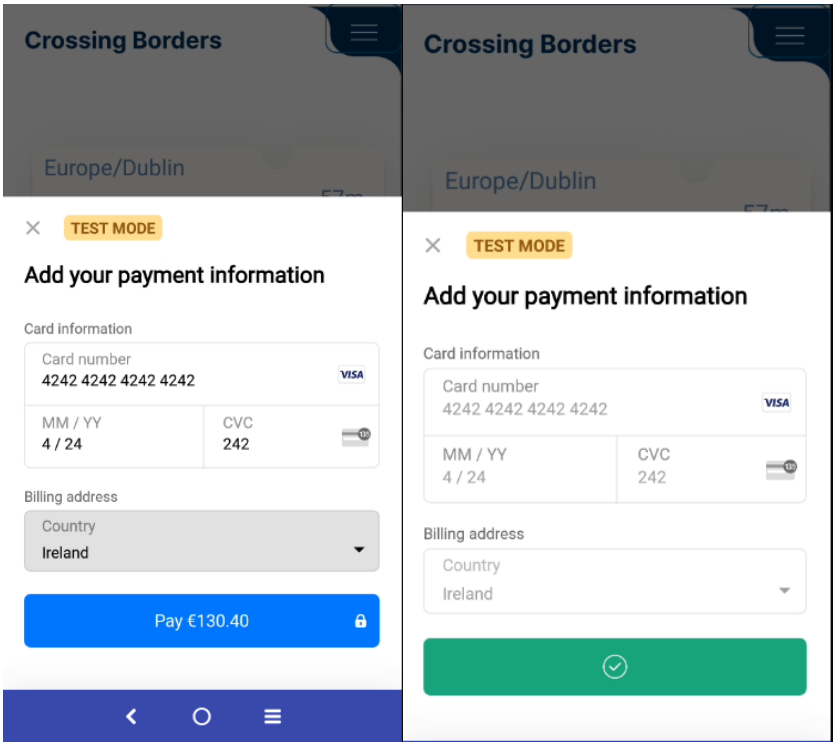


Figure 50: Card information.

Navigation screen (Menu) -> This screen displays a list of functions that can be accessed. The screen appears when a user clicks the menu in the upper right corner of the screen. The list of functions allows the user to check favourite routes, check travel history, search for tickets (returning to the main screen) and open a map with real-time location.

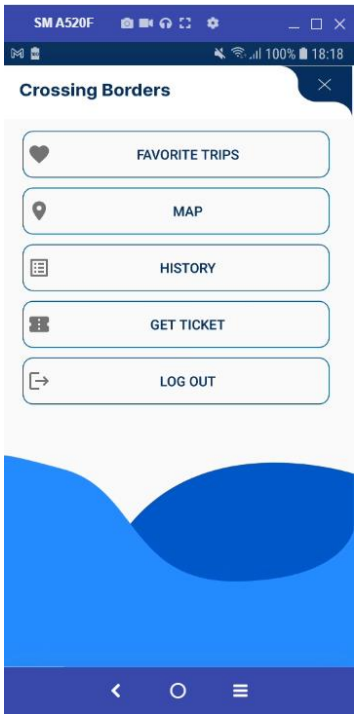


Figure 51: Menu screen

Favourite Screen -> Users have the option to save commonly used routes to their favourites, this screen can be accessed by clicking on the Menu option (upper right) and selecting favourites. This screen displays a list of all trips saved by the user. The user has an option to delete by clicking on the “Remove” button, and also the option to return to the Main screen, by clicking on the “Back” button.

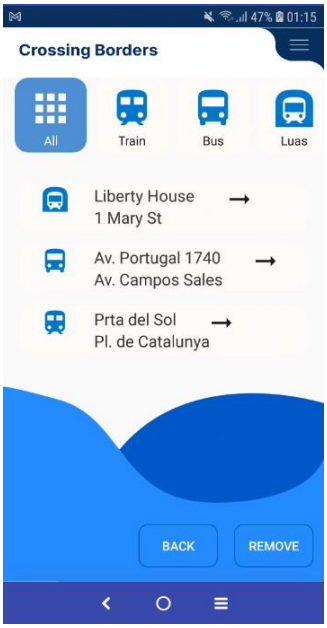


Figure 52: Favourite Screen

History Screen-> This screen will display a list of the last trips completed by the user. There are two buttons; “Remove” allows the user to delete the trip from their history, and the “back” button that returns the user to the main menu.

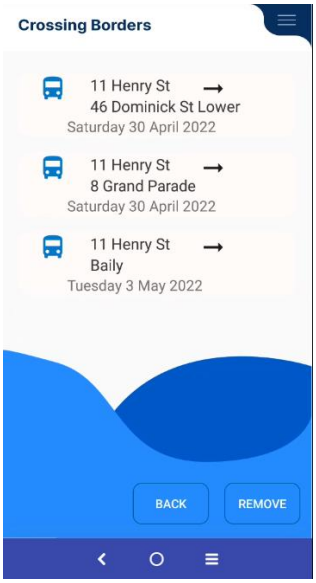


Figure 53: History Screen

Route Screen -> The purpose of this screen is to get the user's location in real-time and display it on the map, which the user can interact with to see nearby locations. Also, the user can search for an address by clicking on the Search button and clear the search bar by clicking on the Clear button as seen below. To leave that screen the user needs to click on the upper right to have access to the Menu and then pick an option.

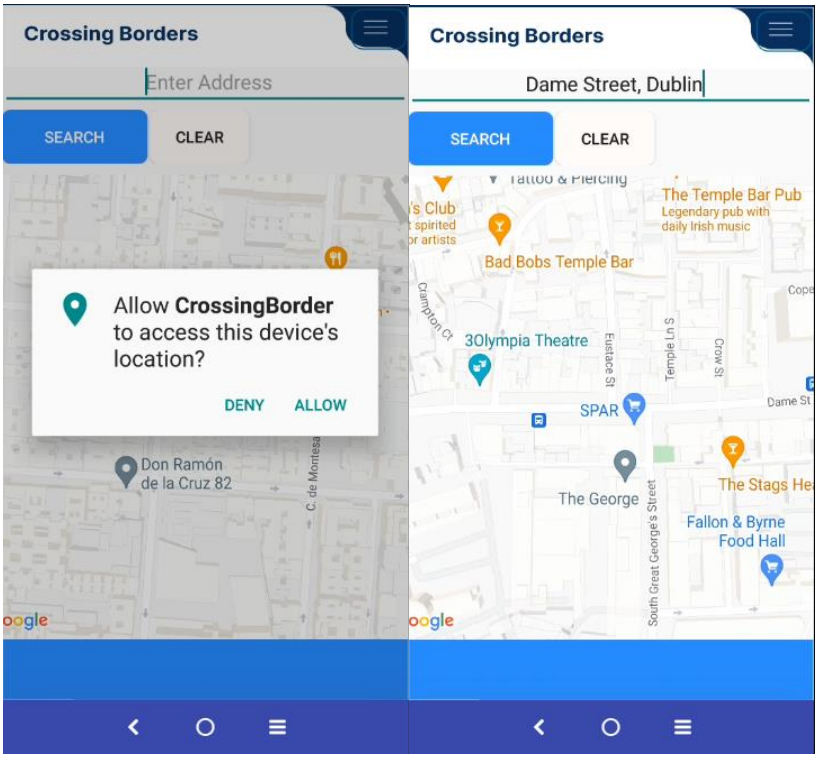


Figure 54: Route screen

2.5. Testing

To ensure that the system works properly, it is necessary to perform the relevant tests on the system. These tests will be carried out as the different features of the system are implemented.

Comprehensive testing is to be carried out on each of the functional requirements using a number of different testing strategies that are detailed further below. For a feature to be considered complete it must have passed each of the tests I have outlined, these include:

Vysor (an application to view and control your Android mobile from your PC through a Chrome extension) was used for testing that each use case worked as expected from the user's perspective. Vysor tests were completed regularly as new features in the code were implemented to ensure that new and older (previously working) features were working as expected.

An excel document was used to record the results of the Vysor (functional) tests as a historical document for the project.

Furthermore, JUnit was used to check that the methods in the code were functioning as intended, each of the core methods of the program was tested in JUnit to give the app good code coverage. Finally, Postman was used regularly throughout the development lifecycle to test the APIs using API calls.

2.5.1 User Testing

Functional testing ensures that all aspects of the software work correctly and in accordance with the project's requirements.

During these tests, valid and invalid data are inserted in order to validate the functionality of the application is working correctly.

To ensure that the initially proposed requirements were met, it's necessary to carry out tests on the mobile application, these tests were planned to validate its correct operation. All the results can be found in an excel table that is attached to this document, it contains Test Log, Test Plan, and Test Data. (The link to the Excel table)

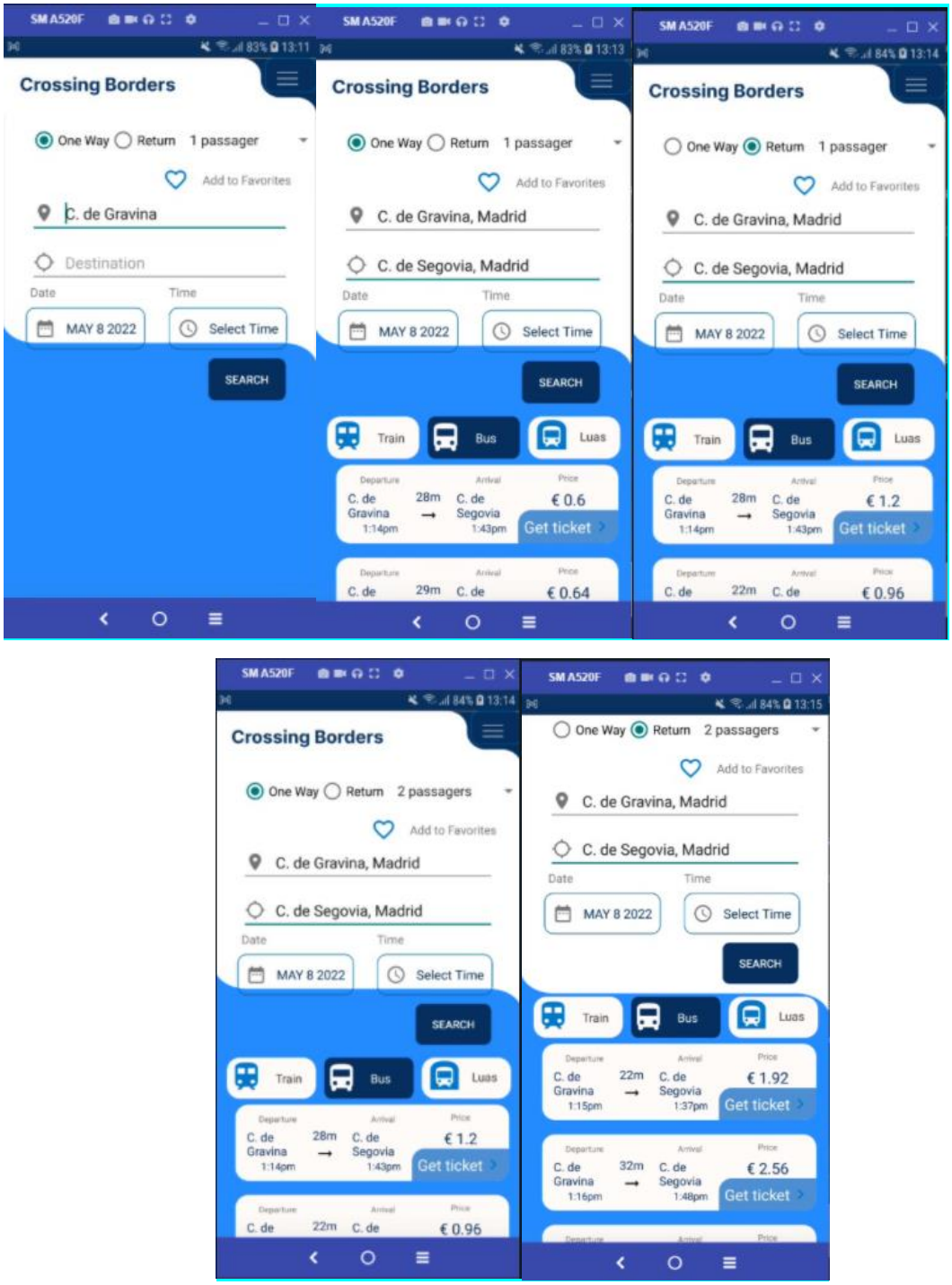


Figure 55: User tests

2.5.2 JUnit Tests

JUnit checks the functionality of classes and their methods. JUnit works on annotations and these annotations indicate whether a method is tested or not, and whether a method should be executed before the class and/or after the class. The annotations also indicate whether the test should be ignored or not and whether the class in question is a test suite, that is, whether the execution of the other test classes is triggered from this class, among other less used annotations.

ExampleInstrumentedTest class is where the JUnit tests can be found. Some tests were created to test the time and directions in the getRoutes() method, for login, save history, and favourites, where a new database is created named “databaseTeste”, a test for the QR code and stripe as well. A simple test was created useAppContext() method where I am testing the project package name.

-> assertEquals([message,] expected, actual): Tests whether two values (expected and actual) are the same. In the case of arrays, the check is against the reference and not the content.

```
@RunWith(AndroidJUnit4.class)
public class ExampleInstrumentedTest {
    private Context context;

    @Before
    public void useAppContext() {
        // Context of the app under test
        context = androidx.test.core.app.ApplicationProvider.getApplicationContext();
        assertEquals( expected: "florence.migliorini.crossingborder", context.getPackageName());
    }
    @Test
    public void timer(){
        Log.d( tag: "Junit Test", msg: "print");
        new Timer().schedule(() -> { Log.d( tag: "Junit Test", msg: "Teste"); }, delay: 500L);
    }
    @Test
    public void getRoutes(){...}
    @Test
    public void login(){...}
    @Test
    public void saveHistory(){...}
    @Test
    public void saveFavorite(){...}
    @Test
    public void createQrCodeWithZxing(){...}
    @Test
    public void configStripe(){...}
}
```

Figure 56: ExampleInstrumentedTest Class.

```java
@Test
public void getRoutes(){
    RequestQueue requestQueue= Volley.newRequestQueue(context);
    String API_KEY = "AIzaSyDfQVjDNvyjLXEj-6AqMHUaCK6ZTc45EeE";
    String urlBase = "https://maps.googleapis.com/maps/api/directions/json?";
    urlBase+="key="+API_KEY;
    urlBase+="&alternatives=true";
    urlBase+="&destination=Montreal";
    urlBase+="&origin=Toronto";
    urlBase+="&mode=transit";
    JsonObjectRequest jsonRequestTrain=new JsonObjectRequest(Request.Method.GET,
            url: urlBase+"&transit_mode=train",
            jsonRequest: null, new Response.Listener<JSONObject>() {
        @RequiresApi(api = Build.VERSION_CODES.O)
        @SuppressLint("ResourceAsColor")
        @Override
        public void onResponse(JSONObject response) {
            if(response!=null) {
                Gson gson=new Gson();
                DirectionsMainDTO modelMaps = gson.fromJson(response.toString(),DirectionsMainDTO.class);
                if(modelMaps.getRoutes().size()>0){
                    Log.d( tag: "Test Junit",modelMaps.toString());
                }
            }else {
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) { Log.d( tag: "Test Junit", msg: "Error"); }
    });
    requestQueue.add(jsonRequestTrain);
}
```

Figure 57: getRoutes() method

```java
@Test
public void login(){
    SQLiteMan.getInstance(context, databaseTag: "databaseTeste").logoutUser();
    String emailUser = SQLiteMan.getInstance(context, databaseTag: "databaseTeste").getUserConnected();
    System.out.println(emailUser);
    SQLiteMan.getInstance(context, databaseTag: "databaseTeste")
            .setUserConnected("TestJunit@gmail.com");
    emailUser = SQLiteMan.getInstance(context, databaseTag: "databaseTeste").getUserConnected();
    System.out.println(emailUser);
}
@Test
public void saveHistory(){
    TravelDTO travel = new TravelDTO( id: null, location: "Localizacao", destiny: "Destino"
            , LocalDate.now(), cdTransport: 1, duration: "Duration", value: 100,
            dtDuration: "Dt Duration", dsTitleTicket: "Titulo", dtHourDeparture: "Departure",
            dtHourTravel: "Tempo", numPassengers: 10);
    SQLiteMan.getInstance(context, databaseTag: "databaseTeste")
            .addHistoric(travel);
}
```

Figure 58: login() and saveHistory() methods.

```
    public void saveFavorite(){
        TravelDTO travel = new TravelDTO( id: null, location: "Localizacao", destiny: "Destino"
                , LocalDate.now(), cdTransport: 1, duration: "Duration", value: 100,
                dtDuration: "Dt Duration", dsTitleTicket: "Titulo", dtHourDeparture: "Departure",
                dtHourTravel: "Tempo", numPassengers: 10);
        SQLiteMan.getInstance(context, databaseTag: "databaseTeste")
                .addFavorite(travel);
    }
    @Test
    public void createQrCodeWithZxing(){
        MultiFormatWriter multiFormatWriter = new MultiFormatWriter();
        try{
            BitMatrix bitMatrix = multiFormatWriter.encode( contents: "CodedExample", BarcodeFormat.QR_CODE, width: 200, height: 200);
            BarcodeEncoder brEncod = new BarcodeEncoder();
            Bitmap bit = brEncod.createBitmap(bitMatrix);
            Log.d( tag: "APP",bit.toString());
        }catch(WriterException e){
            assertTrue( condition: false);
            e.printStackTrace();
        }
    }
    @Test
    public void configStripe(){
        ConfigStripe config = new ConfigStripe();
        Gson gson = new Gson();
        StripeKeyDTO strpe = gson.fromJson(config.getKey(Long.parseLong( s: "100"))
                ,StripeKeyDTO.class);
        Log.d( tag: "Test Junit",strpe.getPaymentIntentClientSecret());
    }
}
```

Figure 59: saveFavorite(), createQrCodeWithZxing(), and configStripe() methods

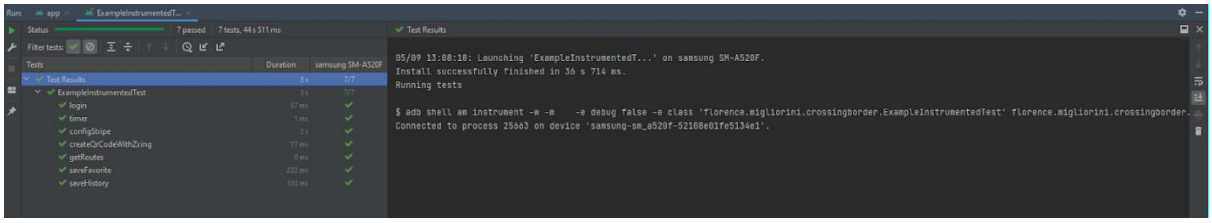


Figure 60: Test Results

2.5.3 POSTMAN - API Tests

In order to test RESTful services (Web APIs) I used postman to send HTTP requests and analyse their return. In Postman, it is possible to easily consume local and internet services, sending data and performing tests and gathering the response. A good reason to use this tool is that it can drastically reduce the time needed to test and develop APIs. Figure 61 below shows my workspace, where I created a collection called Dublin and Madrid, to test the APIs. Also, Figure 61 shows a "404 Not Found" request to the Dublin Bus API. And in Figure 62, the Dublin Rail return data.

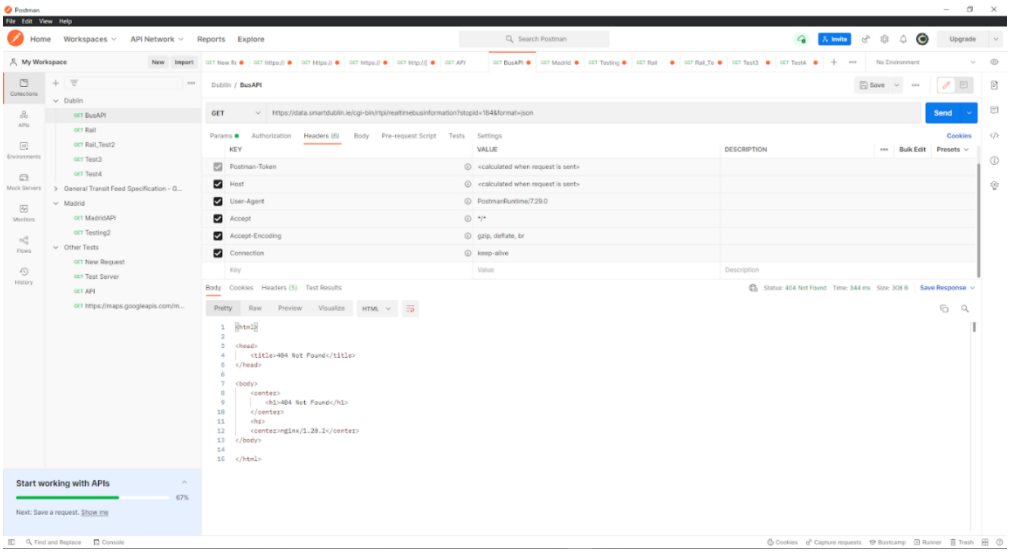


Figure 61: Dublin Bus Postman Test

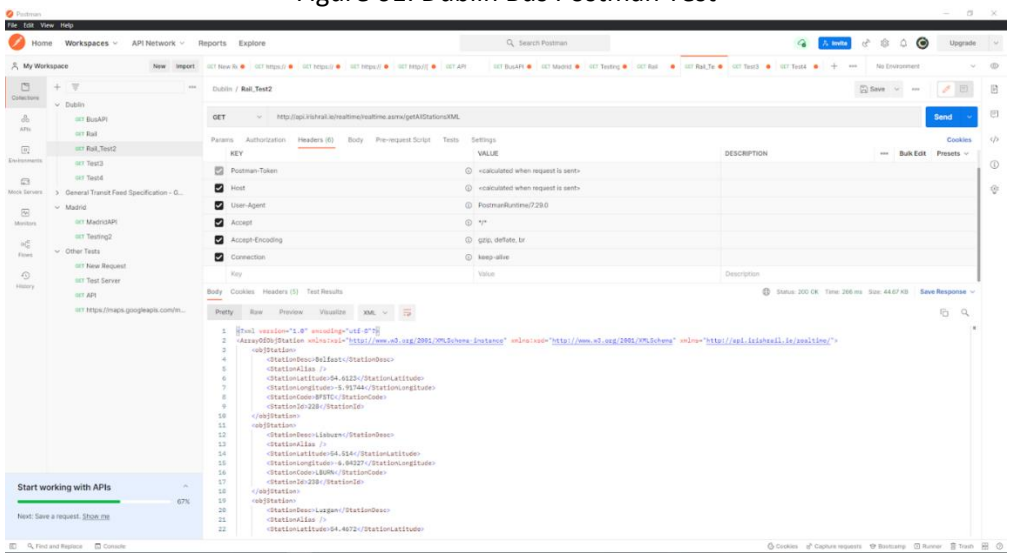


Figure 62: Dublin Rail Postman Test

2.6. Evaluation

Regarding the evaluation of this project, I believe I have accomplished many of the objectives that I had initially established. Initially, the project went as planned, following the schedule, but due to other projects and deadlines, I ended up getting lost in the development and delaying the schedule. Finally, I managed to implement the features, algorithms, UI design, and testing. In the first semester, I presented the prototype and documentation, the result of great effort and focus, getting positive feedback from my supervisor, and I felt excited to complete and deliver the application as promised.

I think the final product of my application has met the expectations of what I had initially set, and I have worked hard to meet my own expectations. The app, however, is not perfect, but it does what it promises. I believe that with more time, it would be possible to add more to the application. Although, I'm satisfied with the project I'm delivering.

3.0 Conclusions

Crossing Borders emerged from the intention of having an application to facilitate the daily life of those who use public transport. During the initial development of this project, it was intended to develop a service that manages the open transport data provided by the city of Dublin, Ireland, and an application that allows the consultation and purchase of the ticket. As I couldn't get the Dublin Public Transport Data API, I had to look for another alternative, which was to use Google's Directions API. With this, it was possible to obtain routes and thus meet the proposal of the initial idea in an alternative way. Although there are similar applications, these do not have the option to search for public transport, and the one that comes closest to the idea would be Omio, which does not work well in all countries. With that, I found a limitation, making this application a good idea, which allows you to have access to a travel ticket of your choice in a few clicks, and eliminate the use of cards and money, after all, nowadays, we use the cell phone for everything. In addition to having a technology, which is on the rise at the moment, the QR code.
I managed to increase my knowledge by using the client-server architecture using the exchange of RESTful requests as a form of communication, the use of SQLite, and also the use of libraries and frameworks that helped in the development of the project.

The app is not perfect, but I'm happy with the final result. One of the biggest challenges was adapting with the IDE and learning how to build the layouts. The goal was to develop a mobile app for Android that the user could search for routes and buy tickets and receive a QR code as a voucher, and that's exactly what is being delivered.
The university provided me with the necessary tools to carry out the project, not only in the coding part, but also in the project management part, essential lessons were provided for carrying out the project.
From the above conclusion, it can be stated that my project objectives were achieved, and I can say that it was possible to apply knowledge acquired in the course and learn new technologies, methodologies, and frameworks.

4.0 Further Development or Research

Concluding the work and taking a constructive self-criticism perspective, the following limitations and potential for future development are noted:

Add a profile, which the user can have control over their own profile, being able to add a photo, or change a password.

Plan options, such as for 3 or 4 days, full month ticket, etc.

Ongoing development and research will continue to add new features, such as an analysis of how to improve user location accuracy. And along with that making it a complete travel app.

5.0 References

Developers. 2021. Guide to app architecture. [online]
Available at: <https://developer.android.com/topic/architecture>
[Accessed in December 2021]

Developers. 2021. *Manifest.permission*. [online]
Available at: < https://developer.android.com/reference/android/Manifest.permission>
[Accessed in December 2021]

Developers. 2022. *Build location-aware apps*. [online]
Available at: <https://developer.android.com/training/location>
[Accessed in January 2022]

Developers. 2022. *Animatinos and Transitions*. [online]
Available at: <https://developer.android.com/training/animation>
[Accessed in May 2022]

Firebase.2021. Add Firebase to your Android Project. [online]
Available at:<https://firebase.google.com/docs/android/setup>
[Accessed in December 2021]

Firebase.2021. *Authenticate with Google on Android*. [online]
Available at:<https://firebase.google.com/docs/auth/android/google-signin>
[Accessed in December 2021]

Firebase.2022. *Fireabse Authentication*. [online]
Available at:<https://firebase.google.com/docs/auth>
[Accessed in April 2022]

Google Maps Platform. 2021. *Maps SDK for Android Quickstart*. [online]
Available at: < https://developers.google.com/maps/documentation/android-sdk/start>
[Accessed in December 2021]

Google Maps Platform. 2021. *Set up an Android Studio project*. [online]
Available at: <https://developers.google.com/maps/documentation/android-sdk/config>
[Accessed in December 2021]

Google Maps Platform. 2021. *Getting started with Google Maps Platform*. [online]
Available at: <https://developers.google.com/maps/get-started>
[Accessed in December 2021]

Google Maps Platform. 2022. *Directions Services*. [online]
Available at: < https://developers.google.cn/maps/documentation/javascript/directions>
[Accessed in April 2022]

Google Maps Platform. 2022. *Directions API overview*. [online]
Available at: < https://developers.google.com/maps/documentation/directions/overview>
[Accessed in April 2022]

Google Maps Platform. 2022. *Geolocation API*. [online]
Available at: <https://developers.google.com/maps/documentation/geocoding>
[Accessed in April 2022]

Google Maps Platform. 2022. *Geocoding request and response*. [online]
Available at: < https://developers.google.com/maps/documentation/geocoding/overview>
[Accessed in April 2022]

Stripe. 2022. *StripeDOCS*. [online]
Available at: <https://stripe.com/docs/google-pay>
[Accessed in April 2022]

Zxing. 2022. *Zxing-android-embedded*. [online]
Available at: <https://github.com/journeyapps/zxing-android-embedded>
[Accessed in April 2022]

Volley. 2022. *Volley overview*. [online]
Available at: <https://google.github.io/volley/>
[Accessed in April 2022]

Gson. 2022. *Gson User Guide*. [online]
Available at: <https://sites.google.com/site/gson/gson-user-guide>
[Accessed in April 2022]

Datasets. 2022. *Datasets* [online]
Available at:<https://data.gov.ie/dataset?theme=Transport&api=true>
[Accessed in April 2022]

EMTMadrid. 2022. MobilityLabs API References. [online]
Available at:<https://apidocs.emtmadrid.es/#api-reference-for-a-submenu-entry>
[Accessed in April 2022]

6.0 Appendices

6.1. Project Proposal

Executive Summary

The purpose of this document is to provide an overview of the technical details, explaining in detail the functionality of the application. As the project is in the middle of development, it is not possible to explain in detail how the implementation will be done.

The key focus for the preparation of this documentation is in the requirements part, which I tried to explain in detail.

1.0 Introduction

1.1 Background

I wanted to create an app to connect all forms of public transport in one simple and easy to use app in the hands of the user. This app would eliminate the need for extra cards or tickets as payment, would be completed digitally (Google Pay, Stripe) and a QR code would be used as proof of purchase.

Transport is something that I'm very interested in but sometimes when travelling in other EU cities I find it difficult to know what public transport to use and what area each different ticket covers. This is when I came up with an idea for an all-in-one public transport app that can be used across multiple cities in the EU. This would prevent me from being unsure when travelling in other cities and would also be a very useful tool for me to use everyday getting to work or college. Going all digital and avoiding unnecessary cards and tickets is something I've always wanted for public transport.

I wanted to create a mobile app because of its usefulness in everyday scenarios. Our phones are almost always next to us at any given time, making them a great tool to develop for. Mobile app development is also completely new to me which will provide a great challenge for me and I'm looking forward to seeing how my mobile app development skills develop during the project. I think this project offers me a great opportunity to do extra research and complete tasks that I would otherwise not do as part of my module.

1.2 Aims

Crossing Borders is a mobile application that aims to streamline the user's experience when taking public transport. The app's goal is to help facilitate a quick and comprehensive way to travel. The app will be available for use on multiple modes of transport across a few selected cities in the European Union. The city I will base my project on is Dublin and afterward, I will look to implement the app in other cities. The modes of transport I will be targeting for this app in Dublin are the Luas, Dart, and Dublin Bus.

1. The main objective of this project is to develop a mobile application that can facilitate day-to-day travel in an intuitive way. This is true domestically as well as for international travel. The app will work in the same way when in another supported city in the European Union. This means there is no extra stress when taking transport in an unfamiliar part of Europe.

2. The app will make daily traveling more convenient for users by helping them save time for their routine commutes to work or school. Every step of the user's daily commute is handled through the app. The user can purchase tickets for their preferred method of travel completely digitally using either PayPal, Google Pay, or Stripe. This means the user does not have to carry around any extra cards or a physical ticket to travel, Instead, the user will receive a QR code that lasts for the duration of their trip.

3. Tourism is a large industry across the EU and this app is aiming to make transport as easy as possible for tourists. The route planner calculates multiple ways for the user to get from their starting point to a designated destination. This will allow tourists to easily figure out what public transport to take, how long the journey will last and what the cost of the trip will be. Purchasing of tickets takes place on the app so there is no need to deal with a convoluted ticket machine UI.

1.3 Technology

The Crossing Borders application will be a mobile application that works on Android devices, as it is an open-source language. The app is developed using the android studio IDE with the Java development language. This tool also includes an SDK (Software Development Kit) that helps in the development of certain applications, since it has several packages such as Google APIs.

Google API

Since this application needs to display maps and locations on maps, it is necessary to use the Google Maps API. This API allows you to place maps in the application interface. In addition, it allows you to obtain the location of the mobile device and add points on the map, moving along the longitude and latitude. This API is free, but it only allows you to make 2,500 requests per day. If you want to make additional requests every day, you must upgrade to Google Maps for Business API.

Additionally, the Google Sign-In and Google Pay API will also be used. Google Sign-In API works by generating a JWT token with the Google Profile data and metadata attached that is validated using a backend Firebase authenticator. Google Pay is used as an option for payment in the app and works similarly to the Google Sign-In API. Google Pay sends a payment token with details about the purchase to the app which is then processed by the backend and a payment token is sent to the payment service provider.

Dependencies

To develop the entire implementation of the system, it is necessary to use a series of libraries that will help in the development of the system.

Firebase to serve as a backend.

Android SDK to run devices and emulators.

Google Mobile Services (GMS) -> APIs that help functionality across the device.

Stripe will be offered as an alternative method for payment. Stripe works by creating a source object that stores the user's payment information. Next Stripe checks if any further action is required such as payment verification from the user’s payment source. Once the payment source is ready to be used a charge request is completed in the backend using the source object information.

SQLite database to ensure that data is persisted for users to access when they use the app.

1.4 Structure

1.4.1 Introduction

Background – In this section I explain my reasoning behind choosing my project idea and how that idea came to be. I also discuss why I decided to use the technologies that I did and how it’ll benefit the project. Finally, I discussed why I think this app is important and how it will challenge me.

Aims – This section addresses the overall aims and objectives of the project and represents what I would like to have accomplished by the end of the development cycle.

Technology – Here, I briefly discuss the technology that I plan to use.

1.4.2 System

Requirements – This section encompasses all the different requirements/functionality of the proposed project. This includes all the functional and non-functional requirements, which have been split into different headings to address each one individually. Use cases have been designed to illustrate and further explain each requirement and the flow of operations.

Design & Architecture – Here, I list all the screens and display them all in a structure chart to highlight the apps design and how to navigate between the different screens. I have also included a system architecture diagram to show the relationship between the app, database, and required components such as API's.

Implementation – This is where I took some snippets of my core functionalities and elaborated on how they were developed.

Graphical User Interface – I provided screenshots of my GUI and explained the process of how the screens were designed and created using Adobe.

Testing – From the planning stage, I had testing in mind and in this section, I explain how testing was a constant in the project. Whether that be unit testing, regression testing GUI testing or black box testing.

1.4.3 Conclusions

In this section I draw my final conclusions about the project development lifecycle from the initial planning stage all the way to the deployment of the app. I reflect on what I accomplished and what I would do differently had I the chance to work on the project again with more experience and development time.

1.4.4 Further Development or Research

Finally, I discuss additional features that I would have liked to develop but were beyond the scope of the project. If I was to continue with more time and resources these are the features I would like to complete.

2.0 System

2.1 Requirements

In this section, the requirements analysis will be performed to determine the needs to be satisfied for the application creation. First, the functional requirements that describe the functionality that the system must provide will be discussed. Second, the non-functional requirements that define the emerging properties of the system, such as response time, reliability, etc., will be detailed.

2.1.2 Functional Requirements

This section defines the functionality of the system. In each of the features will be detailed what to do, the criteria for customer satisfaction. The system features are as follows:

Requirement 1: Login

Requirement 2: Single transport journey

Requirement 3: Multiple transport journey

Requirement 4: Payment

Requirement 5: Favourites

2.1.3 Use Case Diagram

The use-case diagram, which represents the operations the user can perform on the system

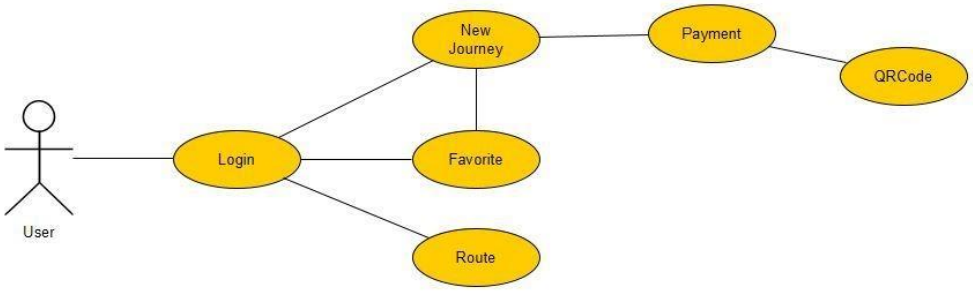


Figure 1: Use Case

Use Case – Requirement 1: Sign-In

Scope

The scope of this use case is to verify the actor's google account using a backend firebase server to authenticate the actor's ID token and credentials obtained from the Google Sign-In API.

Description & Priority

The Google Sign-In API will be used to identify users and assign them a JWT access token that can be used to access the app's features while the token is still active. After a JWT token has expired the token will need to be refreshed to grant access to the app again. Initially, when a user signs in a GET request is made to retrieve the JWT token and then the ID token must be sent to the firebase backend using a POST request to validate the integrity of the token. If a token is valid a HTTP 200 successful response will be returned, and the user will be redirected to the home screen where they can access the main menu and create a new journey. The JWT token will be stored in local storage and will contain important meta data for the token's validation as well as profile information for the user. This profile information can be used to retrieve the user's name, email, and Google profile picture.

This requirement is a CSF (critical success factor) for the project and acts as a gateway to allow users access to the core functionality of the app. This requirement provides the app with secure authentication using the Google API Sign-In, ensuring that user accounts are secure and validated.

Technical issues

I have initial worries about implementing the API with the firebase backend to enable the validation of the JWT tokens. I have read the documentation of the Google Sign-In API but I'm still unclear about some of the concepts around the token validation such as the token expiry and refresh and how they will be reflected back to the user.

Dependencies with other requirements

This use case is critical as it's a requirement for all other use cases in the application. Before the actor can access the app, they must be issued a valid and authenticated JWT token. While a JWT token is active the user will be automatically signed in allowing them access to the other services.

Use Case Diagram

Figure 2: Login use case

Precondition

The actor must have a pre-existing google account to sign in and access the apps services.

Activation

The actor clicks on the google sign in button and selects their google account.

Main flow

1. The actor opens the app
2. The actor clicks on the Google Sign-In Link
3. The actor selects their google account
4. The system retrieves the JWT token using a GET request
5. The system sends the ID token to the firebase backend using a POST request
6. The system validates the ID token using the tokens metadata to ensure the integrity of the token

7. The system sends back a HTTP 200 successful request
8. The actor is redirected to the home screen

Alternate flow

A1: Active JWT token found

1. The actor opens the app
2. The JWT token is found in local storage
3. The use case continues at step 5 of the main flow

Exceptional flow

E1: User does not have a google account

1. The actor opens
2. The actor clicks on the Google Sign-In Link
3. The actor does not have a google account

4. The system prompts the actor to create a Google account before continuing

Termination

The system redirects the actor to the home screen.

Post condition

JWT token is stored in local storage to be retrieved by the system the next time the actor opens the app (as long as the token is still valid).

The actor's Google profile information is stored as JSON data in the JWT token and can be retrieved and displayed by the system to the actor.

The actor must choose what service they would like to choose next for example, view favourites or start a new journey.

Use Case – Requirement 2: Single Transport Journey

Scope

This use case will allow the user to create journeys that are possible using only one transportation service.

Description & Priority

The actor begins by entering a starting and ending location for the journey and then clicking a button to search for possible routes. The button press starts an event that retrieves the data the actor entered

into the search bars and uses that information to search the database. As it is early in development, I still have not decided how to base my searches, whether it be by location name or geolocation. This will be decided when I start development on this feature and will probably be heavily influenced by which method is easier to connect multiple forms of transport as this will be the most complex part of my project. I anticipate I will need to implement data structures and algorithms to make this possible. For a single transport only journey the information gathered will be used to see if a route can be found in either the dart, luas, or bus alone. The dart and luas are relatively straightforward, on the other hand the bus data will need to be divided by bus number to ensure that only one bus is required to be taken for the journey. For example, the 27 bus from Tallaght to Town. If a route was found it is returned to the actor.

This requirement is a CSF (critical success factor) for the project and represents the core functionality of the project. I believe use case 2 and 3 will take critical priority for the project and will account for a large majority of the development time. Use case 3 in particular I believe will be very complex and difficult to achieve.

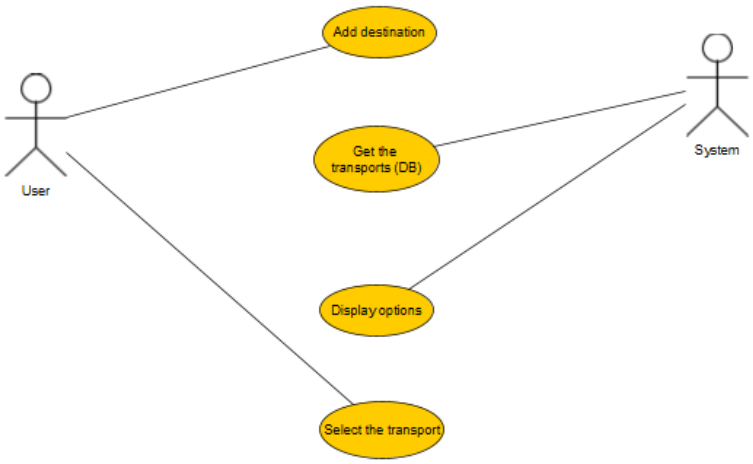
Technical issues

Dublin is a large city with many different small and large towns. As a result of this the search parameters are very large. An issue I may face is the quantity of data present in the Dublin Bus API and the specificity of that data. The data relates to bus stops which in the majority of cases have a street location. This is fine if users end up searching by street, but this will not always be the case as it'll be more common for a user to search by town instead, for example Tallaght. As a result, it may be necessary to associate bus stop locations with their surrounding area. It may also be possible to do this using geolocation coordinates as the bus stop data has a longitude and latitude associated with each bus stop. I will need to explore the possibilities while developing this feature in the next semester.

Dependencies with other requirements

Firstly, the actor must have signed in using their google account to access this feature. This use case is required in order to develop the favourites and payments use cases.

Use Case Diagram



Precondition

The actor must have a starting location and destination in mind for their journey

Activation

The actor searches for a new journey using a start location and destination.

Main flow

1. The actor enters in a start location and destination for the journey
2. The actor searches for the journey
3. The system checks the database for a matching starting point and destination
4. The system checks to see if the starting point and destination are both a part of the same transportation method
5. The system returns the transportation method
6. The actor selects the transportation method returned by the system

Alternate flow

A1: See Multiple Transport Journey Use Case

Exceptional flow

E1: User does not have a google account

1. The actor enters in a start location and destination for the journey
2. The actor searches for the journey
3. The system checks the database for a matching starting point and destination
4. The system doesn't find a matching pair in the database

5. The system prompts the actor to enter another journey
6. The actor begins the process again

Termination

The system will present a list of possible transport routes that the actor can take for their journey, the routes will be listed in order of shortest journey duration.

Post condition

The actor chooses one of these routes based on a number of factors such as time, cost, and convenience.

Once the actor has chosen their preferred route, they can choose to add the route to their favourites or simply continue to payment.

Use Case – Requirement 3: Multiple Transportation Journey

Scope

This use case will allow the user to create journeys that require using multiple transportation services.

Description & Priority

The actor begins by entering a starting and ending location for the journey and then clicking a button to search for possible routes. The button press starts an event that retrieves the data the actor entered into the search bars and uses that information to search the database. As it is early in development, I still have not decided how to base my searches, whether it be by location name or geolocation. This will be decided when I start development on this feature and will probably be heavily influenced by which method is easier to connect multiple forms of transport as this will be the most complex part of my project. I anticipate I will need to implement data structures and algorithms to make this possible. If it's not possible to complete a journey using one transportation method, then a combination of multiple transportation methods will be needed to complete the journey. At maximum three transportation methods can be connected as I believe the algorithm and search operations taking place on the database may have a performance effect on the app after this point. I think it will be easier to work backwards finding first the list of possible transport methods that will arrive at the destination point and assuming from there the best way to arrive at the starting point. This may be possible by checking all the prior stops and confirming if the coordinates are in an acceptable distance from another transportation service. If the journey is possible with 3 or less transportation methods the results are sent back to the user.

This requirement is a CSF (critical success factor) for the project and represents the core functionality of the project. I believe use case 2 and 3 will take critical priority for the project and will account for a large majority of the development time. Use case 3 in particular I believe will be very complex and difficult to achieve.

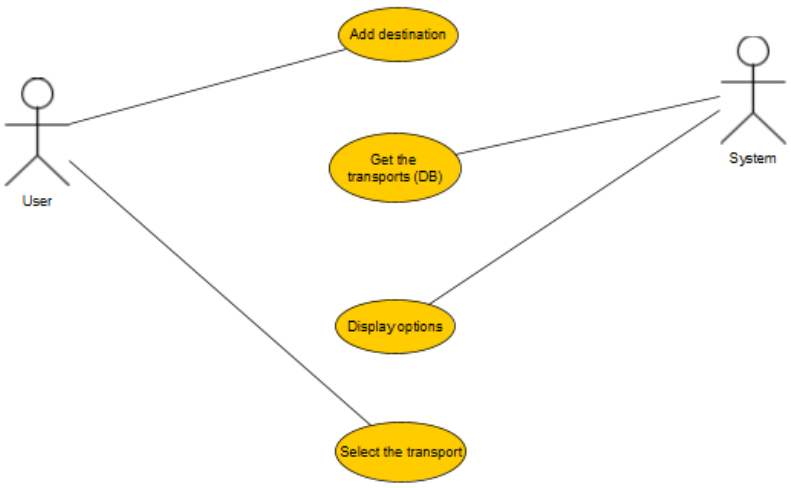
Technical issues

A journey can be very complex when joining multiple forms of transport together deciding when and how to connect these transports will be very difficult to achieve and I anticipate that the algorithm will be very complex and intensive as the number of possibilities available is huge. I hope that these operations don't have serious negative effects on the performance of the app. One way I'm thinking to counter this is by evaluating whether a stop's coordinates value represents a + or a – compared to the current coordinate value of a stop. If the value of the starting position is positive relative to the ending point then all negative values can be ruled out as they're going in the wrong direction.

Dependencies with other requirements

Firstly, the actor must have signed in using their google account to access this feature. This use case is required in order to develop the favourites and payments use cases.

Use Case Diagram



Precondition

The actor must have a starting location and destination in mind for their journey

Activation

The actor searches for a new journey using a start location and destination.

Main flow

1. The actor enters in a start location and destination for the journey
2. The actor searches for the journey
3. The system checks the database for a matching starting point and destination

4. The system checks to see if the starting point and destination are both a part of the same transportation method

5. The system confirms this is not possible

6. The system finds all transport routes that have the destination specified by the actor

7. The system runs an algorithm to identify all transport routes that connect to the destination list of routes

8. The system identifies all stops that trend in the same direction as the starting point using coordinates.

9. The system identifies the transport method that gets the actor closest to their starting point

10. If required the system repeats steps 8 and 9 one more time

11. If the result gets the user to an acceptable distance from their starting point the system accepts the transport route.

12. The system returns the transportation method

13. The actor selects the transportation method returned by the system

Alternate flow

A1: See Single Transport Journey Use Case

Exceptional flow

E1: User does not have a google account

1. The actor enters in a start location and destination for the journey

2. The actor searches for the journey

3. The system checks the database for a matching starting point and destination

4. The system doesn't find a matching pair in the database

5. The system prompts the actor to enter another journey

6. The actor begins the process again

Termination

The system will present a list of possible transport routes that the actor can take for their journey, the routes will be listed in order of shortest journey duration.

Post condition

The actor chooses one of these routes based on a number of factors such as time, cost, and convenience.

Once the actor has chosen their preferred route, they can choose to add the route to their favourites or simply continue to payment.

Use Case – Requirement 4: Payment

Scope

Allows the actor to pay for a journey including all methods of transportation in one transaction. As a proof of purchase the actor will receive a QR code that will expire after a certain amount of time.

Description & Priority

The user selects their journey and selects the get ticket button which redirects the actor to the payment screen. The system calculates the cost of the trip using an algorithm, taking into account the duration and transportation methods of the journey. The price, starting point and destination are all included on the ticket. If the ticket is acceptable for the actor, they can choose to pay using either Stripe or Google Pay. Google Pay is used as an option for payment in the app and works similarly to the Google Sign-In API. Google Pay sends a payment token with details about the purchase to the app which is then processed by the backend and a payment token is sent to the payment service provider. Stripe will be offered as an alternative method for payment. Stripe works by creating a source object that stores the user's payment information. Next Stripe checks if any further action is required such as payment verification from the user's payment source. Once the payment source is ready to be used a charge request is completed in the backend using the source object information. After payment is confirmed the system sends a QR code to the user as proof of purchase for the journey.

This use case is a CSF for the project and as a result has a very high priority level. A payment feature for an app like this is expected from users and the absence of this feature would put the app at a big disadvantage to competitors and alternative travel options.

Technical issues

Two different payment methods are accepted: Stripe and Google pay. Both are implemented using an API. I will need to follow the documentation to implement these features. Stripe is primarily used for web apps and as a result I may need to make additional changes compared to a web application.

Dependencies with other requirements

Firstly, the actor must have signed in using their google account to access this feature. The requirement to search for a new journey will be required and the add to favourites use case is an optional requirement that will help speed up the process of this use case.

Once a payment has been made and the QR code has expired the journey will be added to the history screen.

Use Case Diagram

Figure 3: Payment use case

Precondition

The actor selects a route and their preferred method of transportation(s)

Activation

The actor clicks on the get ticket button which redirects the actor to the payment screen.

Main flow

1. The actor clicks get tickets on their selected journey
2. The actor clicks on the Google Pay icon
3. The actor selects a payment method
4. The system gets a payment token from Google pay
5. The system sends the payment token and payment details to the apps backend
6. The system processes the purchase using the payment details in the payment token
7. The system sends the payment token to the payment service provider
8. The system sends confirmation of the purchase to the user
9. The system generates and sends a QR code for the journey

Alternate flow

A1: Pay using Stripe

1. The actor clicks get tickets on their selected journey
2. The actor enters their card details and clicks Pay
3. The system creates a source object with the payment details entered by the actor
4. The system checks if any further action is required by the actor
5. The actor completes the extra action if required
6. The system completes a charge request in the backend
7. The use case continues at step 8 of the main flow

Exceptional flow

E1: Invalid payment option

1. The actor clicks get tickets on their selected journey
2. The actor clicks on the Google Pay icon
3. The actor selects a payment method
4. The system gets a payment token from Google pay
5. The system sends the payment token and payment details to the apps backend
6. The system processes the purchase using the payment details in the payment token
7. The system can't verify the purchase using the payment details provided in the payment token.
8. The system sends an error message to the actor and prompts them to try again

9. The actor starts the payments process over again

Termination

The system confirms the payment and sends the actor the QR code for their journey.

Post condition

The system will generate a QR code that is active for the course of the journey and add it to the actor's active journeys.

The actor can use the QR code as proof of purchase or to scan at security gates to gain access to the transportation.

Use Case – Requirement 5: Add to Favourites

Scope

The scope of this use case is to implement a feature for the actor to save a trip to their favourites, allowing them to save time for their most regularly searched trips.

Description & Priority

The favourites list will act as a quick and convenient way for the actor to access their most frequently used journeys. The actor can search for a journey and add it to their favourites by clicking on the favourite icon located next to the journey information. This button press will serve as the trigger for the event to add the journey to the actors' favourites. The system will then collect the associated information from the journey and the actor (token profile information) and insert them to the favourites table using the database connection that has been established. If all the data is as expected the data will be inserted into the favourites table and the system will send back a confirmation message to the actor. The actor will be able to view the newly added favourite journey in their favourites.

Out of the 5 core functionalities, I believe this requirement has the lowest priority and as a result I will tackle this requirement last.

Technical issues

I will need to ensure that the method of relating the data to the user works as intended and that the search operation performed on the table does not affect the overall performance of the app.

Dependencies with other requirements

Firstly, the actor must have signed in using their google account to access this feature. Once a trip has been added to the actors' favourites, they will be able to quickly create that trip and pay for their ticket without having to search.

In terms of development use cases 1-3 will need to be finished before development can begin on this use case. Sign in is required to generate the foreign key in the favourites database table to connect the user to their favourites. Also, the requirement to search for a new journey will be required.

Use Case Diagram

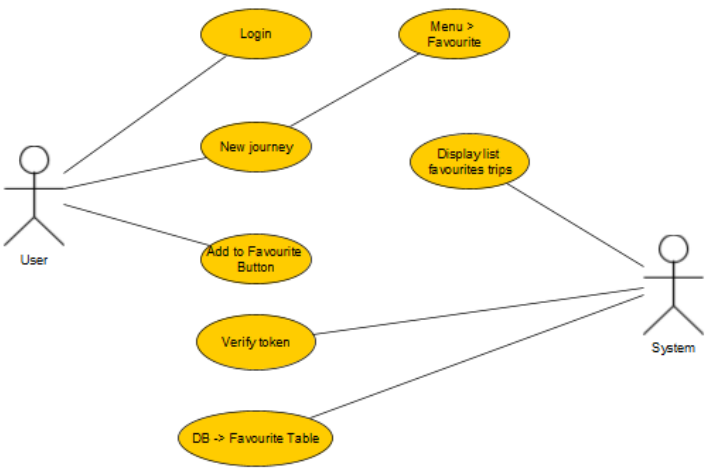


Figure 4: Favourite use case

Precondition

The actor must have a pre-existing google account to sign in and access the apps services.

The actor must search for the journey that they wish to favourite.

Activation

The actor clicks on the favourite icon next to the journey.

Main flow

1. The actor searches for a journey with a start and end point
2. The actor clicks on the icon to favourite the journey
3. The system gathers the associated data into an object
4. The system inserts the object into the database favourites table
5. The system notifies the actor that the journey was added to their favourites
6. The user checks their favourites to verify the journey has been added to their favourites list
7. The system gets the favourites data associated with the actor's email
8. The actor confirms the new journey has been added to their favourites

Alternate flow

A1: Get current location as start point

1. The actor searches for a journey with an end point only
2. The system generates the start point from the user's current geolocation
3. The use case continues at step 2 of the main flow

Exceptional flow

E1: User enters an invalid start or end point

1. The actor searches for a journey

2. The system searches the routes table but can't find any occurrence of the start or end point

3. The system prompts the actor to enter another location as no results were found

Termination

The system adds the trip to the actors' favourite trips.

Post condition

The system will add the trip to the actors' favourite trips list. The trip will be stored in a database table called favourites which will have a one-to-many relationship with the user table to ensure the correct user data is retrieved.

The actor will be able to view their favourite trips with the newly included trip added to the list.

2.1.4 Data Requirements

2.1.4.1 Open data

Most cities provide data about public transport and transport stations in their cities in certain formats. The data offered by the custom cities have a similar attribute in common which is the exact location of each stop, so there are other attributes that depend on what each city wants to offer. In this section, the formats provided by the Dublin government will be analysed.

2.1.4.2 Public data

Dublin -> The open data offered by the city is available on the Data.gov.ie website, in various formats.

Below is a screenshot from the website to demonstrate all the options available, see figure 5.

Figure (5): Public data Formats

An example of one of these data is shown in the figures below. In the illustration 6 you can see the data offered for buses in xlsx.

A	B	C	D	E	F
ShapeId	Operat	StopSequenc	RouteNan	RouteDescription	Directi
10-100-e19-1.253.O	BE	1	100	Strand Street - The Long Walk	O
10-100-e19-1.253.O	BE	2	100	Strand Street - The Long Walk	O
10-100-e19-1.253.O	BE	3	100	Strand Street - The Long Walk	O
10-100-e19-1.253.O	BE	4	100	Strand Street - The Long Walk	O
10-100-e19-1.253.O	BE	5	100	Strand Street - The Long Walk	O
10-100-e19-1.253.O	BE	6	100	Strand Street - The Long Walk	O
10-100-e19-1.253.O	BE	7	100	Strand Street - The Long Walk	O
10-100-e19-1.254.O	BE	1	100	Monasterboice - The Long Walk	O
10-100-e19-1.254.O	BE	2	100	Monasterboice - The Long Walk	O
10-100-e19-1.254.O	BE	3	100	Monasterboice - The Long Walk	O
10-100-e19-1.254.O	BE	4	100	Monasterboice - The Long Walk	O
10-100-e19-1.254.O	BE	5	100	Monasterboice - The Long Walk	O
10-100-e19-1.254.O	BE	6	100	Monasterboice - The Long Walk	O
10-100-e19-1.254.O	BE	7	100	Monasterboice - The Long Walk	O
10-100-e19-1.254.O	BE	8	100	Monasterboice - The Long Walk	O
10-100-e19-1.254.O	BE	9	100	Monasterboice - The Long Walk	O
10-100-e19-1.254.O	BE	10	100	Monasterboice - The Long Walk	O
10-100-e19-1.254.O	BE	11	100	Monasterboice - The Long Walk	O
10-100-e19-1.254.O	BE	12	100	Monasterboice - The Long Walk	O
10-100-e19-1.254.O	BE	13	100	Monasterboice - The Long Walk	O
10-100-e19-1.254.O	BE	14	100	Monasterboice - The Long Walk	O
10-100-e19-1.255.O	BE	1	100	Drogheda Bus Station - The Long Walk	O
10-100-e19-1.255.O	BE	2	100	Drogheda Bus Station - The Long Walk	O
10-100-e19-1.255.O	BE	3	100	Drogheda Bus Station - The Long Walk	O
10-100-e19-1.255.O	BE	4	100	Drogheda Bus Station - The Long Walk	O
10-100-e19-1.255.O	BE	5	100	Drogheda Bus Station - The Long Walk	O

active_route_sequences_report_2

Figure (6): Bus data in XLSX

2.1.4.3 Data Formats

In the previous section I analyzed the different data formats that Dublin city has made available to developers like myself. I will now highlight and explain some formats, which I will consider using in the next semester.

API -> defines a REST-based protocol that allows data to be queried, updated, and exposed using standardized syntax from a wide variety of data stores. API provides access to different forms of data using technologies such as HTTP, XML or JSON. This protocol provides metadata that provides an electronic description of the data model.

CSV (Comma Separated Values) -> is an open format for representing data in a tabular form. This data format structures the data using commas to separate values.

GEO is a microformat used to refer to geographic coordinates in HTML, this is achieved by getting the longitude and latitude values. To do this, you just need to define a section in the HTML that the geo class uses to indicate the latitude and longitude within the section.

XML Extensible Markup Language -> is a plain text format developed by the World Wide Web Consortium (W3C). This format structures information into a prologue describing the XML version, document type, etc. Additionally there are tags of type <name>, where name refers to the referenced element. These elements, in many cases, can contain attributes that provide information about the characteristics of a particular element.

KML Keyhole Markup Language -> is a format used to display 2D or 3D geographic data. KML files can be viewed in geospatial information software such as Google Earth or Google Maps. The KML format uses XML grammar to encode and transport data; therefore, it uses a tag-based structure with nested elements and attributes.

This format contains information such as latitude, longitude and relevant information about a place. In addition, it can support data for user navigation control in order to direct a user on where to go and where to search.

XLS is an extension of a spreadsheet used in the Microsoft Excel program. An XLS file displays information in cells arranged in rows and columns. In addition, cells can contain formulas or references to other cells.

2.1.5 User Requirements

1. An Intuitive design that is easy to use and pleasant to look at.
2. A simple way to find out the fastest and cheapest way to get from a starting location to a destination.
3. An easy and stress free way to pay for the transportation required for the journey.
4. An app that can be used for regular daily commutes but also for discovering new transport routes in unfamiliar locations.

2.1.6 Non-Functional Requirements

This section presents requirements that do not focus on system behavior but are qualities or properties that the application must have.

1. Perception requirements

The. Style requirements

Application #: 1

Requirement Type: 1st

Description: The system has a simple design

Rationale: The system must have a simple design so that the user feels comfortable using it.

Origin (interested): User

Satisfaction criteria: The user can understand application functionality.

2. Humanity and usability requirements

The. Ease of Use Requirements

Application #: 2

Type of requirement: 2nd

Description: The system should be easy to use for users.

Justification: As the user profile is diverse, the system must be easy to use to allow greatest possible number of users.

Origin (interested): User

Satisfaction criteria: The user can use the application easily and simply.

3. Performance requirements

The. Speed and latency requirements

Application #: 3

Requirement Type: 3rd

Description: The system must be able to respond quickly to user requests.

Justification: The system must respond to user requests as quickly as possible, otherwise the user may get tired and stop using the application.

Origin (interested): User

Satisfaction criteria: The user can quickly access requests.

B. Confidentiality and Availability Requirements

Application #: 3

Requirement Type: 3b

Description: The system will be available most of the time.

Justification: The user must be able to use the system at any time.

Origin (interested): User

Satisfaction criteria: The system must be available and operate flawlessly 99% of the time.

4. Preservation and support requirements

The. Preservation and Support Requirements

Application #: 4

Type of requirement: 4th

Description: The system must have a user manual for the application.

Justification: The user must be able to consult the user manual whenever he wants to know how the application works.

Origin (interested): User

Satisfaction criteria: A user manual will be created to provide information on how to use the app.

2.2 Design & Architecture

2.2.1 Physical architecture

The system will have a client-server architecture, since the client will be the mobile application that connects, through the Internet, to a server to retrieve data from the data source. The drawing of this architecture is shown in the figure 7.

The client-server architecture was chosen since a mobile device has little memory to store public transport data in many cities, so a server is needed to store a large amount of data and process it. In addition, with this architecture, it will not be necessary to replicate the data of each city on different mobile devices, as it will only be stored on the server. Having a server allows for the number of concurrent users to be increased and results in higher scalability.

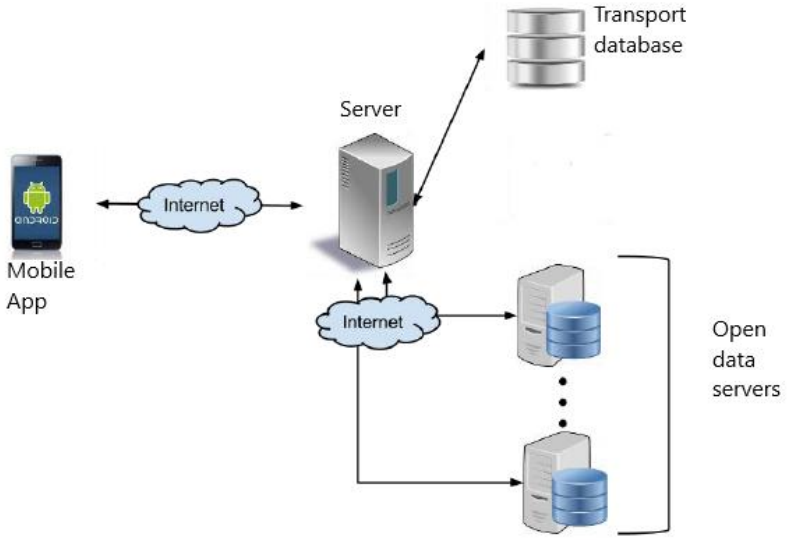


Figure 7: Client-server architecture

In the illustration 7 you can see a mobile application, which contains the complete user interface. This mobile application will allow the user to interact with the application and obtain all the information related to public transport.

For the application to obtain the information, it is necessary to connect to the server through the Internet. This server will be made up of three components: the service layer, which is where the application and the server will exchange requests; the business layer, in which all the necessary operations will be carried out to satisfy the needs of the system's functionalities; and the data layer. This last layer will be in charge of accessing the databases and services to obtain and save the data.

The data tier will connect to a database and the open data services seen in the illustration. The city transport data obtained from the open data services must be extracted and then integrated into the database.

The app also uses a number of API's that require a backend infrastructure. I will be using firebase as a backend to help authenticate the tokens obtained from the Google Sign-In API and the Google Pay API.

2.2.4 Navigation map

The structure diagram shown in the figure below showcases the design of the app and clearly demonstrates how the user will be able to navigate from screen to screen while using the app. The menu acts as an easy and intuitive way to navigate between screens and acts similarly to a navbar on a website.

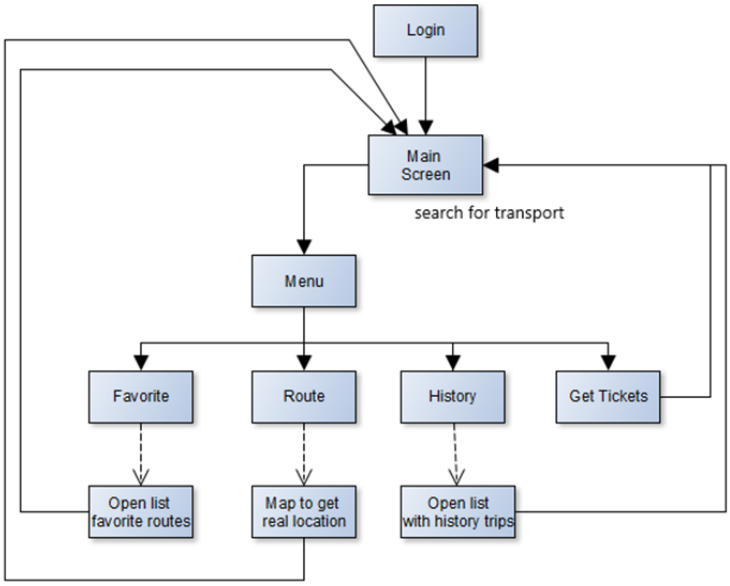


Figure (8). Navigation map

2.3 Implementation

Once the initial design was defined, I continued with the implementation phase. To carry out this phase, it is necessary to select the technologies to use and implement the system following the previously defined projects.

2.3.1 Deployment details

The implementation process started from scratch, meaning that there was no code that could be reused to continue its development. With the use of previously defined technologies, it was possible to implement some of the system's functionality by this point in the development lifecycle of the project. These parts are explained in detail below.

2.3.2 Data extraction and integration

An important part of the system is the data that can be accessed to show the user how they can achieve the journey they wish to complete.

The problem with obtaining public transport data is that when accessing the data catalog provided by Dublin, the formats and variables of the data obtained are different depending on the transport. Therefore, it is necessary to integrate the data into a single schema that is more easily accessible.

This part will be implemented in the next semester.

2.3.3 Maps on Android

To offer an application that shows the real location on the map, there is a Google API that allows you to work with maps.

To work with maps, you need to create a layout that contains a fragment in which a map is defined. But for this map to display in our application, a Java class must be implemented to display the map view. An example implementation for displaying the map appears in the figure 9, in which a View class is created and assigned the shard defined in the layout.

```
package florence.migliorini.crossingborder;

import ...

public class RouteActivity extends AppCompatActivity implements OnMapReadyCallback {
    GoogleMap map;
    Button btnGetDirection;
    Polyline currentPolyline;
    LocationListener locationListener;
    private LocationManager locationManager;
    private final long MIN_TIME = 1000;
    private final long MIN_DIST = 5;
    private LatLng latLng;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_route);
        btnGetDirection = findViewById(R.id.btnGetDirection);
        MapFragment mapFragment = (MapFragment) getFragmentManager().findFragmentById(R.id.mapFrag);
        mapFragment.getMapAsync(this);

        ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.ACCESS_FINE_LOCATION}, PackageManager.PERMISSION_GRANTED);
        ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.ACCESS_COARSE_LOCATION}, PackageManager.PERMISSION_GRANTED);
    }

    @Override
    public void onMapReady(GoogleMap googleMap) {
        map = googleMap;
```

Figure 9: Map code

2.3.4 Favourite Class

The favourites list will act as a quick and convenient way for the actor to access their most frequently used journeys. The actor can search for a journey and add it to their favourites by clicking on the favourite icon located next to the journey information. This button press will serve as the trigger for the event to add the journey to the actors' favourites. The system will then collect the associated information from the journey and the actor (token profile information) and insert them to the favourites table using the database connection that has been established. If all the data is as expected the data will be inserted into the favourites table and the system will send back a confirmation message to the actor. The actor will be able to view the newly added favourite journey in their favourites.

```java
package florence.migliorini.model;

public class Favorite {
    private int id;
    private String location;
    private String destination;

    public Favorite(){}

    public Favorite(int id, String location, String destination) {
        this.id = id;
        this.location = location;
        this.destination = destination;
    }
    public Favorite(String location, String destination) {
        this.location = location;
        this.destination = destination;
    }

    public int getId() { return id; }

    public void setId(int id) { this.id = id; }

    public String getLocation() { return location; }

    public void setLocation(String location) { this.location = location; }

    public String getDestination() { return destination; }
```

Figure 10: Favourite Class

2.3.5 Login Class

The Google Sign-In API will be used to identify users and assign them a JWT access token that can be used to access the app's features while the token is still active. After a JWT token has expired the token will need to be refreshed to grant access to the app again. Initially, when a user signs in a GET request is made to retrieve the JWT token and then the ID token must be sent to the firebase backend using a POST request to validate the integrity of the token. If a token is valid a HTTP 200 successful response will be returned, and the user will be redirected to the home screen where they can access the main menu and create a new journey. The JWT token will be stored in local storage and will contain important meta data for the token's validation as well as profile information for the user. This profile information can be used to retrieve the user's name, email, and Google profile picture.

```java
package florence.migliorini.crossingborder;

import ...

public class LoginActivity extends AppCompatActivity {

    private FirebaseAuth mAuth;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
        Button bLogin = findViewById(R.id.button);
        Button bSignup = findViewById(R.id.button3);
        Button bInfo = findViewById(R.id.button4);
        EditText etEmail = findViewById(R.id.editTextTextEmailAddress2);
        EditText etPasword = findViewById(R.id.editTextTextPassword2);

        // Firebase Auth
        mAuth = FirebaseAuth.getInstance();
        bSignup.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String sEmail = etEmail.getText().toString();
                String sPassword = etPasword.getText().toString();
                mAuth.createUserWithEmailAndPassword(sEmail, sPassword)
                        .addOnCompleteListener( activity: LoginActivity.this, new OnCompleteListener<AuthResult>() {
                            @Override
```

Figure 11: Login Activity

2.4 Graphical User Interface (GUI)

Screen design

In order for a user to access all the functions specified above, an Android application was created for the user to interact with. A series of screens have been designed to create an intuitive and easy to use app with the primary goal to retrieve data from all public transport in Dublin based on the users search. The app also allows for the payment of the journey in a very convenient way and the allocation of a ticket in the form of a QR code.

The screens used in the app are:

Splash Screen -> When the Application is opened and is launching, the first start-up screen is Splash Screen with the logo, this gives the app a professional look while the app is getting the data leaving the user with a good impression of the application.

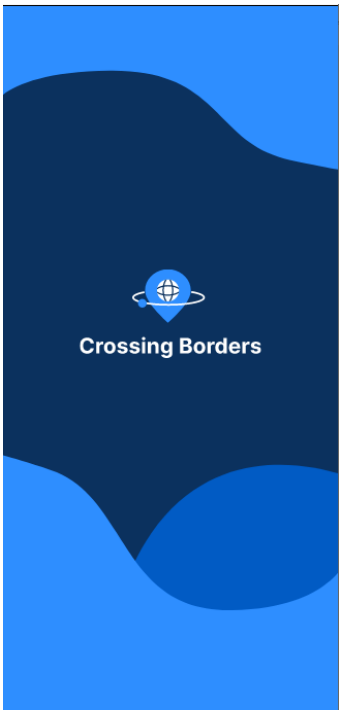


Figure 12: Splash Screen

Login Screen -> The user has the option of filling in their email and password for their registered account or to link their Google account. The google account is authenticated by validating the JWT token using a firebase backend.

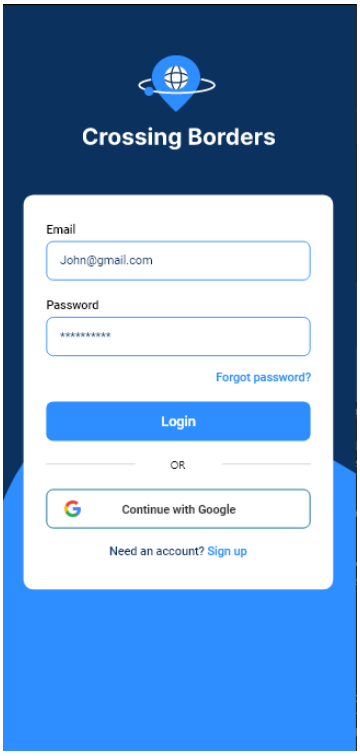


Figure 13: Login Screen

Main screen -> this screen is the most important of all since it is where the user enters their desired destination and then the transport available in the city appears. This screen will be accessed whenever

the user wishes to get a transport option for their journey and to buy the associated ticket. The user can click on their preferred transport method and the results will be filtered accordingly.

This screen will appear after the user logs in, the geolocation will determine the city that the user is in ,for this reason it is necessary to have GPS and WIFI activated.

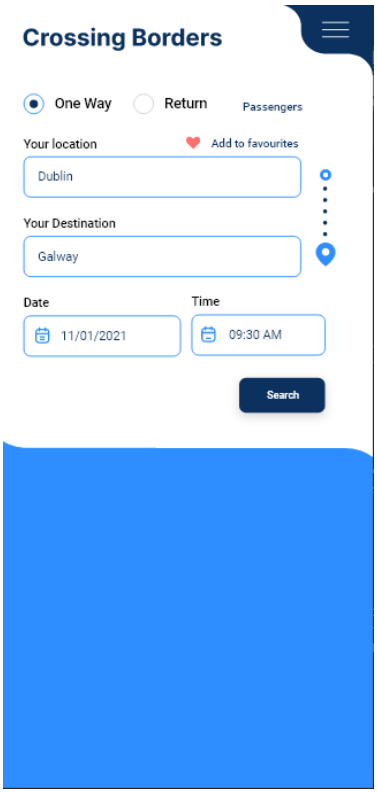


Figure 14: Main Screen

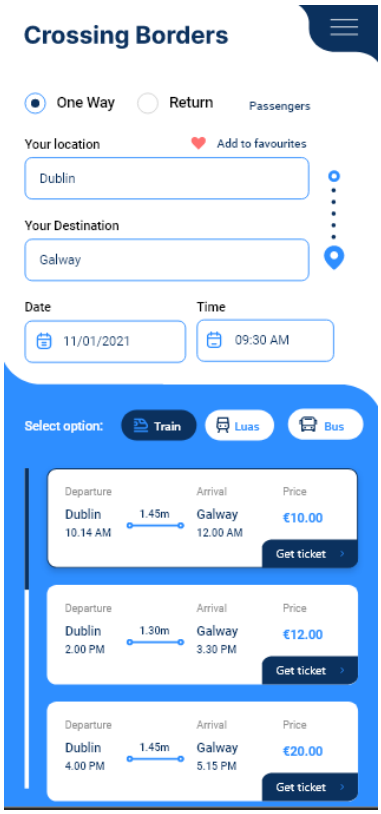


Figure 15: Home Screen displaying options

When the user chooses the best option for their journey, the system displays the payment method.

Payment Screen -> This screen displays the ticket with the options chosen by the user. At this point the user can choose to pay with their card details using stripe or pay with Google Pay using a payment option available in their digital wallet. After the payment is complete, the user will be given a QR Code.

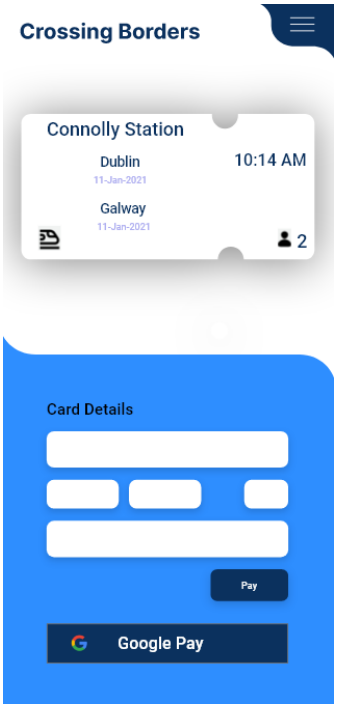


Figure 16: Payment Screen

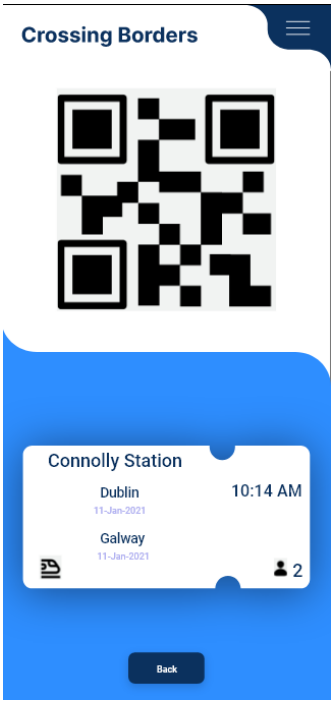


Figure 17: QR Code Screen

Navigation screen(Menu) -> This screen displays a list of functions that can be accessed. The screen appears when a user clicks in the upper right corner of the screen. The list of functions allows the user to check favourite routes, check travel history, search for tickets (returning to the main screen), and open a map with real-time location.

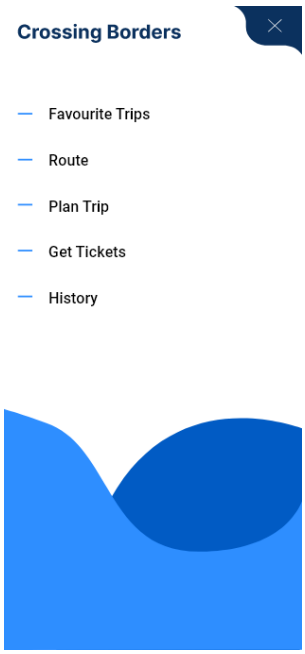


Figure 18: Menu screen

Favorite Screen -> Users have the option to save commonly used routes to their favourites, this screen can be accessed by clicking on the Menu option (upper right) and selecting favourites. This screen displays a list of all trips saved by the user. The user has an option to delete by clicking on the "Remove" button, and also the option to the Main screen, by clicking on the "Back" button.

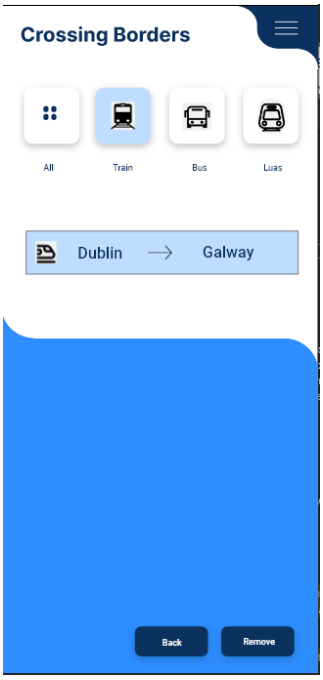


Figure 19: Favourite Screen

History Screen-> This screen will display a list with the last trips completed by the user. There are two buttons; "Remove" allows the user to delete the trip from their history, and the "back" button that returns to the main menu.

Figure 20: History Screen

Route Screen -> The purpose of this screen is to get the user's location in real time and display it on the map, the user can interact with the map to see nearby locations. To leave that screen the user needs to click on the upper right to have access to the Menu and then pick an option.

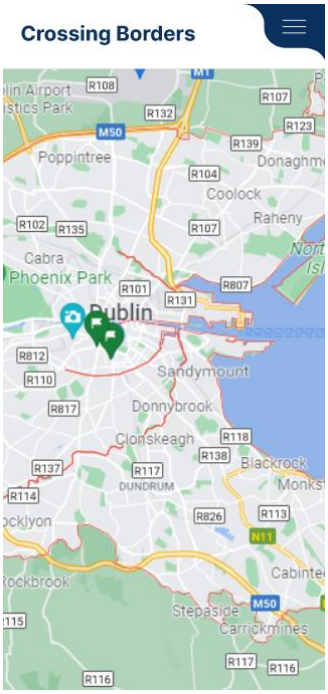


Figure 21: Route screen

2.5 Testing

To ensure that the system works properly, it is necessary to perform the relevant tests on the system. These tests will be carried out as the different features of the system are implemented.

The tests to be carried out for each characteristic will consist of three phases. In the first phase, what will be done is to test the main scenarios of the different use cases. Later, in the second phase, the extensions of each use case will be tested to see if they behave as expected. Finally, the third and final phase will consist of testing some unusual scenarios that are not included in the use case specification.

For a feature to work properly, it must go through the three testing phases described above. If a feature fails in any of these phases, it will be addressed and testing will be resumed from the first phase. Regression testing will be used to ensure that new changes or newly added features do not have a knock-on-effect on previously functional and tested features.

2.5.1 Evidence

This section will be carried out in the second semester, and its purpose is to show the tests have been carried out to verify if the functionalities are working correctly. To capture these tests, this document shows a table for each requirement in which three columns appear indicating the purpose of the test, the tests to be performed, and whether the test completion is successful or not.

2.5.2 Functional requirements

At this point, the results of the tests performed for each of the functional requirements specified in the Requirements section are shown in the table (number here).
Example of how the table will work next semester.

Test objective	Tests to be carried out	Result
#1 Check public transport in a city		
Verify that a user can consult the public transport in the city and buy the ticket	Phase 1: The user indicates to the system that he wants to consult all the transport options in the city and the system shows him all that is available.	Phase 1: Tests will be taken in the second semester.
	Phase 2: User selects transport, pays and gets the ticket.	Phase 2: Tests will be taken in the second semester.

Table 1: Test

2.6 Evaluation

This section will be carried out in the second semester.

3.0 Conclusions

During the initial development of this project, the intention was to develop a service that would manage the open transport data made available by the city of Dublin, Ireland, and an application that would allow the consultation and purchase of the ticket. This purpose is still under development, although the goals defined above of showing the user the best transport options between a point of origin and a point of destination will not be something easy to achieve, as the entire project is at a very advanced level, and the size of this project would be better suited for a development team and not just one person. What I have noticed so far is that time is not the main factor for delays in some phases, but also the lack of experience in planning and knowledge of the new tools used.

To manage open data in the system, the best way to integrate the data must be decided.

By the end of the project, some research must be done to use the open data. With the layouts ready, for the second semester, the focus will be the code and the tests. And with that, I'm satisfied with what I have managed to develop in a short period of time, along with other projects I have completed in the same time span.

4.0 Further Development or Research

The focus for the next semester will be code and testing. The plan is to format the open data in a way that it can be used to find the best transport routes based on a user's search and evaluate the best way to access the data. My aim is to complete this in the first few weeks of January.

5.0 References

Developers.
Available at: https://developer.android.com/docs
[Accessed in December 2021]

Firebase
Available at: https://firebase.google.com/
[Accessed in December 2021]

Google Maps
Available at: https://developers.google.com/maps/documentation/android-sdk/start
[Accessed in December 2021]

6.2. Reflective Journals

6.2.1 Reflective Journals April

April was a busy month, and the focus was on the software project, splitting between documentation and code. I got the application working at the end of April, and I was fixing details, special the layouts that were not responsive. The user tests were in progress.

6.2.2 Reflective Journals May

For this month, I focus on fixing some details in the application, code, and layout, for the favourite layout, the one that I had a lot of work to try to keep nice, was the last one I got working the way I would like it. I worked on slides and videos. I spent more than I was expecting on the documentation, but in the end, I am glad about the time invested in it.

_____

Signature