FINAL YEAR PROJECT REPORT

ROUTEGUIDE – NAVIGATION APP

Eliza Airinei

Eliza.Airinei@city.ac.uk

Academic Year 2023/ 2024

BSc Computer Science

Benedickt Wagner

Contents

CHAPTER 1 – INTRODUCTION

This opening chapter introduces the concept of RouteGuide, detailing the thoughts behind its development and the specific user challenges it is designed to overcome. It sets out to define who will benefit from the app, the objectives that the creator aims to achieve, and provides a review of the subsequent chapters.

1.1 PROBLEM TO BE SOLVED

Existing maps often provide generic travel routes that don't consider individual traveler preferences, interests or time constraints.

Tourists frequently miss out on attractions that are less known, local experiences, and cultural insights due to the lack of integrated local knowledge in most travel apps.

Moreover, travelers are often overwhelmed by the amount of information available online and find it challenging to plan a trip efficiently.

A few similar applications found are: "Google Maps"[1], "Wanderlog"[2]. These two have been chosen because both of them are suggested for travelers, but they have different objectives.

1.2 PROJECT BENEFICIARIES

This project will help travelers and people using navigation maps. Moreover, this project can be used as inspiration for further development of existing apps.

Many people find the current navigation maps confusing when they try to find nearby amenities, but this project aims to make a map more user friendly. Also, people that travel tend to search attractions on websites and create routes by themselves, but that can be exhausting because of the amount of information that can be found online and sometimes the weather might not match the plan, so rearrangements are required. RouteGuide will have AI-driven route suggestions based on user preferences and it will make planning the journeys less stressful. The users will be able to choose the time availability, type of attractions, activity level, transportation preferences, budget, cultural interests, food preferences, accessibility needs, family-friendly options, shopping interests and nature.

[1] Google Maps: maps.google.com
[2] Wanderlog: wanderlog.com

1.3 PROJECT OBJECTIVES

The project shall provide an interactive, AI-powered mobile app designed for people that travel. The app will improve exploration and navigation by offering personalized route suggestions and providing en-route information, such as amenities and attractions. Furthermore, the app intends to have a better user experience compared to the existing software. The full Project Definition Document has been attached to this report (Appendix A). List of objectives and subobjectives the author wants to achieve while developing the project:

1.3.1 Primary Objective: Build a Navigation App with AI-Driven Itinerary Suggestions

The primary goal is to create a mobile application that improves travel experiences by integrating Artificial Intelligence technology. This technology will analyze user preferences, and real-time data to suggest personalized travel routes and itineraries. The app aims to provide a seamless navigation experience that guides users through their journeys.

1.3.2. Sub Objectives:

1.3.2.1 Account (Login and User Info Management)

The app will include a secure and user-friendly account management system allowing users to create and manage their profiles. Users can sign up, log in, and manage personal information with ease. The system will also ensure the secure storage and handling of user data, adhering to privacy regulations.

1.3.2.2 Navigation Map

The navigation map will offer detailed and accurate maps with directions and routes tailored to different travel modes such as walking, driving, or public transit. It will incorporate real-time traffic updates and condition adjustments to provide the best possible routes.

1.3.2.3 Searching Places

The search functionality will enable users to find places both globally and locally. It will support searches with various filters and provide options for discovering attractions near the user's location.

1.3.2.4 AI-Driven Suggestions (Chat Format)

An AI-driven chatbot will be integrated into the app to provide interactive travel suggestions. The chatbot will engage users in a conversational manner, adapting its recommendations based on user queries.

1.3.2.5 Save Favorite Places or Itineraries

Users will have the ability to save and manage their favorite places and itineraries. This feature will allow for easy access to preferred locations and routes, the ability to organize travel plans, and the flexibility to customize and revisit saved itineraries for future trips.

Together, these objectives aim to transform RouteGuide into a highly functional, personalized travel companion that AI technology to significantly change the travel planning and navigation experience.

CHAPTER 2 - OUTPUT SUMMARY

In this chapter are highlighted the outputs of RouteGuide, which include a summarized description, the usage and the beneficiaries of the project.

The main output is a navigation app, developed for Android devices, that will allow people around the world to navigate easier and plan their journeys more efficiently.

Title	Description	Usage	Beneficiary
Application	The successful implementation of the project will offer a compiled application that will be available for downloading on Google Play	The application can be used by anyone that needs a navigation app	Everyone
Source Code	The source code has 5560 lines of Java code and 1000 lines of XML code. 3400 lines of code adapted/ borrowed from external sources.	The source code used for implementing the application’s functionalities	Author, Supervisor, Readers
Database	The Database (Firebase) securely manages user profiles and preferences	The database supports the application's need for real-time data syncing and user authentication	Author, Supervisor, Readers
Software Requirements	The Software Requirements documents provides a roadmap for development	Created by the author and used during the project development to align the progress with its objective	Author, Supervisor

CHAPTER 3 - LITERATURE REVIEW

In this chapter, the author explains the research that is shaping RouteGuide, a travel map app designed to make exploring new places easier for tourists. Key areas:

3.1 Existing Software

The following are two popular travel apps that have been analyzed by the researcher. The similarities with the proposed project are described below.

Google Maps

Google Maps is one of the most popular navigation apps known for its accuracy and comprehensive coverage. The app is available on all devices, and it is designed for a broad range of users with different purposes. It offers route planning and suggestions, but it lacks personalization. Itinerary suggestions would be more useful if they would include individual preferences and interests. Moreover, most of the users are using the app while travelling for searching en-route places, but Google only offers a small range of amenities, while some tourists might find useful to find en-route attractions.

Wanderlog

Wanderlog is designed as a travel planner, and it offers the possibility of creating itineraries. It is available on both; Apple Store and Android Play and it is intended only for tourists. This app is not a navigation map, it focuses on organizing the trips, by including reservations and collaboration with friends. Although it includes itineraries and they can be adjusted by the user (the order can be changed, the users select the activities), the suggestions are not AI-driven, which may make the search complex in some cases.

RouteGuide aims to offer a more personalized experience for tourists.

3.2 Project Development

The project is structured in several steps to successfully build the mobile application.

3.2.1 UML Implementation & Visual Paradigm

Visual Paradigm supports the creation of diagrams, which are essential in planning the app's interface and user journey. By using this tool, the developer has been able to effectively conceptualize the components of the app, ensuring a coherent and user-friendly design. The knowledge acquired in university has been directly applied to construct the interface for the travel map application, demonstrating the practical application of academic skills in real-world development.

3.2.2 Figma for Design

In the development of RouteGuide, Figma is used for the UI/UX design. Figma's intuitive platform allows for the efficient creation of prototypes and user interface designs. The focus will be on creating a visually appealing interface with Figma's capabilities in interactive design elements, enhancing the overall user experience of the travel map application.

3.2.3 Firebase database

RouteGuide uses Firebase's Realtime Database to ensure the synchronization of user data. This setup allows for immediate updates, as every user interaction prompts the latest data to be instantly shared with every connected client, ensuring that all users have access to the most up-to-date information for their travel planning and navigation needs.

3.2.4 Android Studio

As the application is intended to be a mobile application for android devices, Android Studio is the most reliable software to use. Its environment allows the programmer to use different tools, SDKs and programming languages.

3.3 Resources

Google Maps Platform

Google Maps offers a large set of documentation and guidance for using the APIs(Application Programming Interface) provided by them. This resource played a key role in my application as I learned how to integrate maps, places and directions.

Link: Google Maps Platform Documentation | Maps SDK for Android | Google for Developers

OpenAi Documentation

OpenAi platform shows how the API calls can be made and moreover, it presents the multiple types of Artificial Intelligence services that can be used. The information provided helped me to integrate a chatbot in RouteGuide without struggling to identify the most suitable service.

Link: API Reference - OpenAI API

Firebase Documentation

Firebase Documentation presents full guides for implementing backend functionalities within the app, such as setting up and managing the Firebase Database, Authentication, Storage, which were essential for the secure and efficient operation of RouteGuide.

Link: Firebase Documentation (google.com)

Youtube

Youtube serves as a great platform for learning. There are multiple channels available in different languages which help users improve their skills in any area, including Android Studio, Java, Firebase, Google Maps API. Some of these channels helped me enhance my knowledge.

Link: YouTube

Stackoverflow

Stackoverflow is a forum where users can search for answers or help others find answers regarding software development. There are many questions with answers about IDE issues, Java errors, API requests and others, which helped me solve errors that I could not find solutions to.

Link: Stack Overflow - Where Developers Learn, Share, & Build Careers

CHAPTER 4 – METHODS

In this chapter is presented the project's lifecycle. It will describe the techniques used from the analysis phase to the testing phase, including the design and the implementation of the project. The results of each step can be found in Chapter 5.

4.1 Development Methods

Considering the project's complexity, I chose to adopt the Agile methodology, specifically the iterative development process. This approach allows for setting and achieving small milestones, which not only ensures steady progress but also facilitates continuous integration and testing of functionalities. Such a step-by-step approach is critical for early detection and resolution of issues, which, if left unaddressed, could become more complex and problematic. The project was systematically divided into four primary phases: analysis, design, implementation, and testing. Each phase was further decomposed into small steps to guarantee completeness at every stage of development. This breakdown allowed for frequent reassessments and refinements, contributing significantly to the overall quality of the final product.

4.2 Work Plan

Before starting the work, a workplan has been done to visually represent the timeline of the project, taking into consideration courses and other projects. The initial project plan can be found in Appendix A. While completing the project, there were changes to the plan which affected the overall timeline so there was a necessity to update it, and the final version can be found in Appendix C.

4.3 Analysis

4.3.1 Analysis of Existing Applications and Opinions

Prior to doing the requirements for the project, I did a thorough analysis of various applications similar to my concept. Although only a couple are referenced to highlight diverse perspectives, examination was undertaken on a multitude of apps across both iOS and Android platforms. This research informed me of the gaps present in the current market.

4.3.2 Requirements Analysis

During this phase of the project, it was essential to collect details on the functionalities needed to meet the established objectives and to fulfill potential user expectations. The requirements phase was crucial in accurately setting the deadlines. As outlined in Chapter 3, the analysis was performed via online research.

4.4 Design

This stage was dedicated to the design of the application's various segments and how they would integrate with one another. The design process included the creation of multiple diagrams that offered a high-level perspective of the application's architecture. As previously detailed in Chapter 3, Visual Paradigm was utilized to craft these diagrams.

4.4.1 System Architecture

The system design for this project combines various tools and methods to build a complete mobile app that's meant for Android devices like phones and tablets. Throughout the development phase, it was clear that the app was going to be for wireless devices only. The setup of the app was carefully planned out and split into different parts, as described below. The design of the system has several main parts:

- User Interface: This is where the user interacts with the app
- Application Layer: It processes user inputs, handles navigation, and communicates with the service layer
- Service Layer: This is a set of services that the application uses to perform more complex operations, like communicating with the database or external APIs.
- Data Access Layer: Manages the retrieval and storage of data
- Database: Stores user data, preferences
- External Services: Services the app uses that are not built into the app
- Security Layer: Handles authentication, authorization, and data encryption to protect sensitive user data.

You can find all the details of how we put the system together in Appendix D of the report.

4.4.2 Use Case Specifications

A Use Case Specification is a written explanation of the system's capabilities. Crafting strong use case specifications is crucial as they facilitate the identification and structuring of functionalities. Leveraging these specifications, I was able to design the user interface, organize the development process, and establish points of reference for testing.

4.4.3 Use Case Diagram

Another important step is creating the Use Case Diagram. At its core, a use case diagram is a simple way to show how a user interacts with the system and the various scenarios that a user might be involved in. Use cases essentially lay out what the user intends to do with the system. This approach made it possible to clearly display all the app's features and how they connect with the background systems and the app itself. I developed a use case diagram to reflect and guide the evolution of the app's functionality and design.

4.4.4 Entity Relationship Diagram

An entity-relationship diagram, or ERD for short, is a visual that maps out what data a database will hold and how it all links together. It details the different types of information, called entities, their characteristics, known as attributes, and how these entities interact with one another through various relationships. For this app project, I created an ERD to get a clear picture of the database structure I would need for Firebase, making sure we knew exactly what data we'd be storing and how it should be organized.

4.4.5 Class Diagram

A class diagram is a useful tool that shows us the layout of a system. Think of it like a blueprint that shows the different parts of the system, what each part is responsible for, and how they all talk to each

other. I created this blueprint from the project's code using a tool that can take the code we wrote and lay it out in an easy-to-understand diagram. This makes it a lot easier to see how the pieces fit together and how they'll work in harmony. The diagram includes the parts of the system, their characteristics, what actions they can perform, and how they link up with one another.

4.4.6 Sequence Diagram

A sequence diagram is a kind of chart that shows the sequence of actions or interactions between parts of a system. It lays out the order in which things happen and how different elements work together. I created one of these diagrams by using the methods from project's code. I also used the sequence diagram to dig into the specifics of UML use case diagrams, which helped me get a solid grasp on the complex functions we're dealing with.

4.5 Implementation

The application was developed for Android devices only. Below are more details about the steps taken for the implementation process:

4.5.1 Graphic User Interface (GUI)

The Graphic User Interface was designed in two parts. Firstly, I used Figma to visually represent my main idea before beginning to code. It was a useful guidance for the actual implementation of the GUI, but it was also a key part in organizing the work needed for functionality implementation. Secondly, the actual design has been done in Android Studio, using XML. It was done in springs, along with each functionality.

4.5.2 Development Environment

The application development was done in Android Studio. To proceed, it was necessary to install the IDE for Windows operating system from Android Studio website and configure the necessary SDK, version 34 (API 34 (“UpsideDownCake”; Android 14.0)). As part of the installation, the AVD (Android Virtual Device) had to be configured for emulating and testing the app.

4.5.3 Programming Language

For this project, the programming language used for building and managing the app’s functionality was Java. It handles complex data structures and APIs. Meanwhile, JSON (JavaScript Object Notation) is used for data handling, such as storing the data in the database and communicating with the APIs.

The source code is divided into packages: Fragments, Activities, Adapters and Models + other classes which are not included in packages. The models and adapters work together by transforming the data from the models into viewable formats. They have been used mainly for implementing the maps along with its functionality and the chatbot.

4.5.4 Firebase

For managing the data, I chose to use Firebase because it provides a suite of tools to help the developers, such as Authentication, Databases, Hosting and Analytics and the data is protected against unauthorized access, ensuring that it is secure. Firebase handles the complexity of authentication and

offers real-time data storage and syncing, making it a suitable platform for efficiently developing RouteGuide.

4.5.4.1 Authentication

Firebase Authentication makes authentication easy for end users and developers. It allows you to focus on your users, and not the sign-in infrastructure to support them. (Firebase, 2024) The sign-in method used is Email/Password and the email verification function is added to it. Once a user registers and attempts the first login, she/he must confirm the email address and then will be able to access the app.

4.5.4.2 Realtime Database

"The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in real-time to every connected client, with cross-platform support for iOS, Android, Web and more." (Firebase, 2024) The project's Realtime Database holds the user's information such as the username and email, saved places and saved itineraries. Whenever the user marks a location as a favorite or saves a travel plan, the details are sent to the database in real-time. This means that the updates can be seen instantly, in case the user is switching between devices. The data can be updated, deleted, added or just read.

4.5.5 Google Maps Libraries

In this application, I integrated APIs provided by Google Maps. Along with Maps SDK for Android, I enabled the Google Maps API, Directions API and Places API to create an interactive map and to enhance the functionality. The Maps API allows the integration of map along with its styles and lets the users track their location, meanwhile the Directions and Places APIs together add the navigation functionality to the maps, offering directions between places, travel modes, such as walking, driving and public transport, detailed information about locations, including names, addresses, ratings. There was no need of creating a database for holding places information because Google Maps offers access to its data through the APIs.

4.5.6 OpenAI Library

Integrating OpenAI's GPT-4 into the project's chatbot significantly improved the user experience by offering personalized itineraries through advanced capabilities. Users can input their preferences, such as destinations, interests, available time, budget by answering the questions and once all of them are completed, the chatbot will generate a customized itinerary complete with latitude and longitude. . *"Chat models take a list of messages as input and return a model-generated message as output. Although the chat format is designed to make multi-turn conversations easy, it's just as useful for single-turn tasks without any conversation." (OpenAI Docs, 2024).* This integration allows the app to display the itinerary on the map. The use of v1 Chat Completions helped provide detailed information making the trip planning efficient and user-friendly.

4.5.7 Other Libraries

Navigation Drawer simplified the implementation of navigation. A navigation drawer provides access to different screens of the application, such as "Chatbot", "Saved Itineraries", "Help", "About", "Settings" and "Logout".

Retrofit is a HTTP client and allows the app to make network requests, enabling the efficient use of APIs.

The Glide library was intended to be used for loading images when updating the user profile photo. This library is effective for fetching and displaying images, videos or GIFs, minimizing the complexity of image processing and The CircleImageView is a library used for displaying images in a circular frame.

4.6 Testing

Android Studio comes with built-in emulator and it's part of the Android Device (AVD) Manager. It allows the developers to create a virtual device that has the configuration of Android. After selecting and setting up the emulator, the application must be built, and it will automatically be installed on the device. I used the emulator to test the app each time I implemented a functionality to check if it works and if it affects other parts of the code. In order to test the project, I had to enable the location for the app and to have access to internet.

CAHPTER 5 – RESULTS

Chapter 5 presents the outcomes of the methods mentioned in the previous chapter. It will clearly describe each stage of the implementation.

5.1 Requirements Analysis

Below are the requirements described in detail, completing the Section 4.3. This part of the project refers to the results of the methods which were used in collecting and processing the requirements.

5.1.1 Research of existing software

The first step of analyzing the requirements was to research existing software with similar functionalities and collect information that would form a solid foundation for RouteGuide. In this process, the existing applications were compared, the pros and cons were identified, aiding understanding of what features would make the application user-friendly. The analysis was performed on different platforms, Android and iOS. The author encountered challenges with the excessive information encountered during itinerary planning, which was often distracting and complicated. This led to the decision to create an application that consolidates AI-driven suggestions for itineraries, aiming to centralize all necessary details in one location and thereby simplifying travel for users. Other applications tended to present an overload of information, sometimes resulting in a confusing interface. The intent for RouteGuide was to streamline the data presented, focusing on satisfying the end-user's needs. The finalized requirements encompassed a feature for searching nearby places, which proves useful when discovering unexpected local spots, and the ability to search for attractions along a route, enhancing the travel experience. Furthermore, the application was designed to generate personalized itinerary suggestions through Artificial Intelligence and to allow users to save both places of interest and itineraries, providing easy access to directions.

5.1.2 Functional and Non-Functional Requirements

Following the collection of requirements, the author separated them into functional and non-functional. The list of these requirements contains 9 functional and 5 non-functional and can be found in Appendix C.

5.1.3 Requirements prioritization

Once the functional and non-functional list of requirements was finalized, the author analyzed the importance, complexity and the connection between them and set the priority level for each. This prioritization helped the author to manage the time and to focus on what required more attention. The table including this information can be found in the Appendix C.

5.2 Design

In the design stage, the author made the necessary documentation, which was talked about in the previous Chapter. Coming up next, the author will give a detailed description of the diagrams created while designing the app.

5.2.1 System Architecture

The architecture of the mobile application, specifically developed for Android devices, is constructed with a focus on a wireless experience. It features a user-friendly interface for interactions, an application layer that manages input and navigates services, and a service layer that executes complex operations with the database and external APIs. The data access layer oversees data transactions, while the database securely stores user details and preferences. The application integrates with external services, enhancing its capabilities, and a dedicated security layer ensures user data protection with robust authentication and authorization processes. The complete architectural details are documented in Appendix D.

5.2.2 Use Case Diagram

The author developed a use case diagram to illustrate how users interact with the system and the various use cases they engage with. This diagram serves as a visual tool, outlining the user's involvement across the system. In its final form, the diagram includes numerous use cases, a primary actor representing the user, and secondary actors, such as the database and APIs, highlighting the different interactions and responsibilities. Throughout the progression of the project, a series of use case diagrams were created, reflecting additional functionalities and adjustments deviating from the initial plans as development and testing advanced. All iterations of the use case diagrams generated throughout the project are compiled in Appendix D for reference.

5.2.3 Use Case Specifications

Once the use case diagram was complete, the author proceeded to detail a set of use case specifications for the main scenarios depicted in the diagram, tailored to their role in the application. Each specification is linked to a selected use case, facilitating a clearer understanding of the app for readers. These specifications cover information about both primary and secondary actors, their roles, any necessary preconditions, and document the standard and alternate sequences of interactions. The compilation of Use Case Specifications can be found in Appendix C for further information.

5.2.4 Entity Relationship Diagram

The Entity-Relationship (ER) Diagram in the application showcases the structured layout of the database, where entities like Itineraries, Places, and Users are interconnected. The Itineraries entity, defined by unique identifiers includes multiple Places, illustrating a one-to-many relationship where a single itinerary may include several locations with latitude and longitude details. The Users entity captures user-specific details, including saved locations, facilitating personalized experiences. Together, these entities are interwoven to form a cohesive database, instrumental in the app’s functionality for trip planning and navigation. This diagram was essential to visually clarify the relationships between entities.

5.2.5 Class Diagram

To represent the relationship between functionalities within the app, the author created a Class Diagram. This diagram describes the Java classes and their methods and attributes, and it was built along with the source code in order to help identifying the method invocations across the classes. For example, the “HomeFragment” plays an important role in the application by allowing the users to search

places using autocompletion or to search nearby places using the buttons available. This class uses models and adapters to structure and retrieve data from Google Maps. Similarly, the "ChatBotFragment" class engages users in dialogue powered by artificial intelligence to suggest personalized itineraries. It collects the user responses and AI is processing them to suggest a sequence of travel activities. The functionality uses a ChatAdapter to render messages from both the user and the bot within the application's user interface, utilizing a list of ChatBotModel objects that contain the message content and a flag to determine the message source. The full diagram produces by the author can be found in Appendix D.

5.2.6 Sequence Diagram

A sequence diagram was created during the design phase of the software development process, before writing the actual code. It's part of the requirements gathering and system design activities, which are key components of the Software Development Life Cycle. This sequence diagram provides an illustrative breakdown of the interactions between the user and the application's various subsystems during the login and search processes. Initially, the user prompts the app's interface to open the login page. After inputting their credentials and initiating the login, the request is processed by the Login Subsystem. If authentication fails, an error message is displayed. Successful login, on the other hand, redirects the user to the homepage. From here, the user can search for nearby places based on certain criteria. This action calls upon the Search Subsystem, which then interacts with Maps and Location Services to retrieve the relevant data. Once the places data is obtained, it is presented to the user. Creating this diagram helped in outlining the step-by-step flow of operations. The sequence diagram can be visualized in Appendix D.

5.2.7 User Interface

The final step prior to implementation involved designing the user interface to accurately display the screen contents. This has been done using Figma and provided a good starting point of this development. However, as the project progressed, the initial user interface design was revised to better align with user requirements and enhance user-friendliness. These modifications were implemented directly within Android Studio, using XML for layout adjustments. The layout designs, including Linear and Relative configurations, along with interface elements like Buttons, ImageViews, TextViews, and EditViews, were selected to meet the project's needs. Both the Figma design and the final detailed design of the user interface can be found in Appendix D.

5.3 Implementation

After completing the requirements analysis and designing stage, the author began the implementation of the application. For this phase, various tools and languages were used.

5.3.1 Firebase

The initial phase of the project implementation involved integrating Firebase to manage the data effectively. As noted earlier, the author chose Firebase for its comprehensive services that simplify backend management. Additionally, the support for Firebase integration provided by Android Studio was a significant advantage in the development process. To initialize Firebase in the project, it was necessary to include a JSON file in the app's folder. This JSON file, typically named “google-

services.json", contains configuration details that link the app with the Firebase project, enabling Firebase services such as databases, authentication, and analytics to function within the app. This setup not only facilitated a smoother development experience but also ensured that the app could manage data dynamically and efficiently right from the start. See the figure 5.3.1.i for reference. To ensure seamless Firebase integration, modifications were also made in the Gradle files. Specifically, Firebase SDK dependencies were added to the "build.gradle" files, enabling the app to connect with Firebase services effectively and access the configurations specified in the "google-services.json" file. See the dependencies in the figure 5.3.1.ii

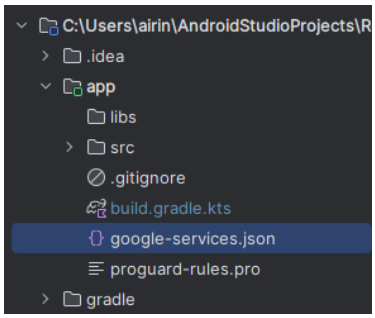


FIGURE 5.3.1.I

```
//Firebase
implementation(platform("com.google.firebase:firebase-bom:32.8.0"))
implementation("com.google.firebase:firebase-analytics")
implementation("com.google.firebase:firebase-database:20.3.1")
implementation("com.google.firebase:firebase-auth:22.3.1")
implementation("com.google.firebase:firebase-storage:20.3.0")
implementation ("com.firebaseui:firebase-ui-database:7.1.1")
```

FIGURE 5.3.1.II

5.3.1.1 Authentication

Firstly, the author set up the Authentication by selecting the Email/Password on the platform (see Figure 5.3.1.1.i). By opting for Email/Password authentication, the setup ensured a straightforward and widely accessible method for user registration and login, without the complexities and additional privacy concerns that can come with third-party authentication providers like Facebook, Google, or Apple The Authentication was initialized in the app as shown in the figure 5.3.1.1.ii.

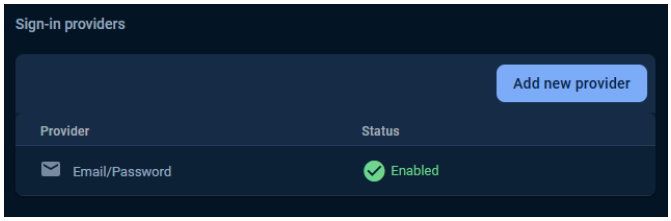


FIGURE 5.3.1.1.I

```
firebaseAuth = FirebaseAuth.getInstance();
```

FIGURE 5.3.1.1.II

5.3.1.2 Realtime Database

The integration of Firebase Realtime Database into the app was a key step in enabling real-time data synchronization across all user devices. This cloud-hosted database allows for updates without manual refreshing, ideal for features that require dynamic data changes, such as saved places or itineraries or

username updates. Setting up the database involved configuring security rules in the Firebase console to ensure data integrity and privacy, followed by adding the necessary Firebase database dependencies in the Android Studio project's "build.gradle" file. Once set up, the app interacts with the database by initializing it within the application code and using database references to read from and write to the database. Additionally, listeners are implemented to react to data changes in real-time, updating the user interface accordingly.

See the figure 5.3.1.2.i for an example of using the Firebase Realtime Database to retrieve and load saved itineraries specific to the logged-in user, identified through Firebase Authentication, enhancing personalization within the app.

```
//Load the saved itineraries from the Firebase database
1 usage  Eliza9a
private void getSavedItineraries() {
    FirebaseUser currentUser = FirebaseAuth.getInstance().getCurrentUser(); //Get the current logged-in user
    if (currentUser != null) {
        String userId = currentUser.getUid();
        DatabaseReference reference = FirebaseDatabase.getInstance().getReference( path: "Itineraries").child(userId);

        Eliza9a
        reference.addListenerForSingleValueEvent(new ValueEventListener() {
            Eliza9a
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                itineraryList.clear();  //Clear the existing data
                for (DataSnapshot childSnapshot : snapshot.getChildren()) {
                    ItineraryModel itinerary = childSnapshot.getValue(ItineraryModel.class);
                    if (itinerary != null) {
                        itineraryList.add(itinerary); //Add itinerary to the list
                    }
                }
                itineraryAdapter.notifyDataSetChanged();  //Notify the adapter that the data has changed
            }
```

5.3.1.2.I

5.3.2 Splash Activity

This splash screen, the first point of interaction, shows the application's name and serves as a loading page. It is displaying while the app initializes and checks the user's authentication status with Firebase. If a user is already logged in, the app transitions to the main activity. If not, the user is directed to the login screen. This integration not only contributes to the aesthetic appeal of the app but also serves a functional purpose by managing the initial loading process and user authentication in a user-friendly manner.

5.3.2 Login, Signup and Forget Activity

The integration of the User Login, Signup, and Forget Password activities into the app are important components of the user management system. The Login activity provides a secure entry point for returning users to access their accounts. Upon entering their credentials, users are authenticated via Firebase. If a user's email isn't verified, the app prompts them to check their inbox, adding an extra layer of security through email verification.

The Signup activity is designed for new users to create an account. It collects essential information like email, username, and password, and creates a new user record in Firebase Authentication. It then sends an email verification link to ensure the user's email address is valid. When entering the information, the users must complete all the fields, and set a password of minimum 5 characters.

For users who have forgotten their passwords, the Forget Password activity provides a simple and user-friendly way to reset their credentials. Users are asked to enter their registered email address, and Firebase triggers a password reset email, guiding them through the process of setting a new password. This integration ensures that users can regain access to their accounts easily, improving the overall user experience.

5.3.4 Navigation Drawer

The navigation drawer, a sliding menu, is configured using XML. In the “nav_menu.xml”, different menu items are defined, each associated with a fragment represented in the “drawer_nav_graph.xml”. This graph outlines the relationship between menu items and their corresponding screens within the app. Each menu item, when clicked, triggers navigation to a specified fragment, such as Home, Chatbot, or Saved Places.

The “drawer_nav_graph.xml” defines the app's paths, which the `NavigationView` in “nav_drawer_layout.xml” uses to switch between different app sections. This layout includes the “NavigationView”, responsible for displaying the navigation items and a header layout, containing user information.

The “header_layout.xml” specifies the layout for the header of the navigation drawer, with and an icon which represents the user and user-related information like username and email address. It’s displayed at the top of the drawer.

These files create a user-friendly navigation drawer, with a visually pleasing header that enhances the app's usability, allowing users to move smoothly between different pages of the app.

5.3.5 Settings and User Profile

In terms of implementation, the Settings fragment can be accessed from the navigation drawer, providing a centralized and easily accessible location for all app configurations. The user interface for the Settings fragment is designed to be intuitive, using standard Android UI elements like “TextView” and “ImageView” and additional XML layouts (“password_dialog_layout”, “username_dialog_layout”) for actions that trigger more complex operations, such as password or username changes. In figure 5.3.5.i can be visualized how a dialog is setup for changing the password. This setup asks the user to input the email, the old password and the new password.

The integration of Firebase into the Settings fragment allows for real-time updates to user information. When a user makes changes, such as updating their username or password, these changes are immediately synchronized with Firebase, ensuring that the user's profile is always up to date across all devices.

```
//Displays a dialog for updating the password
1 usage   Eliza9a
private void showPasswordChangeDialog() {
    View dialogView = LayoutInflater.from(getContext()).inflate(R.layout.password_dialog_layout,  root: null);
    EditText emailEdit = dialogView.findViewById(R.id.emailEdit);
    EditText oldPasswordEdit = dialogView.findViewById(R.id.oldPasswordEdit);
    EditText newPasswordEdit = dialogView.findViewById(R.id.newPasswordEdit);

    // Setup dialog with custom view and buttons
    new MaterialAlertDialogBuilder(getContext())
            .setView(dialogView)
            .setTitle("Change Password")
            .setPositiveButton( text: "Update", (dialog, which) -> {
                String email = emailEdit.getText().toString().trim();
                String oldPassword = oldPasswordEdit.getText().toString().trim();
                String newPassword = newPasswordEdit.getText().toString().trim();
                changePassword(email, oldPassword, newPassword);
            })
            .setNegativeButton( text: "Cancel",  listener: null)
            .show();
}
```

5.3.6 Updating the user data in Firebase

Next step was managing the updates of username and password. For changing the username, the Settings screen provides a “Change Username” option. When the users select this, they're usually required to input the username they wish to set. After entering the required information, the app communicates with Firebase Authentication to update it. Once the username or password is updated successfully, Firebase automatically ensures that the new credentials are in effect immediately. All changes are securely stored in Firebase, ensuring that the user's data remains consistent and secure.

5.3.7 Home Screen & Map Integration

The next step was to design the Home Fragment where the map is displayed. Integrating a Google Map into an app's home screen involved a few essential steps, starting with adding the necessary dependencies. It was necessary to include the Maps SDK for Android in the app’s “build.gradle” file as shown in figure 5.3.7.i and declared as in figure 5.3.7.ii. Next, an API key from the Google Cloud Platform was required; this was obtained by creating a project, enabling the Google Maps API, and generating the key, which was then added to the “AndroidManifest.xml” under the “meta-data” tag. In the layout XML, the author added a `SupportMapFragment` to host the map, configuring it to fill the parent view. This setup was linked to the fragment where was implement the “OnMapReadyCallback” interface, initializing the map with a call to “getMapAsync(this)”. Once initialized, the map was customized within the “onMapReady()” method, using a style created on the Google Platform. The author chose a style that did not include all the road names and place’s names. It was applied to the map by loading a JSON style configuration from the app's “raw” folder user “res” using “MapStyleOptions.loadRawResourceStyle(getContext(), R.raw.map_style)” to fetch it. This integration process was important for displaying and interacting with a Google Map on the home screen and allowed for further customization and functionalities as needed to meet the objectives.

```
//Google Maps
implementation ("com.google.android.libraries.places:places:3.4.0")
implementation ("com.google.maps.android:android-maps-utils:2.2.0")
implementation ("com.google.android.gms:play-services-maps:18.2.0")
implementation("com.google.android.gms:play-services-location:21.2.0")
implementation ("com.google.maps:google-maps-services:0.9.3")
```

5.3.7.I

```
buildConfigField( type: "String",  name: "API_KEY",  value: "\"AIzaSyDTLowd6S3PrpETeL-0Ep4ZvKJd3co0odk\"")
```

5.3.7.II

5.3.8 Location Updates, Camera Animation and Markers

In the app, location updates, camera animation, and basic marker placement were fundamental components to offer an interactive map experience. Location updates were managed using the “FusedLocationProviderClient”, which was initialized in the activity or fragment where the map is. The “LocationRequest” is configured to specify update intervals and accuracy, ensuring that the app receives precise location data efficiently. When a new location is detected, the app responds by updating the map's camera position to center on the user's current location. This camera movement was configured using “CameraUpdateFactory.newLatLngZoom(new LatLng(location.getLatitude(), location.getLongitude()), 14)”, which not only moves the camera but also applies an appropriate zoom level to provide a clear view of the surrounding area. Simultaneously, basic markers were added to the map to show specific points of interest or the user's location. These markers were created using “MarkerOptions”, setting the position, title, and optional custom icons to make them easily identifiable.

5.3.9 Map Types, Traffic Layer and Current Location Button

In the Home Fragment of the app, a few features were integrated to enhance the mapping experience: map types, a traffic layer, and a current location button. Users can switch between Default, Satellite, and Terrain map types via a “PopupMenu” implemented in the code, which adjusts the map's appearance with the “gMap.setMapType()” method based on the user's selection. The traffic layer functionality is toggled by a button that enables or disables real-time traffic data on the map using “gMap.setTrafficEnabled()”, depending on a Boolean variable. Additionally, the current location feature is facilitated by a button that, when clicked, invokes the “FusedLocationProviderClient” to fetch the user's latest geographical position and centers the map on these coordinates using “gMap.animateCamera()”. These features are effectively set up within the “onViewCreated” and “onMapReady” lifecycle methods, ensuring they are ready as soon as the map is loaded, thereby providing a responsive user experience.

5.3.10 Nearby Places Search

The implementation of Nearby Places included adding “LocationModel”, “GeometryModel”, “GooglePlaceModel”, and “GoogleResponseModel”, to fetch the data from Google's Places API. Retrofit, a network library, handles the API call, ensuring a good communication with Google's services. As responses are received, they're parsed into “GooglePlaceModel” objects, each holding information

about a place, such as its name, location, and user ratings. These models are then added to a list. In the Home Fragment of the application, the “getPlaces()” method supports the RetrofitAPI, constructing a URL that includes the user's current location, the search radius, the type of place being queried, and the API key to send a GET request to the Google Places API. Once a response is received, the application parses the JSON data using the “GoogleResponseModel” class, which encapsulates a list of “GooglePlaceModel” objects. Each “GooglePlaceModel” contains information about a specific place, with “GeometryModel” and “LocationModel” classes extracting the detailed geographical coordinates. Another functionality in the fragment was provided by a ChipGroup component that allows the user to select different types of places to search for. When a new place type is selected, the “getPlaces()” method is triggered, sending a fresh request to the API. The visibility of a RecyclerView, which displays these places, is toggled based on whether any places are currently available for the selected type. Markers for places were added to the Google Map via the “addMarker()” method. This method constructs a “MarkerOptions” object using the location information and the name from each “GooglePlaceModel”. The markers followed to be customized using the “getCustomIcon()” method, making the map visually appealing. The RecyclerView was paired with a “LinearLayoutManager” and a “PagerSnapHelper”, which improved the user experience by providing a smooth swiping action. As users scroll through the list of nearby places, the map automatically updates to focus on the selected location. On the other hand, interacting with the map's markers adjusts the RecyclerView's scroll position to the corresponding place in the list. By integrating these components, the Home Fragment becomes interactive. Users can browse nearby locations, get information about each place, and even initiate navigation directly from the map. This integration of the map enchances the usability.

5.3.11 Custom Marker

Creating custom markers in the Home Fragment involved several steps. Firstly, a drawable resource was used to define the appearance of the marker. This resource is a vector that visually represents the place on the map and it was stored in the drawable folder of the application's resources. When adding a marker to the map, the “addMarker()” method of the “GoogleMap” instance is called. For each place, a “MarkerOptions” object is created, specifying the position of the marker using latitude and longitude from the “GooglePlaceModel”. The title and snippet of the marker are set to the name and vicinity of the place, respectively. To apply the custom icon, a method “getCustomIcon()” was implemented. This method creates a “BitmapDescriptor” from the drawable resource that was set for the marker. Within this method, a “Drawable” was obtained from the resources, its properties such as tint and bounds were set, and then it was drawn onto a “Canvas” which is backed by a “Bitmap”. This “Bitmap” was then converted into a “BitmapDescriptor” using “BitmapDescriptorFactory.fromBitmap(bitmap)”. The “BitmapDescriptor” is what the Google Maps API used to display the custom image as a marker icon. Once the “BitmapDescriptor” was ready, it was set as the icon for the “MarkerOptions”. The “MarkerOptions” is then passed to the “addMarker()” method of the “GoogleMap” instance, adding the marker to the map with the custom icon. Each marker was tagged with an identifier, which was used to link the marker with the corresponding place in the RecyclerView. Finally, an event listener was added to the markers to handle clicks. When a marker is clicked, the application scrolls the RecyclerView to the selected place and displays additional information in an InfoWindow. This approach improves the visual experience and makes it easier for users to identify different types of locations.

5.3.12 RecycleView for Places & Map animations

As mentioned previously, a RecycleView was implemented in the Home Fragment, to display a list of items, each corresponding to a place of interest such as attractions, restaurants or hotels. For each item in this list, two buttons were included as part of the item layout. These buttons were designed to offer additional interactions for each place displayed in the RecyclerView. The layout of these buttons is styled to match the overall design of the application. The first button was intended to allow users to save the locations for easy access later. This feature aimed providing a way for users to keep track of places they find interesting or want to visit again. The second button was designed to facilitate navigation by providing route directions from the user's current location. This button initiates the process of getting directions, making it efficient and user-friendly.

5.3.13 Custom InfoWindow

The “InfoWindowAdapter” class provides custom information window that appears when the user clicks on markers on the map in the app. First it was necessary to design the layout to display the distance and the estimated time. The author had to create a class adapter which extends the InfoWindow Class. The constructor of InfoWindowAdapter accepts a Location object representing the user's current location and a Context object. It inflates the layout using “InfoWindowLayoutBinding.inflate”, initializing the binding with the layout which allows the setup of text views within the info window. The “getInfoWindow” method is overridden to provide a custom view for the entire info window, not just the contents. It sets the title of the info window to the marker's title, which represents the location name. It calculates the distance from the current location to the marker using “SphericalUtil.computeDistanceBetween”, converting this distance from meters to kilometers if necessary. This distance is then displayed in the info window. Additionally, if the user’s current speed is available, it estimates the time required to reach the destination and displays it; otherwise, it shows "N/A" for not available. The “getInfoContents” method is similar to “getInfoWindow” but was intended to format the contents inside the window. The adapter uses the same calculations for distance and time, setting the respective views in the info window.

5.3.14 Estimated Time & Distance

In the app, the estimated time and distance to a destination were calculated using Google's Map Utils class. This utility class uses methods to calculate distances between points on a map, which is useful for showing how far away each place is from the user’s current location. The app displays this distance directly in the info window of each marker. For a more precise journey time and route, the app uses Google's Directions API. When directions are requested, this API considers various factors like roads, traffic, and travel mode (like walking or driving) to provide an exact route and travel time. This feature is especially handy because it gives users a detailed idea of how long it will take to reach their destinations, helping them plan their time more effectively.

5.3.15 Save & Delete Place from Firebase

The process of saving and deleting places using Firebase Realtime Database is managed through specific methods. To save a place, when the "Save" button is clicked in the app, the “onSaveClick()” method is invoked. This method first checks if the place already exists in the user's saved locations on Firebase at “/Users/{userId}/SavedLocations”. If not present, a new “SavedPlaceModel” is created, populated with details such as name, vicinity, latitude, longitude, and rating. This model is then serialized and pushed to Firebase using the “setValue()” method, which stores it under a unique key

generated by the push() method. If a place is already saved and the user clicks the "Save" button again, this triggers the delete option, which invokes the “removePlace()” method. This method accesses the specific place's key under the user's saved locations in Firebase and uses the “removeValue()” method. This action deletes the entry from the database and from the user's saved locations. Both operations are integrated with real-time updates, ensuring that any changes are immediately reflected.

5.3.16 User Saved Places

For Saved Places, the author created a SavedPlacesFragment which used Firebase authentication and set up a RecyclerView in the “onCreateView” method. This RecyclerView was designed to display saved location entries that were stored in Firebase under the user's specific node, identified by their authentication ID. The main functionality was developed in the “getSavedPlaces” method. Here, a Firebase query retrieves all saved locations for the logged-in. This method set up a FirebaseRecyclerAdapter, which was directly linked to the Firebase query. The adapter listens for real-time updates to the Firebase database, ensuring that any changes are immediately reflected in the RecyclerView displayed to the user. Each entry in the RecyclerView is represented by a ViewHolder, which is bound to the data fetched from Firebase. The “onBindViewHolder” method in the adapter is responsible for taking data retrieved from Firebase and applying it to the ViewHolder. For each location entry, it sets up views such as text fields or image views. Interactions are also managed in this setup. Clicking on the image view of a saved location in the list will display the map with the route between the current location of the user and the selected place, along with step details. Moreover, if the saved location data changes, the adapter handles these changes on the fly without needing to reload the entire dataset manually.

5.3.17 Direction Activity

In the “DirectionActivity”, the “getDirection” method was intricately designed to handle the retrieval of directional data from the Google Maps API. The author started by extracting the list of locations from the “ItineraryModel” which was passed to the activity through an intent. This list included the starting point and the destination. The method then constructed a query URL for the Google Maps Directions API. This URL included parameters specifying the origin (starting point), destination (end point), the travel mode (such as driving, walking, or public transit), and the API key required for authentication. Once the URL was constructed, the author employed Retrofit to manage the network request. The “RetrofitAPI” interface defined the necessary endpoints, and the “getDirection” method used this interface to enqueue a request to the API. This asynchronous request fetched the JSON response containing route details such as the estimated travel time, distance, and step-by-step navigation instructions. Upon receiving a successful response, the author parsed this JSON data into the model classes (“DirResponseModel”, “DirRouteModel”, etc.). Each segment of the route (referred to as a "leg") and each step within those legs were extracted and stored. The method also included error handling to manage any issues with the API request, such as failures due to network errors or incorrect API responses. In case of an error, appropriate user feedback was displayed using Toast messages. Finally, the directional data extracted from the response was used to populate a RecyclerView through the “DirStepAdapter”. This adapter laid out each step of the journey in a user-friendly manner, allowing users to scroll through the detailed route directions. The RecyclerView was set up in the activity layout and updated dynamically with the route details once the data was available. This method not only

fetched and parsed the route data but also integrated it nicely into the app's interface, ensuring that users received clear navigation guidance.

5.3.18 Search Bar

The autocompletion search bar in the app is implemented using Google's Places API, specifically tailored for Android applications. This feature enhances the user experience by providing suggestions as they type, making it easier to find specific locations quickly. The author created, below the toolbar, a CoordinatorLayout which contains a TextInputLayout and TextInputEditText, forming a search bar. This search bar allows users to input and search for places directly on the map. The search bar includes an end icon for clearing text. When the search text is cleared, the “onTextChanged” method checks if the text is empty. If so, the RecyclerView, which displays the search results, is set to invisible by setting its visibility to “GONE”. This effectively removes any previous search results from the screen for a tidier appearance. At the same time, the data list used for the RecyclerView (“googlePlaceModelList”) is cleared, and the adapter is updated to reflect this change, ensuring no outdated data remains visible. Additionally, any search-related markers are removed from the map to reset the UI to its initial state without any leftover search indicators. In the app, the search bar is set up to trigger the Autocomplete functionality. This is achieved by setting an “OnClickListener” on the search bar input field. When a user taps on the search bar, the app creates an intent to start an activity provided by the Google Places API, specifically designed for autocompletion of location searches. This activity is called “Autocomplete.IntentBuilder”, which is configured to handle all the setup needed for the autocomplete feature to function. The “IntentBuilder” is configured with specific options to customize the autocomplete suggestions. The API allowed setting a geographical bias, which the app used to prioritize results closer to the user's current location or a specific area. Once the intent was configured, it began with a request code that uniquely identified the autocomplete request. This allowed the app to handle the result in the “onActivityResult” method. When a user selects a place from the suggestions, the activity returns the selected place's details, including its name and location coordinates. Using the “onActivityResult” method, the app checks the request code to ensure it processes the correct result. Upon successful selection of a place, the app retrieves the place details from the intent data. These details are then used to update the user interface, such as displaying the selected place's name in the search bar and marking the location on the map with the custom marker. Additionally, the app uses the RecycleView to perform further tasks like saving the place or initiating a navigation route, just as it is done for the Nearby Places. This integration of the autocomplete search bar with Google Places API significantly streamlines the process of searching for locations within the app, making it convenient and user-friendly for its users.

5.3.19 Chatbot

The chatbot integration within the app is designed to use a predefined set of questions that guide the conversation towards generating a personalized itinerary. In the app's chatbot integration, the conversation begins with an initial question asking the user if they need assistance in planning their itinerary. If the user responds affirmatively, the chatbot proceeds to the next question, continuing the structured dialogue designed to gather specific details about the user's travel preferences and requirements. This chat helps gather information to generate a personalized travel itinerary. However, if the user responds negatively, indicating they do not need assistance, the chatbot delivers a message such as "Okay, if you need help later, just let me know!". Following this, the input field is disabled,

preventing further typing. This design decision ensures the chatbot does not continue to prompt the user unnecessarily, maintaining a clean and user-focused interface. This approach demonstrates thoughtful design by allowing the chatbot to handle both engagement and disengagement gracefully, ensuring the interface remains user-friendly and responsive to the user's current needs. Moreover, messages from the user and the system (chatbot) are differentiated using colors, which helps users easily distinguish between their inputs and the chatbot's responses. Above each message, titles such as "You" for the user and "Your Agent" for the chatbot are displayed. This feature not only improves readability but also makes users feel like they are interacting with a responsive entity. The ChatBot uses the "ChatAdapter" and "ChatBotModel" to manage the flow and display of the conversation in the chat interface. The `ChatBotModel` class defines the structure for each chat message, holding information such as the message text and a flag indicating whether the message is from the user or the bot. The "ChatAdapter", on the other hand, takes a list of "ChatBotModel" instances and binds each one to a view within a "RecyclerView". This setup ensures that messages are displayed in the correct order and style. This structured question-and-answer format encourages users to continue interacting with the chatbot, providing the necessary details to receive a personalized travel plan directly within the app.

5.3.20 Integration of AI

To integrate AI integration into the chatbot, the author utilized OpenAI's GPT-4 model to process and generate responses. The backend setup involves sending user inputs to the OpenAI API, which are formatted as structured prompts including the system's predefined questions. This method allows the AI to contextually understand and generate relevant responses. Specifically, the system captures user inputs through the app's UI, sends them to a server where the API call to OpenAI is made, ensuring that the data is handled securely and efficiently. The server receives JSON responses from OpenAI, which include the AI-generated text that the chatbot then presents to the user in the app's chat interface. When generating travel itineraries, the system separates latitude and longitude coordinates from the rest of the AI-generated response. This is implemented to enhance the usability of the itinerary suggestions by extracting precise geographical data. This separation allows the app to plot points on a map, providing directions. The rest of the response, which includes descriptive text about the itinerary, is displayed to the user in a readable format without the embedded technical details of the coordinates.

```
//Parsing the response from OpenAI
1 usage   ♟ Eliza9a
private void handleOpenAIResponse(final String responseData) {
    getActivity().runOnUiThread(() -> {
        try {
            JSONObject jsonObj = new JSONObject(responseData);
            JSONArray choices = jsonObj.getJSONArray( name: "choices");
            if (choices.length() > 0) {
                JSONObject choice = choices.getJSONObject( index: 0);
                if (choice.has( name: "message")) {
                    JSONObject message = choice.getJSONObject( name: "message"); // Parsing as JSONObject
                    String content = message.getString( name: "content"); // Accessing content directly
                    finalResponse = content;

                    // Removing Lat Longs
                    String cleanedText = content.replaceAll( regex: "\\(Lat:\\s*-?\\d+\\.\\d+,\\s*Long:\\s*-?\\d+\\.\\d+\\)", replacement: "");

                    Log.e( tag: "finalResponse", finalResponse);
                    addMessageToChat(new ChatBotModel(cleanedText, isUser: false));
                    ChatHistory.getInstance().addMessage(new ChatBotModel(content, isUser: false));
                }
            }
        } catch (JSONException e) {
            Log.e( tag: "JSON Parsing Error", msg: "Error parsing response from OpenAI", e);
        }
    });
}
```

5.3.21.I

The method in figure 5.3.21.i processes the JSON response from OpenAI's API call on the main UI thread. It extracts the message content, cleans it by removing latitude and longitude coordinates, and then updates the chat interface with the new, cleaned message. It also logs the original response for debugging purposes. If there's an issue with parsing the JSON, it logs an error.

5.3.21 Reload Button

The author decided to implement a reload button that could reset the conversation, clearing previous interactions and preparing the system for a new session. When the reload button is clicked, the following actions are performed: the internal list maintaining the chat messages is cleared, the RecyclerView adapter (which displays the chat messages) is notified of the data change, clearing the user interface. The adapter is then reinitialized to its starting state, ready to accept new user inputs as if the chatbot session had just begun. It provides a quick way to start the conversation over.

5.3.22 Save Button

The save button in the app provides a straightforward way to save itineraries for future reference. Upon clicking this button, a dialog box asks the user to enter a title for their itinerary, ensuring it can be easily recognized later. After a title is given and the user confirms the save action, the itinerary is stored and accessible from the “Saved Itineraries” page. The implementation details include setting up an OnClickListener for the save button, which triggers the AlertDialog. The user's input is then captured from the dialog, validated to ensure it is not empty, and stored in a structured format in the database. This process involves not only capturing and validating user input but also handling the storage and retrieval operations that allow users to manage their saved data efficiently. This feature adds personalization and organization to the app.

5.3.23 User Saved Itineraries

The “SavedItineraryFragment” in the app allows users to access and manage their saved travel itineraries. The implementation of this feature was focused on providing a user-friendly interface where saved itineraries were displayed in a list format. This was achieved through the setup of a RecyclerView within the fragment, which lists each itinerary item efficiently. The process began in the fragment’s “onCreateView” method, where the RecyclerView was initialized and configured with a LinearLayoutManager. This setup ensured that the itinerary items were displayed in a vertical list, making it easy for users to scroll through and select their desired itinerary. An “ItineraryAdapter” was then instantiated and set as the adapter of the RecyclerView. This adapter was important as it converted the saved itinerary data into visual elements within the RecyclerView. To populate the RecyclerView with data, the “getSavedItineraries” method was invoked. This method fetched the list of saved itineraries from Firebase Realtime Database, using data specific to the logged-in user by using their unique user ID. The retrieval and handling of data were performed with Firebase's ValueEventListener attached to the database reference. Inside the “onDataChange” callback, the snapshot of the itinerary data was processed. Each itinerary entry, stored in Firebase as JSON, was deserialized into an “ItineraryModel” object. These models were collected into a list and then assigned to the adapter, prompting the RecyclerView to update and display the newly loaded itineraries. Real-time updates were important, as any changes made to the itinerary data in Firebase are immediately reflected in the app's interface. This ensures that the user interface is always in sync with the backend database, enhancing the application's responsiveness. Upon selecting an itinerary from the list, users

are directed to the “MultiDirectionActivity”, which handles the display of detailed directions for each itinerary point. This activity is an advancement over the simpler “DirectionActivity”, as it supports routing through multiple waypoints, a common requirement for complex travel plans. The necessity for “MultiDirectionActivity” arose from the need to manage multiple waypoints within a single itinerary, which the “DirectionActivity” was not initially designed to handle. In “MultiDirectionActivity”, the “getDirection” method played an important role. It constructed a request to the Google Maps Directions API, incorporating all waypoints of the selected itinerary into the request URL. The method loops through each waypoint, formatting them into a string that meets the API’s requirements. Once the directions data was retrieved from the API, the “MultiDirectionActivity” parsed the response to extract essential details for each leg of the journey, such as distance and duration. Each leg's details are then displayed in a separate RecyclerView, providing users with step-by-step navigation instructions. The transition from “DirectionActivity” to “MultiDirectionActivity” marks an improvement in the app's functionality, accommodating more complex navigation needs. This change not only broadens the app's utility but also enhances user engagement by offering a more detailed and interactive travel planning experience. The ability to handle multiple waypoints effectively makes `MultiDirectionActivity` a pivotal component of the itinerary management system in the app, providing users with comprehensive and easy-to-follow travel directions.

5.3.24 Help Fragment

The author decided to incorporate a Help section to clarify the app's features and functionalities. This section is organized as a series of questions (highlighted), allowing for easy navigation, coupled with comprehensive answers that showcase the app's range of capabilities.

5.3.25 About Fragment

The About section in the app provides a concise description of the application. It serves as a reference point for new users to understand the intent behind the app and how it can be utilized effectively in their daily activities or specific needs.

5.4 Testing

The author conducted testing on the emulator within the development environment. The testing was done at each stage of the project in order to provide valuable results. This approach allowed for thorough testing of the functionalities and user interface without the need for a physical device. By using the emulator, the author could iterate quickly and refine the app’s features based on the testing results. External involvement was unnecessary as the developer conducted comprehensive testing.

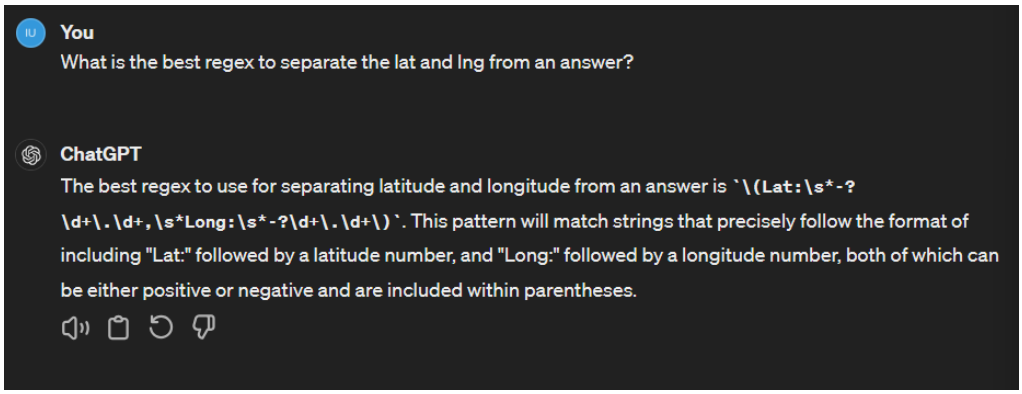
5.5 Evaluation

The author evaluated the app once it was completed. Evaluation of the system involved testing procedures to ensure functionality, usability, and reliability. The process included manual testing techniques to identify and rectify any potential issues or bugs. The system was subjected to various test scenarios, including input validation, edge cases, and stress testing to assess its performance under different conditions.

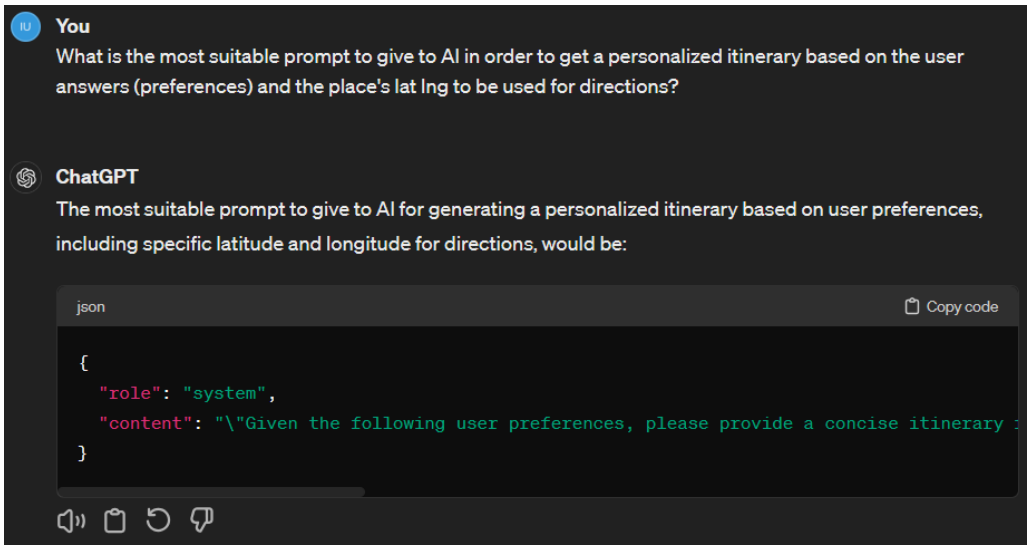
5.6 Use of AI

AI was employed in this project to tackle two specific challenges: identifying the correct regex for extracting latitude and longitude from responses and creating the best prompt for querying AI to receive personalized itinerary suggestions.

The author found it difficult to find the right regex for separating the latitude and longitude from the AI answer. After several attempts and tests with manually written regex patterns, the decision was made to seek assistance from AI. Below is the regex solution that was found to be effective:



Another case for which the AI has been used, was in formulating the prompt to the OpenAI. The goal was to ensure the query would offer the most tailored responses, including specific details like transport timing, meal suggestions, activities, and notable points of interest, all complete with latitude and longitude coordinates. Here's the refined prompt developed with AI assistance:.



“{ "role": "system",

"content": "\"Given the following user preferences, please provide a concise itinerary for the specified time, including transport timing, meal suggestions, activities, and any notable points of interest with Latitude and Longitude like this (Lat: ,Long:) in decimal degrees.\""

}”

5.6 Usability Scale

Question	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
I think that I would like to use this app frequently.					X
I found the app unnecessarily complex.	X				
I thought the app was easy to use.				X	
I found the various functions in this app were well integrated.				X	
I thought there was too much inconsistency in this app.	X				
I would imagine that most people would learn to use this app very quickly.					X

CHAPTER 6 - CONCLUSIONS AND DISCUSSION

Reflecting on the development of RouteGuide, it’s clear that the app has largely met its main objective: to transform travel with AI-driven personalization. Despite challenges, the author has delivered an app that not only navigates but also suggests and adapts to each user’s preferences. While the search functionality for discovering en-route attractions is still in process stage, the progress made so far is more than the author expected to get done. The app adeptly tailors travel experiences.

6.1 Project Objectives

6.1.1 Primary Objective: Build a Navigation App with AI-Driven Itinerary Suggestions

As stated in section 1.3 of this document, the main objective of the project was to design a more user-friendly navigation app that allows users to get AI-driven itinerary suggestions and to search places on the route. This was achieved by developing an application that allows people to find places and navigate to them and to chat with AI to get a personalized itinerary in two languages. Also, the app offers the possibility of saving favorite places to find them quicker and saving itineraries to keep track of the places that they visited or showing the suggestion to friends. After testing it, it was found that the application met most of the proposed functionalities and the user usability requirements. The source code can be found online in author’s GitHub repository.

6.1.2. Sub-Objectives:

The author created a secure and user-friendly system for account management, enabling users to effortlessly manage their profiles and personal data.

The navigation map feature has been implemented to provide comprehensive and accurate routes, accommodating various modes of travel and integrating dynamic traffic updates.

While the search functionality has proficiently allowed users to locate places both globally and within their immediate vicinity, one ambition remains partially unfulfilled. Although the app allows for the discovery of local and global destinations, this feature will include places along the user's route in future updates, enriching the travel experience further by uncovering hidden gems on the journey.

The integration of an AI-driven chatbot has successfully brought interactive and adaptive travel recommendations to match our user’s preferences. Additionally, the capability to save and revisit favorite places and itineraries has been embedded into RouteGuide and the chat has been made available in two languages with further improvements in updates.

Altogether, these achievements highlight RouteGuide's transformation into a great travel companion, demonstrating how AI technology can profoundly elevate the travel planning and navigation experience. Looking ahead, we are excited to extend the app's capabilities and continue refining the way travelers explore the world.

6.2 Future Work

Moving forward with the RouteGuide project, there are several key steps for future:

The first goal is presenting RouteGuide idea to industry leaders like Google. A collaboration with such pioneers would be great to refine the app further and explore potential integrations or partnerships that could enhance the platform's capabilities and reach.

Secondly, the app must be refined, ensuring high-level features are integrated and perform seamlessly, preparing it for launch on the Google Play Store to make it widely accessible for all users.

To broaden the app's reach, the development of a version for iOS is mandatory, expanding our user base by making RouteGuide available on the Apple Store, thus inviting all smartphone users to experience our personalized navigation system.

Suggested updates:

Version	Description
V1	First release
V2	Make the app available in multiple languages Add offline service
V3	Add places on the route
V4	Add in app ticket purchase for public transport

6.3 Knowledge Gained

This project was a great opportunity to enhance programming in Java skills. There were many hours spent on learning how to use the Android Studio features, OpenAI API or how to use Firebase, but the most difficult was the integration of Google Maps API to complete most of the proposed features. The author learned that even when the desired output seems to be almost completed, there is always something more that can actually bring problems in development.

6.4 Project Management

In managing the project, there was a significant evolution from the initial plans to the current stage. Adjustments made along the way highlighted the importance of adaptability and strategic thinking in software development. Moreover, understanding that a complex application might not be easy to complete was crucial since not all the functionalities were easy to implement, as described in Chapter 4.

6.5 Conclusion

As the project wraps up, I reflect on the journey as an extraordinary learning experience, reinforcing my skills and confirming my commitment to creating solutions that bring ease and personalization to users' travel experiences. The positive feedback received in the evaluation phase has been incredibly encouraging and points towards a promising direction for RouteGuide.

GLOSSARY

A

API (Application Programming Interface): A set of protocols and tools for building software applications, specifying how software components should interact.

Authentication: The process of verifying the identity of a user or process.

C

ChatBot: A software application used to conduct an online chat conversation via text or text-to-speech, simulating a human conversation.

D

Database: A structured set of data held in a computer, especially one that is accessible in various ways

Directions API: A service from Google Maps that provides directions between multiple locations.

F

Firebase: A platform developed by Google for creating mobile and web applications, offering services like analytics, databases, messaging, and crash reporting.

Fragment: A reusable class in Android that represents a portion of the user interface or behavior in an Activity

I

Itinerary: A planned route or journey, showing the path and stops along the way.

J

JSON (JavaScript Object Notation): A lightweight data-interchange format that is easy for humans to read and write, and easy for machines to parse and generate.

L

Latitude and Longitude (Lat, lng): Coordinates used to specify the precise location of something on Earth.

R

RecyclerView: An advanced and flexible version of ListView in Android that is used for displaying large sets of data, providing the ability to recycle and reuse views as they are scrolled off screen.

Retrofit: A type-safe HTTP client for Android and Java that makes it easier to consume RESTful web services.

U

User Interface (UI): The means by which the user and a computer system interact, in particular the use of input devices and software.

V

ViewModel: A class that is responsible for preparing and managing the data for an Activity or a Fragment, handling the communication of the Activity / Fragment with the rest of the application.

REFERENCE LIST

www.youtube.com. (n.d.). *Near Me | Firebase, Google Map, Place & Direction API's | Intro & Setup Project Android Java Part(1)*. [online] Available at: https://www.youtube.com/watch?v=dfTzz5AhBNQ&list=PLpQFhyCcxiCqDFYQabluYIYsNSsCFMNFk [Accessed 1 Apr. 2024].

www.youtube.com. (n.d.). *How to use Google Place Api using AutoComplete?* [online] Available at: https://www.youtube.com/watch?v=Nw8OM5Q0mmg&list=LL&index=10&t=706s [Accessed 1 Apr. 2024].

www.youtube.com. (n.d.). *Google Maps & Google Places Android Course*. [online] Available at: https://www.youtube.com/watch?v=OknMZUnTyds&list=PLgCYzUzKIBE-vInwQhGSdnbyJ62nixHCt.

Android Developers. (n.d.). *Troubleshoot known issues with Android Emulator | Android Studio*. [online] Available at: https://developer.android.com/studio/run/emulator-troubleshooting [Accessed 1 Apr. 2024].

Stack Overflow (2024). *Stack Overflow - Where Developers Learn, Share, & Build Careers*. [online] Stack Overflow. Available at: https://stackoverflow.com/.

APPENDICES

APPENDIX A

Project Definition Document

TABLE OF CONTENTS

PROBLEM TO BE SOLVED

Existing maps often provide generic travel routes that don't consider individual traveler preferences, interests or time constraints.

Tourists frequently miss out on attractions that are less known, local experiences, and cultural insights due to the lack of integrated local knowledge in most travel apps.

Moreover, travelers are often overwhelmed by the amount of information available online and find it challenging to plan a trip efficiently.

A few similar applications I have found are: "Google Maps"[3], "Wanderlog"[4].

PROJECT BENEFICIARIES

This project will help travelers and people using navigation maps. Moreover, this project can be used as inspiration for further development of existing apps.

Many people find the current navigation maps confusing when they try to find nearby, or 'en-route' amenities and this project aims to make a map more user friendly.

[3] Google Maps: maps.google.com
[4] Wanderlog: wanderlog.com

Also, people that travel tend to search attractions on websites and create routes by themselves, but that can be exhausting because of the amount of information that can be found online and sometimes the weather might not match the plan, so rearrangements are required. RouteGuide will have AI-driven route suggestions based on user preferences and it will make planning the journeys less stressful. The users will be able to choose the time availability, type of attrcations, activity level, transportation preferences, budget, cultural interests, food preferences, accessibility needs, family-friendly options, shopping interests and nature.

PROJECT OBJECTIVES

The project shall provide an interactive, AI-powered mobile app designed for people that travel. The app will improve exploration and navigation by offering personalized route suggestions and providing en-route information, such as amenities and attractions. Furthermore, the app intends to have a better user experience compared to the existing softwares.

List of objectives and subobjectives I want to achieve while completing the project:

Objectives	Subobjectives	Testing
Research and project planning	Compare existing maps	Find apps with similar purpose and find what they are missing. Take notes of functionality and improvement needs
	Decide how the app will be developed	Consider the options of programming languages and tools and decide which are suitable.
Design & prototype	User interface	Design a user-friendly interface and ask for feedback frequently to ensure that the map is easy to navigate.
App Development	Learn how to use APIs and Android Studio	Study the tools that can be used for my project and test the capabilities in cycles.
	Implement the map and its features	Building the map will begin early to ensure that all the elements can be added adequately. Feedback will

		be required often during this process.
	Implement AI driven suggestions	Learn how to integrate the AI and test it with different options within the app.
Submission	Collect feedback	Gather feedback from end-users and see if the app lacks something.
	Make last updates	Based on the feedback, make last changes and make sure it meets all the requirements.
	Report	Keep track of work versions, save the feedback received and take notes from the discussions with the supervisor.

WORK PLAN

Research & Planning

Output: Project Definition Document

Method: Compare the existing apps and conduct surveys

Technical Skill Development

Output: Knowledge in required development tools and languages

Method: Complete tutorials (e.g., Android Studio, Java, Google Maps API) and practice exercises to gain familiarity with development tools

Initial App Design and Prototyping

Output: App design prototypes and user flow diagrams

Method: Design UI/UX prototypes (Sketch or Figma) and create user flow diagrams

Development Environment Setup

Output: Creation of a database and connection to GitHub

Method: Install and configure all necessary development tools and platforms

Map Development

Output: Functional map interface with basic features

Method: Program the map interface and integrate Google Maps API[5]

Integration of En-Route Attractions

Output: Improved map

Method: Integrate the En-Route Attractions on the map

Development of Other Features

Output: User profile management and AI algorithms

Method: Implement AI-based personalization features and user authentication

Activity 8: Testing and Improvement

Output: Tested and final app version

Method: Conduct iterative testing cycles, update the app based on feedback

Activity 9: Finalization and Submission

Output: Ready-to-submit app and final project report

Method: Finalize the app, prepare the project report

[5] Google Maps API: Google Maps Platform Documentation | Places API | Google for Developers

Gantt Chart [6]

RouteGuide - Final Year Project

Project Start: Mon, 29/01/2024

Display Week: 1

TASK	PROGRESS	START	END
Before Development			
Requirements Research	100%	29/1/24	31/1/24
Programming Research	90%	31/1/24	2/1/24
PDD	80%	2/1/24	12/2/24
Development Phase 1			
Design User Interface	0%	12/2/24	19/2/24
Databases	0%	19/2/24	22/2/24
Map	0%	22/2/24	27/2/24
Navigation Menu	0%	27/2/24	4/3/24
Test	0%	4/3/24	4/3/24
Development Phase 2			
Main Functionalities	0%	4/3/24	9/3/24
En-Route Attractions	0%	9/3/24	11/3/24
Test	0%	11/3/24	11/3/24
Development Phase 3			
AI Route Suggestions	0%	11/3/24	16/3/24
Test	0%	16/3/24	16/3/24
User Preferences Input	0%	16/3/24	20/3/24
Test	0%	20/3/24	20/3/24
Finalization			
Project Report	0%	20/3/24	1/4/24
Request Feedback	0%	1/4/24	8/4/24
Last Changes	0%	8/4/24	15/4/24
Submission	0%	15/4/24	25/4/24

[6] Gantt Chart: https://www.vertex42.com/ExcelTemplates/simple-gantt-chart.html

ETHICS CHECKLIST

Computer Science Research Ethics Committee (CSREC)

http://www.city.ac.uk/department-computer-science/research-ethics

PART A: Ethics Checklist

A.1 If you answer YES to any of the questions in this block, your consultant/supervisor must have obtained approval for the project from an appropriate external ethics committee, and you need to have received written confirmation of this from him/her. Students cannot themselves apply for ethics approval in this case as the project is considered high risk". This type of research is not covered by City's process, and external approval from an appropriate institution is required.		
1.1	Does your research require approval from the National Research Ethics Service (NRES)?	**NO**
1.2	Will you recruit participants who are covered by the Mental Capacity Act 2005?	**NO**
1.3	Will you recruit any participants who are covered by the Criminal Justice System, for example, people on remand, prisoners and those on probation?	**NO**
A.2 If you answer YES to any of the questions in this block your consultant/supervisor must have obtained appropriate ethics committee approval		
2.1	Does your research involve participants who are unable to give informed consent?	**NO**
2.2	Is there a risk that your research might lead to disclosures from participants concerning their involvement in illegal activities?	**NO**
2.3	Is there a risk that obscene and or illegal material may need to be accessed for your research study (including online content and other material)?	**NO**

2.4	Does your project involve participants disclosing information about protected characteristics (as identified by the Equality Act 2010)?	**NO**
2.5	Does your research involve you travelling to another country outside of the UK, where the Foreign & Commonwealth Office has issued a travel warning that affects the area in which you will study?	**NO**
2.6	Does your research involve invasive or intrusive procedures?	**NO**
2.7	Does your research involve animals?	**NO**
2.8	Does your research involve the administration of drugs, placebos or other substances to study participants?	**NO**
A.3 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee or the Senate Research Ethics Committee (SREC), you must apply for approval from the Computer Science Research Ethics Committee (CSREC) through Research Ethics Online - https://researchmanager.city.ac.uk/. Depending on the level of risk associated with your application, it may be referred to the Senate Research Ethics Committee (SREC).		
3.1	Does your research involve participants who are under the age of 18?	**NO**
3.2	Does your research involve adults who are vulnerable because of their social, psychological or medical circumstances (vulnerable adults)?	**NO**
3.3	Are participants recruited because they are staff or students of City, University of London?	**NO**
3.4	Does your research involve intentional deception of participants?	**NO**
3.5	Does your research involve participants taking part without their informed consent?	**NO**
3.5	Is the risk posed to participants greater than that in normal working life?	**NO**
3.7	Is the risk posed to you, the researcher(s), greater than that in normal working life?	**NO**
A.4 If you answer YES to the following question and your answers to all other questions in sections A1, A2 and A3 are NO, then your project is deemed to be of MINIMAL RISK. **If this is the case, then you can apply for approval through your supervisor under PROPORTIONATE REVIEW. You do so by completing PART B of this form.** **If you have answered NO to all questions on this form, then your project does not require ethical approval. You should submit and retain this form as evidence of this.**		
4	Does your project involve human participants or their identifiable personal data?	**NO**

PART B: Ethics Proportionate Review Form

If you answered YES to question 4 and NO to all other questions in sections A1, A2 and A3 in PART A of this form, then you may use PART B of this form to submit an application for a proportionate ethics review of your project. Your project supervisor has delegated authority to review and approve this application under proportionate review. You must receive final approval from your supervisor in writing before beginning the planned research.

However, if you cannot provide all the required attachments (see B.3) with your project proposal (e.g. because you have not yet written the consent forms, interview schedules etc), the approval from your supervisor will be ***provisional***. You **must** submit the missing items to your supervisor for approval prior to commencing these parts of your project. Once again, you must receive written confirmation from your supervisor that any provisional approval has been superseded by with ***full approval*** of the planned activity as detailed in the full documents. **Failure to follow this procedure and demonstrate that final approval has been achieved may result in you failing the project module and/or result in an academic misconduct investigation.**

Your supervisor may ask you to submit a full ethics application through Research Ethics Online, for instance if they are unable to approve your application, if the level of risks associated with your project change, or if you need an approval letter from the CSREC for an external organisation.

B.1 The following questions must be answered fully.		
1.1.	Will you ensure that participants taking part in your project are fully informed about the purpose of the research?	
1.2	Will you ensure that participants taking part in your project are fully informed about the procedures affecting them or affecting any information collected about them, including information about how the data will be used, to whom it will be disclosed, and how long it will be kept?	
1.3	When people agree to participate in your project, will it be made clear to them that they may withdraw (i.e. not participate) at any time without any penalty?	
1.4	Will consent be obtained from the participants in your project? Consent from participants **MUST** be obtained if you plan to involve them in your project or if you plan to use identifiable personal data from existing records. "Identifiable personal data" means data relating to a living person who might be identifiable if the record includes their name, username, student id, DNA, fingerprint, address, etc.	
1.5	Have you made arrangements to ensure that material and/or private information obtained from or about the participating individuals will remain confidential?	

B.2 If the answer to the following question (B2) is YES, you must provide details	

2	Will the research be conducted in the participant's home or other non-University location?	**NO**

B.3 Attachments **ALL of the following documents MUST be provided to supervisors if applicable.** **All must be considered prior to final approval by supervisors.** **A written record of final approval must be provided and retained.**	***YES***	***NO***	***Not Applicable***
Details on how safety will be assured in any non-University location, including risk assessment if required (see B2)			**X**
Details of arrangements to ensure that material and/or private information obtained from or about the participating individuals will remain confidential (see B1.5)			**X**
Full protocol for any workshops or interviews**			**X**
Participant information sheet(s)**			**X**
Consent form(s)**			**X**
Questionnaire(s)**			**X**
Topic guide(s) for interviews and focus groups**			**X**
Permission from external organisations or Head of Department**			**X**

Further Information

https://www.city.ac.uk/about/governance/committees/cs-research-ethics

https://www.city.ac.uk/research/ethics/how-to-apply/participant-recruitment

https://www.city.ac.uk/research/ethics

Thank you for taking the time to read this information sheet.

APPENDIX B

REUSED CODE DOCUMENTATION

The implementation of the "Nearby Places" and "Saved Places" features in the mobile application, along with supplementary functionalities like directions and camera animations, was significantly guided by an educational tutorial (Near Me | Firebase, Google Map, Place & Direction API's | Intro & Setup Project Android Java Part(1) (youtube.com)). This tutorial served as a foundational resource, providing step-by-step instructions on how to integrate Google Maps APIs and Firebase into an Android application effectively.

Initially, the "Nearby Places" functionality was developed by incorporating the Google Places API. The tutorial detailed how to send requests to the Google Places API to fetch locations within a specified radius based on the user's current geographical position. This involved constructing a request URL that included parameters such as latitude, longitude, radius, type of place, and the API key. The response, containing details of nearby places, was then parsed into a list of location models, which were displayed on the map as markers.

For the "Saved Places" feature, the tutorial introduced the use of Firebase as a real-time database to store and retrieve user-saved locations. It explained how to design the database schema to store location details, and how to implement the functionality to add and remove places from the user's saved list directly from the mobile app interface. Each saved place could be accessed or deleted, with changes instantly reflected in the Firebase database, ensuring real-time data synchronization.

Additionally, the tutorial covered the integration of the Directions API to provide route directions from the user's current location to the chosen place. It demonstrated how to send a request to the API with origin, destination, and waypoints, and then how to parse the response to display a detailed route on the map. The step-by-step directions were also shown in a RecyclerView, enhancing user interaction.

Camera animations were another aspect learned from the tutorial. It taught how to implement smooth transitions for the camera to move and zoom into specific coordinates on the map when a user selects a place from the list. This visual feedback was important for improving user experience by helping users orient themselves with respect to the selected locations on the map.

In addition to the core functionalities, the implementation of data models and adapters within the application was also inspired by the tutorial. The tutorial provided clear examples and templates for creating robust model classes and adapters that facilitate the handling and display of data within the app.

For instance, the location data retrieved from the Google Places API and Firebase database was organized using specifically designed model classes. These classes, such as “PlaceModel” and “LocationModel”, were structured to encapsulate all necessary details about a place, including its name, coordinates, and additional metadata. These models ensured that the data remained manageable and accessible throughout the application.

The tutorial demonstrated how to create adapters, such as the “PlaceAdapter” and “DirectionStepAdapter” to efficiently convert the list of model instances into viewable elements within a RecyclerView. This setup was very important for displaying lists of nearby places and detailed steps of a route in a user-friendly manner.

The integration of these models and adapters, as guided by the tutorial, not only streamlined the development process but also enhanced the application's performance and scalability. By following the patterns and practices laid out in the tutorial, the application was able to efficiently manage and display complex datasets, providing a seamless and responsive user experience.

APPENDIX C

REQUIREMENTS DOCUMENTATION

1.Introduction

This document has been prepared during the planning and requirements gathering phase of the application development project. It details the preliminary work undertaken to set the foundation for the application's design and development.

2.Purpose

This document is intended to outline the initial planning and analysis stages of the application development. It will describe the needs and requirements gathering processes, include a complete UML Use Case Diagram, and list and prioritize all identified use cases.

3.RouteGuide vs. Existing Systems

Feature	RouteGuide	TripIt	Google Maps	Wanderlog
Navigation	Provides turn-by-turn directions	Does not offer navigation	Offers real-time navigation	Limited navigation
Itinerary Management	Allows users to create and manage itineraries	Focuses on itinerary management	Allows itinerary creation but not as detailed	Extensive itinerary planning
Saved Places	Users can save and categorize locations	Trip plans can include saved places	Users can save and label places	Users can save and organize places
Real-time Traffic	May include traffic information	Does not include traffic info	Includes live traffic updates	Does not include traffic info
Offline Access	Limited or no offline functionality	Limited offline functionality	Extensive offline map access	Limited offline access
Social Sharing	Itinerary sharing is possible through screenshots	Itineraries can be shared with other users	Allows sharing of locations and routes	Easy sharing of itineraries
User Interface	Customizable and user-friendly	User-friendly but less customizable	Highly intuitive and customizable	User-friendly and visually appealing
API Integration	Integrates with external APIs for enhanced functionality	Limited API integration	Extensive API integration for developers	Limited API integration
Cost	Free with optional premium features	Free base version, paid Pro version offers more features	Free, charges for API usage beyond free quota	Free with premium features available
Platform	Android	Cross-platform (iOS, Android, Web)	Cross-platform (iOS, Android, Web)	Cross-platform (iOS, Android, Web)

ID	Title	Description
	Functional Requirements	
R01	User Authentication	The application will let the user Login/Signup through Email/Password
R02	Settings	The application will offer the ability of changing the username and the password
R03	Display Map & Locations	The application will offer map and locations displayed on the map on request
R04	Search for Places	The application will allow the users to search places by name or by using buttons for selecting “Nearby Places” or “En-Route Places”
R05	Directions for Navigation	The application will provide routes for the searched places (offering navigation instructions and display the route on the map)
R06	Managing Saved Locations	The application will allow the user to save/remove places from a list
R07	ChatBot Integration	The application will offer a chatbot, driven by AI, to suggest itineraries based on the user’s answers
R08	Managing Saved Itineraries	The application will allow the user to save the itineraries. This offers the possibility of seeing directions for the specific itinerary
R09	Help & About	The application offers help for the users to understand the app.
	Non-functional Requirements	

R10	Performance	The application must run smoothly without major delays
R11	Usability	The application must be user-friendly, easy to navigate, to make the users reuse it
R12	Reliability	The application must provide good performance without bugs/crashes
R13	Scalability	The application must be able to handle a high number of users, without affecting the performance
R14	Security	The application must ensure that the user's data is secure and compliant with the regulations

6.Priority of the Requirements

The table below shows the list of the functional and non-functional requirements along with their priority level.

ID	Priority
Functional	**Requirements**
R01	High
R02	Low
R03	High
R04	High
R05	High
R06	Low
R07	Medium
R08	Medium
R09	Low
Non-functional	**Requirements**
R10	Medium
R11	Medium
R12	Low
R13	Low
R14	Medium

7.Use Case Specifications

UC01	**Sign Up**
Description	Users can create a new account by providing their email, username and password
Primary Actor	User
Secondary Actors	Firebase Database
Preconditions	The app must be installed and on the Sign Up screen
Main Flow	The user enters the email, username and password and clicks on Sign Up button to submit the form. The system validates the data and registers the user
Post Conditions	A new account is created, and the user is redirected to the Login Page
Alternative Flow	If the email is already registered, the user is notified to try with another email

UC02	**User Login**
Description	Users login to the app tp access the user's specific data
Primary Actor	User
Secondary Actors	Firebase Database
Preconditions	User has a registered account
Main Flow	The user enters the email and the password
Post Conditions	User is redirected to the Home Page
Alternative Flow	If the input is wrong, the user is informed and asked to enter other details

UC03	**Password Recovery**
Description	Users can recover a forgotten password by using their registered email
Primary Actor	User
Secondary Actors	Firebase Database
Preconditions	User must have access to the email
Main Flow	The user requests a password reset, enters the email and receives a link
Post Conditions	User receives a link for resetting the password
Alternative Flows	If the email is not registered, the user is

	notified

UC04	**Update Profile**
Description	Allows users to update their profile (username and password)
Primary Actor	User
Secondary Actors	Firebase Database
Preconditions	The user is logged in and
Main Flow	User selects what to be updated, makes changes and saves them
Post Conditions	User's profile information is updated
Alternative Flow	If the updates are not successful, the user is notified

UC05	**View Map**
Description	Users can view and interact with the map
Primary Actor	User
Secondary Actors	Google Maps API
Preconditions	User has allowed location track for the app
Main Flow	User views the map and uses gestures to navigate
Post Conditions	The map is displayed and offers interactivity
Alternative Flow	If the location permission is not granted, notify the user and display a blank view

UC06	**Search Nearby Places**
Description	Users can search for nearby places
Primary Actor	User
Secondary Actor	Google Maps API
Preconditions	User has an internet connection and location services enabled
Main Flow	User selects a place type, and the system retrieves and displays results
Post Conditions	A list of nearby places is displayed
Alternative Flow	If no place Is found, the user is informed

UC07	**Save Favorite Places**
Description	Users can save locations they wish to visit

	again or frequently visit
Primary Actor	User
Secondary Actors	Firebase Database Google Maps API
Preconditions	User is logged in and has found a location they like
Main Flow	User selects the save option on a location, and it's added to their saved places list
Post Conditions	Location is saved to the user's favorites
Alternative Flow	If the location is already saved, the option to remove it is presented

UC08	**Request Direction Instructions**
Description	Users can get detailed directions from their current location to a chosen destination
Primary Actor	User
Secondary Actors	Google Maps API
Preconditions	Users must select the end point
Main Flow	User selects their destination and asks for route and the system calculates and displays the route
Post Conditions	Directions are displayed with the respective data
Alternative Flow	If the route cannot be calculated, alternative suggestions are offered

UC09	**Chat with Chatbot**
Description	Users can get itinerary suggestions, driven by AI, by interacting with the Chatbot
Primary Actor	User
Secondary Actors	OpenAI API
Preconditions	User has opened the chat
Main Flow	User types a question, and the chatbot provides an answer based on the responses
Post Conditions	User receives automated responses
Alternative Flow	If the chatbot cannot understand the question, it skips and uses a default value

UC10	**View Saved Itineraries**
Description	Users can view a list of their saved itineraries
Primary Actor	User
Secondary Actors	Firebase Database Google Maps API
Preconditions	User has got the answer from AI and set a title while saving it
Main Flow	User selects the saved itineraries option to view their list
Post Conditions	A list of itineraries is available to be viewed
Alternative Flow	If no itineraries are saved, the page is blank

UC11	**Real Traffic Updates**
Description	Users can receive up-to-date traffic information
Primary Actor	User
Secondary Actors	Google Maps API
Preconditions	User opts in for traffic layer
Main Flow	The system pushes traffic updates on the map as they occur.
Post Conditions	User is informed of traffic conditions in real-time
Alternative Flow	If there's no traffic data available, inform the user

UC12	**Change Map Style**
Description	Users can select different visual styles for the map interface for enhanced readability or personal preference
Primary Actor	User
Secondary Actors	Google Maps API
Preconditions	User is viewing the map and wants a different visual presentation
Main Flow	The user selects the option to change map styles and chooses from a list of available styles such as default, satellite, or terrain. The map immediately refreshes to display the new style
Post Conditions	Map display updates to reflect the selected

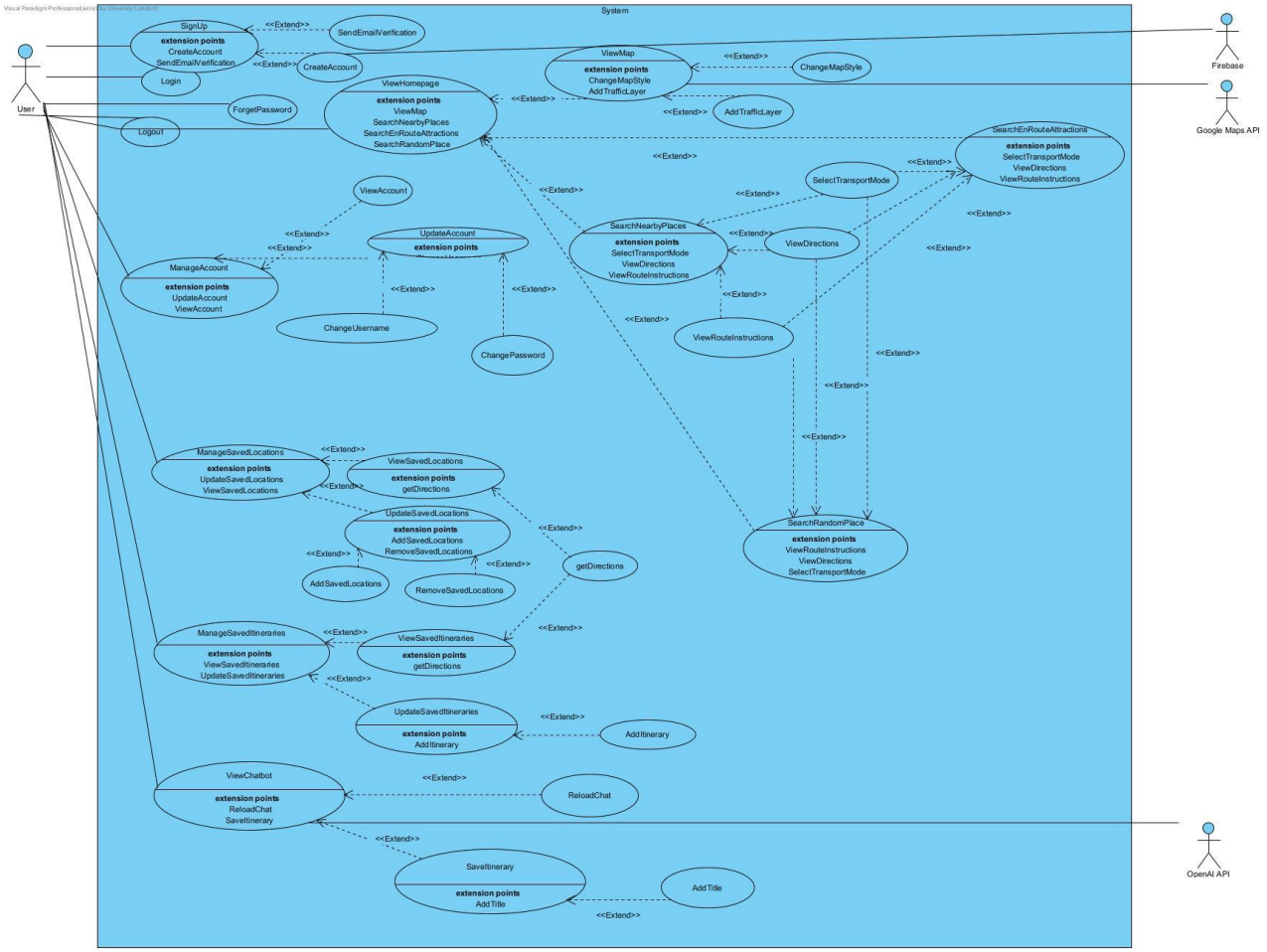
	style
Alternative Flow	If a style fails to load due to an error or connectivity issues, the user is informed and the map reverts to the default style

UC13	**Logout**
Description	Users can securely exit their session
Primary Actor	User
Secondary Actors	Firebase Database
Preconditions	User is logged in
Main Flow	User selects the logout option, and the app confirms session termination
Post Conditions	User session is ended
Alternative Flow	None

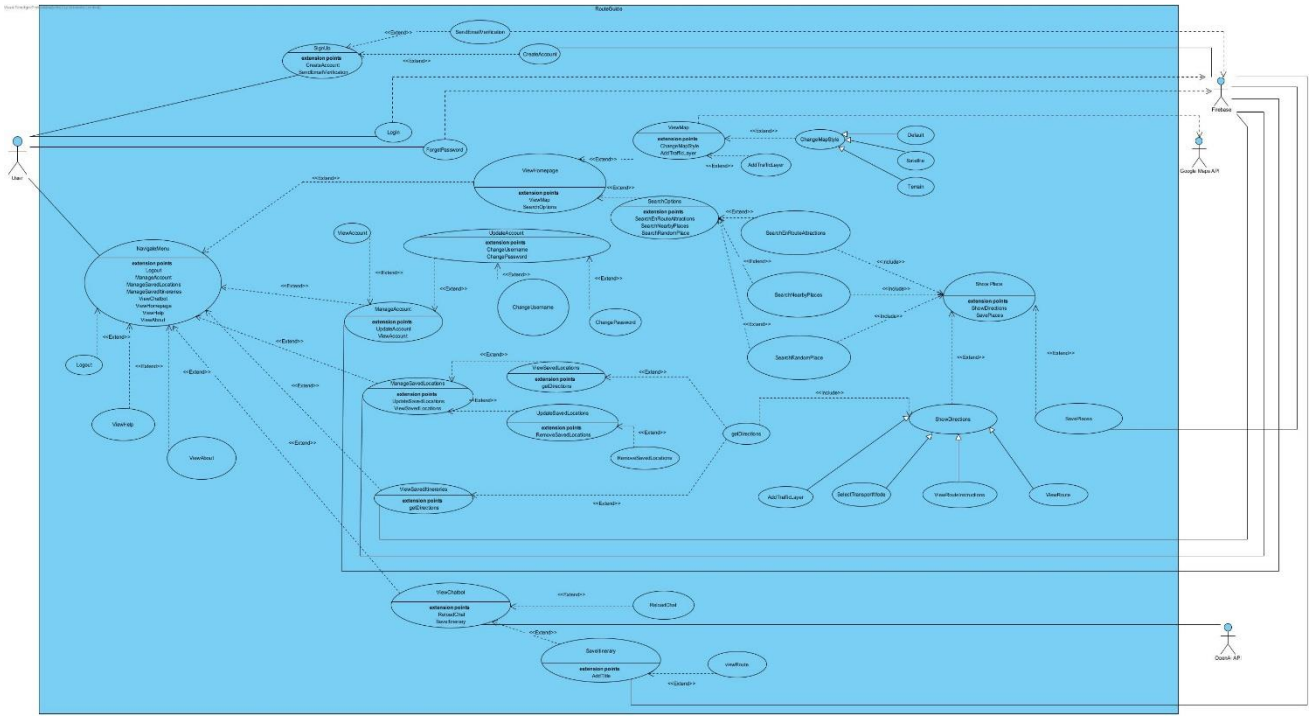
8.Priority of the Use Cases

ID	Priority Level
UC01	High
UC02	High
UC03	Medium
UC04	Low
UC05	High
UC06	High
UC07	Medium
UC08	High
UC09	High
UC10	Medium
UC11	Low
UC12	Low
UC13	Low

9.USE CASE DIAGRAM



USE CASE DIAGRAM V1.0.0



USE CASE DIAGRAM V2.0.0

10.Work Plan

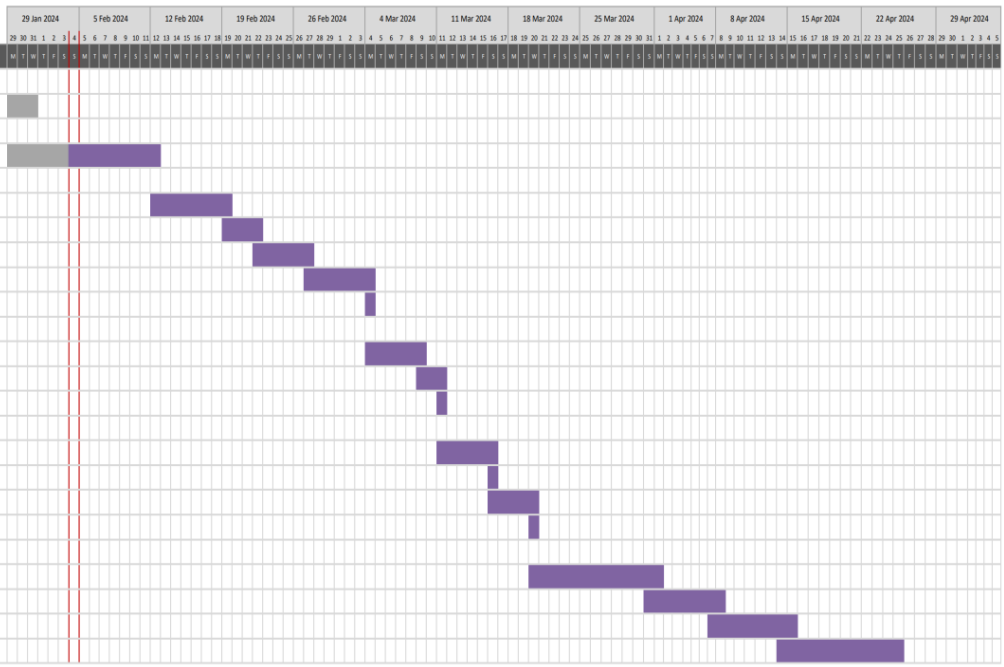
The work plan for the project was carefully set up from the start, which meant that later on, there weren't any big changes needed. The project was broken into smaller parts with specific goals and what needed to be done by when. This made it easier to see how the project was doing and keep everything on track. All of this careful planning meant that the project could go forward smoothly without needing to make big changes to the plan.

RouteGuide - Final Year Project

Project Start: Mon, 29/01/2024

Display Week: 1

TASK	PROGRESS	START	END
Before Development			
Requirements Research	100%	29/1/24	31/1/24
Programming Research	90%	31/1/24	2/1/24
PDD	80%	2/1/24	12/2/24
Development Phase 1			
Design User Interface	0%	12/2/24	19/2/24
Databases	0%	19/2/24	22/2/24
Map	0%	22/2/24	27/2/24
Navigation Menu	0%	27/2/24	4/3/24
Test	0%	4/3/24	4/3/24
Development Phase 2			
Main Functionalities	0%	4/3/24	9/3/24
En-Route Attractions	0%	9/3/24	11/3/24
Test	0%	11/3/24	11/3/24
Development Phase 3			
AI Route Suggestions	0%	11/3/24	16/3/24
Test	0%	16/3/24	16/3/24
User Preferences Input	0%	16/3/24	20/3/24
Test	0%	20/3/24	20/3/24
Finalization			
Project Report	0%	20/3/24	1/4/24
Request Feedback	0%	1/4/24	8/4/24
Last Changes	0%	8/4/24	15/4/24
Submission	0%	15/4/24	25/4/24



WORKPLAN V 1

APPENDIX D

DESIGN DOCUMENTATION

1.Introduction

The Design Documentation was created when planning the project and its requirements and the purpose of this document is to outline the process of designing the application.

2.Graphic User Interface

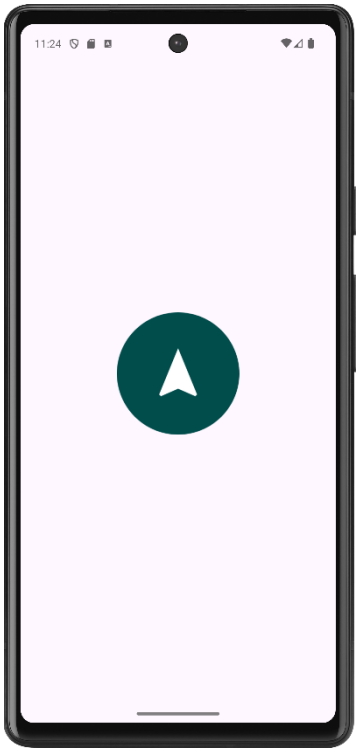


FIGURE 0-1 LAUNCH SCREEN

FIGURE 0-2 LOADING SCREEN

FIGURE 0-3 LOGIN PAGE

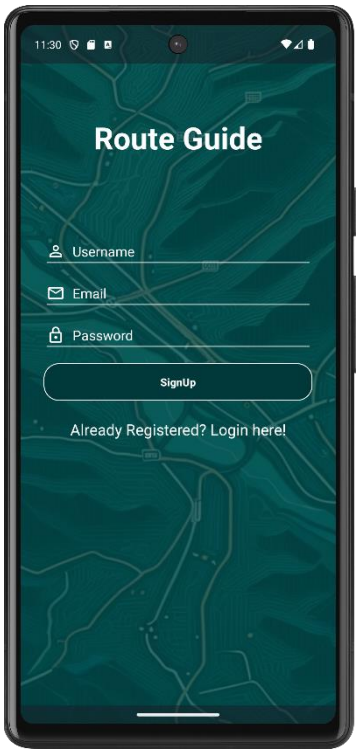


FIGURE 0-4 SIGN UP PAGE

FIGURE 0-5 FORGOT PASSWORD WINDOW

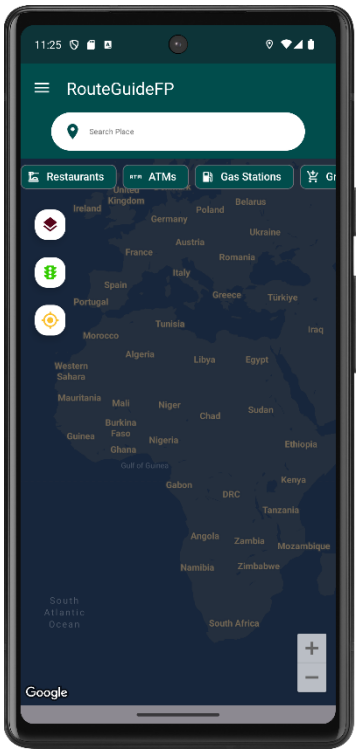


FIGURE 0-6 HOME SCREEN

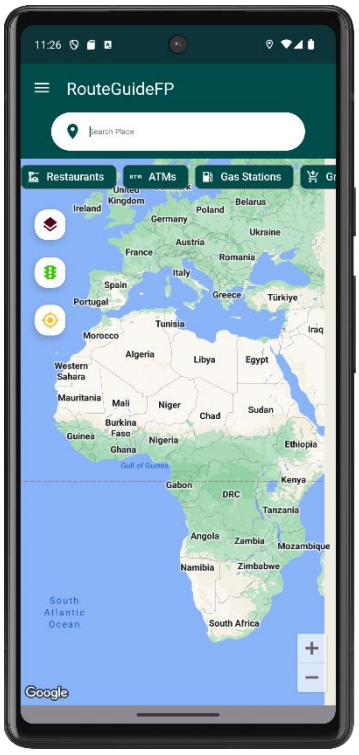


FIGURE 0-7 HOME SCREEN WITH DIFFERENT STYLE (SATELLITE)

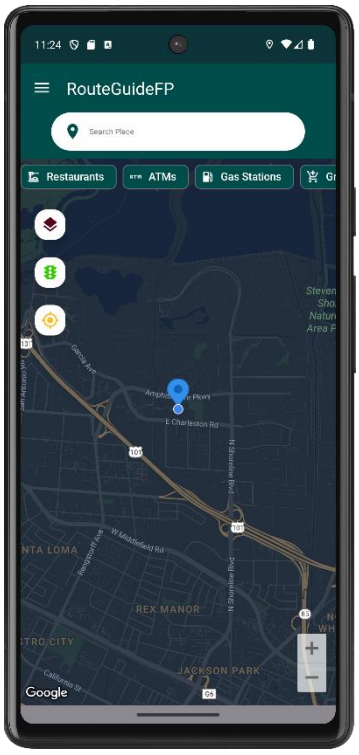


FIGURE 0-8 CURRENT LOCATION SHOWN ON HOME SCREEN

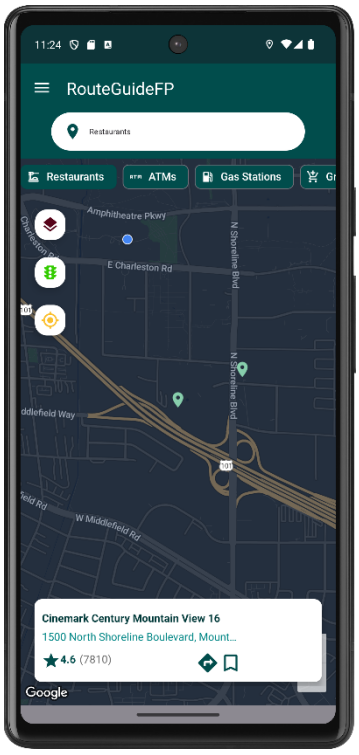


FIGURE 0-9 NEARBY PLACES (MARKERS AND RECYCLEVIEW)

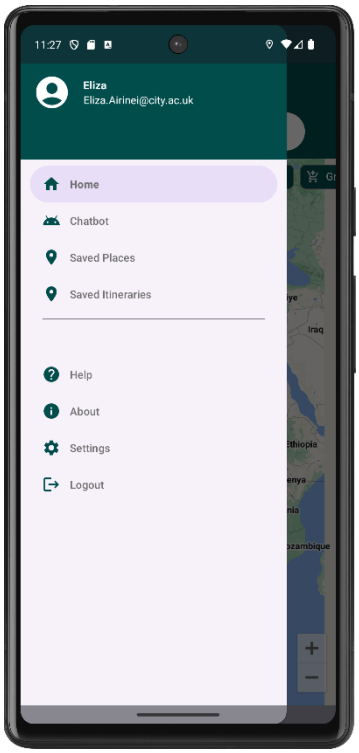


FIGURE 0-10 MENU

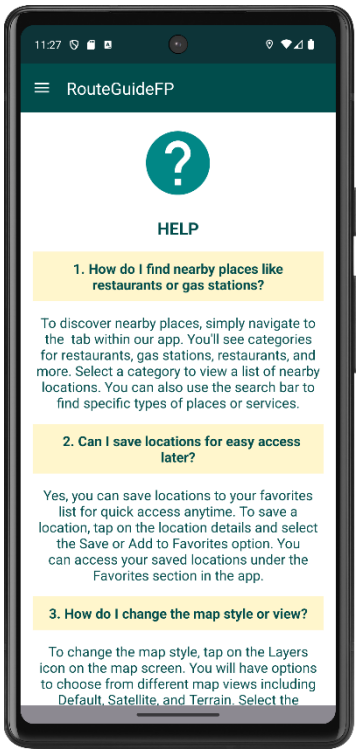


FIGURE 0-11 HELP PAGE

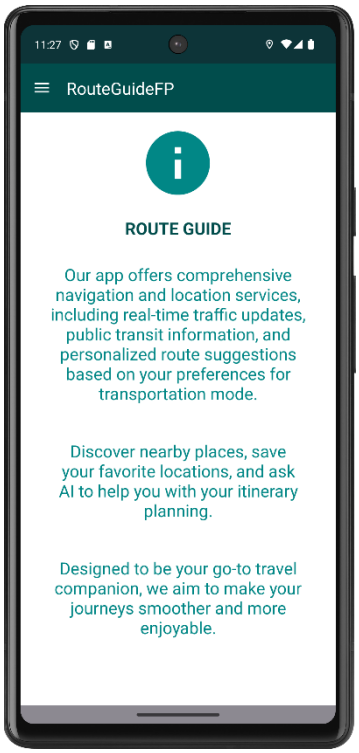


FIGURE 0-12 ABOUT PAGE

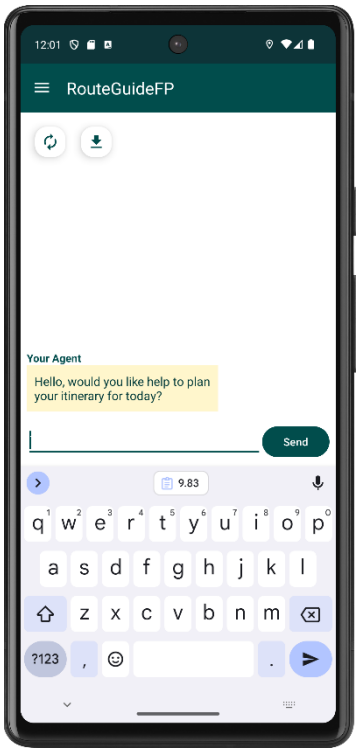


FIGURE 0-13 CHATBOT PAGE

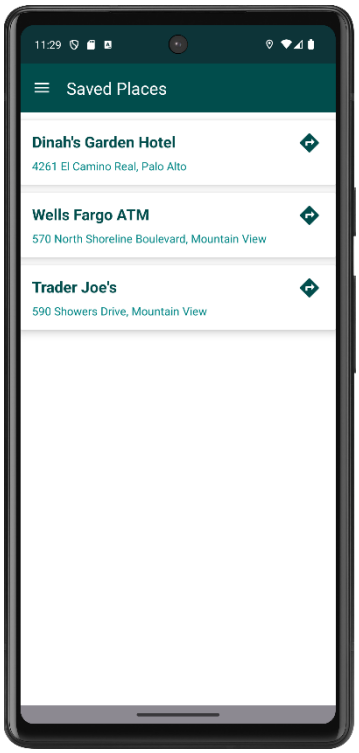


FIGURE 0-14 SAVED LOCATIONS PAGE

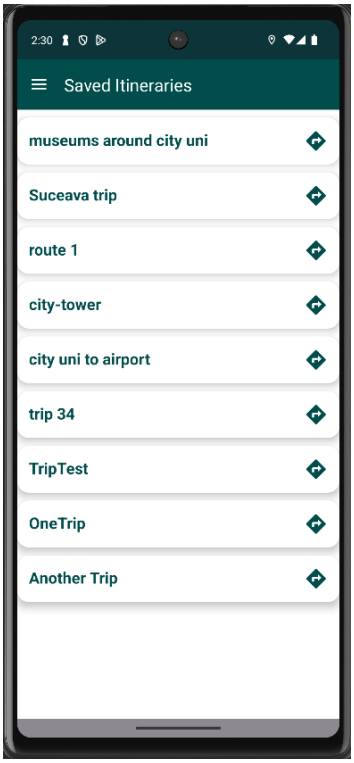


FIGURE 0-15 SAVED ITINERARIES

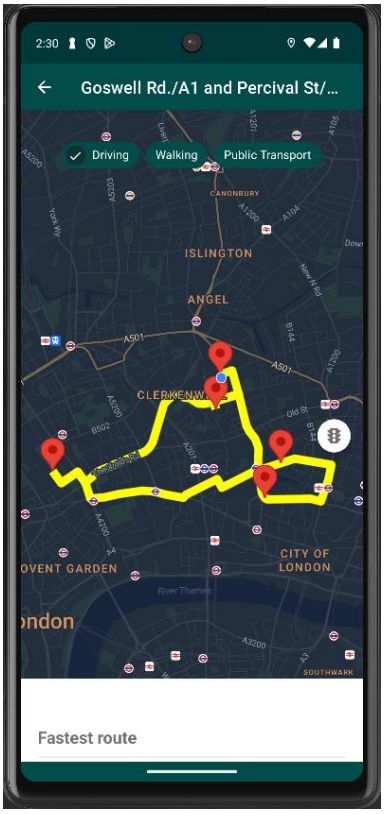
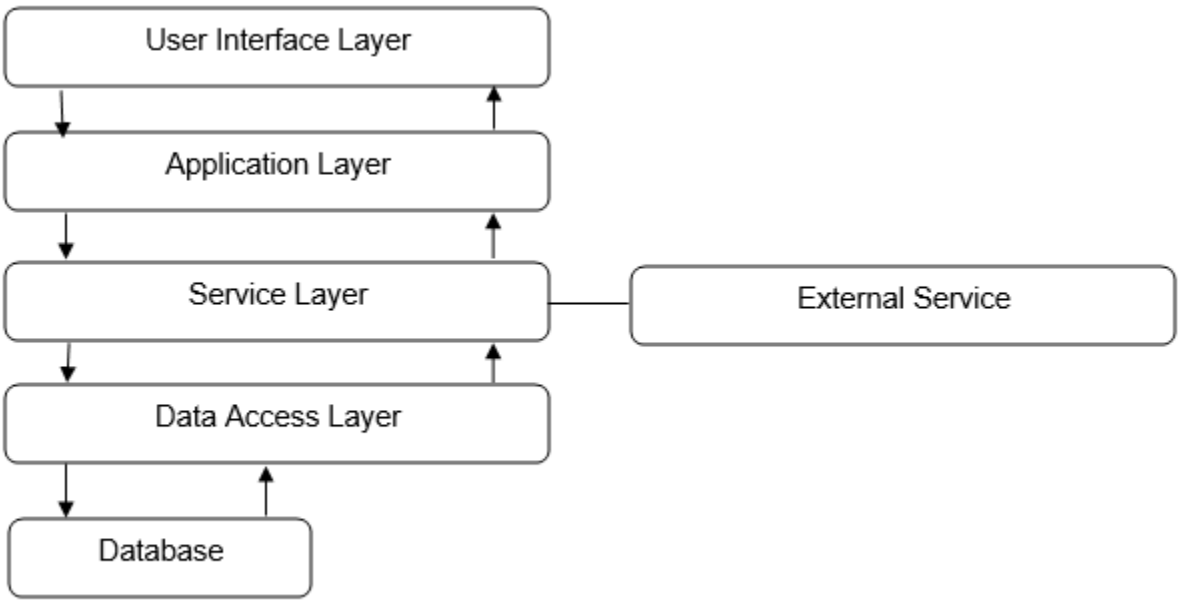
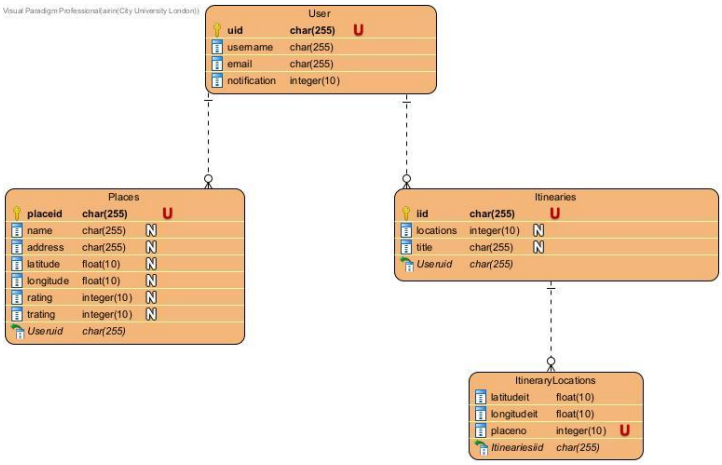


FIGURE 0-16 DIRECTIOSN FOR ITINERARY

3.System Architecture

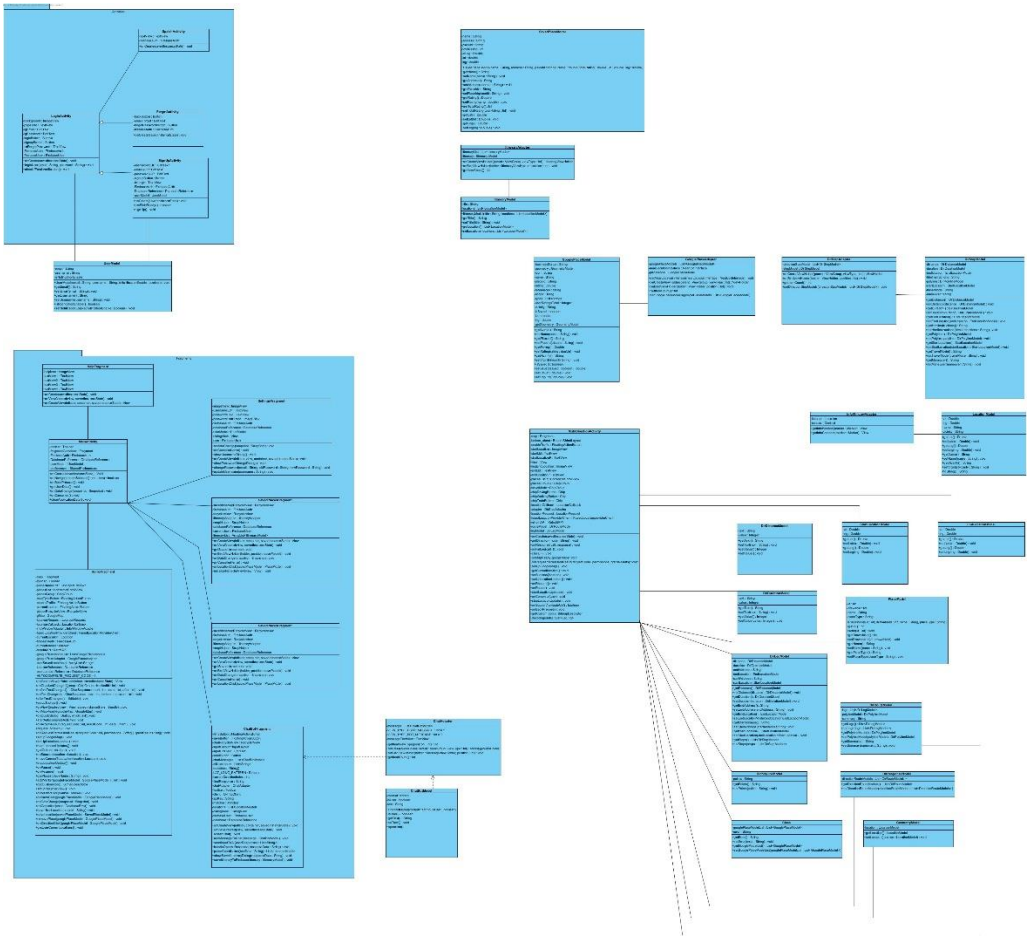


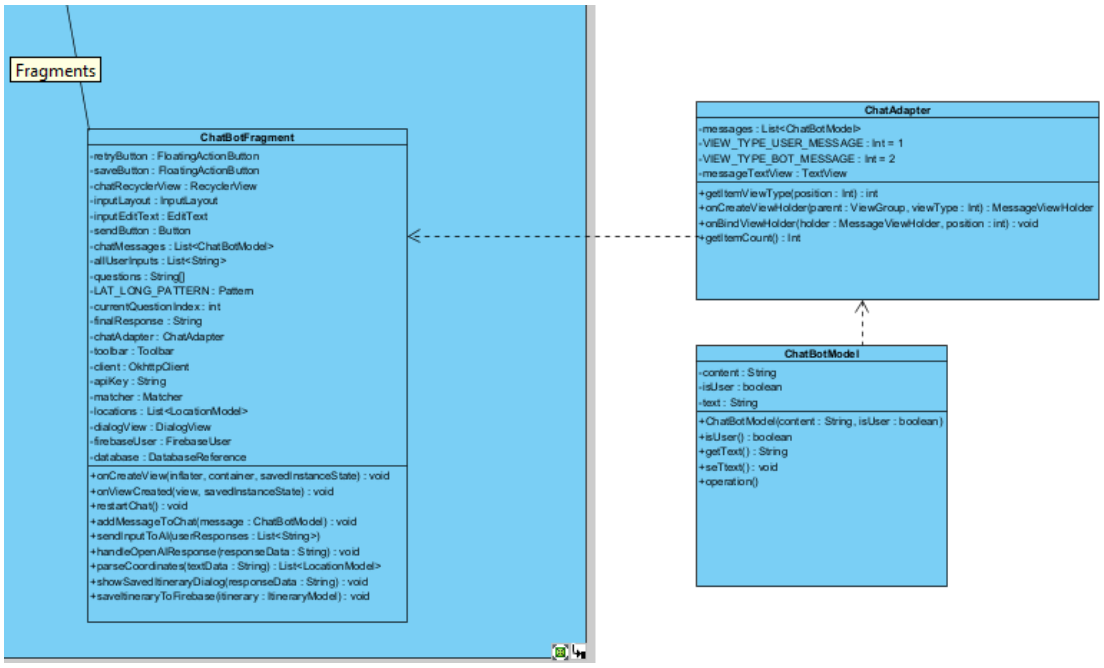
4.Entity-Relationship Diagram

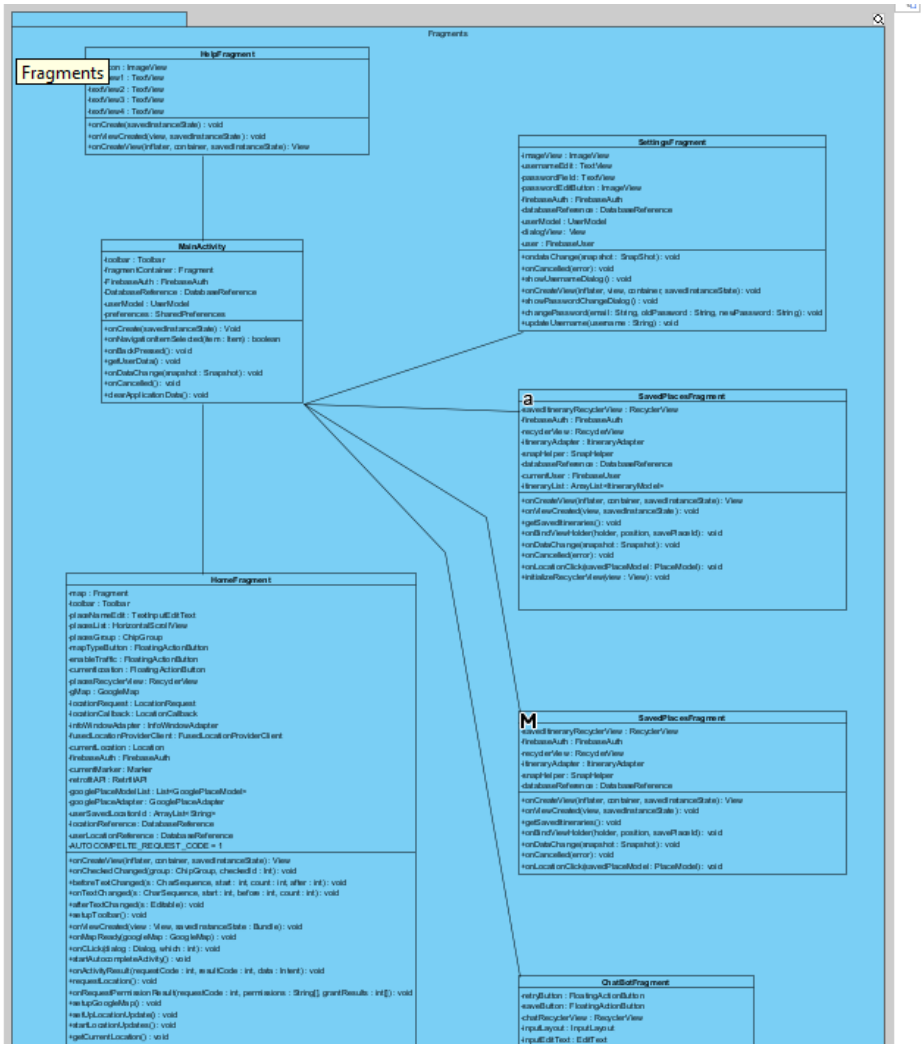


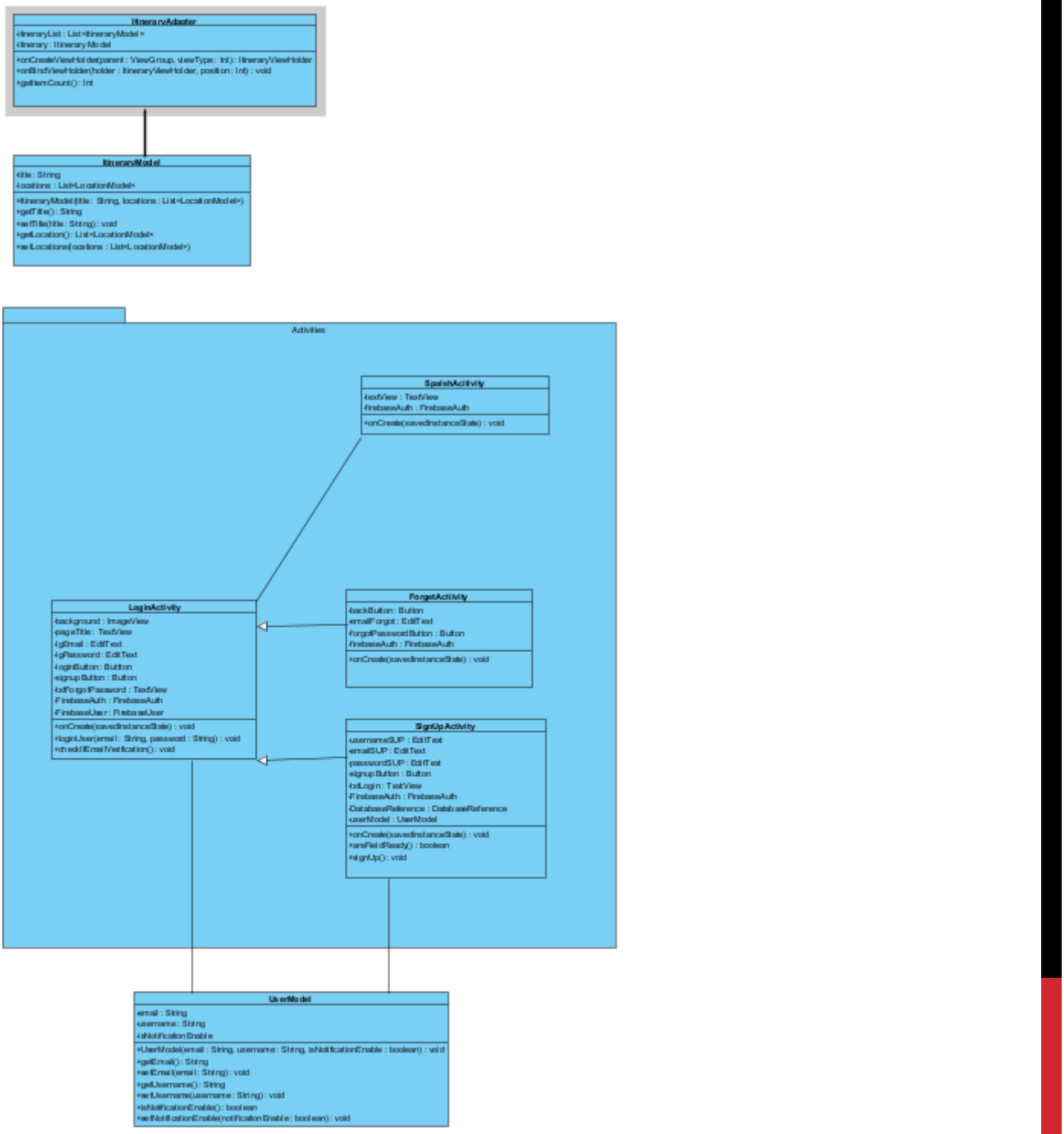
ER DIAGRAM V1.0.0 1

5.Class Diagram

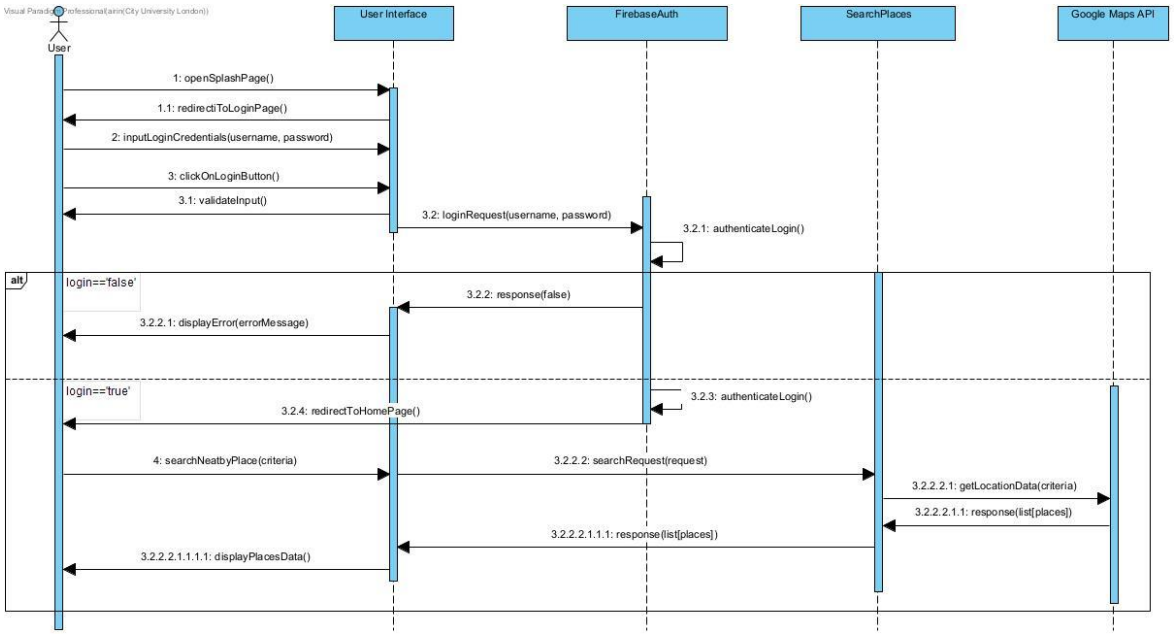



Fragments
ChatBotFragment
-retryButton : FloatingActionButton
-saveButton : FloatingActionButton
-chatRecyclerView : RecyclerView
-inputLayout : InputLayout
-inputEditText : EditText
-sendButton : Button
-chatMessages : List<ChatBotModel>
-allUserInputs : List<String>
-questions : String[]
-LAT_LONG_PATTERN : Pattern
-currentQuestionIndex : int
-finalResponse : String
-chatAdapter : ChatAdapter
-toolbar : Toolbar
-client : OkhttpClient
-apiKey : String
-matcher : Matcher
-locations : List<LocationModel>
-dialogView : DialogView
-firebaseUser : FirebaseUser
-database : DatabaseReference
+onCreateView(inflater, container, savedInstanceState) : void
+onViewCreated(view, savedInstanceState) : void
+restartChat() : void
+addMessageToChat(message : ChatBotModel) : void
+sendInputToAI(userResponses : List<String>)
+handleOpenAIResponse(responseData : String) : void
+parseCoordinates(textData : String) : List<LocationModel>
+showSavedItineraryDialog(responseData : String) : void
+saveItineraryToFirebase(itinerary : ItineraryModel) : void
ChatAdapter
-messages : List<ChatBotModel>
-VIEW_TYPE_USER_MESSAGE : Int = 1
-VIEW_TYPE_BOT_MESSAGE : Int = 2
-messageTextView : TextView
+getItemViewType(position : Int) : int
+onCreateViewHolder(parent : ViewGroup, viewType : Int) : MessageViewHolder
+onBindViewHolder(holder : MessageViewHolder, position : int) : void
+getItemCount() : Int
ChatBotModel
-content : String
-isUser : boolean
-text : String
+ChatBotModel(content : String, isUser : boolean)
+isUser() : boolean
+getText() : String
+seTtext() : void
+operation()


Fragments
Fragments
HelpFragment
MainActivity
SettingsFragment
SavedPlacesFragment
SavedPlacesFragment
HomeFragment
ChatBotFragment


ItineraryAdapter
-itineraryList : List<ItineraryModel>
-itinerary : ItineraryModel
+onCreateViewHolder(parent : ViewGroup, viewType : Int) : ItineraryViewHolder
+onBindViewHolder(holder : ItineraryViewHolder, position : Int) : void
+getItemCount() : Int
ItineraryModel
-title : String
-locations : List<LocationModel>
+ItineraryModel(title : String, locations : List<LocationModel>)
+getTitle() : String
+setTitle(title : String) : void
+getLocation() : List<LocationModel>
+setLocations(locations : List<LocationModel>)
Activities
SpalshActivity
-textView : TextView
-firebaseAuth : FirebaseAuth
+onCreate(savedInstanceState) : void
LoginActivity
-background : ImageView
-pageTitle : TextView
-lgEmail : EditText
-lgPassword : EditText
-loginButton : Button
-signupButton : Button
-txtForgotPassword : TextView
-FirebaseAuth : FirebaseAuth
-FirebaseUser : FirebaseUser
+onCreate(savedInstanceState) : void
+loginUser(email : String, password : String) : void
+checkEmailVerification() : void
ForgetActivity
-backButton : Button
-emailForgot : EditText
-forgotPasswordButton : Button
-firebaseAuth : FirebaseAuth
+onCreate(savedInstanceState) : void
SignUpActivity
-usernameSUP : EditText
-emailSUP : EditText
-passwordSUP : EditText
-signupButton : Button
-txtLogin : TextView
-FirebaseAuth : FirebaseAuth
-DatabaseReference : DatabaseReference
-userModel : UserModel
+onCreate(savedInstanceState) : void
+areFieldReady() : boolean
+signUp() : void
UserModel
-email : String
-username : String
-isNotificationEnable
+UserModel(email : String, username : String, isNotificationEnable : boolean) : void
+getEmail() : String
+setEmail(email : String) : void
+getUsername() : String
+setUsername(username : String) : void
+isNotificationEnable() : boolean
+setNotificationEnable(notificationEnable : boolean) : void

6.Sequence Diagram



SEQUENCE DIAGRAM V1.0.0

APPENDIX E

IMPLEMENTATION DOCUMENTATION

1.Introduction

This document was generated from the development activities that took place during the implementation phase of the RouteGuide application.

2. Purpose

This document outlines the implementation phase of the RouteGuide App development.

3. Source Code

In the Source Code are presented some code snippets of the author's original work. These are just bits of the actual code implemented by the author.

```
1 usage   Eliza9a
private void restartChat() {
    // Reset the conversation index
    currentQuestionIndex = 0;

    // Clear the collected user inputs
    allUserInputs.clear();

    // Clear the chat messages
    chatMessages.clear();
    chatAdapter.notifyDataSetChanged();

    // New instance of chat fragment
    Fragment newChatFragment = new ChatBotFragment();

    // Replace the current fragment with the new one
    if (getFragmentManager() != null) {
        getFragmentManager() FragmentManager
                .beginTransaction() FragmentTransaction
                .replace(R.id.fragmentContainer, newChatFragment)
                .commit();
    }
}
```

```
//Parsing the response from OpenAI
1 usage  Eliza9a
private void handleOpenAIResponse(final String responseData) {
    getActivity().runOnUiThread(() -> {
        try {
            JSONObject jsonObj = new JSONObject(responseData);
            JSONArray choices = jsonObj.getJSONArray( name: "choices");
            if (choices.length() > 0) {
                JSONObject choice = choices.getJSONObject( index: 0);
                if (choice.has( name: "message")) {
                    JSONObject message = choice.getJSONObject( name: "message"); // Parsing as JSONObject
                    String content = message.getString( name: "content"); // Accessing content directly
                    finalResponse = content;

                    // Removing Lat Longs
                    String cleanedText = content.replaceAll( regex: "\\(Lat:\\s*-?\\d+\\.\\d+,\\s*Long:\\s*-?\\d+\\.\\d+\\)", replacement: "");

                    Log.e( tag: "finalResponse", finalResponse);
                    addMessageToChat(new ChatBotModel(cleanedText, isUser: false));
                    ChatHistory.getInstance().addMessage(new ChatBotModel(content, isUser: false));
                }
            }
        } catch (JSONException e) {
            Log.e( tag: "JSON Parsing Error", msg: "Error parsing response from OpenAI", e);
        }
    });
}
```

```
1 usage  Eliza9a
public List<LocationModel> parseCoordinates(String textData) {
    List<LocationModel> locations = new ArrayList<>();
    Matcher matcher = LAT_LONG_PATTERN.matcher(textData);

    // Find all occurrences of latitude and longitude pairs in the text
    while (matcher.find()) {
        double lat = Double.parseDouble(Objects.requireNonNull(matcher.group(1)));
        double lng = Double.parseDouble(Objects.requireNonNull(matcher.group(2)));

        // Create a new LocationModel object and add it to the list
        LocationModel location = new LocationModel();
        location.setLat(lat);
        location.setLng(lng);
        locations.add(location);

        Log.e( tag: "location",  msg: location.getLat()+","+location.getLng());
    }

    return locations;
}
```

```java
//Dialog to save the itinerary to Firebase
1 usage  Eliza9a
private void showSaveItineraryDialog(String responseData) {
    View dialogView = LayoutInflater.from(getContext()).inflate(R.layout.itinerary_layout_dialog, root: null);
    EditText titleEdit = dialogView.findViewById(R.id.itineraryTitleEdit);

    new MaterialAlertDialogBuilder(getContext())
            .setView(dialogView)
            .setTitle("Save Itinerary")
            .setPositiveButton( text: "Save", (dialog, which) -> {
                String title = titleEdit.getText().toString().trim();
                if (!title.isEmpty()) {
                    List<LocationModel> locations = parseCoordinates(responseData);
                    ItineraryModel itinerary = new ItineraryModel(title, locations);
                    saveItineraryToFirebase(itinerary);
                    Toast.makeText(getContext(), text: "Itinerary saved: " + title, Toast.LENGTH_LONG).show();
                } else {
                    Toast.makeText(getContext(), text: "Itinerary title cannot be empty", Toast.LENGTH_SHORT).show();
                }
            })
            .setNegativeButton( text: "Cancel", listener: null)
            .show();
}
```

```java
//Saving the itinerary to Firebase
1 usage  Eliza9a
private void saveItineraryToFirebase(ItineraryModel itinerary) {
    FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();

    //Check if the user is logged in
    if (user != null) {
        FirebaseDatabase database = FirebaseDatabase.getInstance();
        DatabaseReference itinerariesRef = database.getReference( path: "Itineraries").child(user.getUid());
        String key = itinerariesRef.push().getKey();
        //Save the itinerary to the database
        if (key != null) {
            itinerariesRef.child(key).setValue(itinerary)
                    .addOnSuccessListener(aVoid -> Toast.makeText(getContext(), text: "Itinerary saved: " + itinerary.getTitle(), Toast.LENGTH_LONG).show())
                    .addOnFailureListener(e -> Toast.makeText(getContext(), text: "Error saving itinerary: " + e.getMessage(), Toast.LENGTH_LONG).show());
        } else {
            Toast.makeText(getContext(), text: "Error generating key for saving itinerary", Toast.LENGTH_LONG).show();
        }
    } else {
        Toast.makeText(getContext(), text: "User not logged in", Toast.LENGTH_LONG).show();
    }
}
```

```java
//Load the saved itineraries from the Firebase database
1 usage  Eliza9a
private void getSavedItineraries() {
    FirebaseUser currentUser = FirebaseAuth.getInstance().getCurrentUser(); //Get the current logged-in user
    if (currentUser != null) {
        String userId = currentUser.getUid();
        DatabaseReference reference = FirebaseDatabase.getInstance().getReference( path: "Itineraries").child(userId);

        Eliza9a
        reference.addListenerForSingleValueEvent(new ValueEventListener() {
            Eliza9a
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                itineraryList.clear();  //Clear the existing data
                for (DataSnapshot childSnapshot : snapshot.getChildren()) {
                    ItineraryModel itinerary = childSnapshot.getValue(ItineraryModel.class);
                    if (itinerary != null) {
                        itineraryList.add(itinerary); //Add itinerary to the list
                    }
                }
                itineraryAdapter.notifyDataSetChanged();  //Notify the adapter that the data has changed
            }
```

```java
17 usages  Eliza9a
public class ChatBotModel {
    2 usages
    private String content;
    2 usages
    private boolean isUser;
    2 usages
    private String text;

    6 usages  Eliza9a
    public ChatBotModel(String content, boolean isUser) {
        this.content = content;
        this.isUser = isUser;
    }

    // Getters
    Eliza9a
    public String getContent() { return content; }

    Eliza9a
    public boolean isUser() { return isUser; }

    Eliza9a
    public String getText() { return text; }

    Eliza9a
    public void setText(String text) { this.text = text; }
}
```

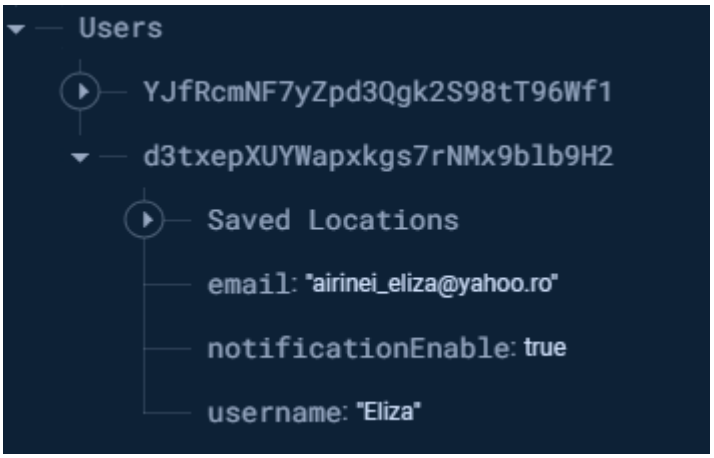
```
1 usage   Eliza9a
public ChatAdapter(List<ChatBotModel> messages) { this.messages = messages; }

//Determine the type of view to be displayed (user or bot)
 Eliza9a
@Override
public int getItemViewType(int position) {
    ChatBotModel message = messages.get(position);
    return message.isUser() ? VIEW_TYPE_USER_MESSAGE : VIEW_TYPE_BOT_MESSAGE;
}

//Create a new view holder based on its type
 Eliza9a
@NonNull
@Override
public MessageViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View view;
    if (viewType == VIEW_TYPE_USER_MESSAGE) {
        view = LayoutInflater.from(parent.getContext()).inflate(R.layout.user_message_item, parent,  attachToRoot: false);
    } else {
        view = LayoutInflater.from(parent.getContext()).inflate(R.layout.bot_message_item, parent,  attachToRoot: false);
    }
    return new MessageViewHolder(view);
}
```

4.Firebase Database:

Identifier	Providers	Created ↓	Signed in	User UID
evenimente....	✉	20 Apr ...	21 Apr ...	YJfRcmNF7yZp...
airinei_eliza...	✉	2 Apr 2...	30 Apr ...	d3txepXUYWapx...



5.Version Control: