00+

National College of Ireland

BSc Computing

Software Development

2022/2023

Adam Regan

x19401956

x19401956@student.ncirl.ie

Travel Planner

Technical Report

Contents

Executive Summary

This report details the project plan and implementation details of a Travel Planner app. The project is an Android application. The aim is to allow the user to plan a travel itinerary by adding places to an itinerary which they can view on a map. The system can help recommend places to the user based on their search behaviour. This report details the requirements specification including all functional and non-functional requirements. It includes implementation and testing details. The aim of this document is to showcase the project and demonstrate its features.

1.0 Introduction

1.1. Background

This project identifies a problem I had when travelling. I enjoy researching places to visit prior to a trip or when on a trip. The method I used was to search for places using Google Maps and note the details of a place in a notes app. The idea I came up with was to integrate the two features together to make the process easier and more efficient.

Upon researching similar applications in the market, I discovered some solutions to the problem which my idea is based. The main app I found is called TripleT. This allows the user to search for places using the Google Maps API and they can store the place on a map and manage the details of the itinerary. The main flaw that I want to address in my application is the ability to get place details without having to manually input them into the system. I also want the app to generate recommendations for the user based on other users searches.

The idea for my app solves some of the specific issues I have with other implementations. It also fits my interest for map-based applications and Android development. I also want to take this opportunity to learn Kotlin which is the primary language for Android development and to use a NoSQL database.

1.2. Aims

This project aims to create an Android application which allow the users to plan a trip itinerary. The user should be able to search for places they want to visit and see them represented on a map. The application can then provide recommendations for place searches based on previous searches and reviews from other users.

The app should streamline the process for planning trip itineraries. The process should be simple and effective to motivate the user to continue using it. By providing recommendations for the user the system can help increase the number of places added to the trip and make it easier for the user to find places they may like based on the interests they present.

1.3. Technology

The application is an Android application and will be developed in Android Studio. The primary implementation language is Kotlin which has become the DeFacto programming language for android development. It is interoperable with the JVM and shares similar features. Kotlin is recommended for Android for apps where performance matters especially on older devices. It is also more easily compiled for multiple platforms which could matter if the app needs to be able to run on IOS at a later stage [1].

The database for the application will be Cloud Firestore. Firestore is a NoSQL database that operates in the cloud. It is a document database which uses JSON like syntax. Firebase operates well with Android and has built in features with Android Studio making it easy and quick to deploy [2]. I choose NoSQL because I wanted to take this opportunity to learn about NoSQL databases. NoSQL is also useful for this project because it is more scalable than traditional SQL and is fast for cloud storage. It is useful for data structures which may need to be changed at a later point. This is much easier in NoSQL than SQL. It is also optimised more for queries than data duplication which can make it faster and provides more options for structuring the data [3].

1.4. Structure

This document details the requirements specification for both functional and non-functional requirements. It shows implementation and testing details. At the end it includes the project proposal and monthly reflections.

2.0 System

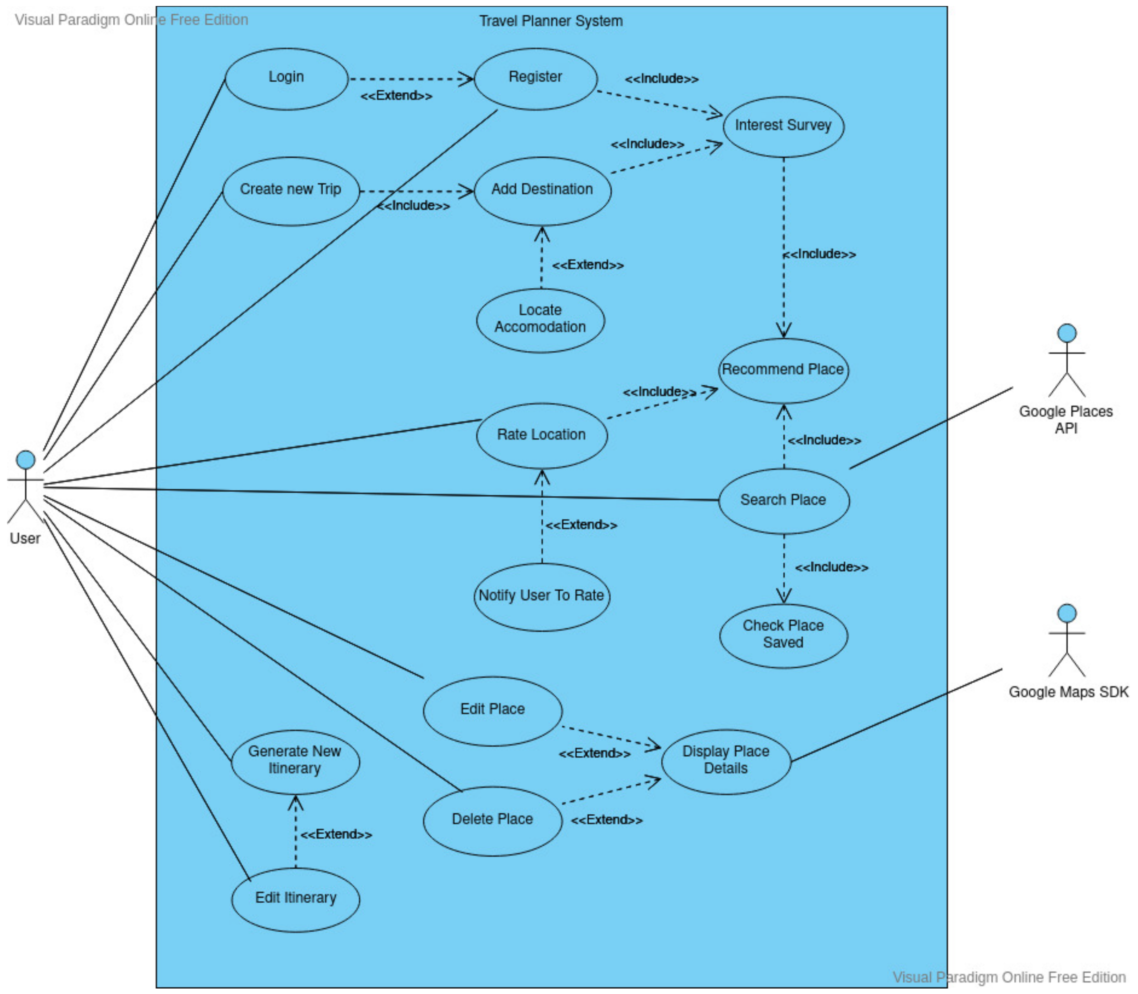
2.1. Requirements

2.1.1. Functional Requirements

1. User must be able to search for a place by name
2. Places must be displayed on a map screen using location markers
3. User can generate an itinerary based on list of places
4. User must be able to edit generated itinerary
5. User can view place details
6. User can edit place details

7. User can delete place
8. User can create new trip
9. User must be able to add a travel destination
10. User must be able to add dates of travel
11. User can edit trip details
12. User can provide accommodation location to improve local recommendations
13. User can register account
14. User can login using account details
15. Registering requires filling out an interest survey to provide recommendations
16. User can rate visited places
17. Place ratings enhance applications recommendation algorithm
18. Searching for a place will generate a recommendation for the user
19. Application notifies user to rate visited places

2.1.1.1. Use Case Diagram



2.1.1.2. Requirement 1: Search Place by Name

2.1.1.2.1. Description & Priority

The user can input a place name into a search bar. The Google Places API will return place information back to the application which can be displayed by the system. This is a key functionality of the system and has a high priority.

2.1.1.2.2. Use Case

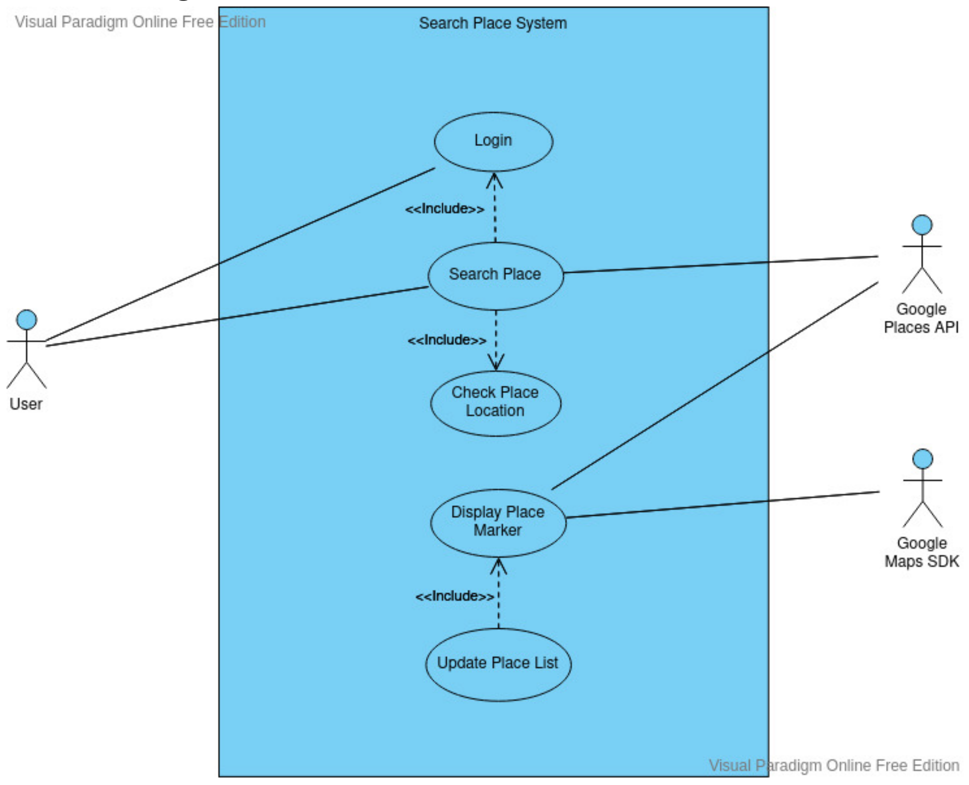
Scope

Allows the user to input a place name into a search bar. The place details are returned by the Google Places API. The coordinates return by the API are displayed on a map using a map marker. Other details provided by the API are not used for this use case.

Description

The user can input a place name into the search bar in the application. The Google Places API will return the place information including the coordinates and other relevant details which can be used in other use cases. This place is then displayed on a map using the coordinates.

Use Case Diagram



Flow Description

Precondition

The user has a registered account and has logged in using their details. The application search bar is available.

Activation

Inputs a place name into the search bar and clicks search.

Main flow

1. The user inputs a place name into the search bar
2. The user clicks search
3. The system sends a GET request to the Google Places API
4. The API returns a result with place details
5. The system checks that the coordinates of the place are within the boundary of the trip created
6. The user confirms they want to add the place to their list of places
7. The system updates the places list in the database under the current user
8. The system displays a map marker using the place coordinates

Alternate flow

A1 : No results from place name

4. The API does not return a result
5. The user finds and inputs coordinates of place
6. The API returns a result
7. The user confirms they want to add the place to their list of places
8. The system updates the places list in the database under the current user
9. The system displays a map marker using the place coordinates

A2 : Place not within trip boundary

5. The system checks that the coordinates of the place are within the trip boundary
6. The system finds that the place is not within the trip location boundary
7. The user is informed that the location cannot be added to the trip
8. The User is prompted to search for a new location

Exceptional flow

E1 : No Search Result

4. The API does not return a result from the search
5. The system displays an error message showing no result
6. The system asks the user to change search parameters

Termination

The system returns to the map screen with pinned locations

Post condition

The system waits for user input

2.1.1.3. Requirement 2: Display Places on Map

2.1.1.3.1. Description & Priority

When the application is open and the user logs in, the system retrieves place locations from the coordinates stored in the database under the current user. It displays the coordinates with map markers. This is high priority feature. It does not affect other functionality too much but is a primary feature defined in the project proposal.

2.1.1.3.2. Use Case

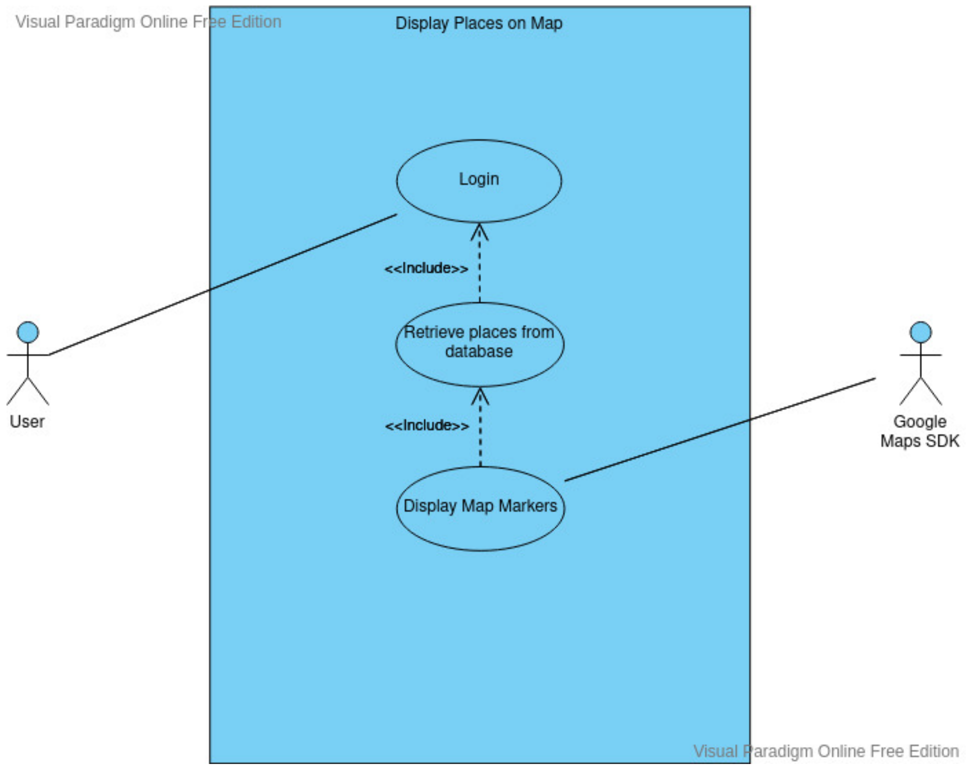
Scope

The user can see a visual representation of their list of places on a map. The coordinates of the places added by the user is retrieved from the database and represents them as map markers. This use case does not include interactive functionality for the map markers. It does not include updating map markers when a new place is added. It only includes the loading of markers when the application loads.

Description

The user can view their list of places for a trip on a map as map markers.

Use Case Diagram


Visual Paradigm Online Free Edition
Display Places on Map
Login
<<Include>>
Retrieve places from database
<<Include>>
Display Map Markers
User
Google Maps SDK
Visual Paradigm Online Free Edition

Flow Description

Precondition

- The user opens the application.

- The user has registers and account.
- The user has created a trip.
- The user has added places to the trip.

Activation

The user opens the application and logs in with their account details

Main flow

1. The user logs in using their account details
2. The system retrieves the user account details
3. The system retrieves the users' created trips
4. The system retrieves place locations from the database
5. The system generates map markers using the Google Maps SDK
6. The user can view the map markers on a map in the application

Alternate flow

A1 : No places to display

1. The user logs in using their account details
2. The system retrieves the user account details
3. The system retrieves the users' created trips
4. The system cannot find any saved places
5. The system notifies the user that there are no saved places
6. The user adds a place to their trip
7. The system is updated to display places

A2 : No trip created

1. The user logs in using their account details
2. The system retrieves the user account details
3. The system cannot find any created trips
4. The system prompts the user to create a trip
5. The user creates a new trip
6. The user adds places to their trip
7. The system is updates to display places

Exceptional flow

E1 : System can’t retrieve place details

8. The system cannot retrieve place details from the database
9. The user is given an error message

Termination

The system shows the users listed places

Post condition

The system waits for user input

2.1.1.4. Requirement 3: Edit Itinerary

2.1.1.4.1. Description & Priority

When an itinerary is generated, the user can edit the order and composition of the itinerary either before they accept it or after. This is a medium priority.

2.1.1.4.2. Use Case

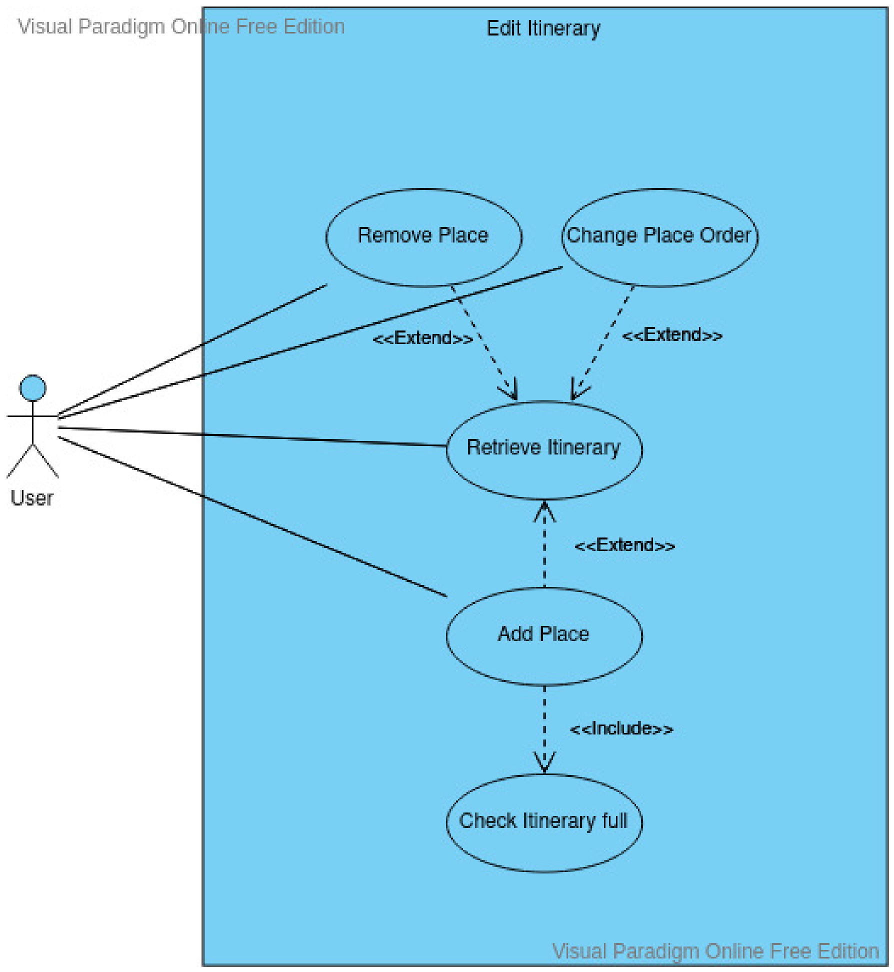
Scope

The user must be able to edit the generated Itinerary. This use case does not handle if other factors affect the itinerary such as the trip length being edited. That is dealt with by another use case

Description

The user can edit the generated itinerary. This includes changing the places in the itinerary and changing the order.

Use Case Diagram



Flow Description

Precondition

- The user is logged in
- The user has generated an Itinerary

Activation

The user requests to edit the itinerary

Main flow

1. The user request to view the itinerary
2. The user requests to edit the itinerary
3. The user makes desired changes
4. The system checks that the changes don't extend the timeframe
5. The user accepts the changes
6. The system updates the Itinerary in the database

Alternate flow

A1 : User does not accept changes

5. The user does not accept changes made
6. The system does retrieves original itinerary from database
7. The user is taken back to view itinerary screen

A2 : Changes exceed timeframe

5. The system finds the changes exceed the timeframe constraints from the trip
6. The user is notified to make changes to fit the tie constraints
7. The user makes the changes
8. The user accepts the changes
9. The system updates the itinerary database

Exceptional flow

E1 : Cannot save new itinerary

5. The user accepts changes
6. The system attempts to update database
7. The system received an error and cannot update database
8. The user receives an error message
9. The system restores the original itinerary
10. The user is prompted to try again

Termination

The user accepts the new itinerary

Post condition

The system displays the map screen and waits for user action

2.1.1.5. Requirement 4: View Place Details

2.1.1.5.1. Description & Priority

The user can view place details after they add a place to their list. They can view the details by clicking on a map marker or by viewing their list in a separate screen and searching for it.

2.1.1.5.2. Use Case

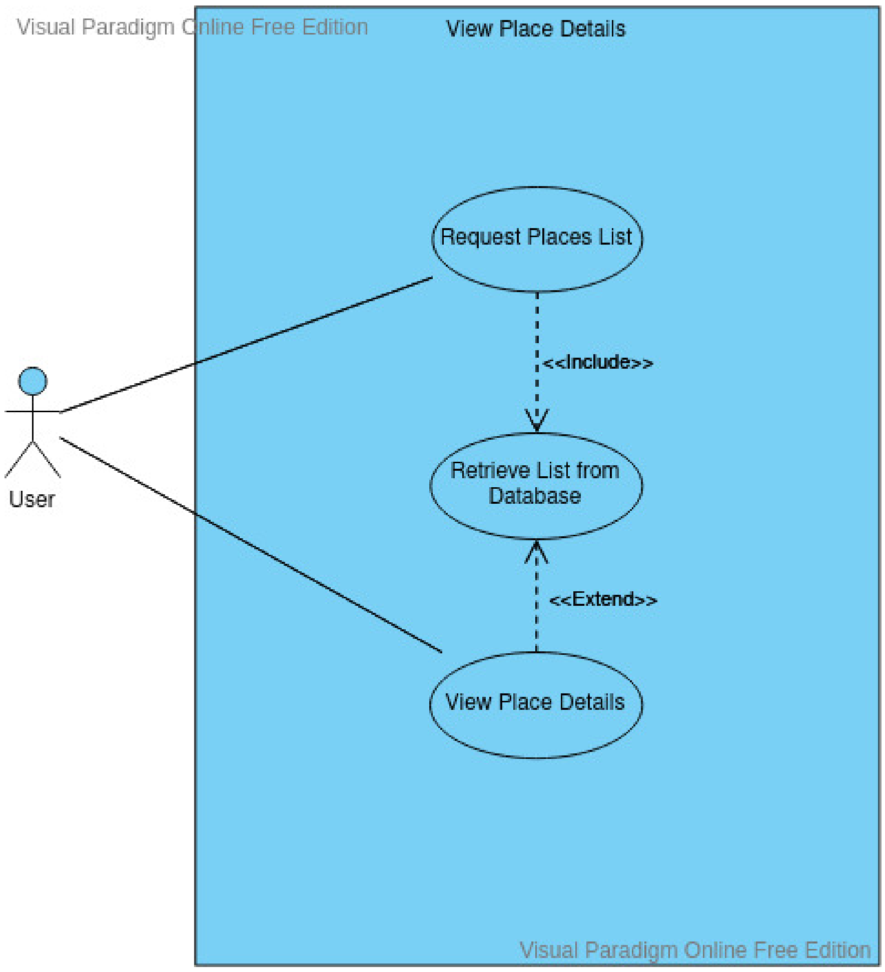
Scope

This use case defines how the user can retrieve their list of places and view the details of place. The user can view place details by two methods. They can click on a map marker, or they can select from a list screen.

Description

The user can retrieve place details from the database and view details of the place provided by the Goole Places API.

Use Case Diagram


Visual Paradigm Online Free Edition
View Place Details
Request Places List
<<Include>>
Retrieve List from Database
<<Extend>>
View Place Details
User
Visual Paradigm Online Free Edition

Flow Description

Precondition

- The user has created a trip
- The user has selected to view that trip
- The user has added at least one place added to their list

Activation

The application has loaded

Main flow

1. The user selects the button to view places
2. The system loads the places list screen
3. The user requests to view the place details for a place from the list
4. The system loads the place details from the database
5. The user can view the place details

Alternate flow

A1 : Use map markers to view

1. The user selects a map marker from the map screen.
2. The system loads the place details from the database
3. The system displays the details on a screen.

Exceptional flow

E1 : Cannot find place details in database

4. System cannot retrieve place details
5. User requests to retrieve details from API
6. System Displays details

Termination

The place details screen has loaded

Post condition

The user exits the screen and the system waits for further input

2.1.1.6. Requirement 5: Edit Place Details

2.1.1.6.1. Description & Priority

The user can edit certain details if they find the details retrieved from the API are incorrect. This is not a critical function for the system but may be necessary to correct small errors such as the place name being incorrect.

2.1.1.6.2. Use Case

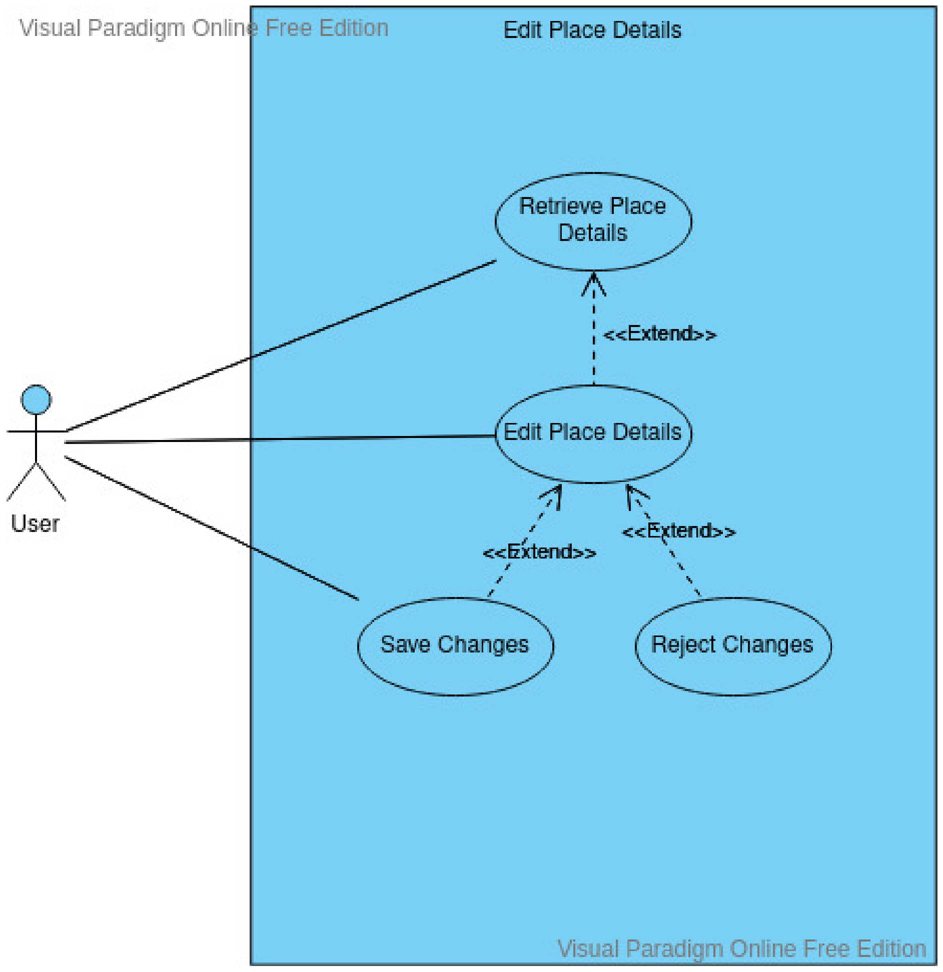
Scope

This use case describes how the user can edit place details provided by the API if they are incorrect. This feature is only limited to certain details such as the title, coordinates and address.

Description

A user should be able to edit certain place details if the user determines them to be incorrect. This will help the algorithm be more accurate in the case that the API returns wrong information. It should be used in some instances but is important to make sure the app functions properly.

Use Case Diagram



Flow Description

Precondition

- The user has created a trip
- The user has added a place to their trip

Activation

View a place and selects the edit button

Main flow

1. The user selects a place
2. The system retrieves the place details
3. The user requests to edit the place details
4. The user makes changes
5. The user chooses to save the changes

Alternate flow

A1 : User rejects changes

4. The user makes changes
5. The user chooses to reject changes

Exceptional flow

E1 : Changes can't save

4. The user makes changes
5. The system cannot save changes
6. The user receives and error saying the system cannot save changes
7. The system goes back to the view place details screen

Termination

The user has saved rejected the changes made

Post condition

The system returns to the view places screen and waits for user input

2.1.1.7. Requirement 6: Delete Place

2.1.1.7.1. Description & Priority

The user should be able to delete a place from their list. This is a medium priority but is important to allow the user to have control over the system.

2.1.1.7.2. Use Case

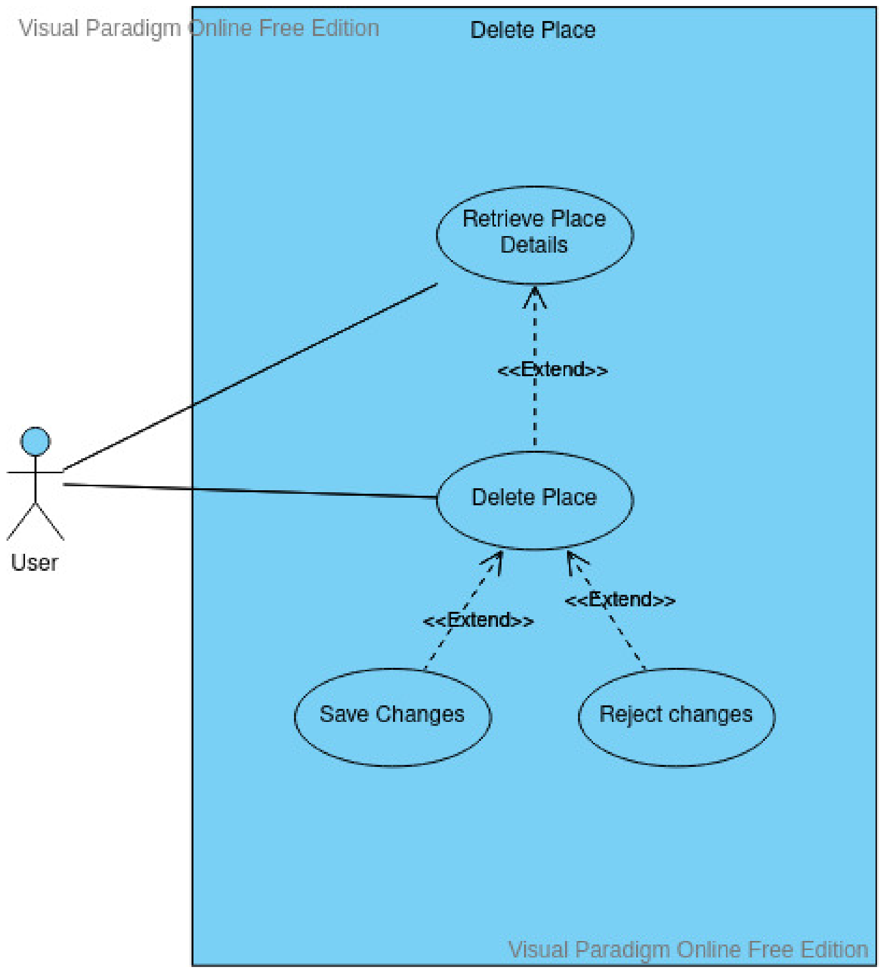
Scope

The user should be able to delete a place from their list.

Description

The user can delete a place from their list.

Use Case Diagram



Flow Description

Precondition

- The user has created a trip
- The user has added a place to their list

Activation

The user views a place and requests to delete

Main flow

1. The user selects a place to view
2. The user requests to delete place
3. The system asks the user to confirm delete action
4. The user confirms deletion
5. The system removes the place entry from the database
6. The system updates the current session to show that place has been removed
7. The system notifies that the place has been removed

Alternate flow

A1 : User rejects deletion action

3. The system asks the user to confirm deletion action
4. The user rejects deletion
5. The system reverts to the view place details screen

Exceptional flow

E1 : Deletion error

6. The user confirms deletion
7. The system tries to remove entry from database
8. Database responds with an error
9. The system shows an error to the user
10. The system reverts to the view place details screen

Termination

The user confirms or rejects deletion

Post condition

The system reverts to view place details screen and waits for user input

2.1.1.8. Requirement 7: Create Trip

2.1.1.8.1. Description & Priority

The user can create a trip. The trip will allow the user to specify a destination and the dates of stay. The user can then add places to their trip by searching for a place using the Google Places API. This is a high priority use case. The ability to add places to a list is dependent on this feature.

2.1.1.8.2. Use Case

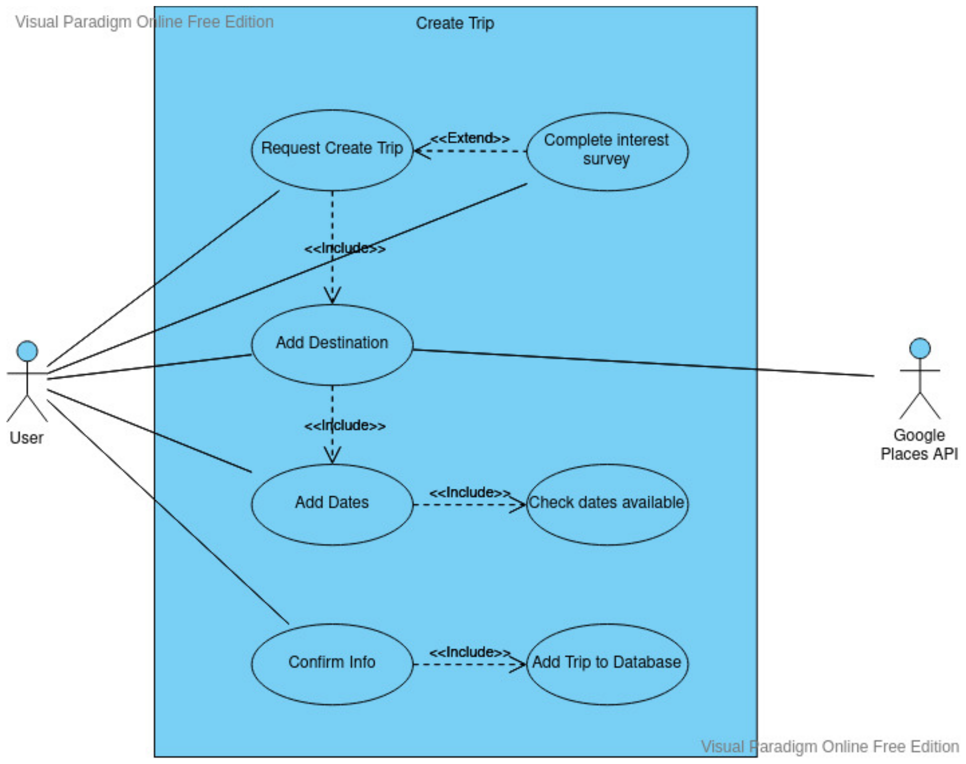
Scope

The user can create a trip and specify the destination and dates. The user can then add places to their trip. This use case does not deal with adding places but only the creation of a trip by adding a destination and dates. It also requires the user to specify interests which the recommendation algorithm can use to recommend places.

Description

This use case defines how the user can create a trip by specifying a destination and travel dates. The user can create multiple trips. The dates on the trips cannot overlap.

Use Case Diagram



Flow Description

Precondition

- The user has registered an account
- The user has logged into their account

Activation

The user has requested to create a trip

Main flow

1. The user requests to create a trip
2. The user searches for a destination using the Google Places API
3. The user inputs travel dates
4. The system checks that the dates are not used by another trip
5. The user fills out interest survey
6. The system adds the trip to the database
7. The user is prompted to add places to their trip

Alternate flow

A1 : Travel dates are already in use

4. The system checks that the dates are not used by another trip
5. The system finds dates are in use
6. The system asks the user to change the dates of the trip
7. The user changes the dates of the trip

A2 : Cannot find destination

8. The user searches for a destination using the Google Places API
9. The API cannot find place
10. The system requests the user to input a new search parameter

Exceptional flow

E1 : API does not return destination after multiple attempts

11. The user searches for a destination using the Google Maps API
12. The system returns a not found error to the user
13. The system prompts the user to input the coordinates of the destination
14. The API returns the location name from the coordinates

Termination

The trip is created and added to database

Post condition

The system displays a page to add places to trip and waits for user input

2.1.1.9. Requirement 8: Edit Trip Details

2.1.1.9.1. Description & Priority

The user can make changes to their trip details. The user can change the destination and the dates of the trip. This is a medium priority. It provides the application with more functionality.

2.1.1.9.2. Use Case

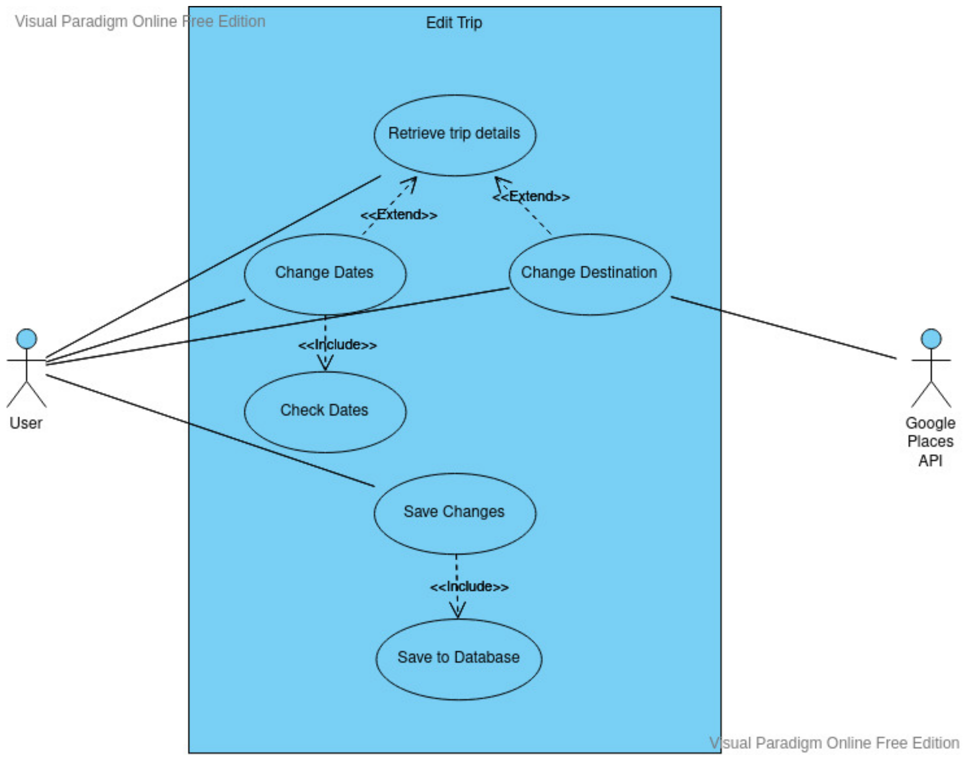
Scope

This use case describes the way a user can edit trip details. The details they can edit are destination and the dates. It does not include editing the place details of the trip.

Description

The user can edit trip details including the destination and the dates of their trip

Use Case Diagram



Flow Description

Precondition

- The user has created a trip

Activation

The user requests to edit trip

Main flow

1. The user requests to edit their trip
2. The user makes changes
3. They system verifies the dates of the trip
4. The user accepts the changes
5. The system updates the database

Alternate flow

A1 : The user rejects the changes

2. The user makes changes
3. The user rejects changes
4. The system reverts to view trip screen

A2 : Date changes match another trip

2. The user changes the date of the trip
3. The system verifies that the dates don't match another trip
4. The system finds a match with the dates
5. The system informs the user that the dates cannot be used

6. The user changes the dates or rejects the changes

Exceptional flow

E1 : Changes cannot be saved

7. The user accepts the changes
8. The system attempts to save the changes to the database
9. The system notifies the user that the changes cannot be saved
10. The system reverts back to the view trip details screen

Termination

The user accepts or rejects changes

Post condition

The system goes back to the view trip screen and waits for user input

2.1.1.10. Requirement 9: Register Account

2.1.1.10.1. Description & Priority

The user can register and account using an email and a password. This account is required to track the users' trips and places in the database. It is a high priority requirement. It is required to allow the user to have a personal set of trips and places stored.

2.1.1.10.2. Use Case

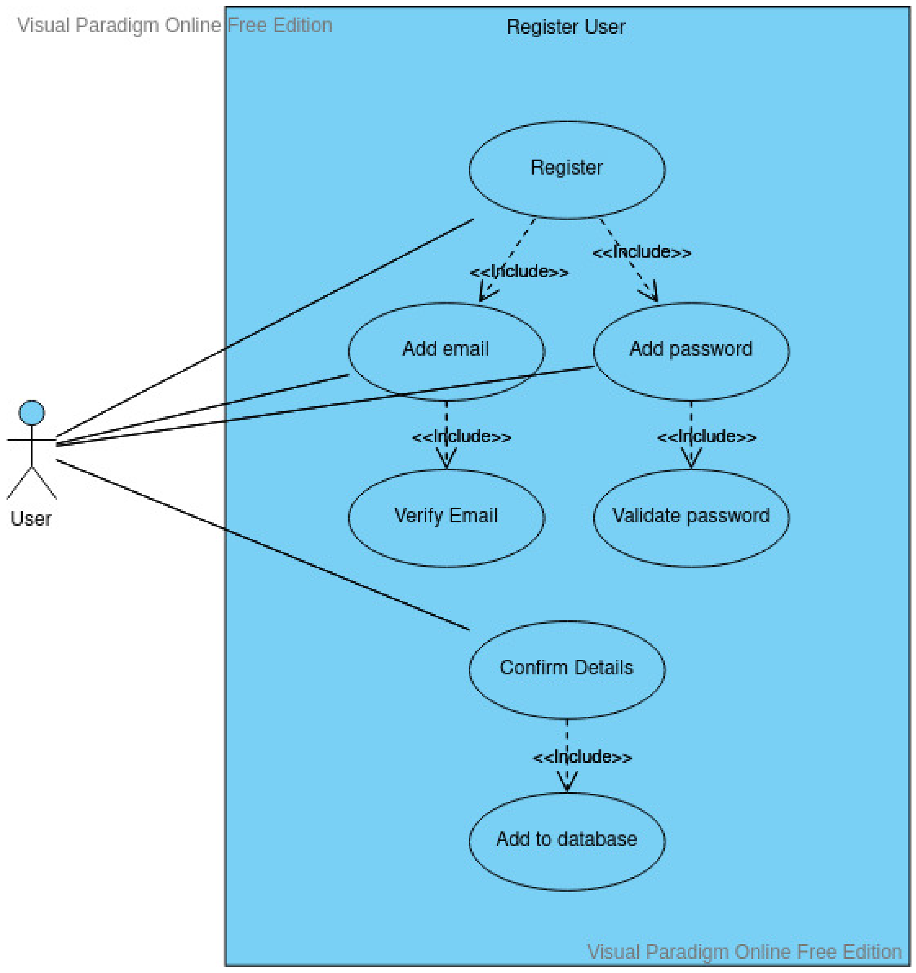
Scope

The user can create and account using their email and a password. This use case only deals with the registration process and not the login function of the app.

Description

The user can register and account to start creating trips and adding places to their trips and using all the other functionality of the app.

Use Case Diagram



Flow Description

Precondition

- The user has an email address
- The user has installed the app

Activation

The user requests to register and account

Main flow

1. The user requests to create an account
2. The user inputs their email and desired password
3. The system sends a verification email.
4. The user verifies their email
5. The system validates the password for desired constraints
6. The user confirms they want to create account
7. The system adds the user to the database

Alternate flow

A1 : No email verification

3. The system sends a verification email
4. The system waits 10 minutes for a response
5. The user does not verify email
6. The system asks the user to register again

A2 : Password does not meet validation constraints

5. The system validates the password for desired constraints
6. The system asks the user to input a password that meets the constraints
7. The user inputs a new password.
8. The system accepts the password
9. The system adds the user to the database

Exceptional flow

E1 : Account cannot save to database

10. The user confirms they want to create the account
11. The system attempts to add the user to the database
12. The database returns an error
13. The user is informed that the account creation failed
14. The system records the error for a bug fix
15. The user is asked to try again

Termination

The user account is created

Post condition

The system waits for the user to login

2.1.1.11. Requirement 10: Login

2.1.1.11.1. Description & Priority

A user who has created an account should be able to login with the details they used to create their account. The details include their email address and their password. When a user is logged in, they can interact with the system functions.

2.1.1.11.2. Use Case

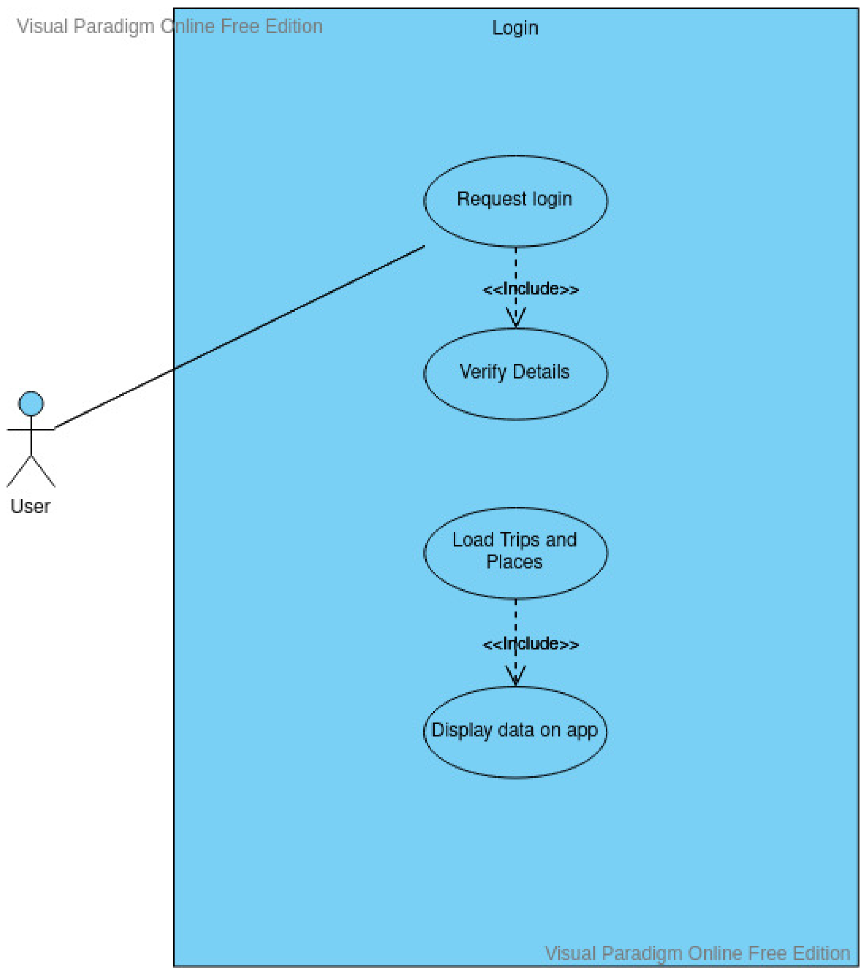
Scope

This use case specifies how a user can login using their account details created when registering. It only includes the process of inputting user details to gain access to the user account.

Description

This use case describes the way in which a user can login to their account using their account details. It specifies the way that the information is validated and how the user can use the login system.

Use Case Diagram



Flow Description

Precondition

The user has created an account

Activation

The user requests to login

Main flow

1. The user requests to login
2. The user input their account email and password
3. The system verifies the details with the record on the database
4. The system loads the trips and places associated with the user

Alternate flow

A1 : Wrong email or password

1. The user inputs their email and password
2. The system verifies the input
3. The user input doesn't match the database record
4. The user is asked to input their email and password again

5. The system verifies the user
6. The system loads the trips and places associated with the user

A2 : Too many login attempts

7. The user is allowed 5 attempts at login
8. The user requests an email verification from the system
9. The system sends a password reset to the user under the email on the account
10. The user clicks the link under the email
11. The user inputs a new password
12. The user attempts to login again
13. The system loads the user account

Exceptional flow

E1 : User account not found

14. The user exceeds 5 login attempts
15. The user inputs email to reset password
16. The email is not found
17. The user is required to send an error report

Termination

The user logs in to their account and can access their trips and places

Post condition

The system waits for user input

2.1.1.12. Requirement 11: Generate Recommendation

2.1.1.12.1. Description & Priority

The user should receive personalised recommendations for places based on their previous searches and the search patterns of other users. The recommendations will appear when a user is searching for a place. It will also consider the interest surveys which users can fill in when they register and when they create a new trip. This feature is designed to enhance the experience of the user and add complexity to the application. This is a medium priority use case. It is a feature which enhances the user experience but is not integral to the core functionality of the app.

2.1.1.12.2. Use Case

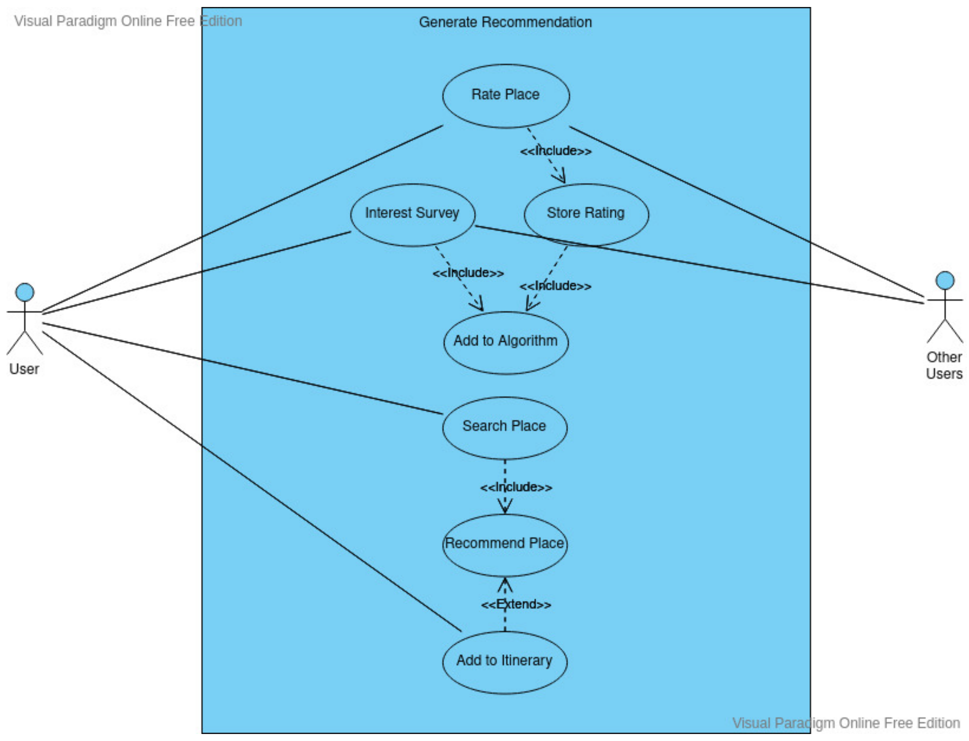
Scope

The user can receive recommendations from the system about places they could add to their itinerary. It is meant to help the user build an itinerary faster than by supplementing the process with recommendations they may not know about. This system is related to the ratings use case which is used to provide the algorithm with parameters. This use case describes how the user will receive recommendation when using the app.

Description

When the user creates a new trip, the user can add places which will feed into the algorithm which recommends more places to the user. This recommendation is based on an interest survey the user fills out when registering and creating a new trip. It is also based on the search patterns of other users and the types of places the user is searching for. A user which provides ratings for places after they visit will enhance personalised recommendations for places.

Use Case Diagram



Flow Description

Precondition

- The user has created a trip
- The user has filled in their interest survey
- The user has added a trip
- The user has added places to their trip

Activation

The user searches for a place

Main flow

1. The user searches for a place

2. The system takes the location and place type and inputs into the algorithm
3. The system searches for a place in the database with good recommendations within the same of similar category using the algorithm
4. The system returns a place from the database and provides it to the user
5. The user views the place details
6. The user adds the place to their itinerary

Alternate flow

A1 : User Rejects Recommendation

1. The system returns a place from the database and provides it to the user
2. The user views the place details
3. The user does not add place
4. The system inputs action into recommendation algorithm

Exceptional flow

E1 : The system does not return a recommendation

5. The system cannot find a recommendation
6. The system does not show any recommendations to the user

Termination

The user receives a recommendation and either adds place or dismisses

Post condition

The system returns to the search screen and waits for user input

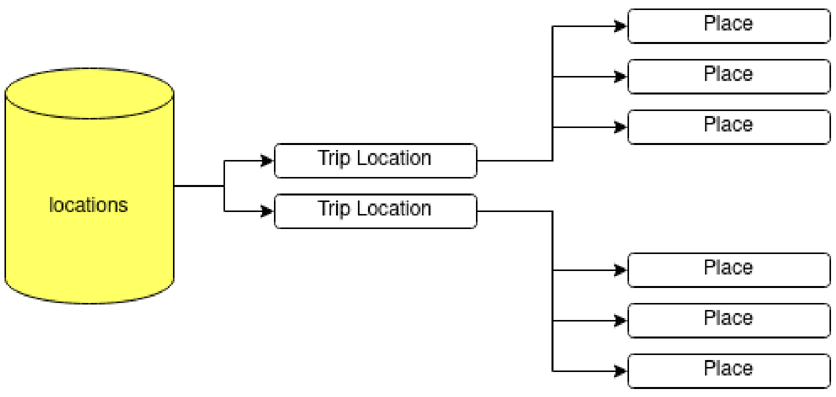
2.1.2. Data Requirements

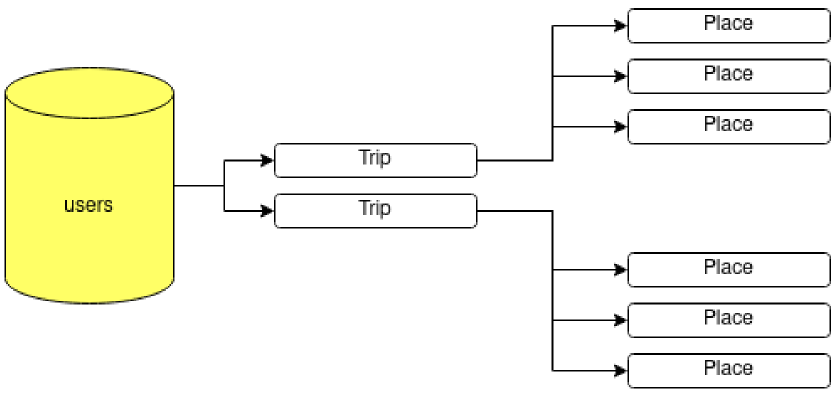
For the database, we will be using Firebase Cloud Firestore. Cloud Firestore is a cloud-based document NoSQL database. It incorporates well with Android and Android Studio.

Requirements:

- User can create many trips
- Trips can have many Places
- We can store Places separately from users to perform actions for the recommendation algorithm
- Flexible schema to allow changes in the type of data being stored
- Prioritise speed
- Structure to reduce read and write operations where possible
- Security features

Below is a diagram of the database structure. Note that the relationships don't explicitly exist in a NoSQL database like in a traditional database. The relationships here represent nested collections in the database.





The database uses a nested structure to store trips and places associated with a user. It also stores all the places searched by all users in the system in a separate collection. The reasoning behind this is to reduce the amount of read operations we need to perform when recommending a place to the user. Firebase bills read and write operations but has less restrictions on the amount of data stored. This makes it more important for data to be more accessible than to reduce data redundancy.

NoSQL databases are less efficient at performing complex queries. If we want to find all the places that and get their reviews, we could have to search every user account and then every trip and then every place in that trip using the given structure. This would require a lot of read operations which Firebase bills and it would slow the operation. By creating a separate collection when the Place is written to the database, we can increase the read speed later when the system must read a lot of data to generate a recommendation for the user.

NoSQL also allows flexibility in the database schema. This is useful when using APIs where the data received could change depending on the response. It can also create a new document when a user adds a new place with a place type that the system has not stored before.

Firebase also provides authentication methods for users. It offers different sign in methods and password reset tokens. Users are authenticated separately to the main database. [2]

2.1.3. User Requirements

- The system should provide useful error messages that help rectify issues
- The user should be able to customise certain features to make the app more accessible.
- The user can provide a valid email address

2.1.4. Environmental Requirements

The user is required to have a mobile Android device with a minimum of Android Lollipop 5.0.

2.1.5. Usability Requirements

- The interface should require low input from the user to achieve results
- The system should provide the user with useful error messages when something goes wrong
- The system should be designed to reduce the possibility of errors occurring
- The user should be able to navigate the User Interface easily

The app will be designed to minimise the input from the user. A primary feature of the app will be to add place details from the API automatically. This will reduce the amount of user input to the system.

The system will be tested to help design useful error messages for the user.

We can reduce errors by limiting user interaction with the system where possible. Testing the system thoroughly will also be important to lower the error rate.

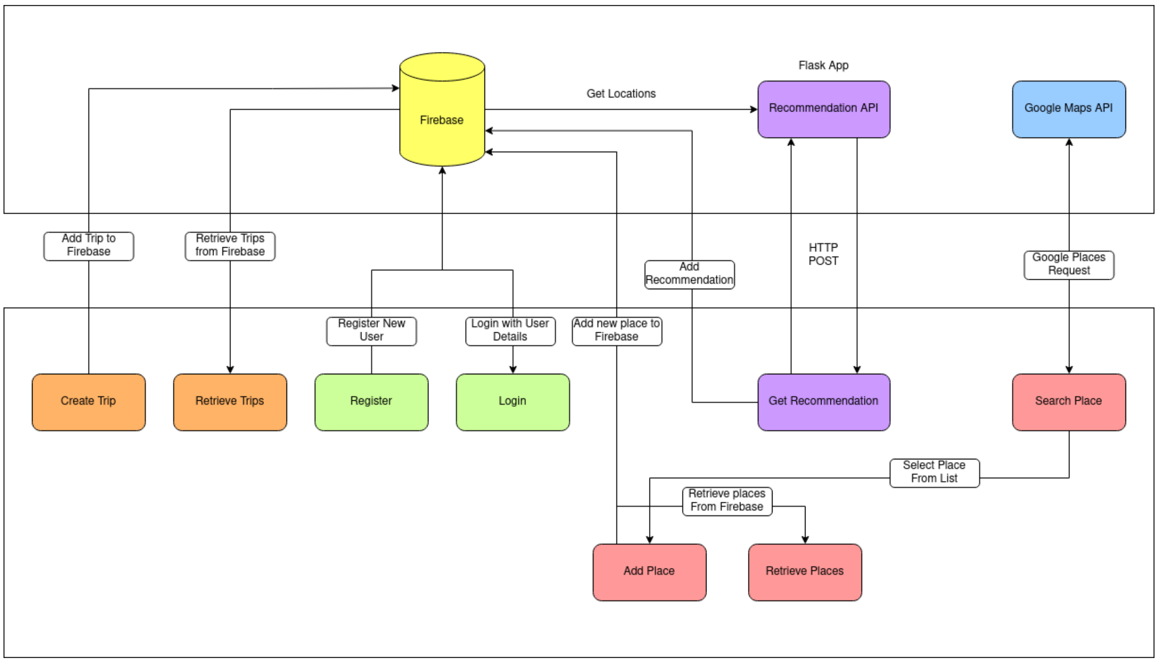
The base design of the application should provide the code to allow the user to navigate easily through the app.

2.2. Design & Architecture

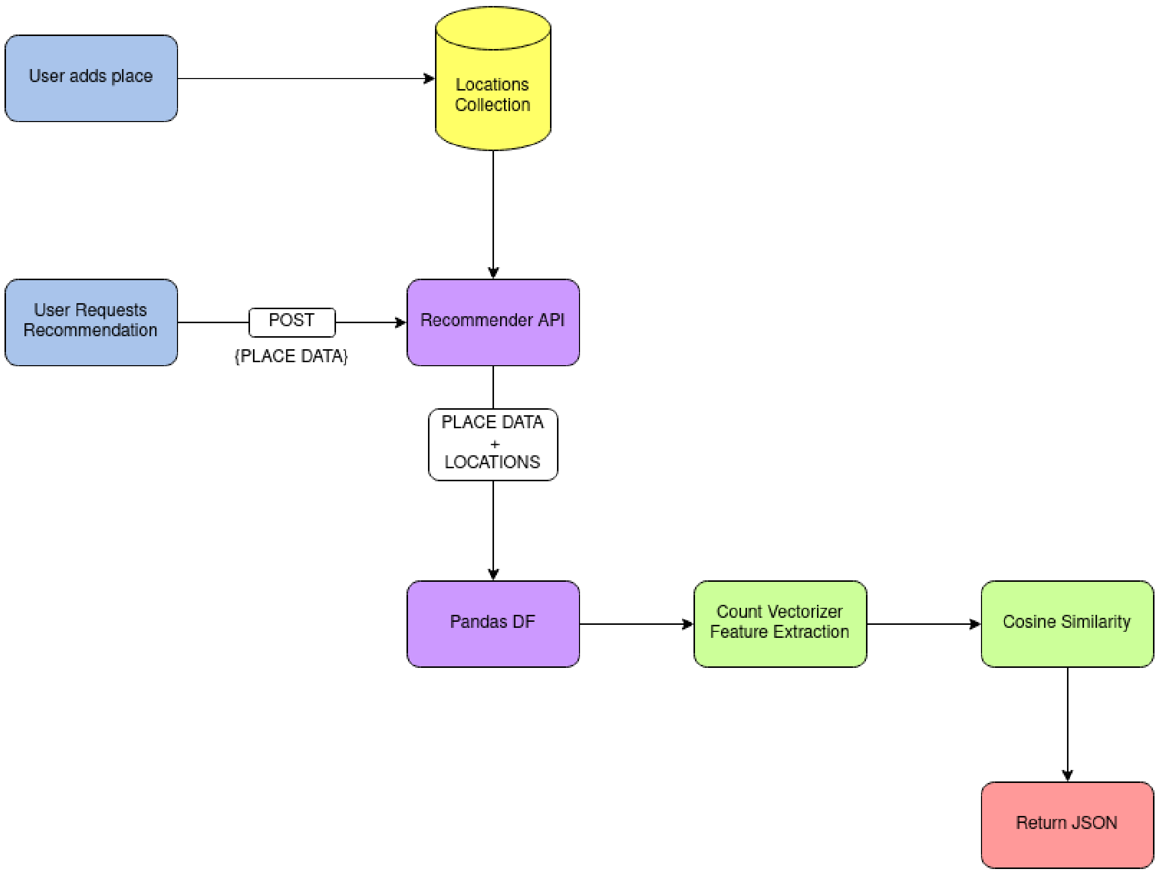
The client side of the application is and Android Application. The figure below shows how the client interacts with the backend of the application. The backend includes a Firebase database, a recommendation API that was made for the application and the Google Maps API.

The trips features and login and registration are handled by Firebase. The add places feature involves both the recommendation API and the Google Maps API. When a user searches for a place, it queries the Google Places API Autocomplete [4]. The user can then choose whether to add the place or not.

The user can also request recommended places from the Recommendation API. The API retrieves all locations under the user's trip location and computes similarity based on the place the user provides as input.



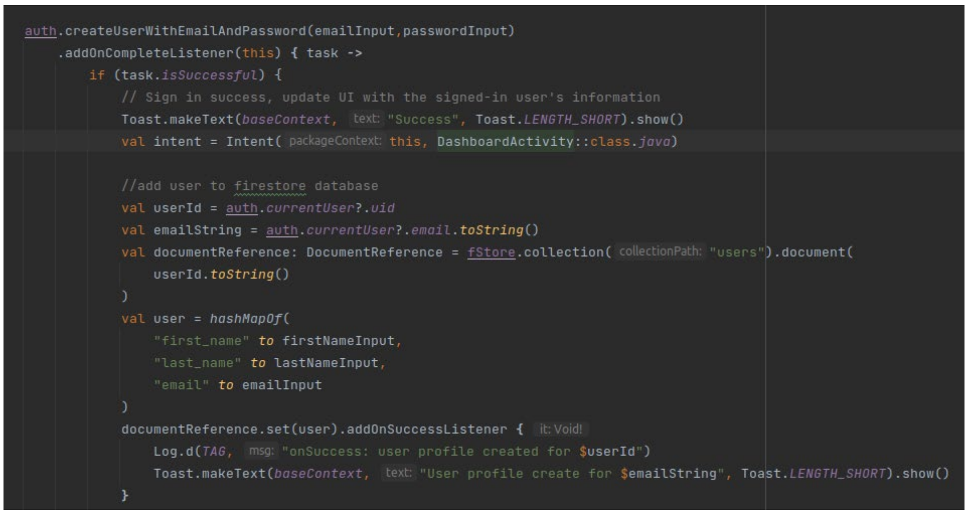
This diagram shows the Architecture of the Recommendation System. First users add places to the locations collection in Firebase. This collects all the places searched under a trip location from all users. When a user requests a recommendation, the API retrieves all the places from the trip location specified in the POST request. The place from the POST request is then appended onto the Firebase data and inserted into a pandas data frame. A cosine similarity algorithm is used to compute the similarity of each place based on the place types provided by the Google Places API. The API returns a JSON response to the Android App.



2.3. Implementation

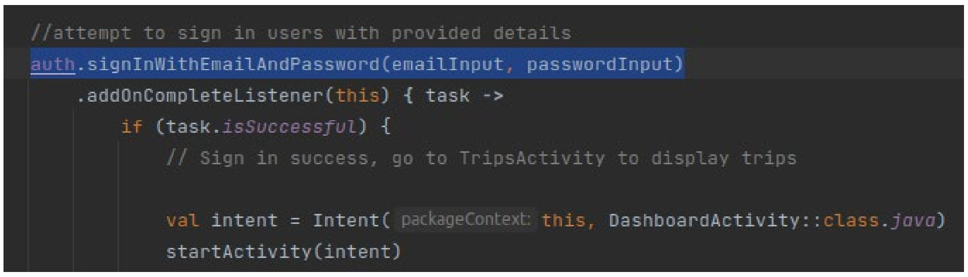
2.3.1. Login and Registration

To register a user, the Firebase auth method is called. This allows us to create a user using the provided email and password. Firebase authentication is handled separately from the main database. To create a collection of users in the main database, we need to get the userId that was just created and create a collection. This will then be used to store user trips and places.



```
auth.createUserWithEmailAndPassword(emailInput,passwordInput)
    .addOnCompleteListener(this) { task ->
        if (task.isSuccessful) {
            // Sign in success, update UI with the signed-in user's information
            Toast.makeText(baseContext, text: "Success", Toast.LENGTH_SHORT).show()
            val intent = Intent( packageContext: this, DashboardActivity::class.java)

            //add user to firestore database
            val userId = auth.currentUser?.uid
            val emailString = auth.currentUser?.email.toString()
            val documentReference: DocumentReference = fStore.collection( collectionPath: "users").document(
                userId.toString()
            )
            val user = hashMapOf(
                "first_name" to firstNameInput,
                "last_name" to lastNameInput,
                "email" to emailInput
            )
            documentReference.set(user).addOnSuccessListener {  it: Void!
                Log.d(TAG,  msg: "onSuccess: user profile created for $userId")
                Toast.makeText(baseContext,  text: "User profile create for $emailString", Toast.LENGTH_SHORT).show()
            }
```

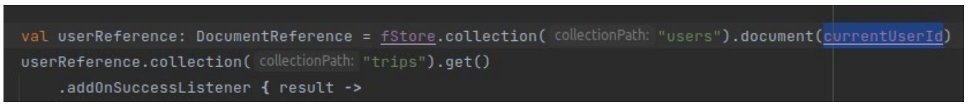
To login we provide valid account credentials and the Firebase auth will sign in with the email and password provided.



```
//attempt to sign in users with provided details
auth.signInWithEmailAndPassword(emailInput, passwordInput)
    .addOnCompleteListener(this) { task ->
        if (task.isSuccessful) {
            // Sign in success, go to TripsActivity to display trips

            val intent = Intent( packageContext: this, DashboardActivity::class.java)
            startActivity(intent)
```

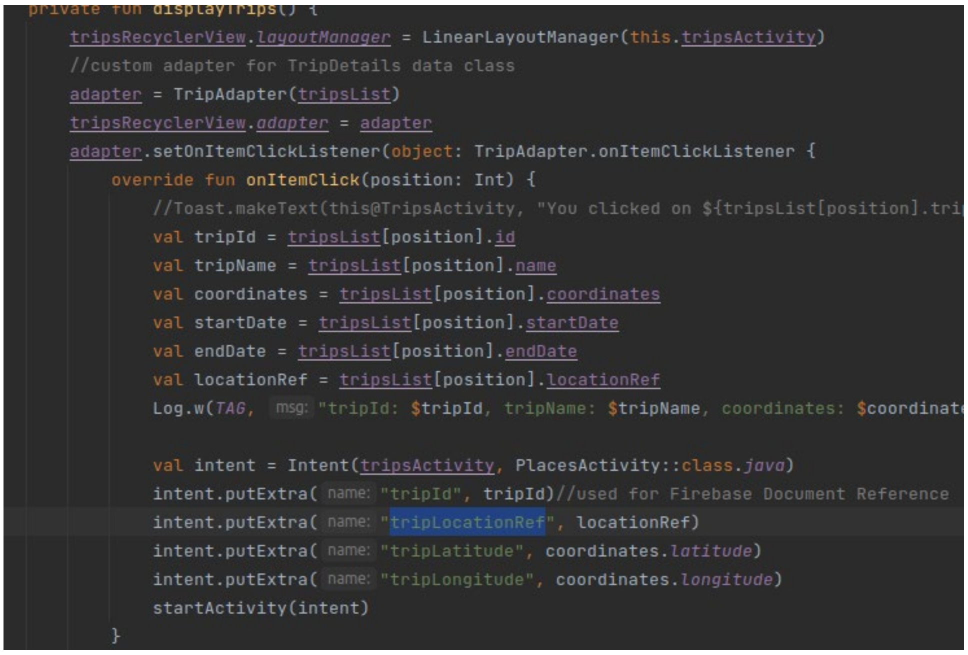
2.3.2. Trips

Retrieve trips from Firebase using the current user id. The current user can be retrieved from Firebase Auth from an activity while the session is alive.



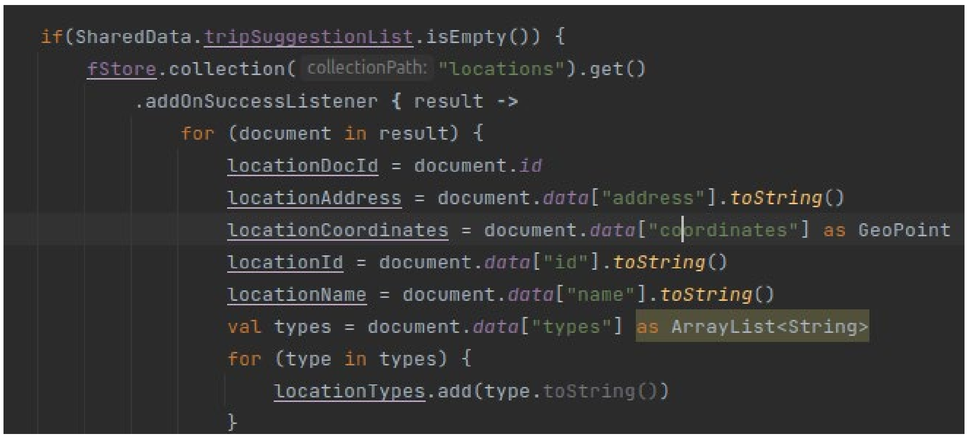
```
val userReference: DocumentReference = fStore.collection( collectionPath: "users").document(currentUserId)
userReference.collection( collectionPath: "trips").get()
    .addOnSuccessListener { result ->
```

The trips from the list are displayed in a recycler view. Clicking an item on the recycler view starts PlacesActivity. Extra information is passed into the intent to identify the item being clicked on the list. This information is used to retrieve the data associated with that specific trip.



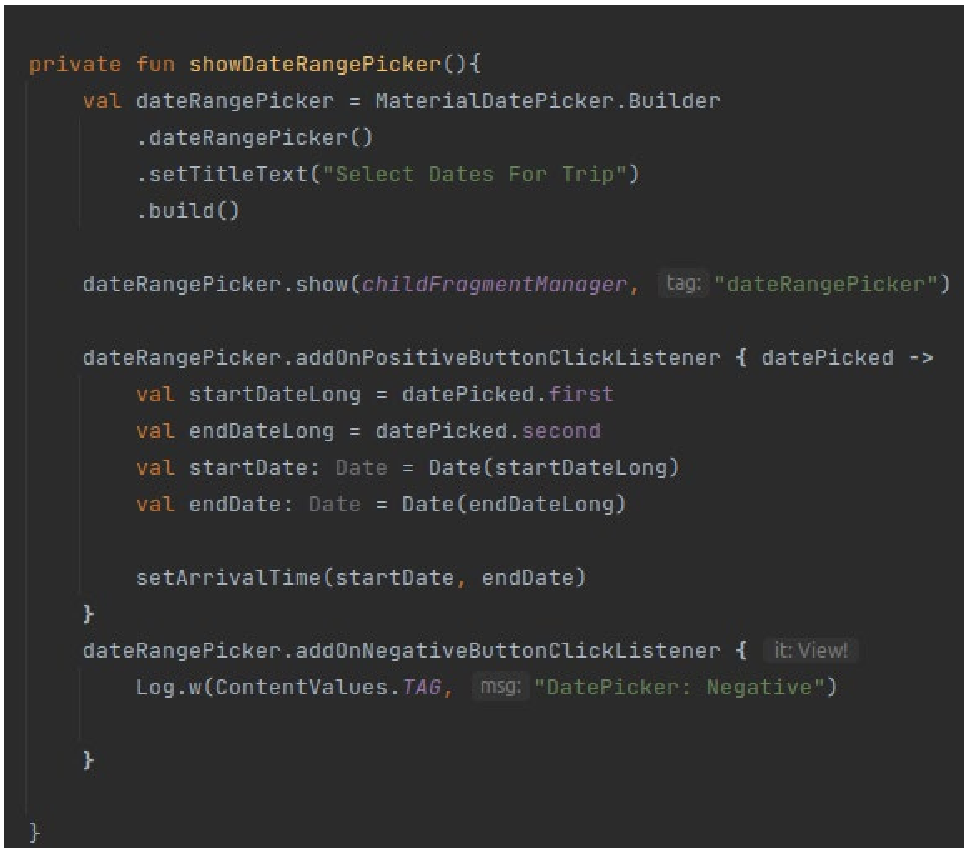
```
private fun displayTrips() {
    tripsRecyclerView.layoutManager = LinearLayoutManager(this.tripsActivity)
    //custom adapter for TripDetails data class
    adapter = TripAdapter(tripsList)
    tripsRecyclerView.adapter = adapter
    adapter.setOnItemClickListener(object: TripAdapter.onItemClickListener {
        override fun onItemClick(position: Int) {
            //Toast.makeText(this@TripsActivity, "You clicked on ${tripsList[position].tri
            val tripId = tripsList[position].id
            val tripName = tripsList[position].name
            val coordinates = tripsList[position].coordinates
            val startDate = tripsList[position].startDate
            val endDate = tripsList[position].endDate
            val locationRef = tripsList[position].locationRef
            Log.w(TAG,  msg: "tripId: $tripId, tripName: $tripName, coordinates: $coordinat

            val intent = Intent(tripsActivity, PlacesActivity::class.java)
            intent.putExtra( name: "tripId", tripId)//used for Firebase Document Reference
            intent.putExtra( name: "tripLocationRef", locationRef)
            intent.putExtra( name: "tripLatitude", coordinates.latitude)
            intent.putExtra( name: "tripLongitude", coordinates.longitude)
            startActivity(intent)
        }
```

When the NewTripFragment is loaded is retrieves trip locations from the location collection in firebase. The locations are displayed in a recycler view which the user can select to add a new trip at that location.



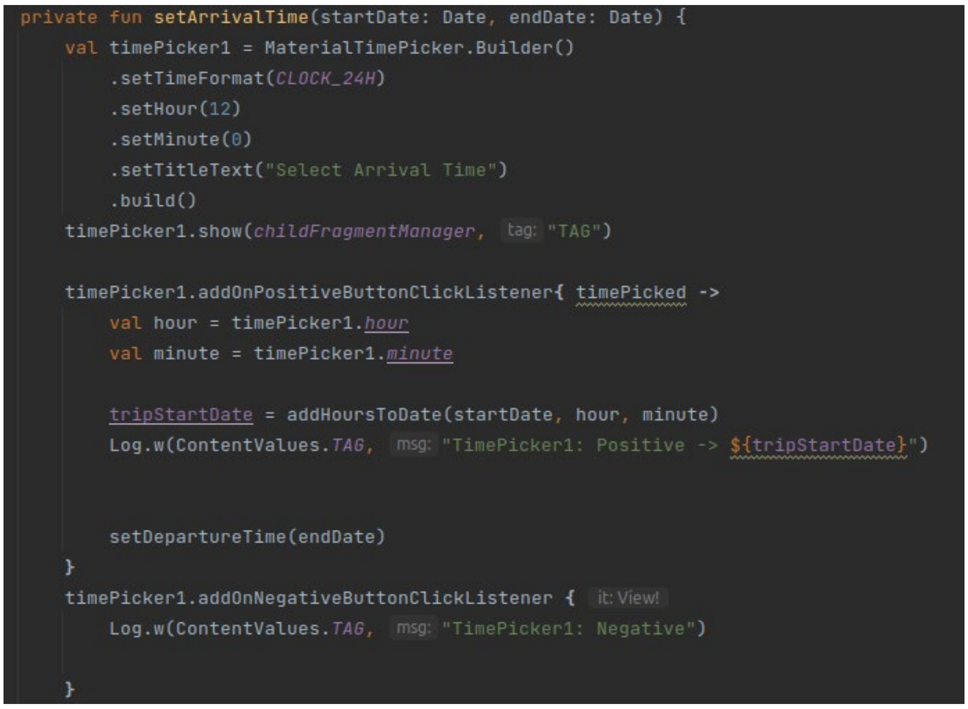
```kotlin
if(SharedData.tripSuggestionList.isEmpty()) {
    fStore.collection( collectionPath: "locations").get()
        .addOnSuccessListener { result ->
            for (document in result) {
                locationDocId = document.id
                locationAddress = document.data["address"].toString()
                locationCoordinates = document.data["coordinates"] as GeoPoint
                locationId = document.data["id"].toString()
                locationName = document.data["name"].toString()
                val types = document.data["types"] as ArrayList<String>
                for (type in types) {
                    locationTypes.add(type.toString())
                }
```

When the user adds a trip, they must specify a start and end date. We use a date range picker to allow the user to select the dates.

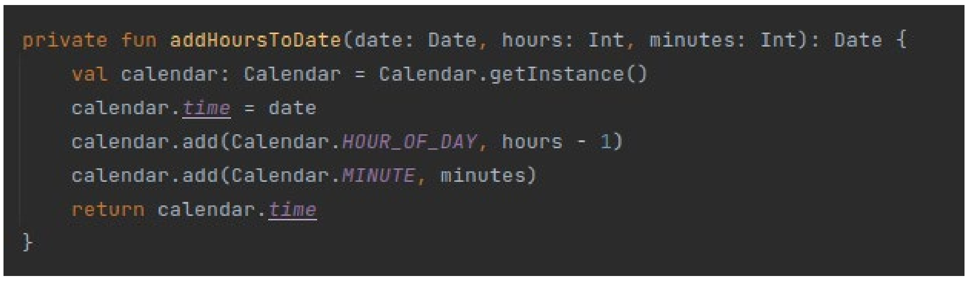


```kotlin
private fun showDateRangePicker(){
    val dateRangePicker = MaterialDatePicker.Builder
        .dateRangePicker()
        .setTitleText("Select Dates For Trip")
        .build()

    dateRangePicker.show(childFragmentManager,  tag: "dateRangePicker")

    dateRangePicker.addOnPositiveButtonClickListener { datePicked ->
        val startDateLong = datePicked.first
        val endDateLong = datePicked.second
        val startDate: Date = Date(startDateLong)
        val endDate: Date = Date(endDateLong)

        setArrivalTime(startDate, endDate)
    }
    dateRangePicker.addOnNegativeButtonClickListener {  it: View!
        Log.w(ContentValues.TAG,  msg: "DatePicker: Negative")

    }

}
```

When the user selects the dates, they are prompted to add an arrival time and departure time. This i achieved using a time picker.

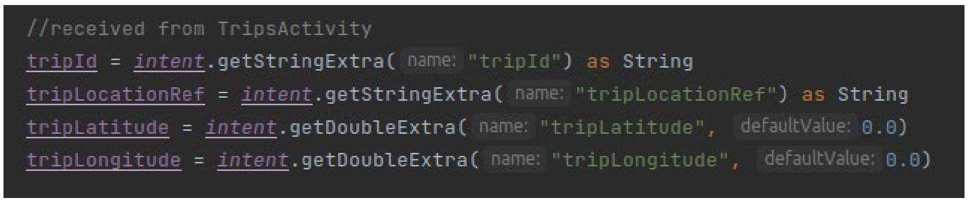

```
private fun setArrivalTime(startDate: Date, endDate: Date) {
    val timePicker1 = MaterialTimePicker.Builder()
        .setTimeFormat(CLOCK_24H)
        .setHour(12)
        .setMinute(0)
        .setTitleText("Select Arrival Time")
        .build()
    timePicker1.show(childFragmentManager,  tag: "TAG")

    timePicker1.addOnPositiveButtonClickListener{ timePicked ->
        val hour = timePicker1.hour
        val minute = timePicker1.minute

        tripStartDate = addHoursToDate(startDate, hour, minute)
        Log.w(ContentValues.TAG,  msg: "TimePicker1: Positive -> ${tripStartDate}")


        setDepartureTime(endDate)
    }
    timePicker1.addOnNegativeButtonClickListener {  it: View!
        Log.w(ContentValues.TAG,  msg: "TimePicker1: Negative")

    }
```

Firebase stores dates as timestamps. To store the time and date in one field, the hours have to be added to the date. This is achieved using the calendar class. The hours and minutes from the time picker is passed into this method which returns a Date object. This can be converted to a Timestamp.

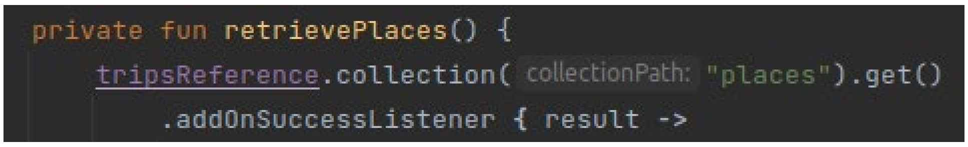

```
private fun addHoursToDate(date: Date, hours: Int, minutes: Int): Date {
    val calendar: Calendar = Calendar.getInstance()
    calendar.time = date
    calendar.add(Calendar.HOUR_OF_DAY, hours - 1)
    calendar.add(Calendar.MINUTE, minutes)
    return calendar.time
}
```

2.3.3. Places

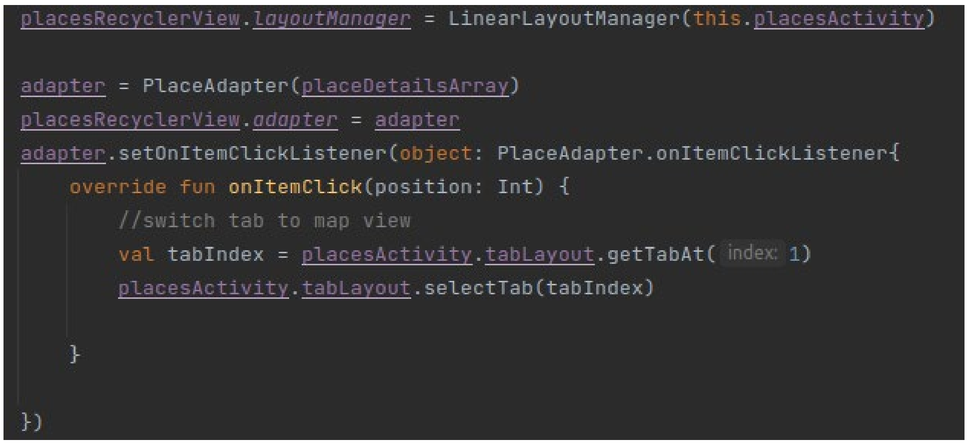
Get trip intent from TripsActivity. This provides the PlacesActivity with some extra information to perform some operations.


```
//received from TripsActivity
tripId = intent.getStringExtra( name: "tripId") as String
tripLocationRef = intent.getStringExtra( name: "tripLocationRef") as String
tripLatitude = intent.getDoubleExtra( name: "tripLatitude",  defaultValue: 0.0)
tripLongitude = intent.getDoubleExtra( name: "tripLongitude",  defaultValue: 0.0)
```

Use the tripId from the intent to get places nested in that trip from the database

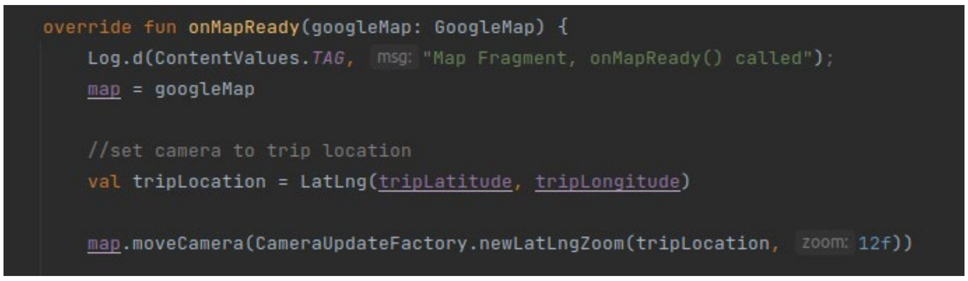

```
private fun retrievePlaces() {
    tripsReference.collection( collectionPath: "places").get()
        .addOnSuccessListener { result ->
```

Display places in a recycler view in PlacesListFragment



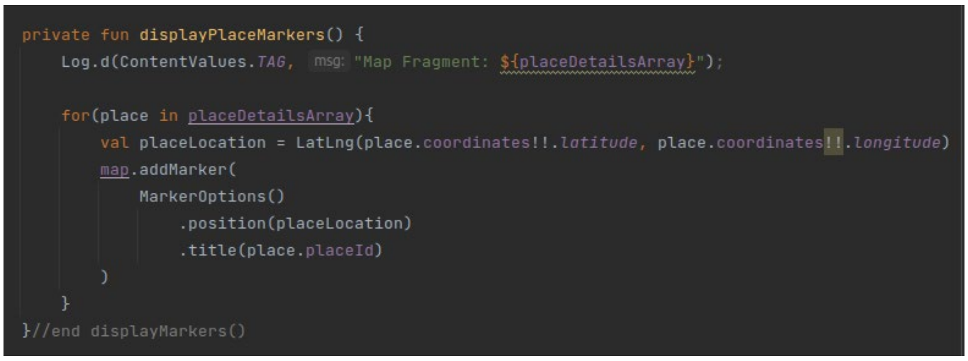
```
placesRecyclerView.layoutManager = LinearLayoutManager(this.placesActivity)

adapter = PlaceAdapter(placeDetailsArray)
placesRecyclerView.adapter = adapter
adapter.setOnItemClickListener(object: PlaceAdapter.onItemClickListener{
    override fun onItemClick(position: Int) {
        //switch tab to map view
        val tabIndex = placesActivity.tabLayout.getTabAt( index: 1)
        placesActivity.tabLayout.selectTab(tabIndex)

    }

})
```

Set map camera on the coordinates provided by the intent



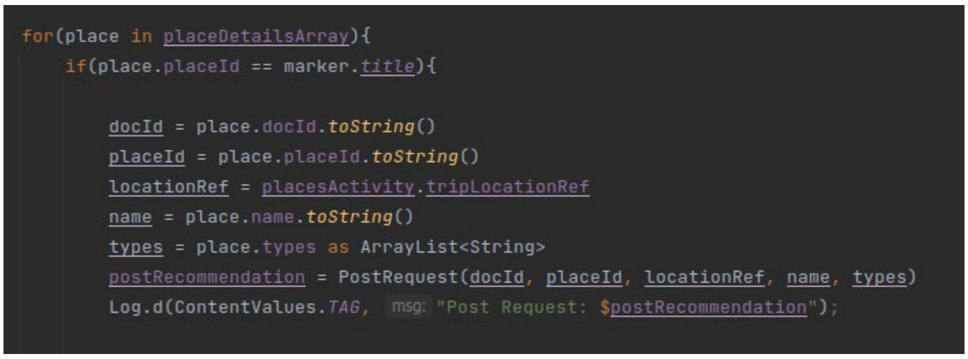
```
  override fun onMapReady(googleMap: GoogleMap) {
      Log.d(ContentValues.TAG,  msg: "Map Fragment, onMapReady() called");
      map = googleMap

      //set camera to trip location
      val tripLocation = LatLng(tripLatitude, tripLongitude)

      map.moveCamera(CameraUpdateFactory.newLatLngZoom(tripLocation,  zoom: 12f))
```

Display map markers from list of places array. Set the marker title to the placeId



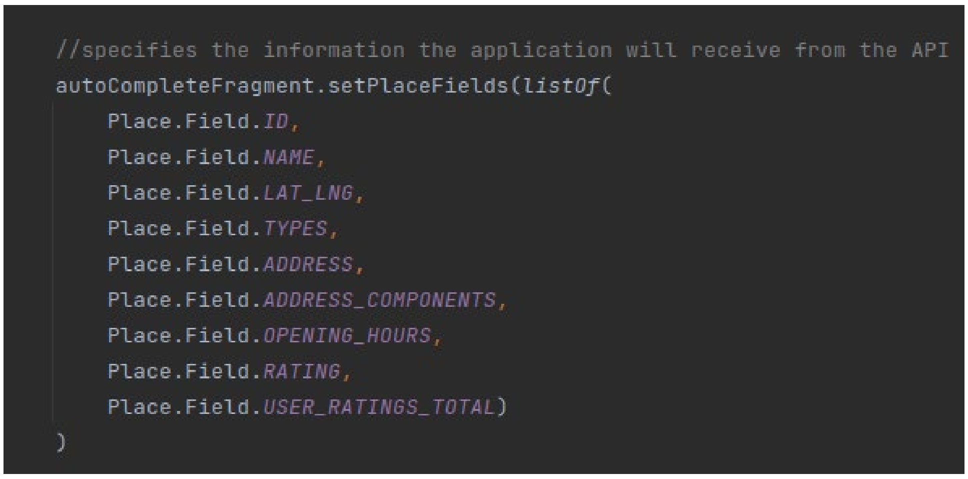
```
private fun displayPlaceMarkers() {
    Log.d(ContentValues.TAG,  msg: "Map Fragment: ${placeDetailsArray}");

    for(place in placeDetailsArray){
        val placeLocation = LatLng(place.coordinates!!.latitude, place.coordinates!!.longitude)
        map.addMarker(
            MarkerOptions()
                .position(placeLocation)
                .title(place.placeId)
        )
    }
}//end displayMarkers()
```

Map marker onClickListener gets place details from the marker using the placeId. The other data can then be retrieved from the placeDetailsArray



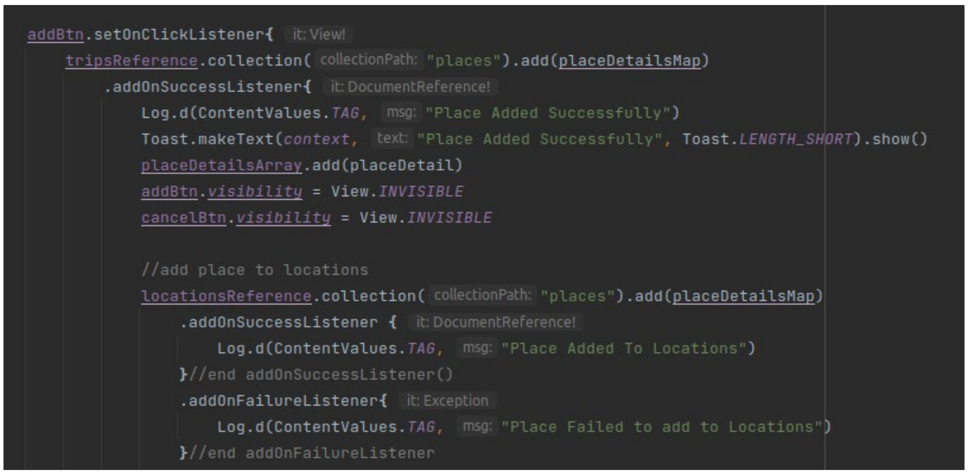
```
for(place in placeDetailsArray){
    if(place.placeId == marker.title){

        docId = place.docId.toString()
        placeId = place.placeId.toString()
        locationRef = placesActivity.tripLocationRef
        name = place.name.toString()
        types = place.types as ArrayList<String>
        postRecommendation = PostRequest(docId, placeId, locationRef, name, types)
        Log.d(ContentValues.TAG,  msg: "Post Request: $postRecommendation");
```

Specifies the data that the Google Places API will return on when searched



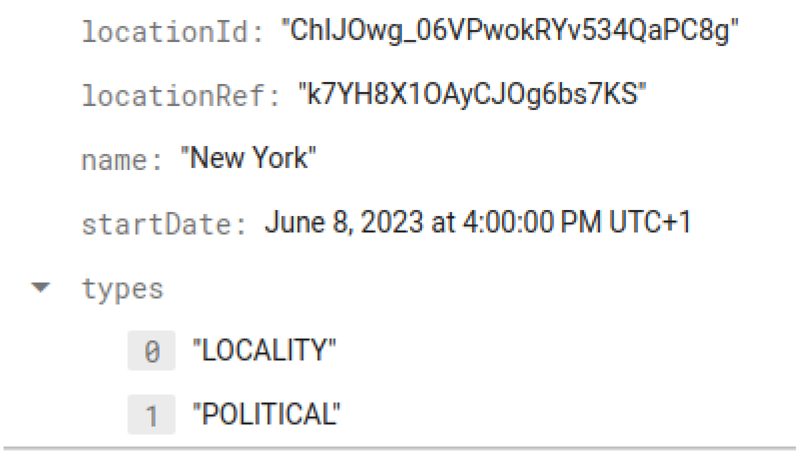
```
//specifies the information the application will receive from the API
autoCompleteFragment.setPlaceFields(listOf(
    Place.Field.ID,
    Place.Field.NAME,
    Place.Field.LAT_LNG,
    Place.Field.TYPES,
    Place.Field.ADDRESS,
    Place.Field.ADDRESS_COMPONENTS,
    Place.Field.OPENING_HOURS,
    Place.Field.RATING,
    Place.Field.USER_RATINGS_TOTAL)
)
```

The data returned can be added to the database using the tripsReference we made using the tripId from the intent. If the place is successfully added to the users trips, the place is added to the locations collection under the trip location. This is done using the location reference provided in the intent.



```
addBtn.setOnClickListener{ it: View!
    tripsReference.collection( collectionPath: "places").add(placeDetailsMap)
        .addOnSuccessListener{ it: DocumentReference!
            Log.d(ContentValues.TAG, msg: "Place Added Successfully")
            Toast.makeText(context, text: "Place Added Successfully", Toast.LENGTH_SHORT).show()
            placeDetailsArray.add(placeDetail)
            addBtn.visibility = View.INVISIBLE
            cancelBtn.visibility = View.INVISIBLE

            //add place to locations
            locationsReference.collection( collectionPath: "places").add(placeDetailsMap)
                .addOnSuccessListener { it: DocumentReference!
                    Log.d(ContentValues.TAG, msg: "Place Added To Locations")
                }//end addOnSuccessListener()
                .addOnFailureListener{ it: Exception
                    Log.d(ContentValues.TAG, msg: "Place Failed to add to Locations")
                }//end addOnFailureListener
```

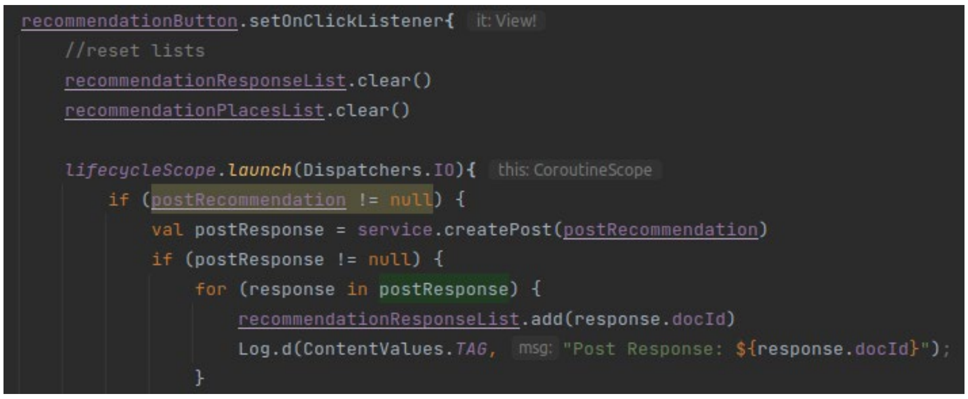
In the trips document of each users trip is a reference to the document in the locations collection. This allows us to add the place to both collections when adding a place from the Places API.



```
locationId: "ChIJOwg_06VPwokRYv534QaPC8g"
locationRef: "k7YH8X1OAyCJOg6bs7KS"
name: "New York"
startDate: June 8, 2023 at 4:00:00 PM UTC+1
▾ types
    0 "LOCALITY"
    1 "POLITICAL"
```

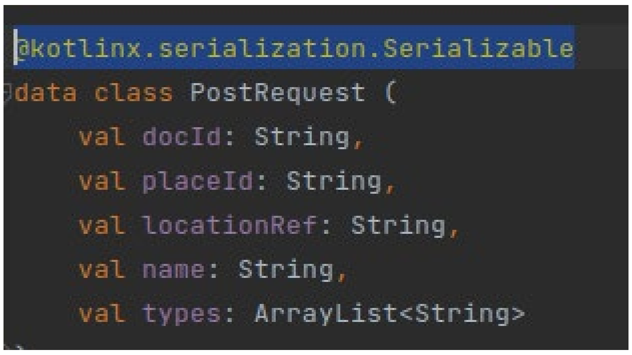
2.3.4. Recommendations Client

To initiate the POST request to the recommendation API, an onClickListener is set which triggers the event. A coroutine is launched to send the request as a worker on the Thread [5]. A null check is performed on the response before the docId is added to an array list. This will be used to search for the place in Firebase later.



```
recommendationButton.setOnClickListener{  it: View!
    //reset lists
    recommendationResponseList.clear()
    recommendationPlacesList.clear()

    lifecycleScope.launch(Dispatchers.IO){  this: CoroutineScope
        if (postRecommendation != null) {
            val postResponse = service.createPost(postRecommendation)
            if (postResponse != null) {
                for (response in postResponse) {
                    recommendationResponseList.add(response.docId)
                    Log.d(ContentValues.TAG,  msg: "Post Response: ${response.docId}");
                }
```

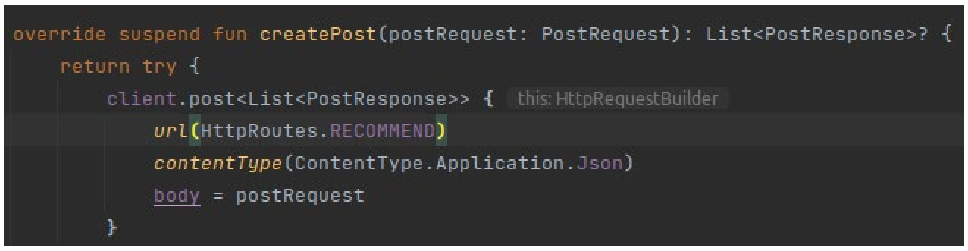
The request body contains 5 parameters. The placeId is a unique id from the Google Places API. The locationRef is used to determine which location the API needs to retrieve places under. The types array is what is used to compute similarity and return a recommendation.



```
@kotlinx.serialization.Serializable
data class PostRequest (
    val docId: String,
    val placeId: String,
    val locationRef: String,
    val name: String,
    val types: ArrayList<String>
```

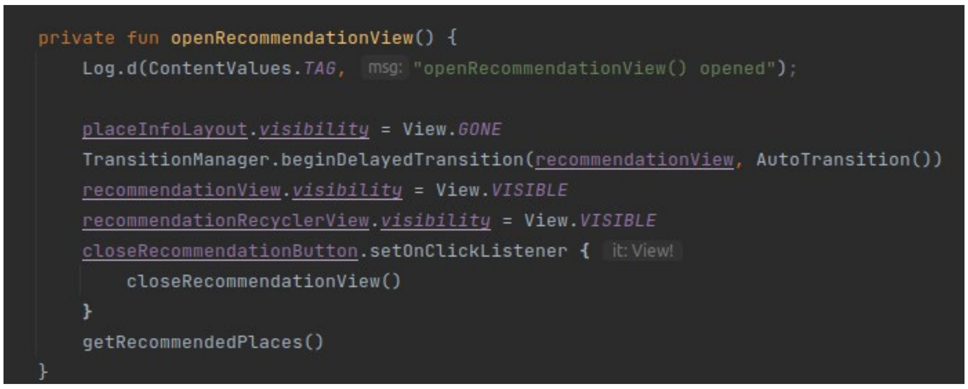
For the HTTP request we are using KTOR. KTOR is a library for Kotlin which can be used to create client and server features. It is asynchronous by using Kotlin coroutines and is fully build in Kotlin. This means it can be fully interoperable with the JVM and can be ported to IOS applications [6].

For the createPost method, it accepts a postRequest object and returns a List of post response objects. The body is specified as application/json.



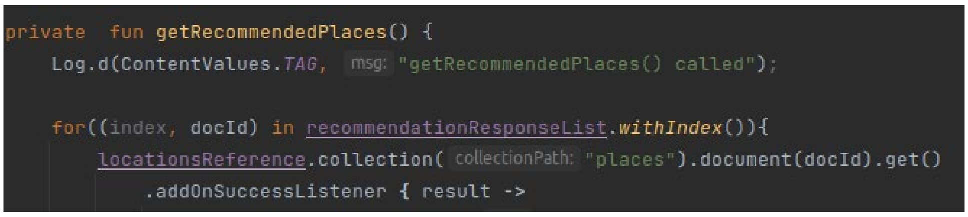
```
override suspend fun createPost(postRequest: PostRequest): List<PostResponse>? {
    return try {
        client.post<List<PostResponse>> {  this: HttpRequestBuilder
            url(HttpRoutes.RECOMMEND)
            contentType(ContentType.Application.Json)
            body = postRequest
        }
```

The recommendation list view is then opened in the UI



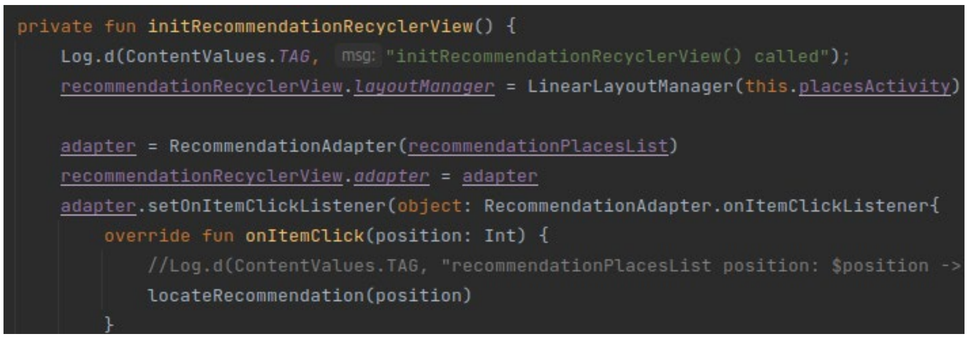
```
private fun openRecommendationView() {
    Log.d(ContentValues.TAG,  msg: "openRecommendationView() opened");

    placeInfoLayout.visibility = View.GONE
    TransitionManager.beginDelayedTransition(recommendationView, AutoTransition())
    recommendationView.visibility = View.VISIBLE
    recommendationRecyclerView.visibility = View.VISIBLE
    closeRecommendationButton.setOnClickListener {  it: View!
        closeRecommendationView()
    }
    getRecommendedPlaces()
}
```

The application then gets the documents from Firebase using the docIds returned from the API.



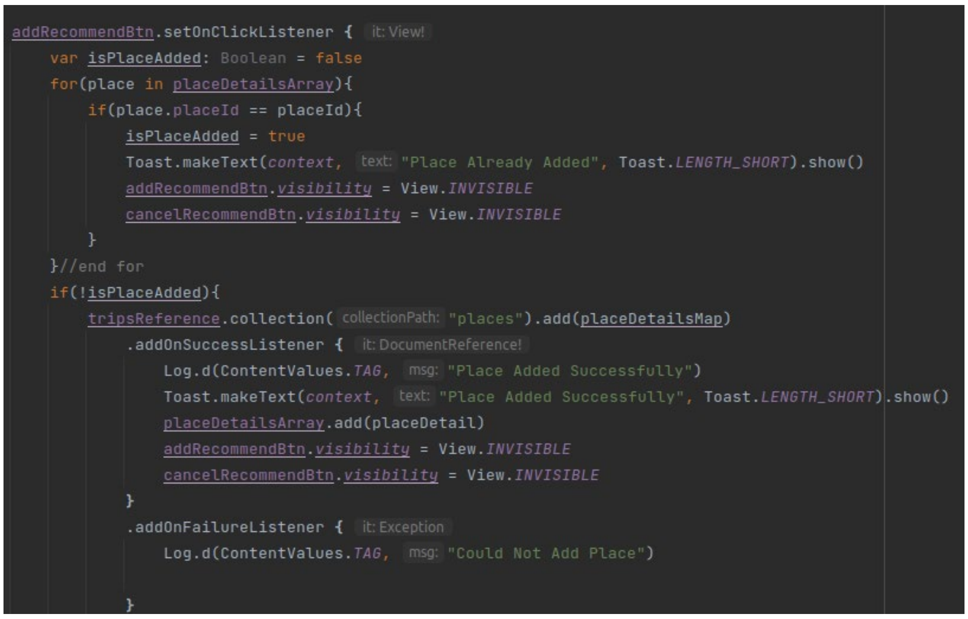
```
private  fun getRecommendedPlaces() {
    Log.d(ContentValues.TAG,  msg: "getRecommendedPlaces() called");

    for((index, docId) in recommendationResponseList.withIndex()){
        locationsReference.collection( collectionPath: "places").document(docId).get()
            .addOnSuccessListener { result ->
```

These values are then put into another ArrayList and displayed in a RecyclerView



```
private fun initRecommendationRecyclerView() {
    Log.d(ContentValues.TAG,  msg: "initRecommendationRecyclerView() called");
    recommendationRecyclerView.layoutManager = LinearLayoutManager(this.placesActivity)

    adapter = RecommendationAdapter(recommendationPlacesList)
    recommendationRecyclerView.adapter = adapter
    adapter.setOnItemClickListener(object: RecommendationAdapter.onItemClickListener{
        override fun onItemClick(position: Int) {
            //Log.d(ContentValues.TAG, "recommendationPlacesList position: $position ->
            locateRecommendation(position)
        }
```

If a place is selected from the Recycler View, the application checks whether the place already exists in the users places list. It checks this using the placeId which is unique ID provided by the Google Places API. If it not already added, then the place can be added to the users places list.

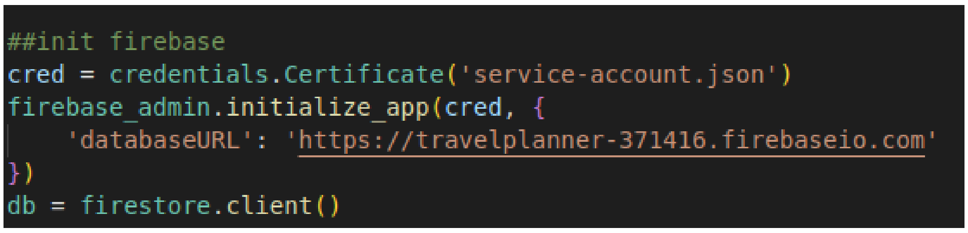


```
addRecommendBtn.setOnClickListener { it: View!
    var isPlaceAdded: Boolean = false
    for(place in placeDetailsArray){
        if(place.placeId == placeId){
            isPlaceAdded = true
            Toast.makeText(context, text: "Place Already Added", Toast.LENGTH_SHORT).show()
            addRecommendBtn.visibility = View.INVISIBLE
            cancelRecommendBtn.visibility = View.INVISIBLE
        }
    }//end for
    if(!isPlaceAdded){
        tripsReference.collection( collectionPath: "places").add(placeDetailsMap)
            .addOnSuccessListener { it: DocumentReference!
                Log.d(ContentValues.TAG, msg: "Place Added Successfully")
                Toast.makeText(context, text: "Place Added Successfully", Toast.LENGTH_SHORT).show()
                placeDetailsArray.add(placeDetail)
                addRecommendBtn.visibility = View.INVISIBLE
                cancelRecommendBtn.visibility = View.INVISIBLE
            }
            .addOnFailureListener { it: Exception
                Log.d(ContentValues.TAG, msg: "Could Not Add Place")

            }
```

2.3.5. Recommendations Server

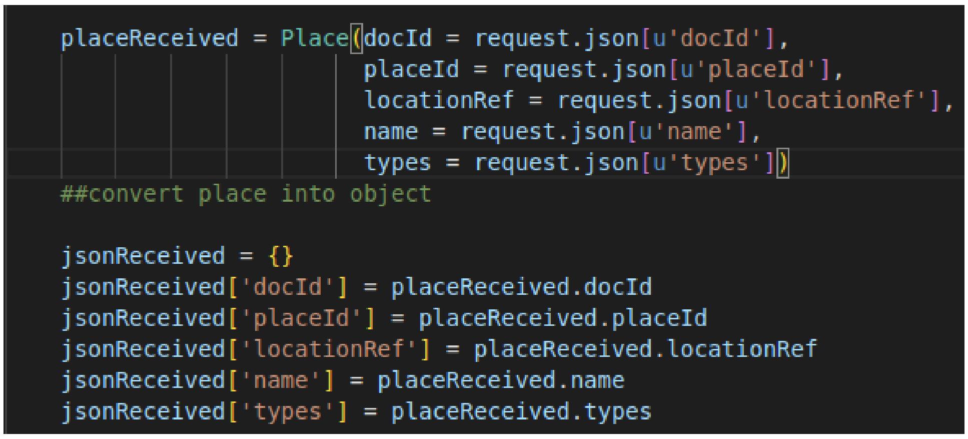
The code for the server side of the Recommendation system is a Python Flask app. We are using python to take advantage of its machine learning libraries. Flask allows us to route the application and create an API which can return a JSON response [7].

The first thing is to validate firebase credentials in python to connect to the database.



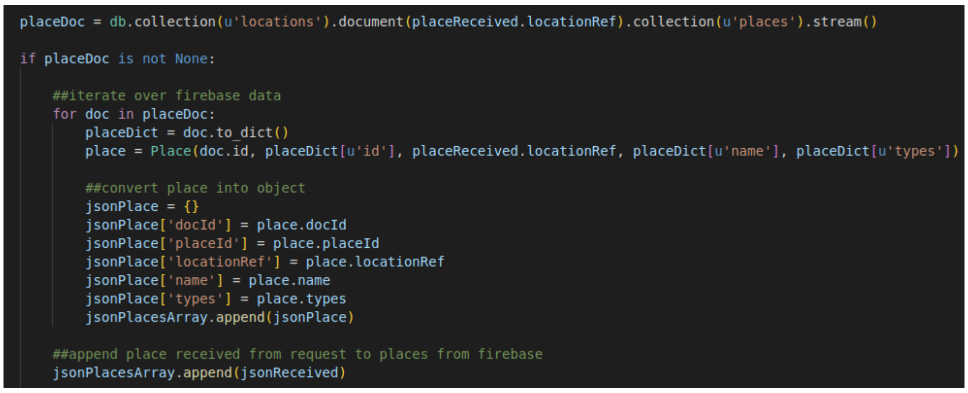
```
##init firebase
cred = credentials.Certificate('service-account.json')
firebase_admin.initialize_app(cred, {
    'databaseURL': 'https://travelplanner-371416.firebaseio.com'
})
db = firestore.client()
```

Then the application processes the request sent from the client using a HTTP POST. This is then converted to a JSON object.



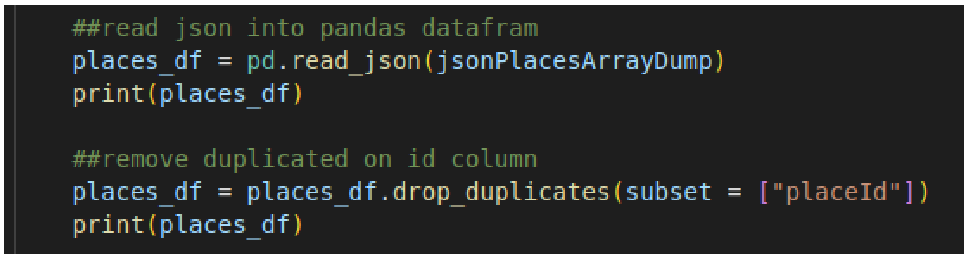
```
    placeReceived = Place(docId = request.json[u'docId'],
                          placeId = request.json[u'placeId'],
                          locationRef = request.json[u'locationRef'],
                          name = request.json[u'name'],
                          types = request.json[u'types'])
    ##convert place into object

    jsonReceived = {}
    jsonReceived['docId'] = placeReceived.docId
    jsonReceived['placeId'] = placeReceived.placeId
    jsonReceived['locationRef'] = placeReceived.locationRef
    jsonReceived['name'] = placeReceived.name
    jsonReceived['types'] = placeReceived.types
```

It then retrieves places from Firebase. It uses the location reference provided in the request to get places from the same location that the user trip is located. This is then converted into a JSON array.



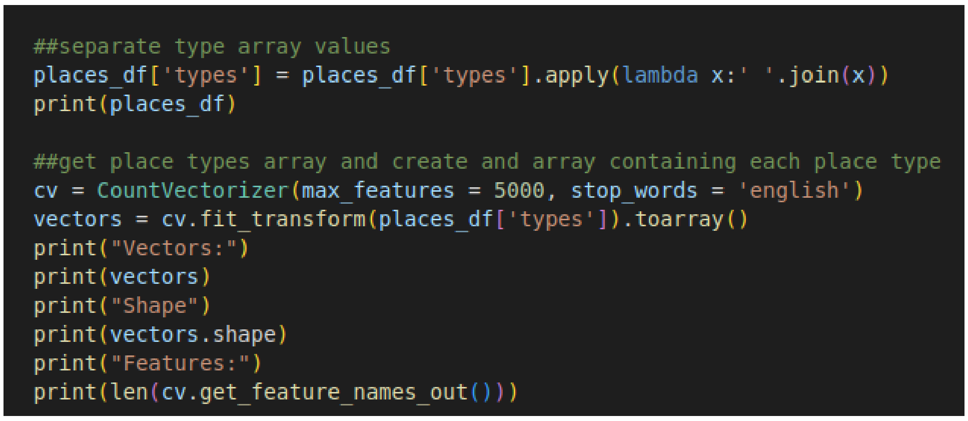
```
placeDoc = db.collection(u'locations').document(placeReceived.locationRef).collection(u'places').stream()

if placeDoc is not None:

    ##iterate over firebase data
    for doc in placeDoc:
        placeDict = doc.to_dict()
        place = Place(doc.id, placeDict[u'id'], placeReceived.locationRef, placeDict[u'name'], placeDict[u'types'])

        ##convert place into object
        jsonPlace = {}
        jsonPlace['docId'] = place.docId
        jsonPlace['placeId'] = place.placeId
        jsonPlace['locationRef'] = place.locationRef
        jsonPlace['name'] = place.name
        jsonPlace['types'] = place.types
        jsonPlacesArray.append(jsonPlace)

    ##append place received from request to places from firebase
    jsonPlacesArray.append(jsonReceived)
```

The request body JSON and the Database places JSON are read into a Pandas data frame. The duplicate values are removed.



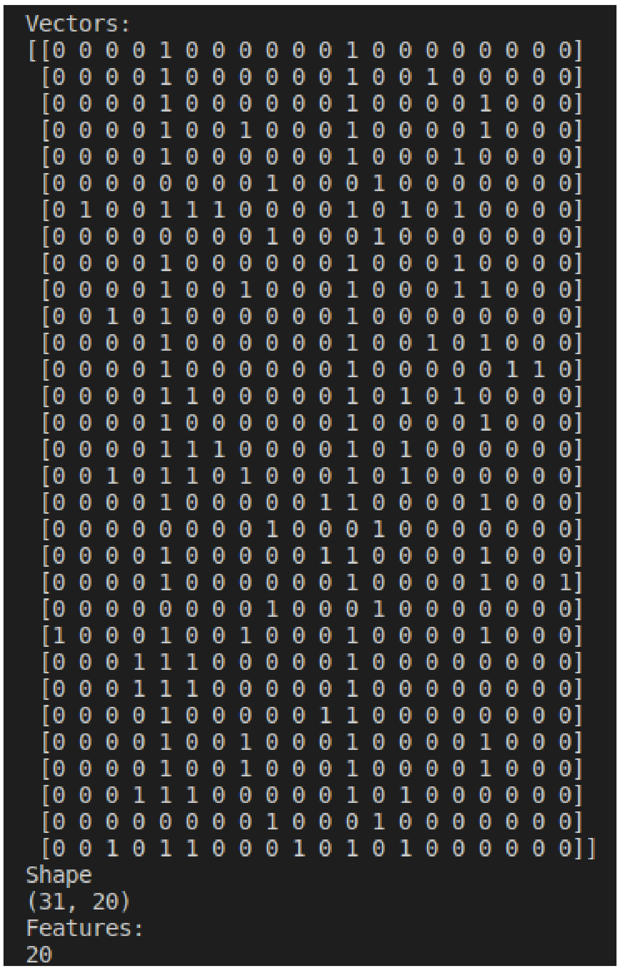
```
    ##read json into pandas datafram
    places_df = pd.read_json(jsonPlacesArrayDump)
    print(places_df)

    ##remove duplicated on id column
    places_df = places_df.drop_duplicates(subset = ["placeId"])
    print(places_df)
```

A count vectorizer algorithm is used to compute the frequency that each word occurs in a dataset [8]. This will be used to compute cosine similarity for generating a recommendation [9]. This is being compute on the types array of the places. This is a list of values provided by the Google Places API showing the type of place for example, if it is a museum or a cafe etc. It provides an array of values for each place which can be used to find similarity.



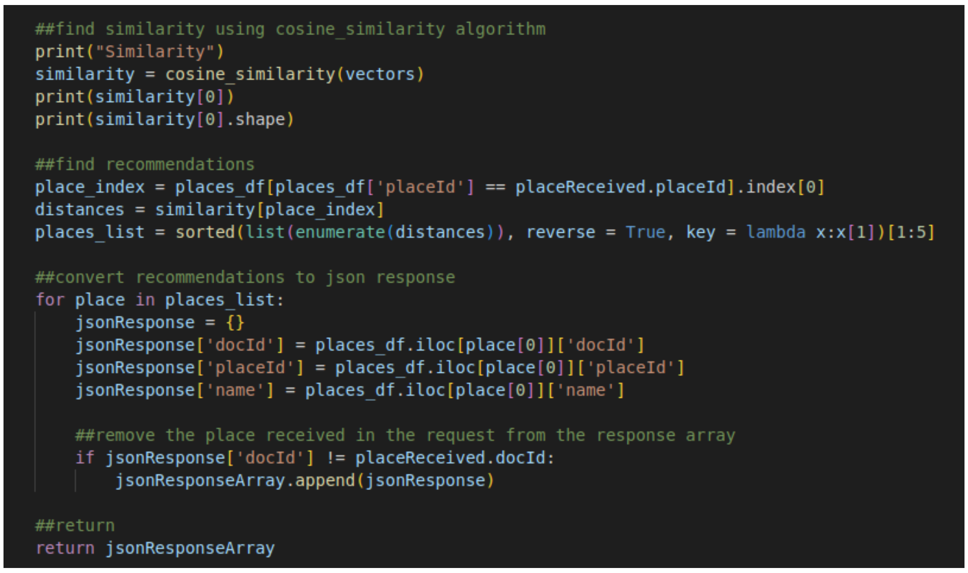
```
##separate type array values
places_df['types'] = places_df['types'].apply(lambda x:' '.join(x))
print(places_df)

##get place types array and create and array containing each place type
cv = CountVectorizer(max_features = 5000, stop_words = 'english')
vectors = cv.fit_transform(places_df['types']).toarray()
print("Vectors:")
print(vectors)
print("Shape")
print(vectors.shape)
print("Features:")
print(len(cv.get_feature_names_out()))
```

We can see from the printout below that there are 20 features given by the Count Vectorizer



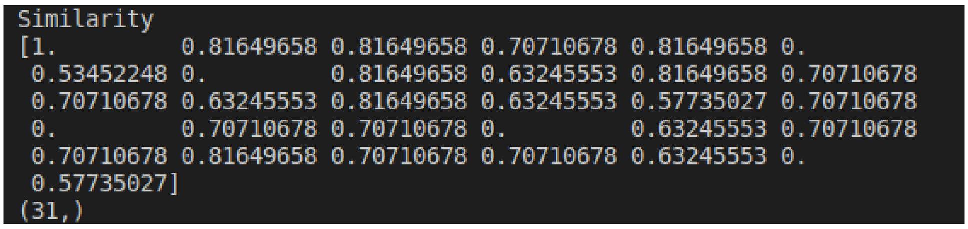
```
Vectors:
[[0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0]
 [0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0]
 [0 1 0 0 1 1 1 0 0 0 0 1 0 1 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0]
 [0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0]
 [0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0]
 [0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 1 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0]
 [0 0 0 0 1 1 1 0 0 0 0 1 0 1 0 0 0 0 0 0]
 [0 0 1 0 1 1 0 1 0 0 0 1 0 1 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1]
 [0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0]
 [1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0]
 [0 0 0 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0]
 [0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0]
 [0 0 0 1 1 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 1 0 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0]]
Shape
(31, 20)
Features:
20
```

Based on this, the cosine similarity can be computed. Cosine similarity is used to determine similarity irrespective of its size. This means that is objects are distant from each other they can still have a high similarity [10]. In this application, we are using a library from scikit-learn [11]. The response is then computed into a list and sorted based on highest similarity. The array is converted to a JSON object which can be returned to the user.



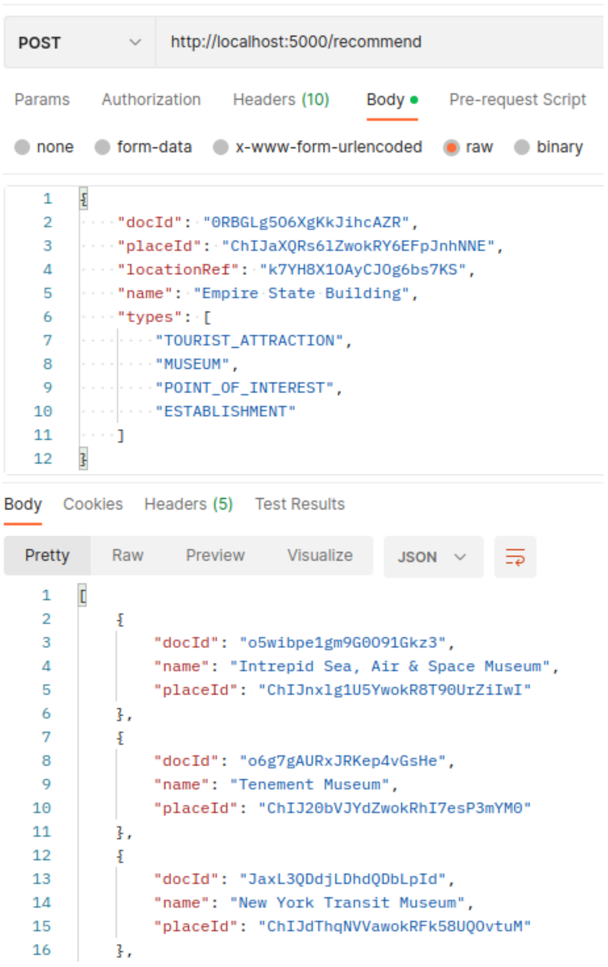
```python
##find similarity using cosine_similarity algorithm
print("Similarity")
similarity = cosine_similarity(vectors)
print(similarity[0])
print(similarity[0].shape)

##find recommendations
place_index = places_df[places_df['placeId'] == placeReceived.placeId].index[0]
distances = similarity[place_index]
places_list = sorted(list(enumerate(distances)), reverse = True, key = lambda x:x[1])[1:5]

##convert recommendations to json response
for place in places_list:
    jsonResponse = {}
    jsonResponse['docId'] = places_df.iloc[place[0]]['docId']
    jsonResponse['placeId'] = places_df.iloc[place[0]]['placeId']
    jsonResponse['name'] = places_df.iloc[place[0]]['name']

    ##remove the place received in the request from the response array
    if jsonResponse['docId'] != placeReceived.docId:
        jsonResponseArray.append(jsonResponse)

##return
return jsonResponseArray
```

This is the output from running the cosine similarity function



```
Similarity
[1.         0.81649658 0.81649658 0.70710678 0.81649658 0.
 0.53452248 0.         0.81649658 0.63245553 0.81649658 0.70710678
 0.70710678 0.63245553 0.81649658 0.63245553 0.57735027 0.70710678
 0.         0.70710678 0.70710678 0.         0.63245553 0.70710678
 0.70710678 0.81649658 0.70710678 0.70710678 0.63245553 0.
 0.57735027]
(31,)
```

We can generate a POST request on postman to test whether it works. We can see from the data below that it is returning an array of places.

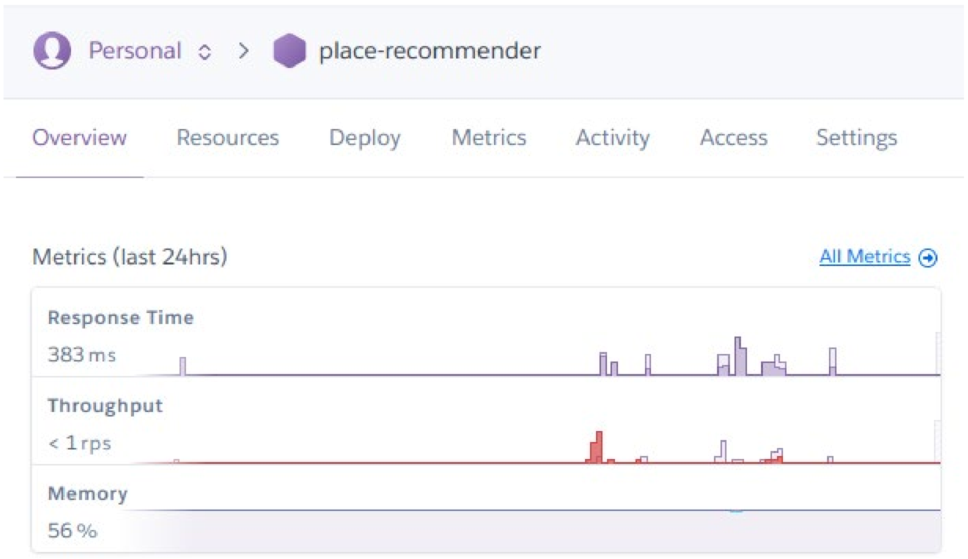


POST http://localhost:5000/recommend

```
{
    "docId": "0RBGLg5O6XgKkJihcAZR",
    "placeId": "ChIJaXQRs6lZwokRY6EFpJnhNNE",
    "locationRef": "k7YH8X1OAyCJOg6bs7KS",
    "name": "Empire State Building",
    "types": [
        "TOURIST_ATTRACTION",
        "MUSEUM",
        "POINT_OF_INTEREST",
        "ESTABLISHMENT"
    ]
}
```

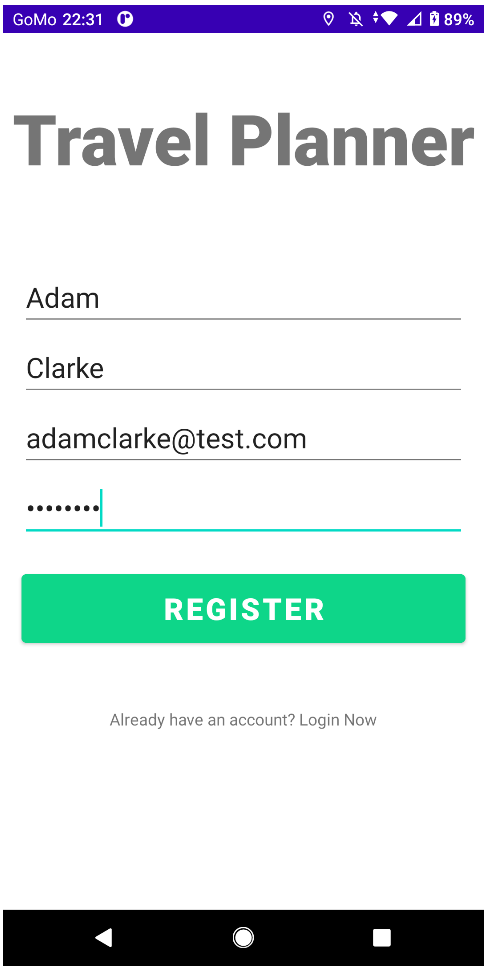
```
[
    {
        "docId": "o5wibpe1gm9G0O91Gkz3",
        "name": "Intrepid Sea, Air & Space Museum",
        "placeId": "ChIJnxlg1U5YwokR8T90UrZiIwI"
    },
    {
        "docId": "o6g7gAURxJRKep4vGsHe",
        "name": "Tenement Museum",
        "placeId": "ChIJ20bVJYdZwokRhI7esP3mYM0"
    },
    {
        "docId": "JaxL3QDdjLDhdQDbLpId",
        "name": "New York Transit Museum",
        "placeId": "ChIJdThqNVVawokRFk58UQOvtuM"
    },
```

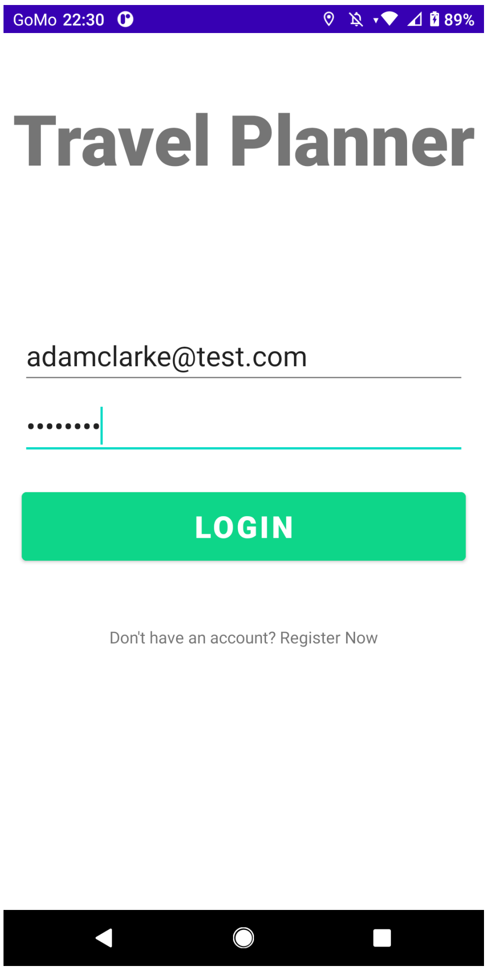
This application is also deployed on Heroku so that the Android Client can interoperate with the API.



2.4. Graphical User Interface (GUI)

Register and Login




GoMo 22:30 89%

Travel Planner

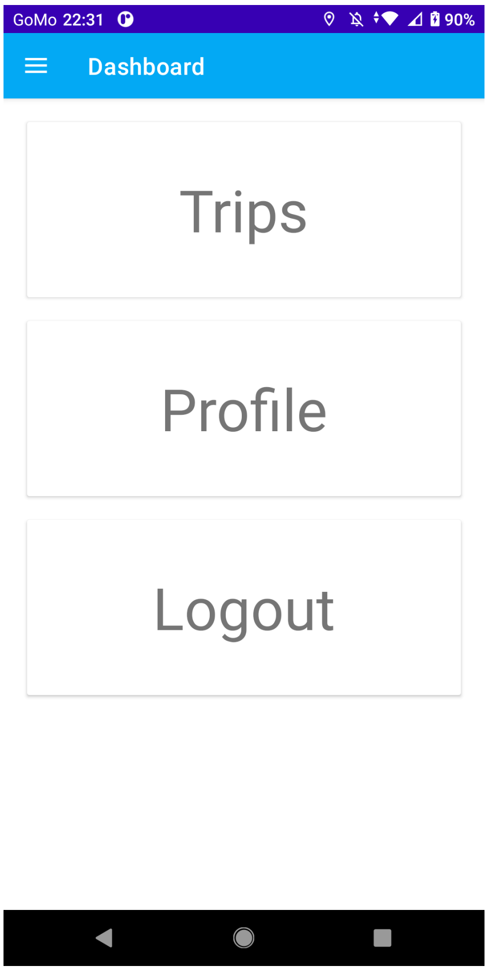
adamclarke@test.com

••••••••

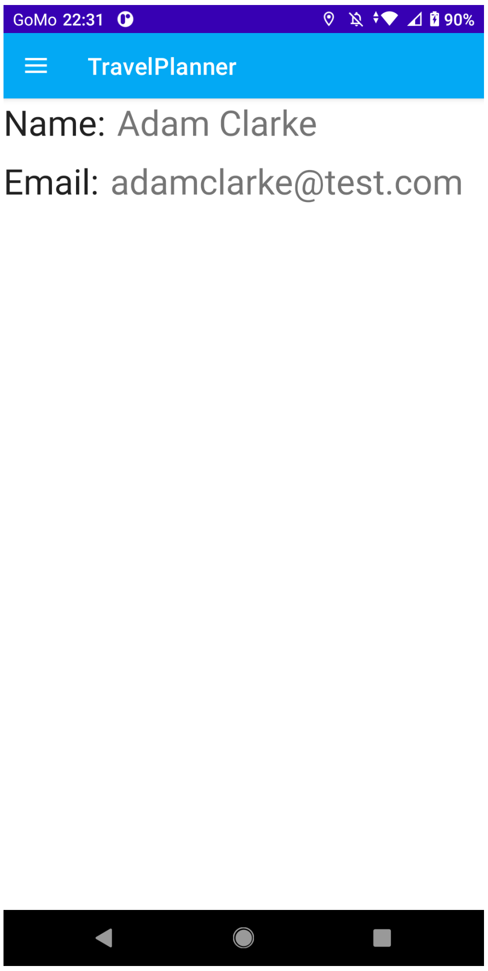
LOGIN

Don't have an account? Register Now

Dashboard



Profile

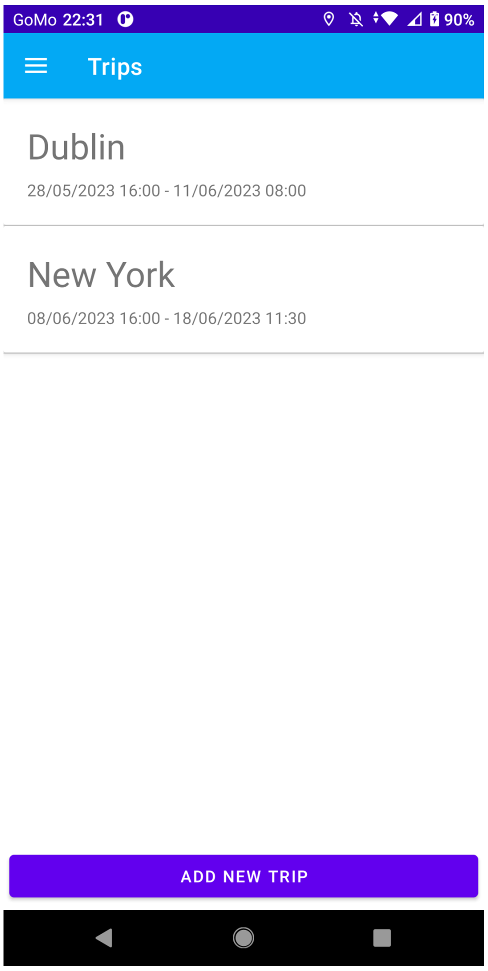


Name: Adam Clarke

Email: adamclarke@test.com

Trips List

User can click on list item to display places user has added to trip or click Add New Trip button to add a new trip


GoMo 22:31 90%

Trips

Dublin

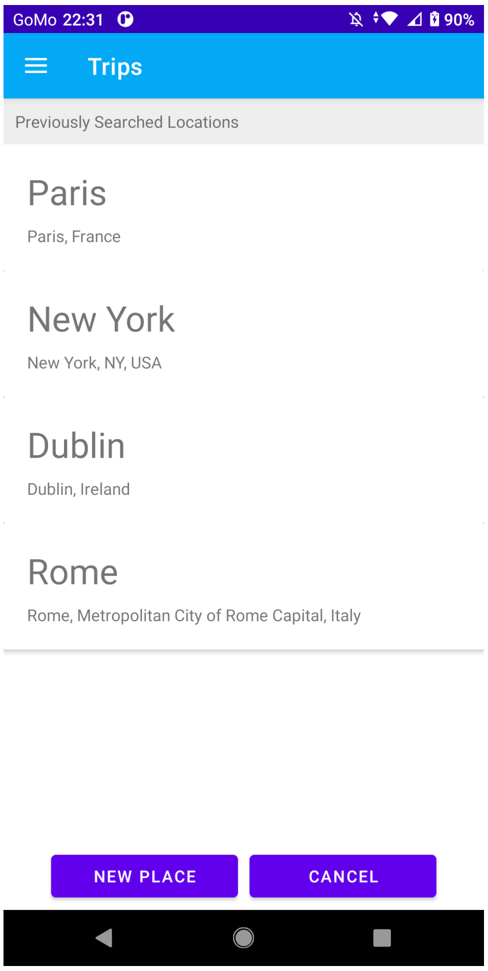
28/05/2023 16:00 - 11/06/2023 08:00

New York

08/06/2023 16:00 - 18/06/2023 11:30

New Trip

User Clicks on a location that they want to create a trip for



Previously Searched Locations

Paris
Paris, France

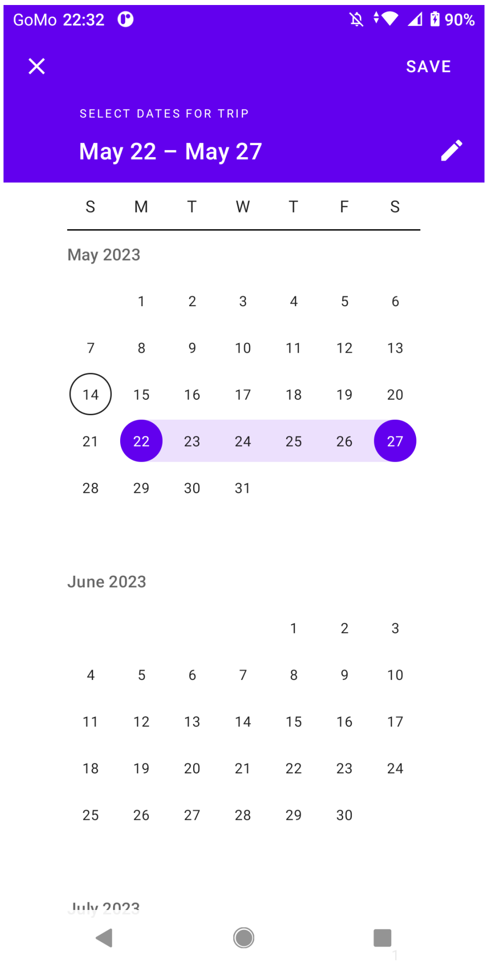
New York
New York, NY, USA

Dublin
Dublin, Ireland

Rome
Rome, Metropolitan City of Rome Capital, Italy

Select Trip Dates

A date range picker is display for the user to select the duration of the trip


GoMo 22:32 90%

SAVE

SELECT DATES FOR TRIP

May 22 – May 27

May 2023

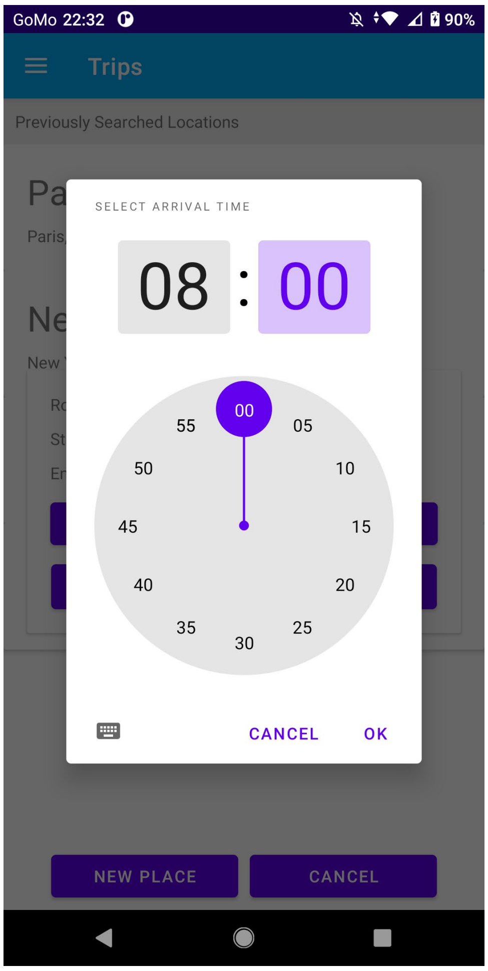
S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

June 2023

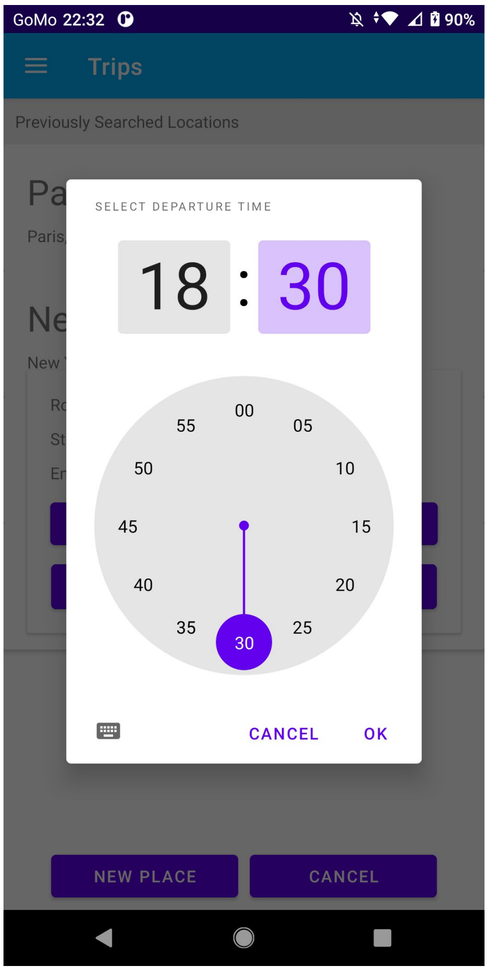
S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

July 2023

Select Arrival Time

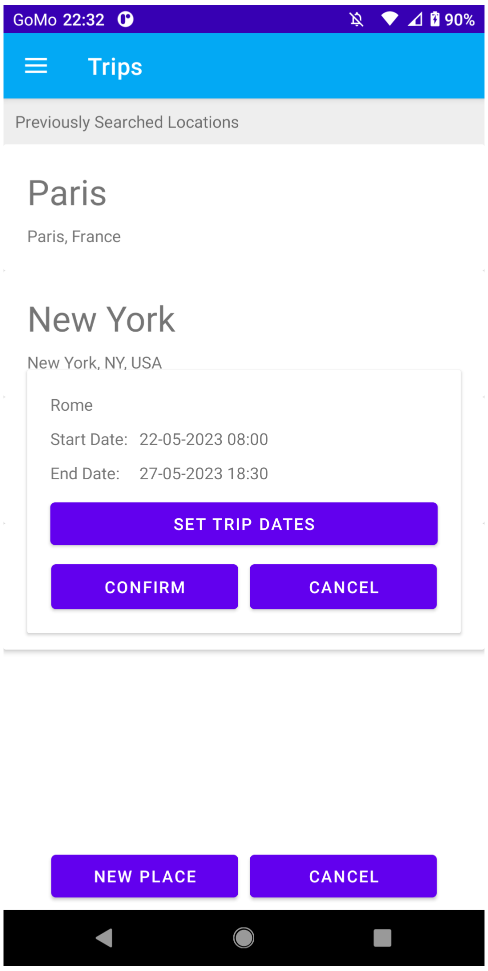


Select Departure Time



Confirm or Cancel Trip

User can confirm the details they provided or edit the dates and times. Changing the location requires the user to cancel and select another place from the list


GoMo 22:32 90%

Trips

Previously Searched Locations

Paris

Paris, France

New York

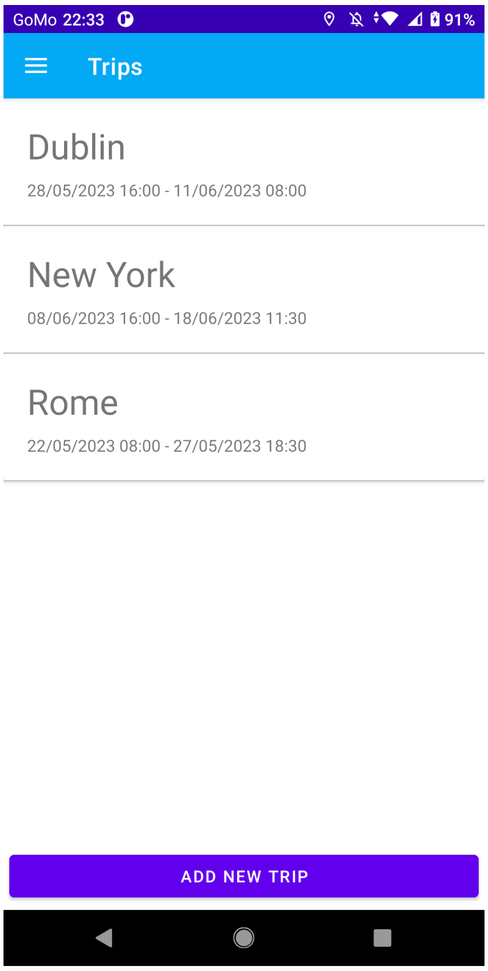
New York, NY, USA

Rome

Start Date: 22-05-2023 08:00

End Date: 27-05-2023 18:30

View New Trip in Updated List


GoMo 22:33 91%

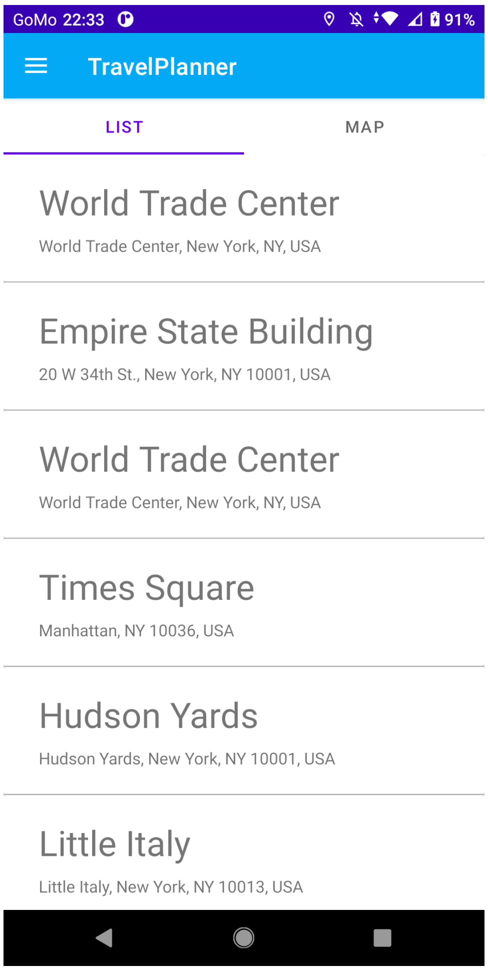
Trips

Dublin
28/05/2023 16:00 - 11/06/2023 08:00

New York
08/06/2023 16:00 - 18/06/2023 11:30

Rome
22/05/2023 08:00 - 27/05/2023 18:30

Select Trip From List to View Places


GoMo 22:33 91%

TravelPlanner

LIST MAP

World Trade Center
World Trade Center, New York, NY, USA

Empire State Building
20 W 34th St., New York, NY 10001, USA

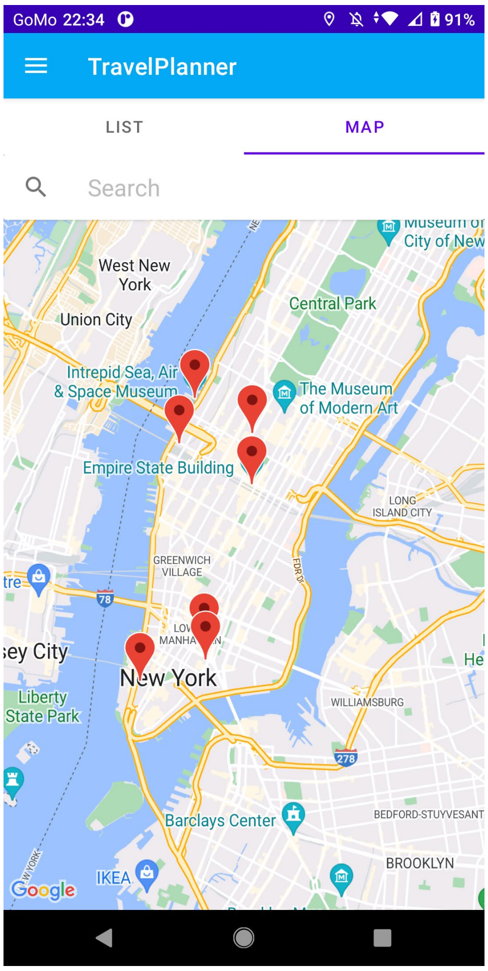
World Trade Center
World Trade Center, New York, NY, USA

Times Square
Manhattan, NY 10036, USA

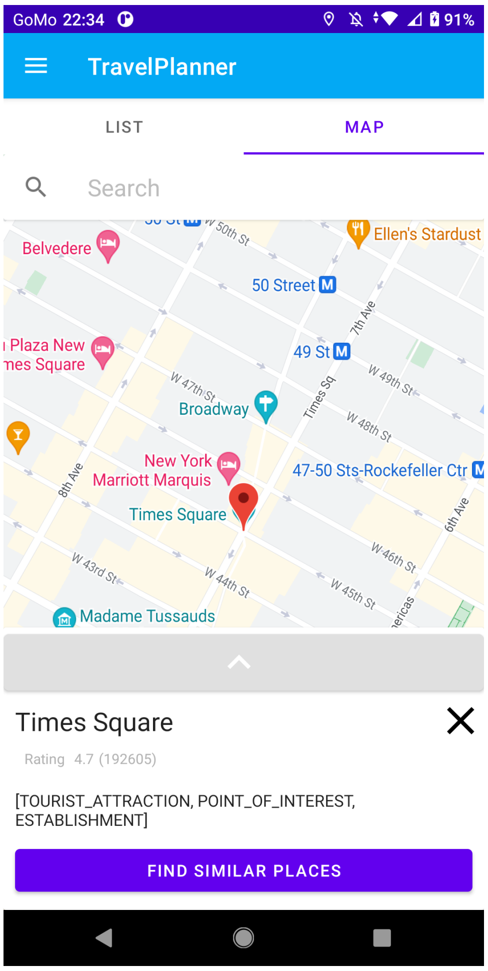
Hudson Yards
Hudson Yards, New York, NY 10001, USA

Little Italy
Little Italy, New York, NY 10013, USA

Select Map Tab to View Places on Map



Click On Place To View Place Details

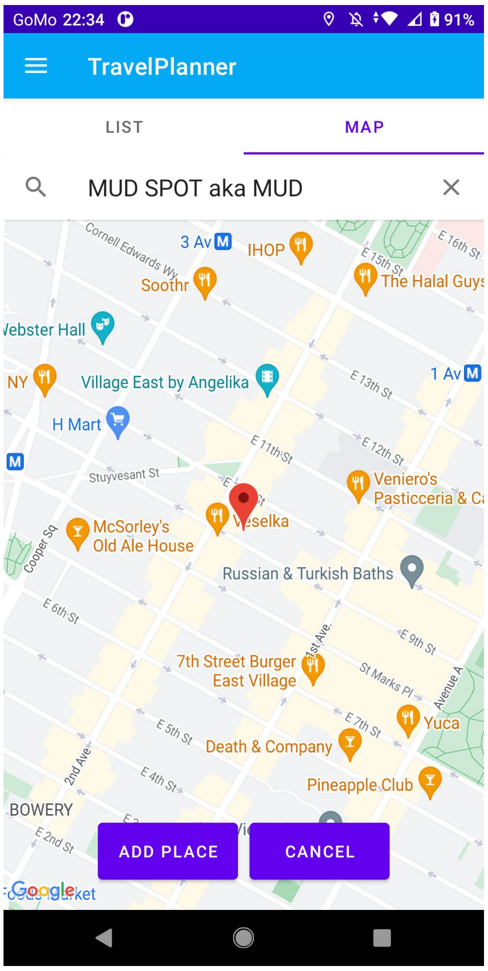


Times Square

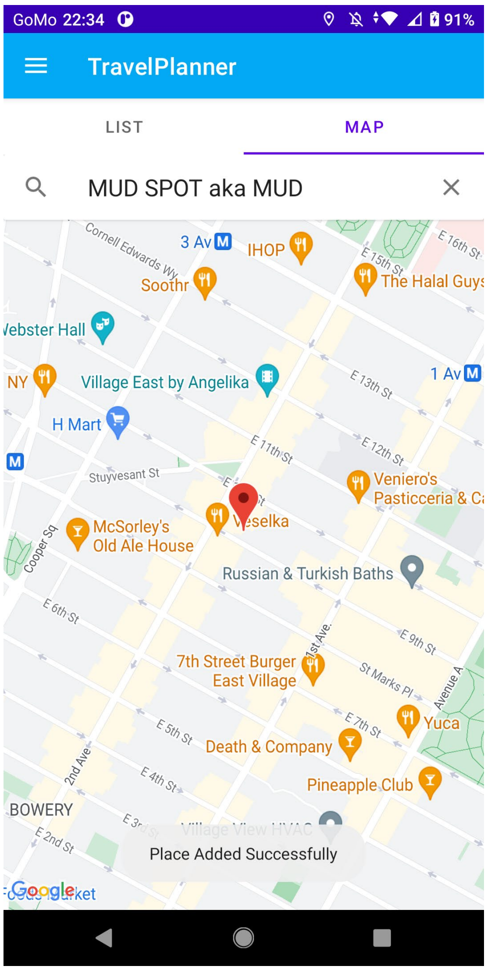
Rating 4.7 (192605)

[TOURIST_ATTRACTION, POINT_OF_INTEREST, ESTABLISHMENT]

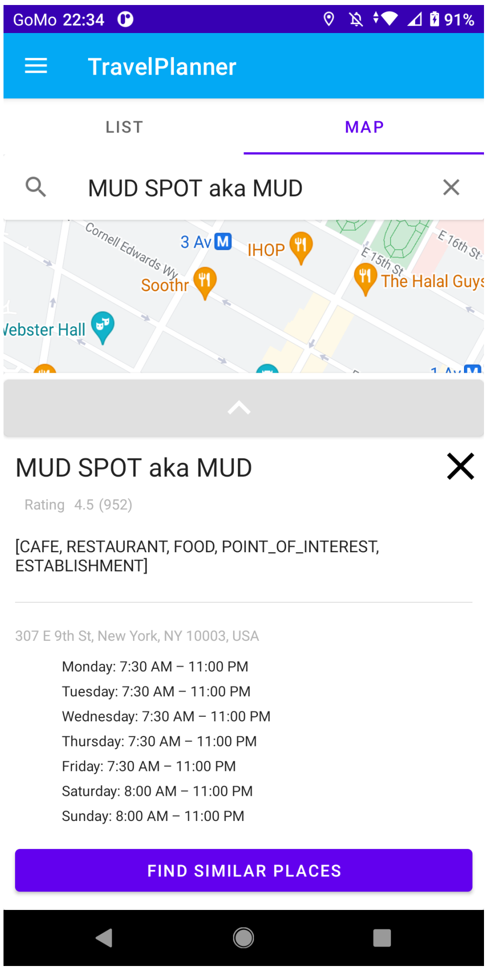
Search For Place Using Google Places API



Confirm Add Place



Click on Map Marker to View Details


GoMo 22:34 91%

TravelPlanner

LIST MAP

MUD SPOT aka MUD

MUD SPOT aka MUD

Rating 4.5 (952)

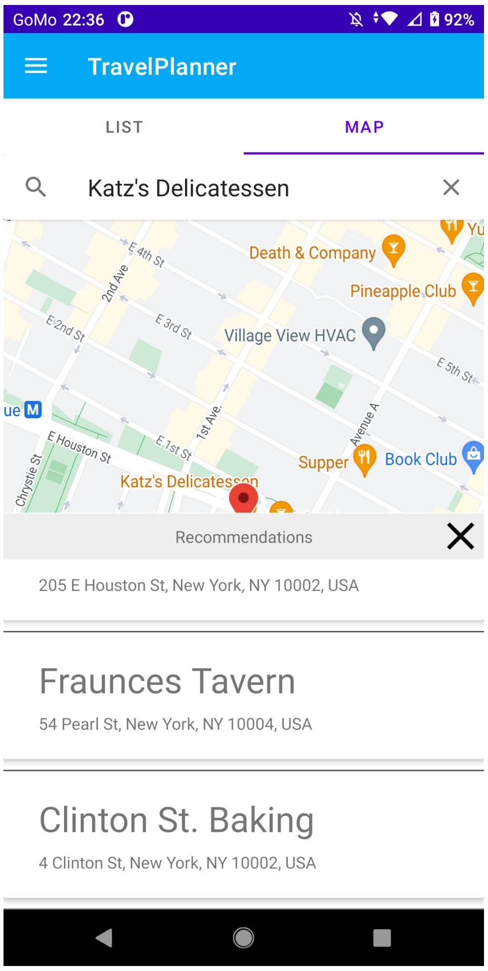
[CAFE, RESTAURANT, FOOD, POINT_OF_INTEREST, ESTABLISHMENT]

307 E 9th St, New York, NY 10003, USA

Monday: 7:30 AM – 11:00 PM
Tuesday: 7:30 AM – 11:00 PM
Wednesday: 7:30 AM – 11:00 PM
Thursday: 7:30 AM – 11:00 PM
Friday: 7:30 AM – 11:00 PM
Saturday: 8:00 AM – 11:00 PM
Sunday: 8:00 AM – 11:00 PM

FIND SIMILAR PLACES

Click Find Similar Places to get Recommended Places



Katz's Delicatessen

Recommendations

205 E Houston St, New York, NY 10002, USA

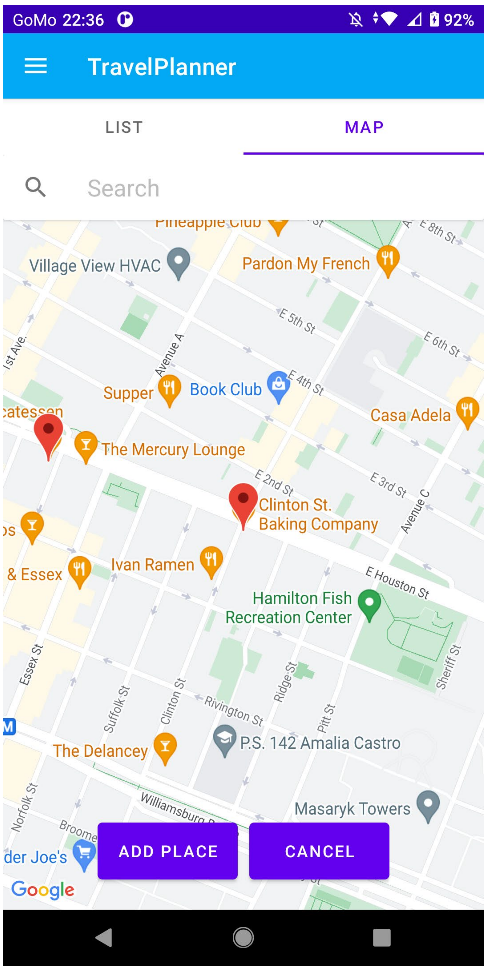
Fraunces Tavern

54 Pearl St, New York, NY 10004, USA

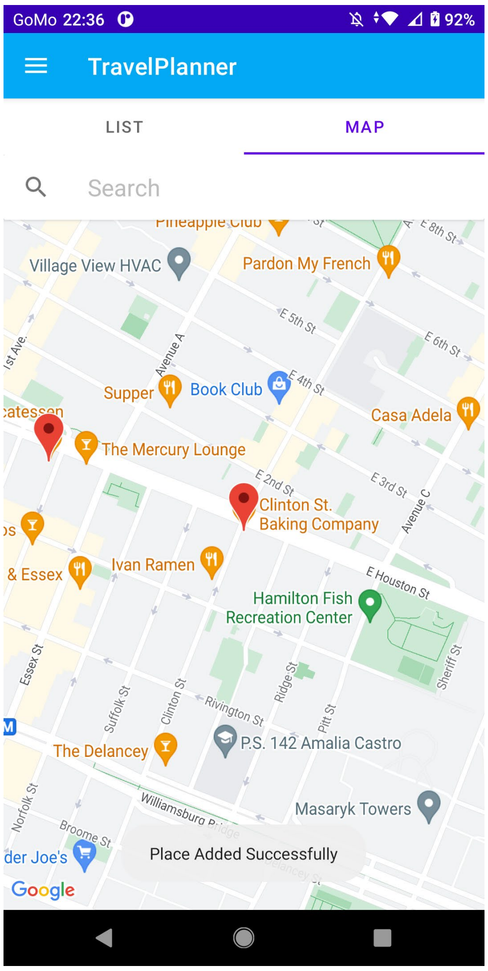
Clinton St. Baking

4 Clinton St, New York, NY 10002, USA

Click on Recommended Place to See Location



Click Add Place to Add Recommended Place to List



2.5. Testing

2.5.1. Register and Login

ID	Name	Assertions	Pass/ Fail	Notes
1.1	Register User	Assert that app dashboard is displayed Assert that profile activity displays name and email of current user	Pass Pass	
1.2	Leave fields empty in Registration Form	Assert that account will not be created Assert that an error message is displayed to user	Pass Pass	
1.3	Login with valid credentials	Assert that dashboard activity is launched Assert that profile activity displayed current users name and email	Pass Pass	

1.4	Login with invalid credentials	Assert that user will not be logged in Assert that an error message is displayed to the user	Pass Pass	
1.5	Logout	Assert that user has logged out Assert that LoginActvity is launched Assert that logging in as a new user displays their details and not signed out user Assert that a newly registered user does not have access to signed out users details	Pass Pass Pass Pass	

2.5.2. Add and Retrieve Trips

ID	Name	Assertions	Pass/ Fail	Notes
2.1	Retrieve Trips	Assert that user is logged in Assert that ‘TripsListFragment’ is loaded Assert that trips displayed matches the trips in the user's database collection	Pass Pass Pass	
2.2	Click on Trip	Assert that ‘PlacesListFragment’ is launched.	Pass	
2.3	Launch New Trip Fragment	Assert that ‘NewTripFragment’ is launched’ Assert that a List of locations are displayed Assert that a cancel button is displayed Assert that cancel button returns user to ‘TripsListFragment’	Pass Pass Pass Pass	
2.4	Select Location from List	Assert that clicking a location from list opens a pop-up window to add trip dates Assert that cancel button on pop-up closes the pop-up window	Pass Pass Pass	

		Assert that clicking confirm before entering trip dates displays a prompt to enter trip dates		
2.5	Set Trip Dates	Assert that clicking on set trips dates displays a date range selector Assert that user must enter two dates Assert that arrival date cannot be before departure date Assert that confirming valid dates displays a time selector window Assert that confirming arrival time displays another time selector window Assert that confirming departure times closes window Assert that selected dates and times are displayed	Pass Pass Pass Pass Pass Pass Pass	
2.6	Confirm Trip Added	Assert that TripsListFragment is loaded after confirming new trip Assert that the database is updated correctly Assert that new trip is displayed in list Assert that clicking on new list item launches PlacesListFragment	Pass Pass Fail Pass	Trips list i did not update but same method worked for updating places list

2.5.3. Add and Retrieve Places

ID	Name	Assertions	Pass/ Fail	Notes
3.1	Display Added Places	Assert that added places are displayed in PlacesListFragment Assert that number of places matches number in database Assert that place row item displays the name and address of the place	Pass	
3.2	Map View	Assert that clicking on the map tab launches PlacesMapFragment	Pass	

		Assert that Map Camera is set on trip location coordinates Assert that map markers are displayed to represent each place in the places list		
3.3	Map Markers	Assert that clicking a map marker moves the map camera to the location of the marker Assert that an info window for the place details is displayed. Assert that the details of this place match the record in the database Assert that the window can be expanded to view the place address and opening hours is they exist Assert that a recommend button is visible at the bottom of the card Assert that the window can be closed Assert that the info changes when a new map marker is selected	Pass	
3.4	Search Places	Assert that clicking the search bar launches the google places autocomplete fragment Assert that entering information into search bar displays a list from the API Assert that selecting a place move the map camera to the location of the place and displays a map marker Assert that an add place button and a cancel button are displayed at the bottom of the view Assert that the cancel button removes the map marker and hides the buttons Assert that the add button adds a place to the database and displays a success message to user	Pass	

		Assert that clicking on the Places List Tab displays the new place in the list Assert that clicking on the new place map marker displays an info window with the correct information		

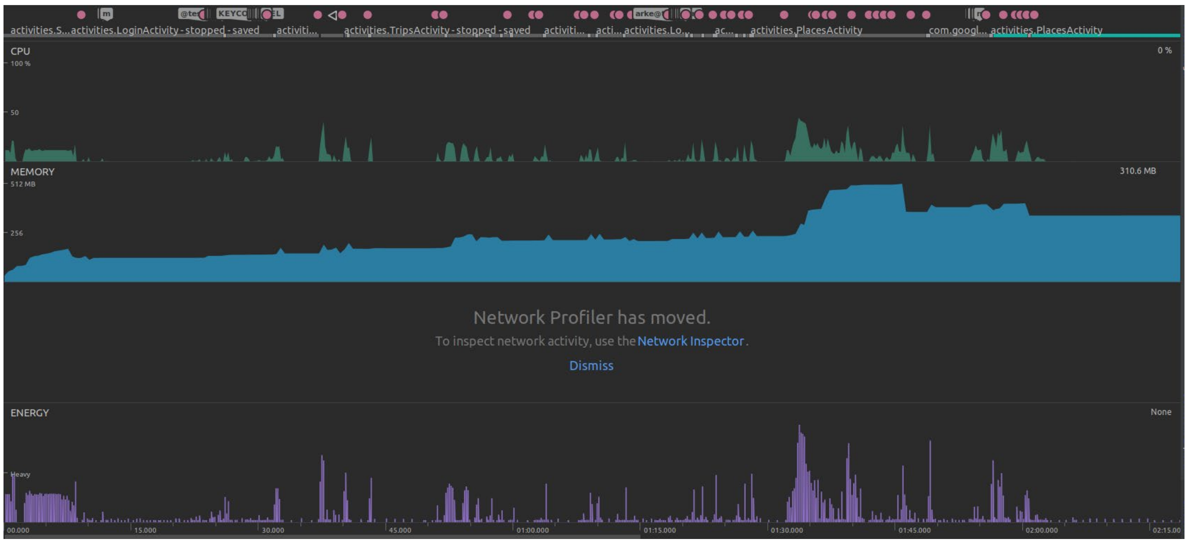
2.5.4. Recommendation API

ID	Name	Assertions	Pass/ Fail	Notes
4.1	Request Recommendation from API	Assert that the API returns data when provided a place from the client Assert that there are 3 recommendations displayed in a recycler view	Pass	
4.2	Add Recommendation	Assert that clicking a recommendation from the list moves the map camera to the location of the place with a map marker Assert that the list closes and add button and cancel button are displayed Assert that the cancel button removes the map marker and the buttons Assert that the add button adds the place to the database and is updated in the list and map Assert that clicking a recommendation that the user has already added to their list displays a message stating 'Place Already Added'	Pass	
4.3	API Not Available	Assert that clicking the find recommendation button displays a message to the user	Pass	

2.6. Evaluation

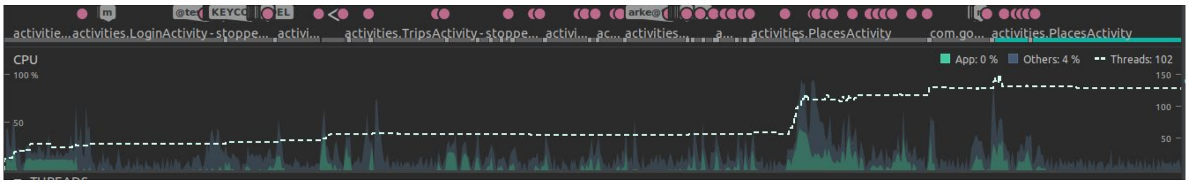
Profile

This is the result from running a profiler while using the applcation.


activities.S...activities.LoginActivity - stopped - saved
activities.TripsActivity - stopped - saved
activities.PlacesActivity
CPU
100 %
50
0 %
MEMORY
512 MB
256
310.6 MB
Network Profiler has moved.
To inspect network activity, use the Network Inspector.
Dismiss
ENERGY
Heavy
None
00.000
15.000
30.000
45.000
01:00.000
01:15.000
01:30.000
01:45.000
02:00.000
02:15.00

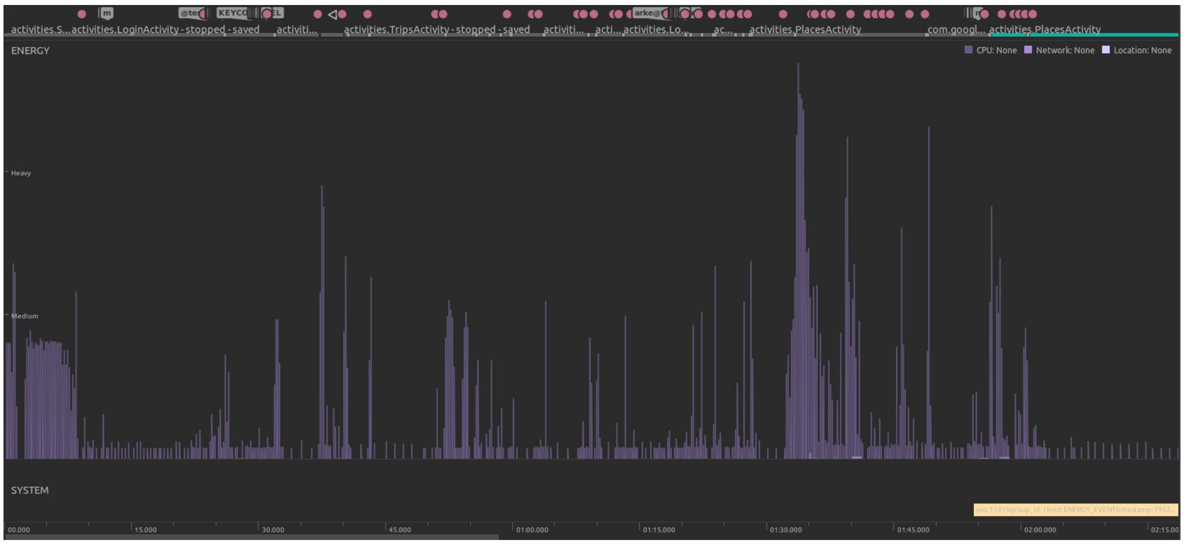
CPU

CPU usage was steady until the PlacesActivity was launched and then it spiked.


activitie...activities.LoginActivity - stoppe...
activities.TripsActivity - stoppe...
activities.PlacesActivity
CPU
100 %
50
App: 0 %
Others: 4 %
Threads: 102
150
100
50

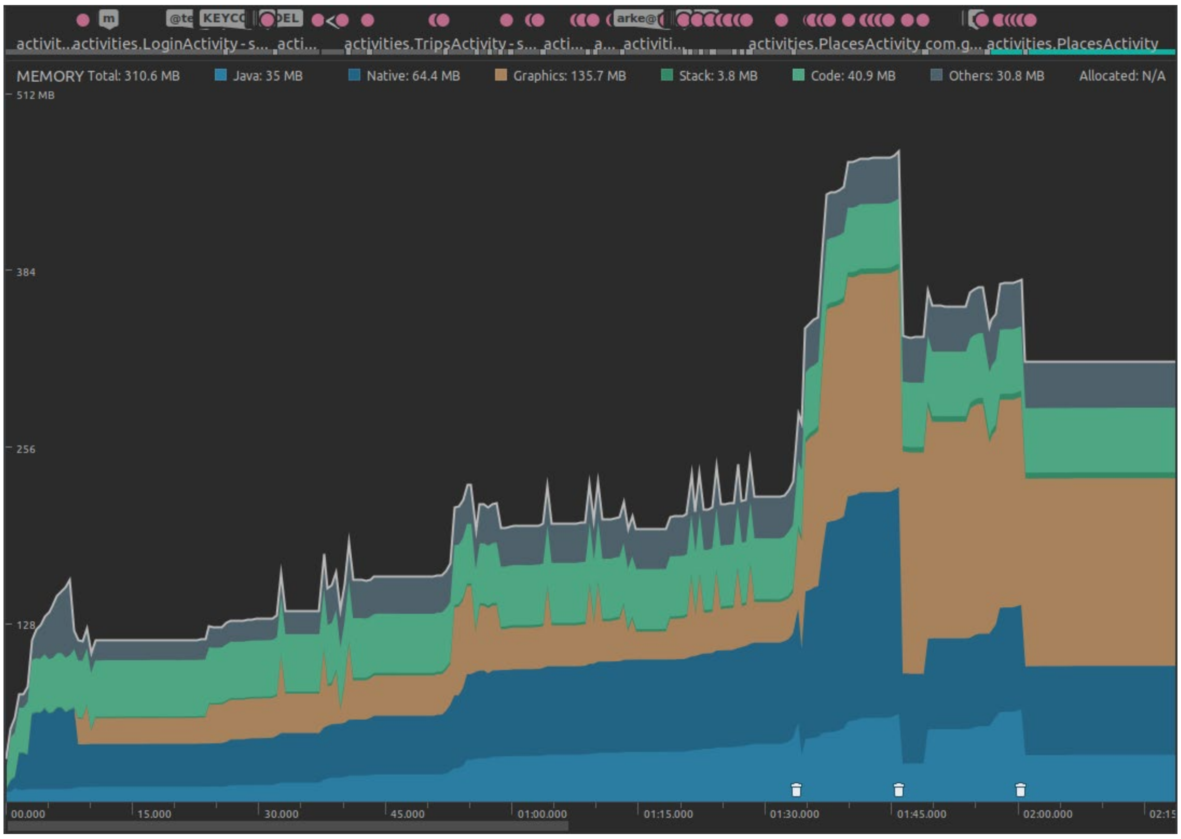
Energy

Energy usage spiked when different activities and fragments were launched or other operations were performed.



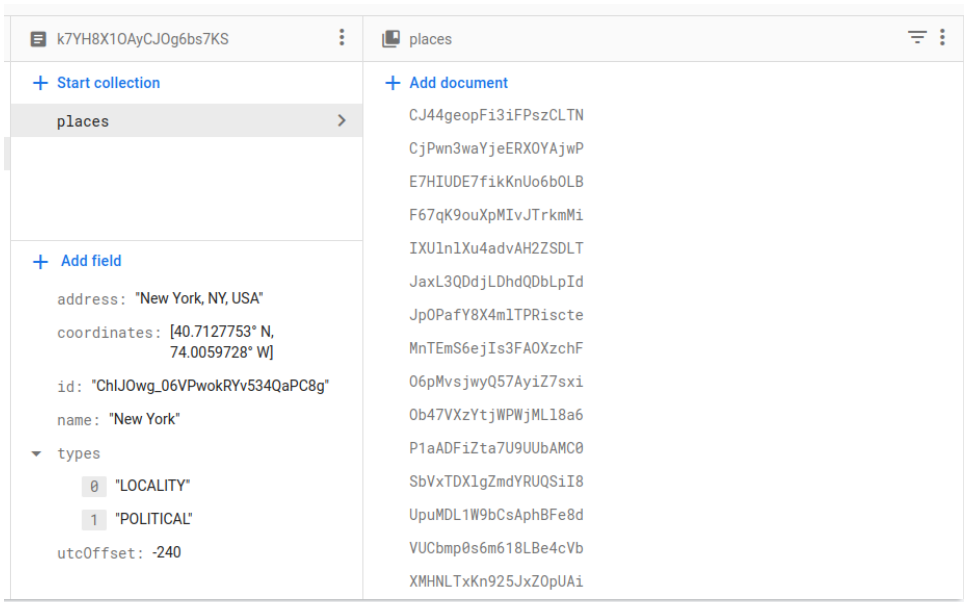
Memory

Memory usage also spiked when PlacesActivity was launched. This correlates with the loading of Google Maps and the loading of places from Firebase which takes more memory.



API

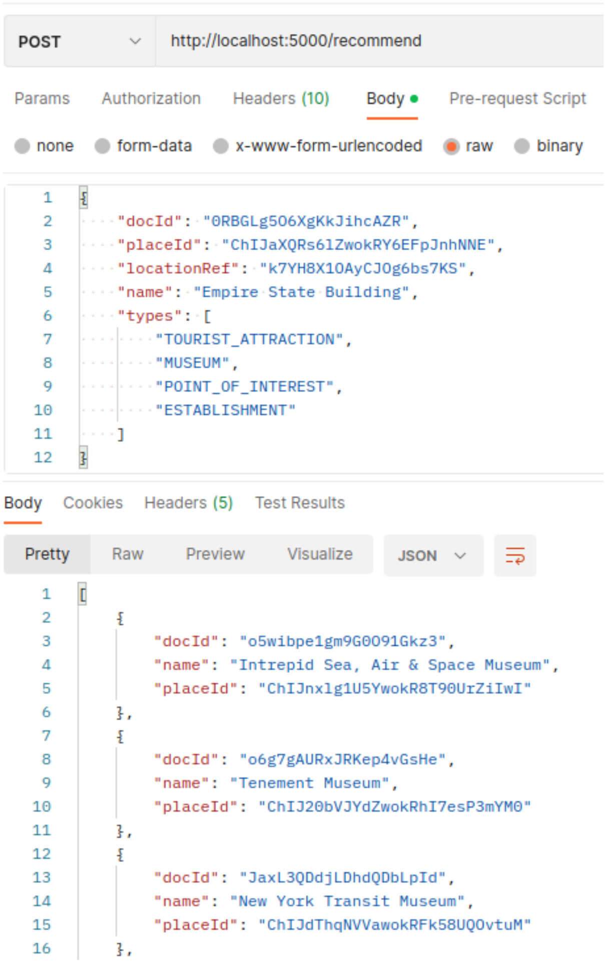
To evaluate the API, 5 users were added and each searched for 10 places. This allowed the Locations collection to be populated with places. This was done under New York so all places searched were stored in that collection. This meant the API had a sample to return more accurate results in testing.



address: "New York, NY, USA"
coordinates: [40.7127753° N, 74.0059728° W]
id: "ChIJOwg_06VPwokRYv534QaPC8g"
name: "New York"
types
0 "LOCALITY"
1 "POLITICAL"
utcOffset: -240

CJ44geopFi3iFPszCLTN
CjPwn3waYjeERXOYAjwP
E7HIUDE7fikKnUo6bOLB
F67qK9ouXpMIvJTrkmMi
IXUlnlXu4advAH2ZSDLT
JaxL3QDdjLDhdQDbLpId
JpOPafY8X4mlTPRiscte
MnTEmS6ejIs3FAOXzchF
O6pMvsjwyQ57AyiZ7sxi
Ob47VXzYtjWPWjML18a6
P1aADFiZta7U9UUbAMC0
SbVxTDXlgZmdYRUQSiI8
UpuMDL1W9bCsAphBFe8d
VUCbmp0s6m618LBe4cVb
XMHNLTxKn925JxZOpUAi

This allowed us to evaluate whether appropriate recommendations were being made

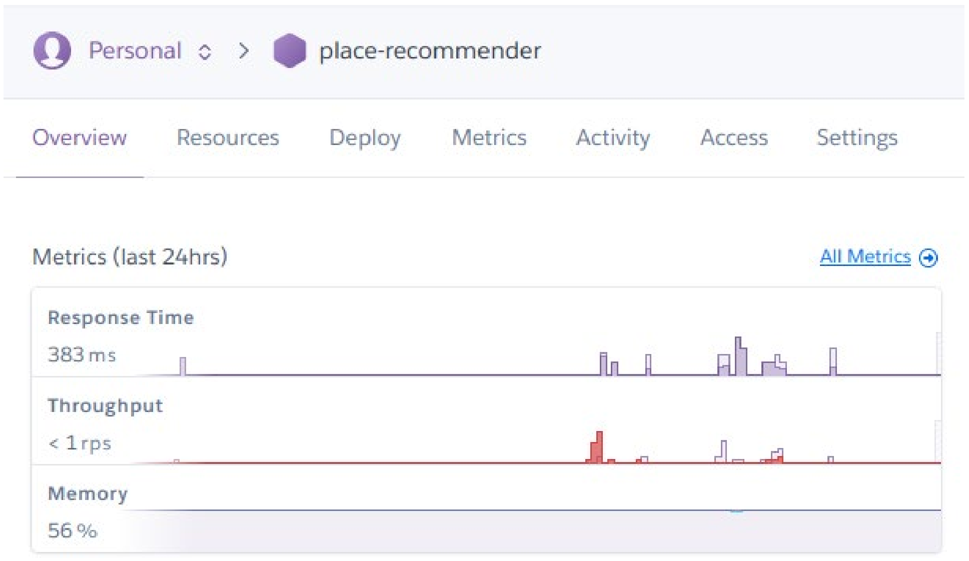


http://localhost:5000/recommend

```
{
    "docId": "0RBGLg5O6XgKkJihcAZR",
    "placeId": "ChIJaXQRs6lZwokRY6EFpJnhNNE",
    "locationRef": "k7YH8X1OAyCJOg6bs7KS",
    "name": "Empire State Building",
    "types": [
        "TOURIST_ATTRACTION",
        "MUSEUM",
        "POINT_OF_INTEREST",
        "ESTABLISHMENT"
    ]
}
```

```
[
    {
        "docId": "o5wibpe1gm9G0091Gkz3",
        "name": "Intrepid Sea, Air & Space Museum",
        "placeId": "ChIJnxlg1U5YwokR8T90UrZiIwI"
    },
    {
        "docId": "o6g7gAURxJRKep4vGsHe",
        "name": "Tenement Museum",
        "placeId": "ChIJ20bVJYdZwokRhI7esP3mYM0"
    },
    {
        "docId": "JaxL3QDdjLDhdQDbLpId",
        "name": "New York Transit Museum",
        "placeId": "ChIJdThqNVVawokRFk58UQOvtuM"
    },
```

The API is deployed on Heroku which provides some metrics about the response times, throughput and memory usage of the API.



3.0 Conclusions

The main strength of the project is that it takes care of a lot of processes for the user. When a user adds a place to their trip, it takes all relevant data from the API, so the user does not have to input it. The user can also get recommended place based on other users searches. This helps the user discover new places more easily and save them if they are interested in visiting.

The main weakness of the project is that it is less customisable than other similar apps. However, the main aim of this idea is to streamline the process of planning an itinerary. In developing this project, it would have been beneficial to reduce the scope in the beginning. A lot of difficulties were faces in the implementation which meant features had to scrapped for time or to work on other features which led to a less complete application and project.

It also would have been beneficial to research more about Android UI and Framework features. Android is so much different to any other programming tool I have used and has a steep learning curve. Knowing what features are possible to implement and what are not would have saved a lot of time and energy.

4.0 Further Development or Research

This project would be best integrated with another service such as TripAdvisor. Using their platform, the recommendation system for the user would be enhanced dramatically. It would also be useful to interface the apps review system with theirs which would make the app more powerful and useful.

There are other features that could be useful in the app such as directions to places. It could also be beneficial to compute the distance between when finding recommendations to find places close to where the user is staying or what their current location is.

If starting from scratch, I would likely opt for using Jetpack Compose for the UI instead of default Android. Default Android has a lot of excess code which makes it difficult to debug and read. Jetpack Compose is an updated way of implementing Android UIs with less code.

5.0 References

[1] Emma Jhonson, ‘Kotlin vs Java: Which is the Best Choice for Android App Development?’, *Medium*, Oct. 26, 2021. https://medium.com/javarevisited/kotlin-vs-java-which-is-the-best-choice-for-android-app-development-7c9fc782d2c9 (accessed Dec. 20, 2022).

[2] Google, ‘ Cloud Firestore’, *Google Cloud*, Dec. 15, 2022. https://firebase.google.com/docs/firestore/ (accessed Dec. 20, 2022).

[3] MongoDB, ‘NoSQL vs. SQL Databases’, *MongoDB*, 2022. https://www.mongodb.com/nosql-explained/nosql-vs-sql (accessed Dec. 20, 2022).

[4] Google, ‘Place Autocomplete’, *developers.google*. https://developers.google.com/maps/documentation/places/android-sdk/autocomplete#bias_results_to_a_specific_region (accessed May 14, 2023).

[5] R. Pandey, ‘Kotlin Coroutines: Getting Started in Android’, *Medium*, Jul. 22, 2020.

[6] Section.io, ‘How to Make HTTP Requests With Ktor-Client in Android’, *section.io*, Dec. 16, 2021. https://www.section.io/engineering-education/making-http-requests-with-ktor-in-android/ (accessed May 14, 2023).

[7] Python Basics, ‘Flask REST API Tutorial’, *pythonbasics.org*. https://pythonbasics.org/flask-rest-api/ (accessed May 14, 2023).

[8] scikit-learn, ‘sklearn.feature_extraction.text.CountVectorizer’, *scikit-learn.org*. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html (accessed May 14, 2023).

[9] Verma Khushali, ‘Using CountVectorizer to Extracting Features from Text’, *GeeksForGeeks*, Jul. 07, 2022. https://www.geeksforgeeks.org/using-countvectorizer-to-extracting-features-from-text/ (accessed May 14, 2023).

[10] S. Prabhakaran, ‘Cosine Similarity – Understanding the math and how it works (with python codes)’, *Machine Learning Plus*, Oct. 22, 2018. https://www.machinelearningplus.com/nlp/cosine-similarity/ (accessed May 14, 2023).

[11] scikit-learn, ‘sklearn.metrics.pairwise.cosine_similarity’, *scikit-learn.org*. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html (accessed May 14, 2023).

6.0 Appendices

6.1. Project Proposal

Objectives

The aim of this project is to create and Android Application that allows the user to map out a travel itinerary. The focus of this application is on locating places of interest in cities which can be searched for by their place name and their coordinates stored and visualised on a map. The app can then generate an itinerary for the user based on the list of places and their location.

The user can also interact with the map markers to see more information about the place such as address, phone number of business etc. They can also filter places on their list by their type. For example, filtering by restaurant if the user is looking for a place to eat.

The app will generate an itinerary for the user which they can edit to suit their specific requirements. When the user opens the app, it can request the users geolocation and determine if they are at a specific destination. If the user does not want their location tracked, they can manually tell the app if they went to a place or not. The user can confirm whether they went to a place and the app will ask them to rate the location. We can use this information to provide recommendations for future users based on the popularity of a place.

It should also be able to recommend places based on the type of places you are searching for and where other users have visited. For example, if you add a cafe to your list, it could recommend cafes that other users have visited and rated. User interaction with the application should enhance the recommendations of other users.

Overall, the app is designed to make it easier to plan a trip by adding places and place details efficiently and getting the app to generate an itinerary based on the location of places on your list.

Background

The idea for this project came from my own experience trying to plan trips in cities. Before a trip, I like to record places of interest in a notes app and look them up on google maps to find directions. What often ends up happening is I visit a place and not realise that there is another place nearby that I had on my list. The aim of this application is to record all the places and have them visualised on a map. This allows the user to make more informed decisions and make the most out of their limited time on their trip. I also want the app to do more for me by automatically generating an itinerary.

The Google Maps API will be used to provide map functionality. The Google Places API will allow us to search for places on the map.

Since this is an android application, it will be developed in Android Studio using Kotlin as the primary language. We will use this to implement app functionality and all algorithms required such as itinerary generation.

The application will also require a database to store place details and user accounts. The storage will be cloud based to provide security and reduce reliance of device storage.

State of the Art

There are travel websites with map functions such as TripAdvisor. My application acts more like a more functional notes app for tourism. It will not have a method for discovering new places but, requires the user to research places outside of the app. The user can then search for a place by its name, and it will store its location and details for personal reference. It will also function as a direction finder, help the user estimate travel time, and navigate the most efficient route to visit multiple places in a day.

Triplt and Trip Case are both travel itinerary planners with similar features. They allow the user to input a travel destination and search for locations by category such as a restaurant or an attraction. It also provides directions and uses map markers for locations. I have some criticisms of these apps which I would like to address in my own. Finding and adding a place is a slow process with these apps. The user must choose the category they are searching for such as restaurant or attraction before inputting the name of the place. The user then inputs more details such as the time they would like to arrive at and leave the place. This method is slow and does not suit my requirements for a travel app. I want to use the Google Places API to assume details about the place based on the information the search results return. The details can be edited by the user after if needed. This way the workflow is faster and requires less user input.

I also want the application to identify areas where there are a lot of places on the list and help design an itinerary based on these locations which these apps do not accomplish. The aim of this is to increase the efficiency of planning a trip and reducing the amount of user input which can be a drawback for using these apps.

Technical Approach

The project would be best served by working with an iterative approach using test driven development. There are essential features to the project such as the map, geolocation and place search functionality. Then there are desired features such as the directions and travel time estimates. These features require a basic set of functionalities to work. By working iteratively, more essential features can be integrated and tested to make a working app. The more complex tasks can then be integrated into the working app more easily. If a feature doesn't work as intended, the idea can be adapted or scrapped but, the application should still function.

Functional Requirements

- Application can display a map with Geolocation of the user.
- User can specify a place they are travelling to and where they are staying either by searching for address or using a movable map marker.
- User can search for a place and store its details and location. The location is then displayed on a map.

- User can create an account using a username and password which tracks their list of places and place details on a database.
- User can edit place details and delete a place.
- User can find directions and get time estimates for travel between locations.
- Application will generate a travel itinerary based on the users list of places and each locations proximity to each other.
- User can change account details such as username and password.
- The generated travel itinerary must account for certain factors such as the opening hours of the business and any input from the user regarding a certain visit.

Non-Functional Requirements

Milestones

1. Basic map functionality with the ability to search for a place and have return relevant information which we can use to display a map marker and place details.
2. Database configuration with ability to register users and display their personal list of places. Includes the ability for users to edit and delete places and place details.
3. Implementation of directions and the algorithm to generate a travel itinerary.

Technical Details

The programming language I would like to use is Kotlin. Kotlin is considered the De Facto language for Android Applications. It is interoperable with Java and is built on the JVM. It can be compiled into Native which is used for running IOS applications. It also supports Coroutines which Java does not. Coroutines are beneficial for Android applications for asynchronous calls. Java supports multiple threads, but they create complexity and use more memory. Coroutines work on a single thread and are easier to manage.

The Google Maps API will be used to provide map functionality for the app. The Google Places API will be used to search for places which the user can store for their trip.

Itinerary generation will involve an algorithm which will use the place location as input and try to plan activities based on their proximity to each other. It will also have to account for opening hours and if the user specifies a time slot they must visit somewhere for example, a restaurant booking. The user should also be able to customise the generated itinerary if it does not suit their needs. We can use Firebase to run a cloud based backend to run the algorithm

Firebase will be used for the database. This is because it is designed to work with Android applications to provide authentication and security. This is important when storing users' location data which must be secure. It is also hosted on the cloud which reduces storage space on the user's device. It also uses NoSQL with JSON syntax which makes it easily compatible with the format of data we receive from the API. It also works with Google Could Storage which we will be using for the Places API.

Special Resources Required

Project Plan

1. **Basic Map Functionality:**
    - Setup a basic Android Studio template with a map using the MapBox SDK for Android.
    - Implement a geolocation function to locate the user's device.
2. **Google Places Api Configuration:**
    - Setup application to process API data from place search.
    - Design a search bar to input a place name and have it return coordinates.
3. **Adding and Displaying a Place:**
    - Create a screen to add a place. This involves a search bar to input a place name.
    - When the API returns the place information, there will be a confirmation screen which allows the user to confirm and edit details provided by the API.
    - The place maker is then displayed on the map. The user can click on the place marker for more place info.
4. **Database setup:**
    - Configure Firebase to store place details. Database should be set up to handle user accounts.
5. **User Accounts and Login:**
    - Design a sign up and login screen using a username and password.
    - User can associate their itinerary with their account.
    - Implement account security measures.
    - Design screen to edit account details
6. **Update and Delete a Place:**
    - Places stored in user database can be updated and deleted
    - Design screen to manage updating place details
7. **Directions:**
    - Add MapBox Navigation API from the SDK to the Android Project.
    - Test direction response between two coordinates.
    - Implement time estimates for walking
8. **Travel Itinerary Algorithm:**
    - Research algorithms for sorting a searching that would help find clusters of places with nearby proximity.
    - Implement and test in code
    - Edit algorithm to help design an itinerary for the user based on places that are clustered.
    - Refine algorithm to account for other factors such as average time spent at location, time to travel, if user has timed booking for location, opening hours and times which may correlate with dining such as a placing a cafe at lunch time for example.

9. **Review System**
    - Create screen for reviewing places
    - Create notification system to ask users to review
    - Design logic to track user engagement for recommendations

Testing

The challenge for testing this application is simulating enough user engagement to allow the algorithm to provide user recommendations. This will be part of system testing and will require creating several user accounts with both common and unique searches to provide recommendations to users on multiple categories. This system is dependent on the app having user accounts implemented.

<u>Functional Testing:</u>

- **Unit Testing:** Each phase set out in the project plan will have to be unit tested to ensure it functions as its own entity. Each component generated in a phase will be unit tested before moving to the next phase. This is part of our iterative development approach.
- **Integration Testing:** Each unit will have to be tested with other components to ensure they function together. For example, we must ensure that the Google Places API works with the MapBox SDK. After each unit test, we integrate the component into the application and see how they work together.
- **System Testing:** System Testing can be done at each milestone. System Testing can only be done when there is a certain amount of functionality to work with. Each milestone is set when there is a level of functionality that makes the application usable. This will be important to compare our test results with the project requirements.

<u>Non-Functional Testing</u>

- **Security Testing:** We need to do some penetration testing for the database to ensure that user details are safe. We need to ensure there is proper validation for user passwords and that proper encryption is used.
- **Performance Testing:** The performance of the system needs to tested using load testing and stress testing. Load testing can be performed by seeing the limit of places that can be added by the user and how that affects performance. This could help define a limit for each user.
- **Usability Testing:** Some different perspectives could be useful here to ensure that the User Interface is usable. We also need to ensure that the system performs to a usable standard.
- **Compatibility Testing:** This will check how the system will work on multiple devices. We are limited to Android devices for this application but, how will it perform on older phones? Will it work on older versions of Android?

There are tools included in Android studio to help meet these testing requirements. Instrumented tests run on the android device to check UI functions. Local Tests run on the development machine and run unit tests on code.

6.2. Ethics Approval Application (only if required)

6.3. Reflective Journals

October

Student Name	Adam Regan
Student Number	x1940156
Course	BSc in Computing
Supervisor	Adriana Chis

Month: October

What?   Reflect on what has happened in your project this month?   This month I did a feasibility analysis on some project ideas. I researched some technical solutions to ideas before settling on my current idea of a travel planner Android application. I then went on to complete my project proposal. I spent some time refining the idea during this to settle on the features I wanted in the application. The app needed to be able to do something technically demanding. I came up with an idea for the itinerary generator based on a list of place searches to meet this requirement. I also set out a basic project plan with a Gannt chart and testing plan which will be refined over the course of the year.
So What?   Consider what that meant for your project progress. What were your successes? What challenges still remain?   I spent a lot of time on my proposal ensuring I had a refined idea that I could execute. I have a plan which should be achievable to follow and is detailed enough. I also settled on the technologies I'm going to use for the programming language and the database. I think the main challenge will be designing the algorithm to generate and itinerary. I need to research different sorting and searching algorithms and figure out how to apply them to my application. I also need to figure out if the Google Places API will work with MapBox.

Now What? What can you do to address outstanding challenges? I need to do some research into different searching and sorting algorithms and try to fit them into the application. I also need to test the Google API and MapBox API to ensure they are compatible. I will have to set up the accounts and test in Android Studio.	
Student Signature	Adam Regan

November

Student Name	Adam Regan
Student Number	x1940156
Course	BSc in Computing
Supervisor	Adriana Chis

Month: November

What?
Reflect on what has happened in your project this month? This month I had to redefine some of my project requirements to meet complexity standards set out in the marking rubric. My supervisor suggested implementing a recommendation algorithm based on user input into the system. This algorithm will help create personalised recommendations for the user based on the type of places they search for. This feature should help bring the complexity of the application to the desired standard. All the functional requirements have been set out describing the flows and providing use case diagrams for each. There are 14 use cases in total. This gives me a clearer picture of how to program the application.
So What? Consider what that meant for your project progress. What were your successes? What challenges still remain? Setting out all the functional requirements was one of the larger sections of the midpoint report. It also helps me understand the system i am trying to create. The next step is to describe the non-functional requirements of the system and to plan the database. I would also like to start building the app at a basic level. The screen designs and the database are the first areas I want to focus on.
Now What? What can you do to address outstanding challenges? Firstly, the application needs to get started as soon as possible to start building some of the features of the app for the midpoint. The database plan is also going to important. I would like to have that complete before I start configuring the database.

Student Signature	Adam Regan

December

Student Name	Adam Regan
Student Number	x1940156
Course	BSc in Computing
Supervisor	Adriana Chis

Month: December

What? Reflect on what has happened in your project this month? This month is when the midpoint submission is due. A lot of work has been done on the implementation of the project. So far, I have done a lot of work setting up firebase for user authentication, implementing the autocomplete search, and a map with places marked on it.
So What? Consider what that meant for your project progress. What were your successes? What challenges still remain? Getting some primary features implemented is an important step. There are some challenges with my understanding of Android Studio which need to be addressed. Certain UI features were difficult to implement, and some caused errors prior to my midpoint submission and had to be removed for the demonstration.

Now What? What can you do to address outstanding challenges? To address this issue, I need to educate myself on Android studio by reading the documentation and studying some app implementations to try and understand them.	
Student Signature	Adam Regan

6.4. Other materials used

Any other reference material used in the project for example evaluation surveys etc.