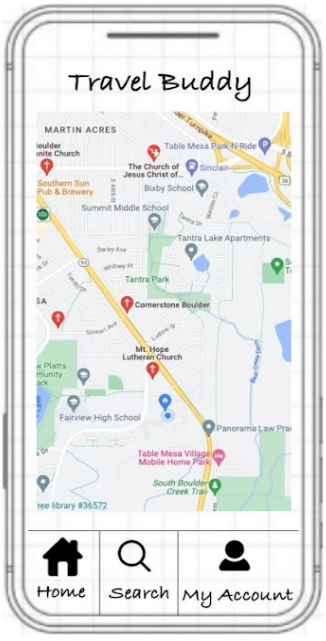
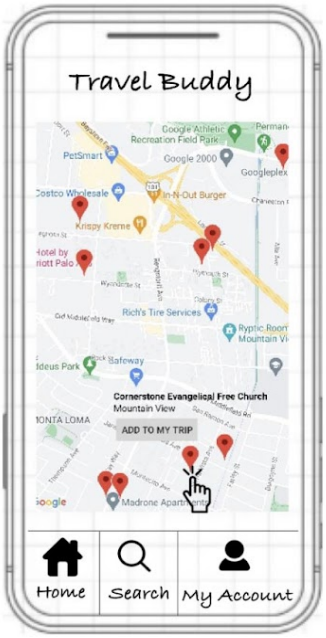
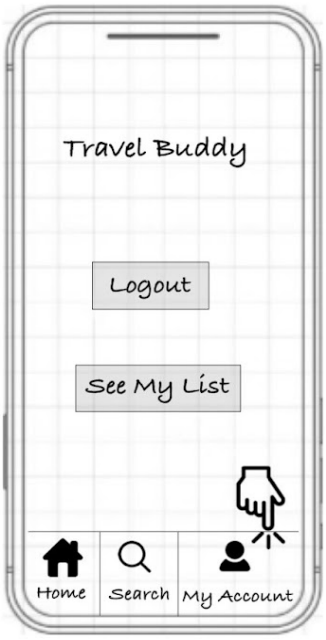
Criterion B: Design

Part A: User Flow and Screen Sketches







User Flow Explained [Exploring Churches Nearby]

- So, for example if any user wished to explore the churches nearby they want to visit, they can:
    1. Enter a valid email and password to sign up.
    2. Enter valid credentials entered while signing up to successfully sign in.

- After successfully signing in, the user now has the option to choose from a list of recommendations, since this particular user wants to explore churches nearby, they can click on the church option from the recommendation list.

- After clicking on this option the user is redirected to a google maps page where they are requested to grant permission to the application to access their location.

- Once the user grants access to their current location, the application displays the user's location and a list of churches nearby on the maps page with the red location pin.

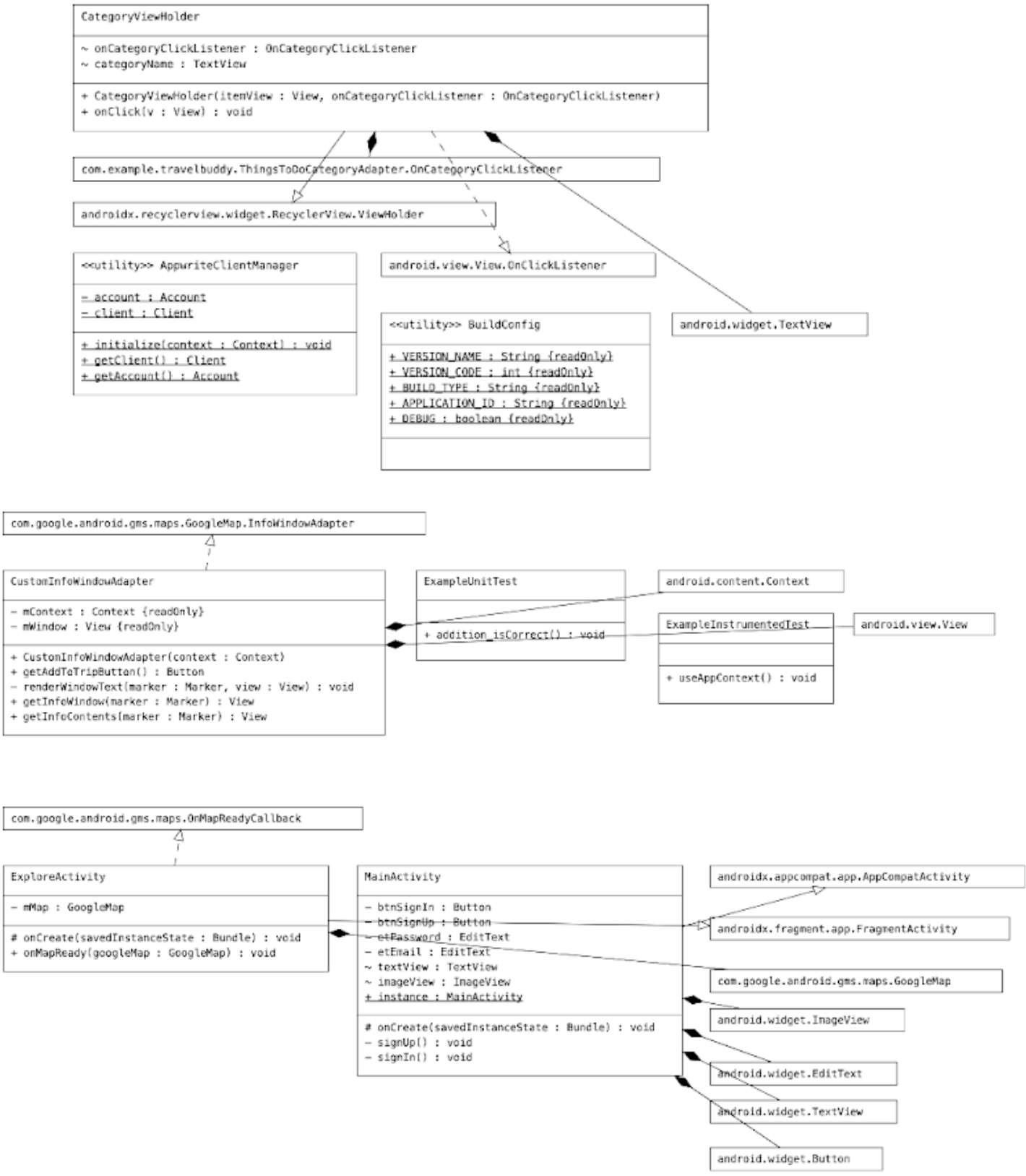
- After exploring the user can click on the red pin of one of the churches they would want to visit. This displays the exact location and Add to My List button.

- After clicking the Add to My List button the user can explore more churches nearby and add it to their list. Once done exploring the user can access their saved locations by clicking on My Account → See My List to see the list of saved locations.

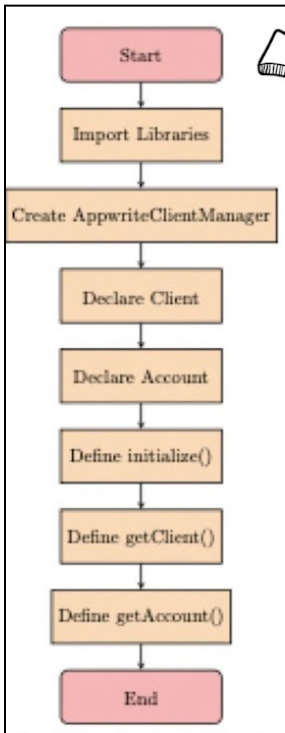
The following criteria are met:

- The client should be able to sign up by entering their email and setting a password. This will save all the information related to their account in our database as well.
- After logging in the client will be directed to an explore page which would have a search bar for tailored searches and a list of recommendations that include locations and things to do nearby.
- After searching for a specific thing to do or location; or by clicking on any of the recommendations the user will then be redirected to a maps page with all the locations nearby related to that particular activity or type of location.
- By simply clicking on the red pop-up location symbol users can see the exact address of the location and click on the "Add to list" button to add it to their list of places to visit.
- If any user would like to have a look at their lists they can access the my account button on the bottom navigation bar where they would have the information related to all the locations in their list and also a button to log out.

Part B: UML Class Diagrams



Part C: Flowcharts [Class Parameters & Functions]



This Java class called "TripPlace" has three fields: name (a String), address (a String), and latLng (a LatLng object). It also has a constructor that takes three parameters: name, address, and latLng, and initializes the corresponding fields of the class.

The parameters for the constructor are:

name: a string that represents the name of a place.
address: a string that represents the address of a place.
latLng: a LatLng object that represents the latitude and longitude coordinates of a place.

This class represents a place with a name, address, and latitude/longitude coordinates. This is useful for creating a list of places to visit or for displaying the location of a place on a map. The class provides three methods to access the fields: getName(), getAddress(), and getLatLng(). These methods return the values of the corresponding fields.

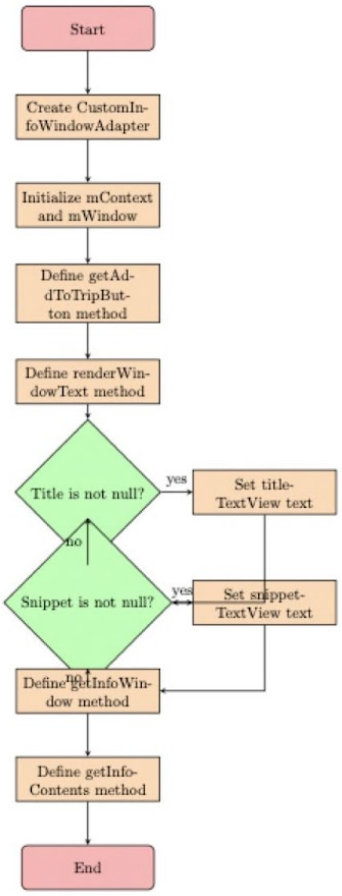
This Java class is called "CustomInfoWindowAdapter" implements the GoogleMap.InfoWindowAdapter interface. This interface is used to customize the appearance of the info window that is displayed when a marker on a Google Map is clicked.

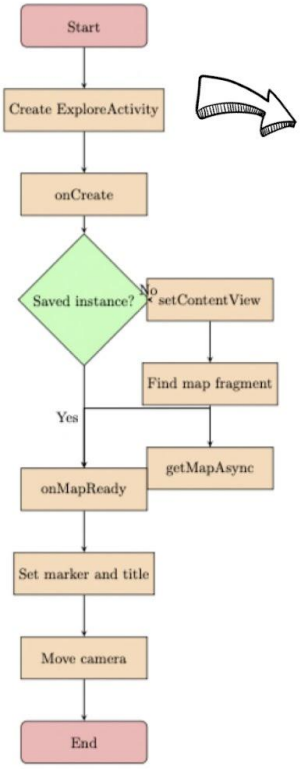
The constructor for this class takes one parameter:
context: a Context object that is used to inflate the custom info window layout.
The functionality of this class is to provide a custom info window layout for a marker on a Google Map. The class provides two methods: getInfoWindow() and getInfoContents(). These methods are called when the info window is opened for a marker. They both call the renderWindowText() method, which sets the text for the title and snippet of the info window. The getAddToTripButton() method returns the button used to add a marker to a trip.

The renderWindowText() method takes two parameters:

marker: a Marker object representing the marker that was clicked.
view: a View object representing the custom info window layout.
The renderWindowText() method sets the title and snippet of the info window by getting the values from the Marker object and setting them on the TextView objects in the custom info window layout.

This class provides a way to customize the appearance of the info window that is displayed when a marker is clicked on a Google Map. It does this by inflating a custom layout and setting the values of the TextViews in the layout based on the data provided by the Marker object.





This activity named ExploreActivity that displays a map using the Google Maps API. The ExploreActivity extends the FragmentActivity class and implements the OnMapReadyCallback interface.

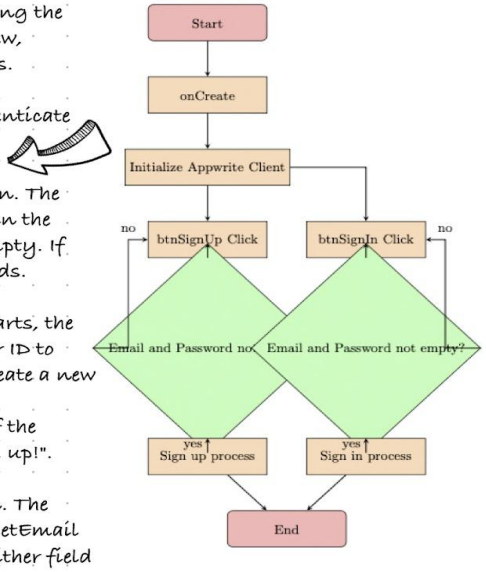
The onCreate method is called when the activity is created. It sets the layout of the activity to be the XML layout file activity_explore.xml, which presumably contains a SupportMapFragment that displays the map. The getSupportFragmentManager() method is called to retrieve the support FragmentManager and findFragmentById() is called to find the SupportMapFragment with the ID R.id.map. The getMapAsync() method is called on this fragment to set up the OnMapReadyCallback interface.

The onMapReady method is called when the map is ready to be used. It receives a GoogleMap object which can be used to manipulate the map. In this method, a LatLng object is created to specify the location of the marker that will be added to the map. A marker is added to the map using the addMarker() method and a MarkerOptions object which specifies the position and title of the marker. The moveCamera() method is then used to move the camera to the specified location with a zoom level of 15.

Its purpose is to display a map and add a marker to a specified location on the map. The code can be modified to display different locations on the map or to add more markers with different titles and positions. Additionally, the logic in the onMapReady method can be modified to implement different functionality for the map, such as displaying user location, customizing marker icons, or displaying additional information when a marker is clicked.

The MainActivity class extends the AppCompatActivity class and overrides its onCreate() method. The onCreate() method sets the layout of the activity using the setContentView() method, and initializes several views (ImageView, TextView, EditText and Button objects) using their respective findViewById() methods.

The code also initializes the Appwrite client manager, which is used to authenticate users with Appwrite, a backend as a service platform.



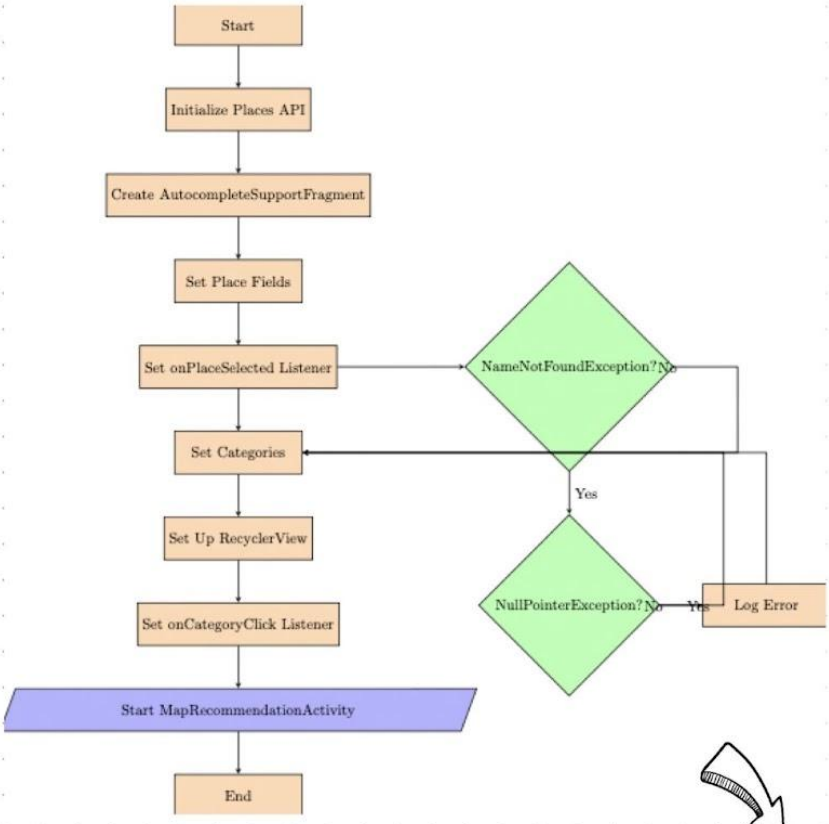
The signUp() method is called when the user clicks on the btnSignUp button. The signUp() method first retrieves the email and password entered by the user in the etEmail and etPassword fields respectively, and checks if these fields are empty. If either field is empty, it displays a message asking the user to fill in all fields.

If both fields are not empty, the signUp() method splits the email into two parts, the email address and the domain name, and uses the email address as the user ID to create a new account with Appwrite. It uses the account.create() method to create a new account, passing in the user ID, email address and password, and a CoroutineCallback object that will be called when the operation is complete. If the operation is successful, it displays a message saying "Successfully signed up!".

The signIn() method is called when the user clicks on the btnSignIn button. The signIn() method retrieves the email and password entered by the user in the etEmail and etPassword fields respectively, and checks if these fields are empty. If either field is empty, it displays a message asking the user to fill in all fields.

If both fields are not empty, the signIn() method uses the account.createEmailSession() method to create a new email session with Appwrite, passing in the email and password, and a CoroutineCallback object that will be called when the operation is complete. If the operation is successful, it displays a message saying "Successfully signed in!", and starts a new activity, ThingstodoActivity, which is used to display things to do in the user's current location.

Overall, this code sets up the main activity of a travel app, allowing users to sign up and sign in using Appwrite, and providing access to things to do in their current location.



This class is MapRecommendationActivity which extends FragmentActivity and implements the OnMapReadyCallback interface. This activity displays a Google Map and searches for nearby places based on a specified category. The user's current location is obtained using the FusedLocationProviderClient from the Google Play Services API.

Parameters:
String category: a string variable that stores the category of places to search for.
Functionality:

onCreate(Bundle savedInstanceState): This method is called when the activity is created. It sets the layout of the activity to activity_map_recommendation.xml and obtains the category from the intent passed to the activity.

onMapReady(@NonNull GoogleMap googleMap): This method is called when the Google Map is ready to be used. It requests permission from the user to access their current location and initializes the custom info window adapter for the markers. It also sets up the OnInfoWindowClickListener and OnMarkerClickListener for the map.

requestLocationPermissions(): This method requests permission from the user to access their current location.

onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults): This method is called when the user grants or denies location permission. If permission is granted, the user's current location is obtained. If permission is denied, a toast message is displayed.

getCurrentLocation(): This method obtains the user's current location using the FusedLocationProviderClient from the Google Play Services API. It adds a marker on the user's current location and moves the camera to that location. It then calls the getNearbyPlacesBasedOnCategory(double latitude, double longitude) method to search for nearby places based on the specified category.

getNearbyPlacesBasedOnCategory(double latitude, double longitude): This method uses the Google Places API to search for nearby places based on the specified category. It constructs a URL to send a GET request to the Google Places API and uses the OkHttp library to make the request. It then parses the JSON response and adds a marker for each nearby place on the map.

addToMyTrips(TripPlace tripPlace): This method adds a trip place to the user's trips.

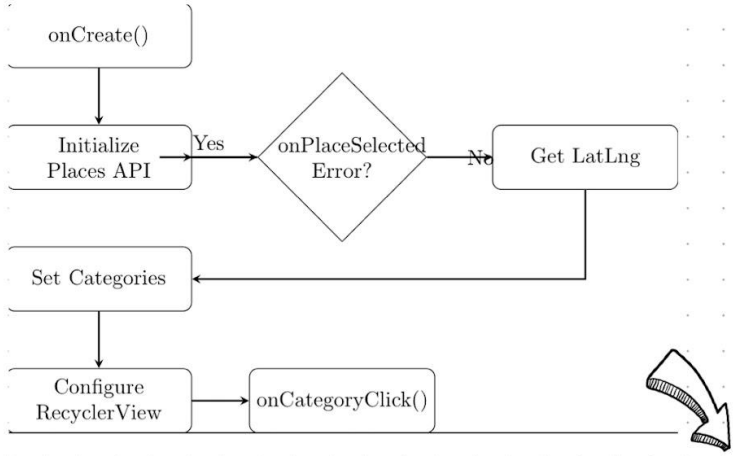
LOCATION_PERMISSION_REQUEST_CODE: A constant integer value used as the request code when requesting location permission.

PLACES_API_BASE_URL: A constant string value that contains the base URL for the Google Places API.

CustomInfoWindowAdapter customInfoWindowAdapter: An instance of the CustomInfoWindowAdapter class which is used to customize the appearance of the info window for each marker.

PlacesClient placesClient: An instance of the PlacesClient class from the Google Places API which is used to fetch details about a specific place.

RectangularBounds bounds: An instance of the RectangularBounds class which represents a rectangular bounding box that can be used to bias search results to a specific area.
TypeFilter typeFilter: An instance of the TypeFilter class which can be used to filter search results by place type.



The ThingstodoActivity class has several instance variables such as placesClient, recyclerView, thingsToDoCategoryAdapter, and categories.

The onCreate() method is the entry point of the activity, and it is called when the activity is created. It performs several tasks, such as initializing the Places API by retrieving the API key from the app's meta-data, creating the PlacesClient instance, and initializing the AutocompleteSupportFragment.

The AutocompleteSupportFragment is a UI component provided by the Places API, and it allows users to search for places using auto-completion. The code sets the fields to be returned when a place is selected and also registers a listener to handle the selected place.

The code also creates a list of categories that are used to populate a RecyclerView with the categories for things to do. The RecyclerView displays a list of categories, and when a user clicks on a category, it launches a MapRecommendationActivity and passes the selected category as an extra to the intent.

This activity class takes no parameters and is responsible for initializing the Places API, handling user input from the AutocompleteSupportFragment, and displaying a list of categories for things to do in a RecyclerView. When a user selects a category, it launches the MapRecommendationActivity to display recommendations for the selected category.

Part D: Testing Plan

Criteria to Test	Testing Methodology	Application Display
Successful Sign-Up Users should be able to register with a valid email address to complete the sign-up process.	Case 1: Enter a valid email id and password. Case 2: Enter an invalid email address, missing an @, domain, etc.	Case 1: Toast notification: Successfully Signed Up. Case 2: Toast Notification: Enter valid email.
Successful Login: Users should be able to log in using the email and password they provided during sign-up.	Case 1: Enter a valid email address and correct password Case 2: Enter a valid email address and incorrect password	Case 1: Toast notification: Successfully Signed In. Displays activities. Case 2: Toast Notification: Failed to Sign In.

The app should display a set of nearby locations related to the user's chosen criteria or preferences.	Enter valid credits to sign in, the app should display a list of activities or things to do in the recycler view.	The app displays a list of things/ activities to choose from.
Location Access: Upon accessing the maps page, users should receive a pop-up requesting permission to access their device's location from the settings.	Click on any of the options from the list of activities to check the functionality of this segment of the success criteria.	On clicking the list users receive a pop-up requesting permission to access their device's location from the settings.
Location Display: After granting permission, the app should accurately display the user's location on the map.	Click on “Only this time” for the location access request pop-up.	The app displays the current location and nearby locations of the category chosen from the list.
Location Information and Addition: After clicking on the red pin location symbol should display the exact location and name of the place, along with an option to add it to the user's list.	Click on the red pin-location symbol, then click on the location displayed by clicking it initially. Click on the Add to My List button.	The app displays the exact location of the place and once the user clicks on the location, the “Add to My List” option is displayed and can be clicked. After clicking on the “Add to My List” button a toast “Added to List” is displayed.
Account Management: Users should be able to access their account by clicking the "My Account" button, which should display a button for logging out and their list of saved locations.	Click on the My Account button in the bottom navigation bar.	The app displays the list of all the saved locations the user saved and a button to log out.