

12/25/2023

PMH

LAB SESSION 5:Geolocation

By:Kritika Bissessur
TO:MR SHIAM BEEHARRY

Contents

1	Task 1	1
1.1	Brief overview	1
1.2	Packages.....	1
1.3	Steps.....	2
1.4	Final output.....	5
2	Task 2	6
2.1	Brief overview:	6
2.2	Packages.....	6
2.3	Steps.....	7
2.4	Final Output	10
3	Task 3	12
3.1	Brief overview	12
3.2	Packages.....	12
3.3	Steps.....	12
3.4	Final Output:	18

1 Task 1

Task-1

Adding Google Maps to a Flutter App

<https://codelabs.developers.google.com/codelabs/google-maps-in-flutter#0>

Provide a documentation with a short description and screenshots for all the steps of the lab and the Dart files.

1.1 Brief overview

Task 1 involves the initial setup of a Flutter project to seamlessly integrate Google Maps. The first step is to create a new Flutter project through the Flutter CLI, laying the foundation for further development. Following project initialization, the crucial step of configuring Google Maps integration begins. This requires modifying the `pubspec.yaml` file to include essential dependencies, notably the `google_maps_flutter` package. This package acts as the bridge, facilitating the seamless integration of Google Maps functionalities into the Flutter application.

Integral to the setup is the acquisition of an API key from the Google Cloud Console. This key serves as the authentication mechanism, enabling the app to make secure requests to the Google Maps API. Once this initial configuration is completed as part of Task 1, developers will have a solid Flutter project, primed for the seamless incorporation of powerful Google Maps features into their applications.

1.2 Packages

In this Flutter code snippet, the following packages are being imported:

1. `flutter/material.dart`:

This package provides the Flutter framework's core material design implementation. It includes widgets and classes needed to build Material Design applications, such as `MaterialApp`, `Scaffold`, `AppBar`, etc.

2. `google_maps_flutter/google_maps_flutter.dart`:

This package is part of the official Google Maps plugin for Flutter. It allows developers to embed Google Maps within their Flutter applications and provides widgets like `GoogleMap`, `Marker`, `InfoWindow`, etc., to work with map-related components.

3. src/locations.dart (aliased as locations):

This seems to be a custom module or file named locations.dart. The use of as locations means that the content of this file can be referred to using the alias locations. It suggests that this file likely contains code related to location information, possibly a set of predefined locations.

1.3 Steps

1. In step 1, we need to add the required dependencies in the pubspec.yaml file. In this case, the required dependencies are http, json_serializable, json_annotation, and build_runner.

```
29 # versions available, run 'flutter pub outdated'.
30 dependencies:
31   flutter:
32     sdk: flutter
33
34
35 # The following adds the Cupertino Icons font to your application.
36 # Use with the CupertinoIcons class for iOS style icons.
37 cupertino_icons: ^1.0.2
38 http: ^0.13.5
39 json_serializable: ^6.6.1
40 json_annotation: ^4.8.0
41 google_maps_flutter: any
42
```

2. We need to create a directory in the lib directory, then we need to create a locations.dart file and describe the structure of the returned JSON data as follows:

```
1 import 'dart:convert';
2
3 import 'package:flutter/foundation.dart';
4 import 'package:flutter/services.dart' show rootBundle;
5 import 'package:http/http.dart' as http;
6 import 'package:json_annotation/json_annotation.dart';
7
8 part 'locations.g.dart';
9
10 @JsonSerializable()
11 class Lating {
12   Lating({
13     required this.lat,
14     required this.lng,
15   });
16
17   factory Lating.fromJson(Map<String, dynamic> json) => _LatingFromJson(json);
18   Map<String, dynamic> toJson() => _LatingToJson(this);
19
20   final double lat;
21   final double lng;
22 }
23
24 @JsonSerializable()
25 class Region {
26   Region({
27     required this.coords,
28     required this.id,
29     required this.name,
30     required this.name
```

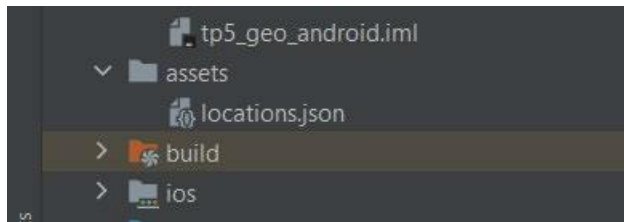
Once we've added this code, our IDE should display some red squiggles, as it references a nonexistent sibling file, locations.g.dart.

To resolve this issue we need to run “flutter pub run build_runner build --delete-conflicting-outputs” this generated file converts between untyped JSON structures and named objects. Our code should now analyze cleanly again.

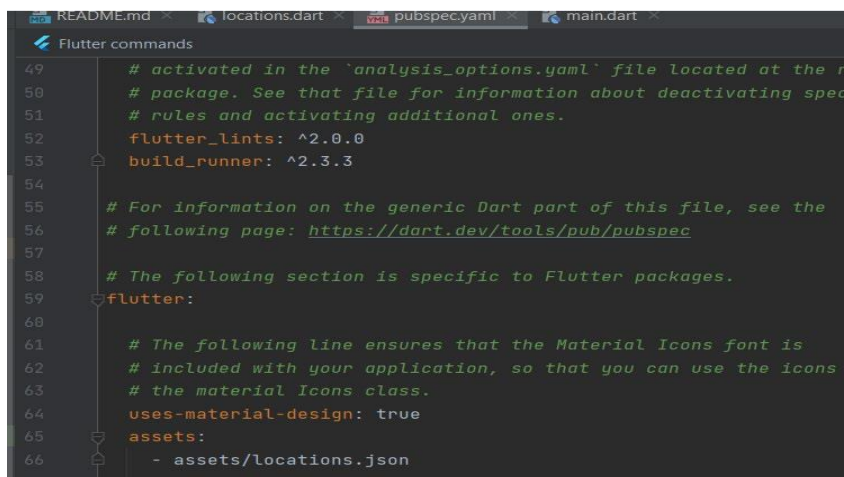
3. We should add in the fallback locations.json file used in the getGoogleOffices function.

One of the reasons for including this fallback is that the static data being loaded in this function is served without CORS headers, and thus will fail to load in a web browser. The Android and iOS Flutter apps don't need CORS headers, but mobile data access can be finicky at the best of times.

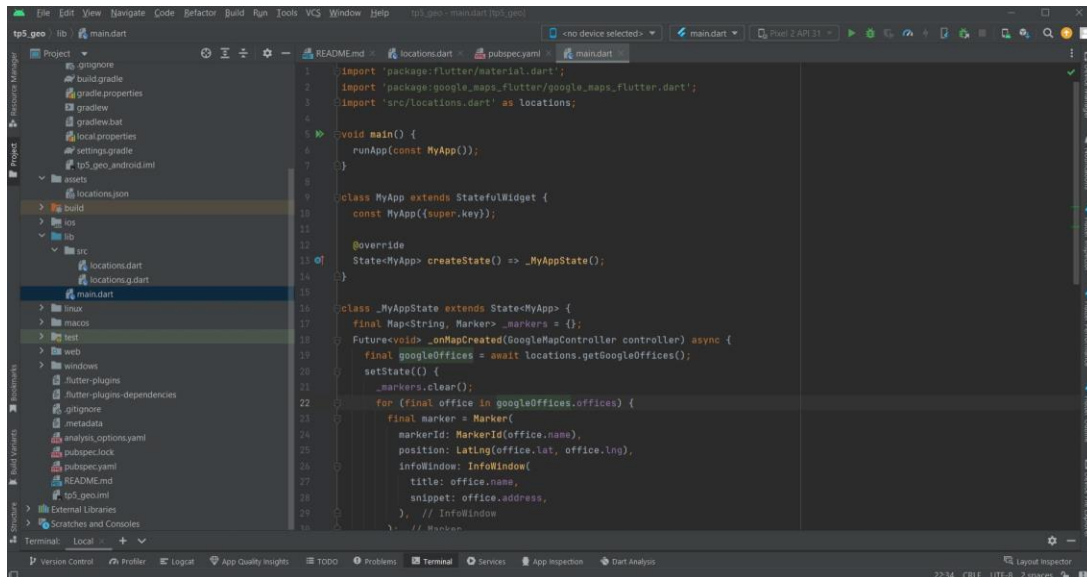
We need to create a directory assets and save the contents locations.json in the directory.



4. Now that we have the asset file downloaded, we need add it to the flutter section of our pubspec.yaml file.



5. Lastly, we need to modify the main.dart file to request the map data, and then use the returned info to add offices to the map:



```
1 import 'package:flutter/material.dart';
2 import 'package:google_maps_flutter/google_maps_flutter.dart';
3 import 'src/locations.dart' as locations;
4
5 void main() {
6   runApp(const MyApp());
7 }
8
9 class MyApp extends StatefulWidget {
10   const MyApp({super.key});
11
12   @override
13   State<MyApp> createState() => _MyAppState();
14 }
15
16 class _MyAppState extends State<MyApp> {
17   final Map<String, Marker> _markers = {};
18   Future<void> _onMapCreated(GoogleMapController controller) async {
19     final googleOffices = await locations.getGoogleOffices();
20     setState(() {
21       _markers.clear();
22       for (final office in googleOffices.offices) {
23         final marker = Marker(
24           markerId: MarkerId(office.name),
25           position: LatLng(office.lat, office.lng),
26           infoWindow: InfoWindow(
27             title: office.name,
28             snippet: office.address,
29           ), // InfoWindow
30         ); // Marker
31       }
32     });
33   }
34 }
```

Explanation of the code:

1. Import Statements:

- Importing necessary Flutter and Google Maps packages.
- flutter/material.dart: Core Flutter material design implementation.
- google_maps_flutter/google_maps_flutter.dart: Google Maps plugin for Flutter.
- src/locations.dart (aliased as locations): A custom module or file providing location-related data.

2. main() Function:

- Entry point of the application.
- Calls runApp() to run the app and specifies the top-level widget (MyApp).

3. MyApp Class:

- A stateful widget representing the entire application.
- MyApp extends StatefulWidget and overrides createState() to create the mutable state (_MyAppState).

4. _MyAppState Class:

- The mutable state class for MyApp.
- Defines a map (_markers) to store markers on the map.

5. _onMapCreated Method:

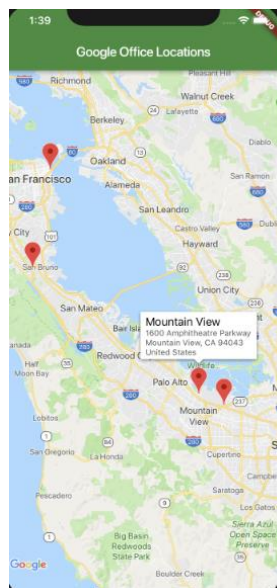
- Asynchronous method triggered when the Google Map is created.
- Retrieves Google office locations asynchronously using the getGoogleOffices() function from the locations module.
- Clears existing markers and populates _markers with new markers based on Google office data.

6. build Method:

- Builds the UI for the app.
- Uses MaterialApp as the top-level widget with a customized theme.
- Creates a Scaffold containing an AppBar with the title "Google Office Locations" and elevation.
- The body of the Scaffold contains a GoogleMap widget with an initial camera position and markers from the _markers map.

In summary, the app fetches Google office locations and displays them on a Google Map with custom markers. It demonstrates the use of Google Maps in a Flutter application along with the integration of a custom locations module.

1.4 Final output



2 Task 2

Task-2

Create a Flutter APP using OpenStreet Map (OSM)
Create a Flutter App with OSM to display a few (5) hotels in 5 different districts in Mauritius using markers. Add some interesting features on the markers such as an info window.
Provide a documentation with a short description and screenshots for all the steps of the lab and the Dart files.

2.1 Brief overview:

Task 2 involves creating a Flutter app using OpenStreetMap (OSM) to showcase hotels in different districts of Mauritius. The app will utilize markers to indicate the locations of these hotels on the map. Additionally, it aims to enhance user interaction by incorporating interesting features on the markers, such as an info window.

2.2 Packages

1. flutter/material.dart: This is the core package for Flutter, providing fundamental UI components and functionalities for building Material Design-compliant apps. It includes widgets for constructing the app's user interface, managing navigation, and handling other aspects of the visual presentation.
2. flutter_map/flutter_map.dart: This package is crucial for integrating maps into the Flutter app. It leverages the Leaflet library, allowing developers to embed interactive maps seamlessly. The flutter_map package simplifies map integration by providing a set of Flutter-specific components and utilities.
3. latlong2/latlong.dart: Responsible for handling geographical coordinates, this package is essential when working with maps. It defines the LatLng class, enabling the representation of latitude and longitude coordinates. This package is particularly useful for specifying locations and markers on the map.
4. location/location.dart: The location package is employed to manage device location services. It facilitates the retrieval of the device's current location, granting the app access to real-time geographic data. This is crucial for tasks such as placing markers on the map based on the device's coordinates.

2.3 Steps

Before we start, we need to add the required dependencies in the pubspec.yaml file. In this case, the required dependencies are flutter_map , latlong2 , geolocator , and location.

```
# versions available, run 'flutter pub outdated'.
dependencies:
  flutter:
    sdk: flutter

  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.2
  flutter_map: ^0.14.0
  latlong2: ^0.8.1
  geolocator: ^7.6.0
  location: ^4.3.0
```

This small snippet of code ensures that the app has the necessary permissions and services enabled to access the device's location. If successful, it updates the UI with the current location.

1. Location Service Check:

- The code uses the Location class to interact with the device's location services.
- It checks whether location services are enabled using `await location.serviceEnabled()`.
- If not enabled, it requests the user to enable location services using `await location.requestService()`.
- If the user denies enabling location services, the function exits.

2. Location Permission Check:

- It checks whether the app has permission to access the device's location using `await location.hasPermission()`.
- If permission is denied, it requests the user for location permission using `await location.requestPermission()`.
- If the user denies location permission, the function exits.

3. Obtaining Location:

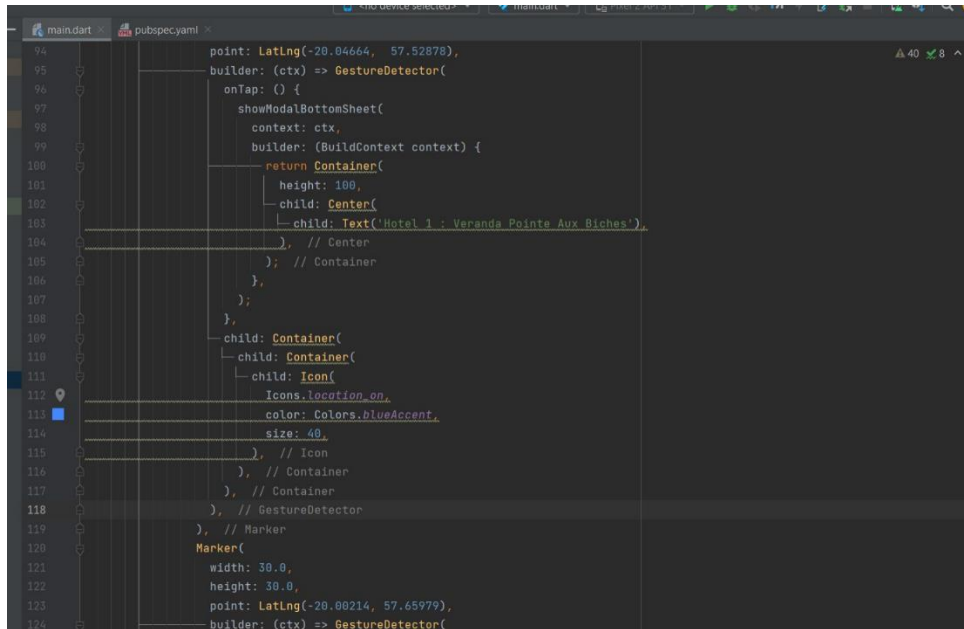
If location services are enabled and the app has permission, it obtains the current location using `await location.getLocation()`.

4. Updating UI:

- The obtained location data is stored in `_locationDataTemp`.
- The UI is updated by setting the state with the new location data, adding a marker to the map at the obtained location.

```
Future<void> _getLocation() async {  
  Location location = Location();  
  
  bool _serviceEnabled;  
  PermissionStatus _permissionGranted;  
  LocationData _locationDataTemp;  
  
  _serviceEnabled = await location.serviceEnabled();  
  if (!_serviceEnabled) {  
    _serviceEnabled = await location.requestService();  
    if (!_serviceEnabled) {  
      return;  
    }  
  }  
  
  _permissionGranted = await location.hasPermission();  
  if (_permissionGranted == PermissionStatus.denied) {  
    _permissionGranted = await location.requestPermission();  
    if (_permissionGranted != PermissionStatus.granted) {  
      return;  
    }  
  }  
}
```

3. We need to modify the provided code in order to to display a few (5) hotels in 5 different districts in Mauritius using markers



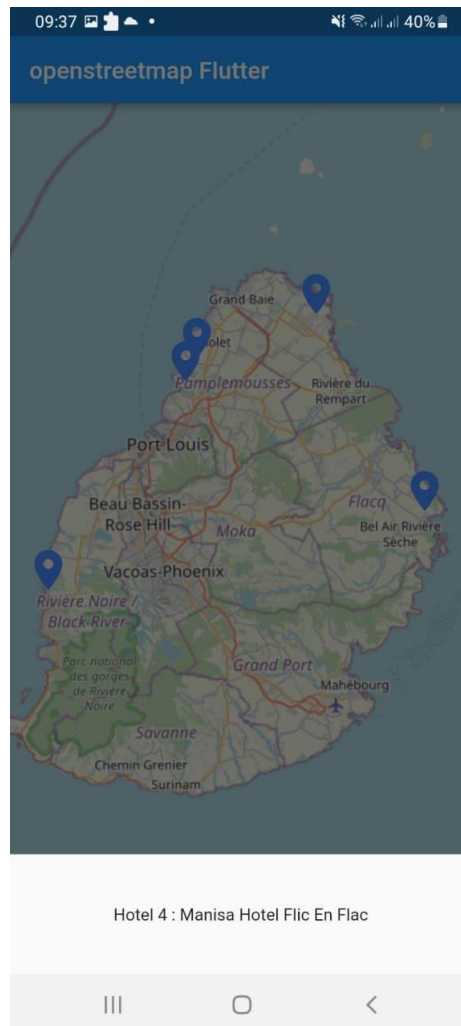
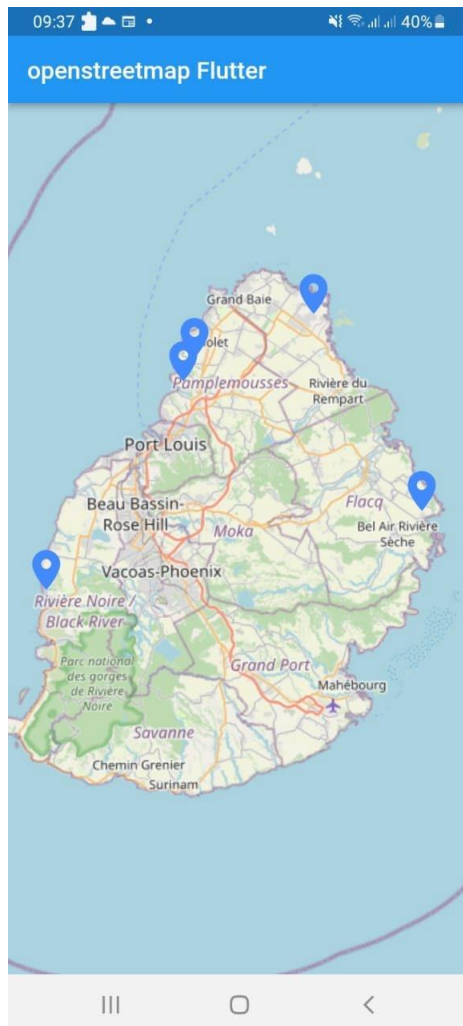
```

94      point: LatLng(-20.04664, 57.52878),
95      builder: (ctx) => GestureDetector(
96        onTap: () {
97          showModalBottomSheet(
98            context: ctx,
99            builder: (BuildContext context) {
100              return Container(
101                height: 100,
102                child: Center(
103                  child: Text('Hotel 1 : Veranda Pointe Aux Biches'),
104                ), // Center
105              ), // Container
106            ),
107          );
108        },
109        child: Container(
110          child: Container(
111            child: Icon(
112              Icons.location_on,
113              color: Colors.blueAccent,
114              size: 40,
115            ), // Icon
116          ), // Container
117        ), // Container
118      ), // GestureDetector
119    ), // Marker
120    Marker(
121      width: 30.0,
122      height: 30.0,
123      point: LatLng(-20.00214, 57.65979),
124      builder: (ctx) => GestureDetector(

```

- Five markers with different latitude and longitude have been added in order to display different hotels in Mauritius.
- The modal, displays a modal bottom sheet when a marker is tapped. In this instance, the builder method returns a Container widget that has details on the relevant hotel.
- The markers list can be expanded by adding new Marker widgets with their own constructor functions and settings.
- A modal bottom sheet is displayed with some text about the location when the user taps on a marker.

2.4 Final Output



Explanation of the code:

This Flutter code creates a mobile app that utilizes OpenStreetMap (OSM) to display a map with markers representing five hotels in different districts in Mauritius. Here's a breakdown of the code:

1. Initialization and State Setup:

- The app is initialized with a `MaterialApp` widget, and the `MyApp` widget is set as the home screen.
- `_MyAppState` is a stateful widget responsible for managing the app's state.

2. Location and Map Initialization:

- The `myMarkers` list holds markers that will be displayed on the map.
- The `_locationData` variable stores information about the device's current location.
- The `initState` method is used to fetch the device's location using the `location` package. It checks and requests necessary permissions and service enablement.

3. Map Widget - FlutterMap:

- The app's UI is built using a `Scaffold` widget with an `AppBar` and a `Container` holding the map.
- The `FlutterMap` widget from the `flutter_map` package is used. It allows for the integration of OpenStreetMap tiles into the app.

4. Tile Layer:

-The map is set up with a tile layer using the OpenStreetMap tile URL template.

5. Marker Layer:

- A `MarkerLayerOptions` widget is used to display markers on the map.
- Markers are added for five different hotels, each represented by a `Marker` widget.
- Each marker includes its own `onTap` functionality to display a bottom sheet with information when tapped.

6. Hotel Markers:

- Each hotel is represented by a `Marker` with specific coordinates and a custom icon (blue location icon).
- When a user taps on a marker, a bottom sheet is displayed with information about the respective hotel.

The overall result is a Flutter app that showcases a map with marked hotel locations in Mauritius, providing an interactive and visually appealing user experience.

3 Task 3

Task-3

Create a Flutter APP to showcase a live location tracking
Reference: https://medium.com/flutter-community/flutter-google-map-with-live-location-tracking-uber-style-12da38771829
Provide a documentation with a short description and screenshots for all the steps of the lab and the Dart files.

3.1 Brief overview

In Task 3, the aim is to create a Flutter app that integrates Google Maps for live location tracking, emulating the style used by Uber. The steps involved in the lab include setting up a new Flutter project, adding dependencies for Google Maps, implementing the live location tracking feature, and styling the app to resemble the Uber app.

3.2 Packages

The code snippet imports three Flutter packages: flutter/material.dart, location/location.dart, and two custom Dart files, map_page.dart and map_simple.dart. The flutter/material.dart package is the core Flutter framework for building user interfaces, providing widgets and tools for designing interactive applications. The location/location.dart package facilitates location-based functionalities, enabling the app to access device location information. The custom Dart files, map_page.dart and map_simple.dart, likely contain specific implementations or features related to mapping and location services.

3.3 Steps

1. The Flutter project relies on three crucial dependencies for its location-based and mapping functionalities. The first one is the location package with version 5.0.3, which facilitates the management of location services on the device, providing methods to retrieve the device's location and handle necessary permissions. The second dependency is google_maps_flutter with version 2.5.0, a fundamental plugin for seamlessly integrating Google Maps into the Flutter app. This package empowers developers to incorporate interactive maps, markers, and various map-related features into their applications. Lastly, the geocoding package is mentioned, indicating its potential use, though specific version details are not provided. Geocoding packages are commonly used to convert between geographic coordinates and human-readable addresses, supporting tasks such as displaying location names on the map. Together, these dependencies enhance the Flutter app with robust location and mapping capabilities.

```
cupertino_icons: ^1.0.2

# GoogleMap APIs Plugins
location: ^5.0.3 # Location Plugin Link https://pub.dev/packages/location
google_maps_flutter: ^2.5.0 # Google Maps for Flutter Link https://pub.dev/packages/google\_maps\_flutter
geocoding: ^2.1.1 # Flutter Geocoding Plugin Link https://pub.dev/packages/geocoding

# After adding your reference dependencies run the following comm
```

2. In the AndroidManifest.xml snippet, crucial permissions are declared for a location-based app. INTERNET enables network communication, ACCESS_FINE_LOCATION grants access to precise location data, and FOREGROUND_SERVICE allows running location services in the foreground. The ACCESS_BACKGROUND_LOCATION permission permits accessing device location in the background. These permissions are essential for features like live tracking and mapping in a Flutter app.

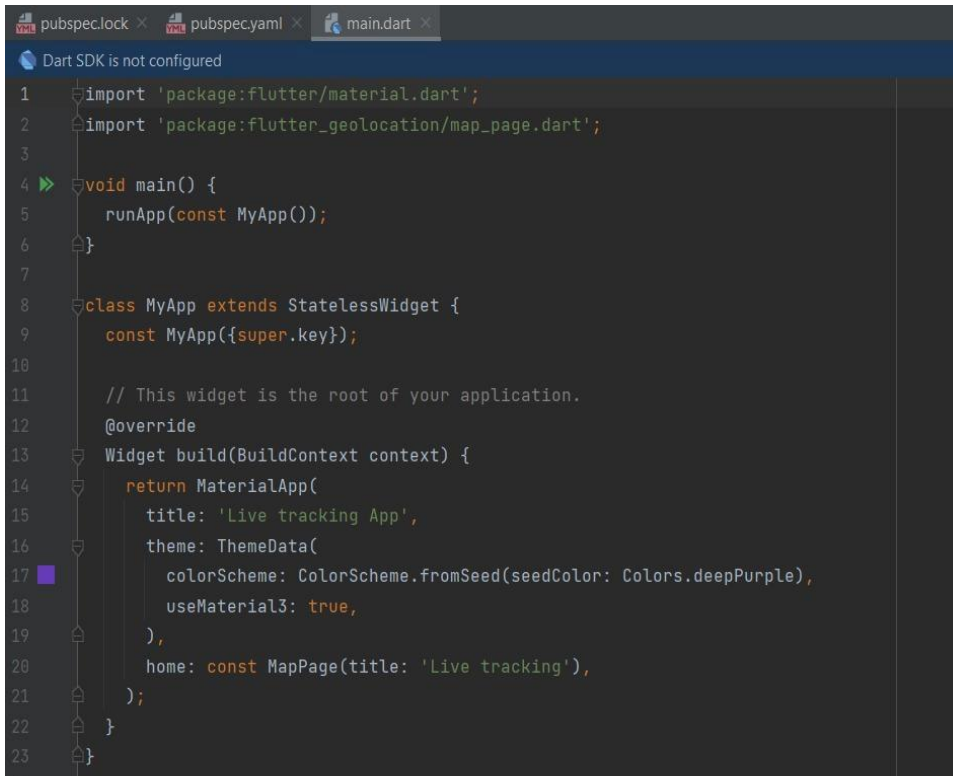
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
  <!-- Request permission in Android application -->
  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
  <uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
  <uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION"/>

  <application
```

3. I then create the main.dart and include the following codes. This Dart code represents the entry point of a Flutter application. It uses the flutter/material.dart library to create a mobile app. The main() function calls runApp() and initializes an instance of the MyApp widget. The MyApp widget is a stateless widget that extends the StatelessWidget class. It defines the overall theme of the app, such as the title and color scheme.

In the build method of MyApp, a MaterialApp widget is returned. This widget is essential for setting up the basic structure of a Flutter app. It includes the app's title, theme configuration, and specifies the initial screen to be displayed, which is an instance of the MapPage widget. The MapPage widget is passed a title of 'Live tracking'.

The theme is configured using ThemeData, where ColorScheme is customized with a primary color derived from the seed color 'Colors.deepPurple'. This snippet provides a clean and structured way to initialize and configure the Flutter app, ensuring a consistent look and feel throughout the application.



```

1 import 'package:flutter/material.dart';
2 import 'package:flutter_geolocation/map_page.dart';
3
4 void main() {
5   runApp(const MyApp());
6 }
7
8 class MyApp extends StatelessWidget {
9   const MyApp({super.key});
10
11   // This widget is the root of your application.
12   @override
13   Widget build(BuildContext context) {
14     return MaterialApp(
15       title: 'Live tracking App',
16       theme: ThemeData(
17         colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
18         useMaterial3: true,
19       ),
20       home: const MapPage(title: 'Live tracking'),
21     );
22   }
23 }

```

4. This Dart code defines a Flutter widget for a map page (MapPage) that allows users to check their location and track it using Flutter and the location plugin. Here's a brief overview of the main components:
- **Stateful Widget:** MapPage is a stateful widget, indicating that it has mutable state. It takes a title as a parameter.
 - **State Class:** _MapPageState is the associated state class. It manages the state for the MapPage, including variables like serviceEnabled, permissionGranted, and userLocation.
 - **getUserLocation Function:** The getUserLocation function utilizes the location plugin to check and request location services and permissions. It updates the userLocation state with the obtained location data.
 - **goToSimpleMap Function:** This function navigates to another page (MapSimplePage) using the Navigator.push method. It is triggered when the user wants to track their location.
 - **Build Method:** The build method constructs the UI for the MapPage. It includes buttons to check and track the user's location, and it displays the latitude and longitude if available. The UI is responsive to changes in the userLocation state.
 - **Scaffold:** The widget is wrapped in a Scaffold, providing a basic structure for the app, including an AppBar, body content, and a floating action button to get the location.


```

1  import 'package:flutter/material.dart';
2  import 'package:flutter_geolocation/map_simple.dart';
3  import 'package:location/location.dart';
4
5  class MapPage extends StatefulWidget {
6    const MapPage({Key? key, required this.title}) : super(key: key);
7    final String title;
8
9    @override
10   State<MapPage> createState() => _MapPageState();
11 }
12
13 class _MapPageState extends State<MapPage> {
14   late bool serviceEnabled;
15   late PermissionStatus permissionGranted;
16   LocationData? userLocation;
17
18   Future<void> getUserLocation() async {
19     Location location = Location();
20
21     serviceEnabled = await location.serviceEnabled();
22     if (!serviceEnabled) {
23       serviceEnabled = await location.requestService();
24       if (!serviceEnabled) {
25         return;
26       }
27     }
28   }
29 }

```

5. **ElevatedButton Widgets:** Two ElevatedButton widgets are included in the column. These buttons trigger the getUserLocation and goToSimpleMap functions when pressed. They have corresponding text labels: "Check your Location" and "Track your location."

6. **SizedBox Widget:** A SizedBox widget adds vertical spacing between the buttons and the next set of widgets. It provides an empty space with a height of 25 pixels.

7. **Conditional Rendering:** The userLocation variable is checked for nullability. If it is not null, a Wrap widget is used to create a new row of widgets. This row includes two Center widgets with Text widgets displaying the latitude and longitude of the user's location.

8. **Fallback Text Widget:** If userLocation is null, a simple Text widget is displayed, informing the user to enable location services and grant permission.

```

Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      backgroundColor: Theme.of(context).colorScheme.secondary,
      title: Text(widget.title),
    ),
    body: Center(
      child: Padding(
        padding: const EdgeInsets.all(20),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            ElevatedButton(
              onPressed: getUserLocation,
              child: const Text('Check your Location'),
            ),
            ElevatedButton(
              onPressed: goToSimpleMap,
              child: const Text('Track your location'),
            ),
            const SizedBox(height: 25),
            userLocation != null
              ? Wrap(
                  children: [
                    Center(
                      child: Text(
                        'Your Latitude: ${userLocation?.latitude}',
                      ),
                    ),
                    // Longitude widget would follow here
                  ],
                )
              : const Text('Enable location services and grant permission.'),
          ],
        ),
      ),
    ),
  );
}

```

- **Stateful Widget:** The `MapSimplePage` is a stateful widget, allowing it to maintain state information.
- **Initialization of Marker Icon:** The `markerIcon` variable is initialized with the default marker icon. It will later be updated to a custom icon loaded from an asset.
- **initState Method:** The `initState` method is overridden to call the `addCustomIcon` function when the widget is initialized. This function loads a custom marker icon from an asset.
- **addCustomIcon Method:** This method loads a custom marker icon from an asset and sets the `markerIcon` variable to the loaded icon.
- **Completer for GoogleMapController:** A `Completer` is used to hold the `GoogleMapController`, providing access to control the map programmatically.
- **Set of Markers:** The `markers` set is initialized to hold map markers. It is later updated in the `_onMapCreated` function.
- **_onMapCreated Method:** This callback function is triggered when the map is created. It adds a marker to the map with specific details, including a custom icon and an information window.
- **Initial Camera Position:** The initial camera position for the map is defined using the `CameraPosition` class.
- **AppBar:** The app bar is displayed at the top of the screen, showing the title "Live tracking."
- **GoogleMap Widget:** The `GoogleMap` widget is used to display the map. It includes various parameters such as `mapType`, `initialCameraPosition`, `onMapCreated`, and `markers`.
- **Map Type and Initial Camera Position:** The map type is set to `hybrid`, and the initial camera position is set using the predefined `_initialCameraPosition`.
- **InfoWindow:** An `InfoWindow` is associated with the marker, displaying the title "Your current location, Olivia."

```

17 // Create a Completer to hold the GoogleMapController, allowing access to the map
18 final Completer<GoogleMapController> controllerMap = Completer();
19
20 // Initialize an empty set to hold map markers
21 Set<Marker> markers = {};
22
23 // Callback function when the map is created
24 void _onMapCreated(GoogleMapController controller) {
25   setState(() {
26     // Add a marker to the map with specific details
27     markers.add(
28       Marker(
29         markerId: const MarkerId("id-1"),
30         position: const LatLng(-20.294359, 57.736938),
31         icon: markerIcon, // Set the custom icon
32         infoWindow: const InfoWindow(title: "Your current location, Olivia"),
33       ),
34     );
35   });
36 }

```

9. **CameraPosition:** The `_initialCameraPosition` variable is defined as a `CameraPosition` object with initial map center coordinates `(-20.294359, 57.736938)` and an initial zoom level of 10.

10. **build Method:** This method returns a `Scaffold` widget, which serves as the basic structure for the page.

11. **AppBar:** The `AppBar` widget is used to create the app bar at the top of the screen. The title of the app bar is set to "Live tracking," and `centerTitle` is set to `true` to center-align the title.

12. **Body:** The body of the `Scaffold` contains the `GoogleMap` widget.

13. GoogleMap Widget:

- **Map Type:** The `mapType` property is set to `MapType.hybrid`, indicating that the map should display satellite imagery with roads and labels.
- **Initial Camera Position:** The `initialCameraPosition` property is set to `_initialCameraPosition`, defining the initial viewpoint of the map.
- **onMapCreated:** The `onMapCreated` property is set to `_onMapCreated`, which is a custom callback function called when the map is created.
- **Markers:** The `markers` property is set to the `markers` set, which contains the map markers to be displayed.

This code sets up the basic structure of the page, including the app bar and the Google Map. The map is centered around the specified coordinates with a predefined zoom level, and the map type is set to hybrid. The `_onMapCreated` function will be called when the map is initialized, and the markers will be displayed on the map

```

map_simple.dart
Dart SDK is not configured
64 final CameraPosition _initialCameraPosition = const CameraPosition(
65   target: LatLng(-20.294359, 27.736938), // Initial map center coordinates
66   zoom: 10, // Initial zoom level for the map
67 );
68
69 @override
70 Widget build(BuildContext context) {
71   return Scaffold(
72     appBar: AppBar(
73       title: const Text("Live tracking"), // Display the app bar title
74       centerTitle: true, // Center align the app bar title
75     ),
76     body: GoogleMap(
77       mapType: MapType.hybrid, // Set the map type to hybrid
78       initialCameraPosition:
79         _initialCameraPosition, // Use the initial camera position
80       // onMapCreated: (GoogleMapController controller) {
81       //   _controller.complete(controller);
82       // },
83       onMapCreated: _onMapCreated, // Call the custom _onMapCreated function
84       markers: markers, // Display the markers on the map
85     ),
86   );
87 }
88
89

```

3.4 Final Output:

