Lab session 5: Create Geolocation Flutter application with Google Maps and Open Street Maps.

Task-3 Create a Flutter APP to showcase a live location tracking

In this task, we will learn how to use Google Maps in Flutter with some customizations, like setting up Custom Image Markers, drawing route direction polylines and adding real-time location updates to the map.

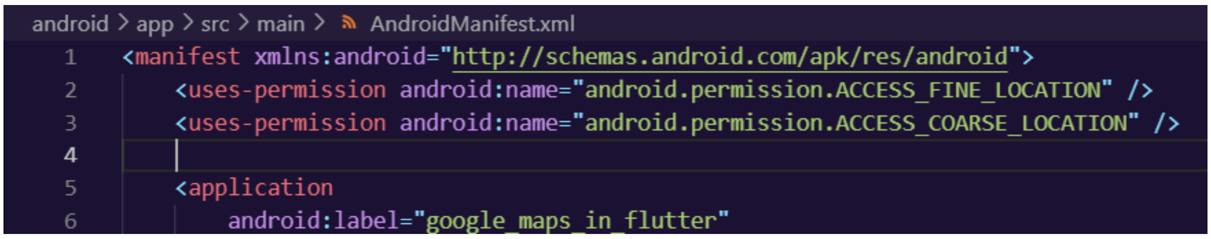
We will the project created in Task-1 as a base.

Environment Setup

To enable location tracking on both iOS and Android for a Flutter app, we need to configure the necessary permissions in the AndroidManifest.xml file for Android and the Info.plist file for iOS.

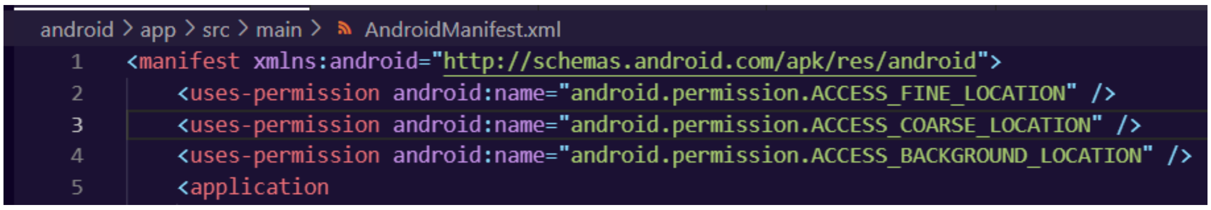
Android (AndroidManifest.xml):

1. Open the **android/app/src/main/AndroidManifest.xml** file.
2. Add the required permissions inside the **<manifest>** element:



```
android > app > src > main > AndroidManifest.xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />

    <application
        android:label="google_maps_in_flutter"
```

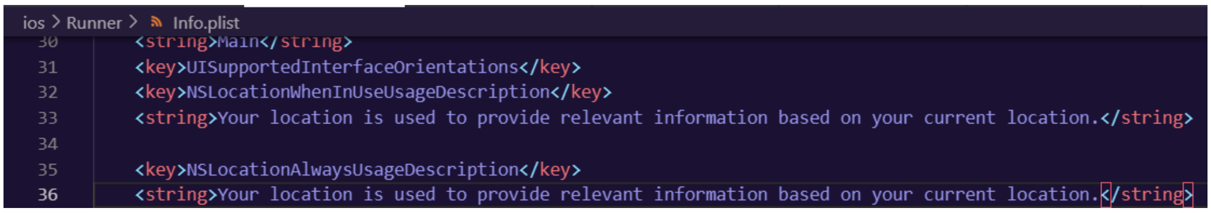
3. If you are targeting Android 10 (API level 29) or higher, add the following line to request background location access:



```
android > app > src > main > AndroidManifest.xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
    <application
```

iOS (Info.plist):

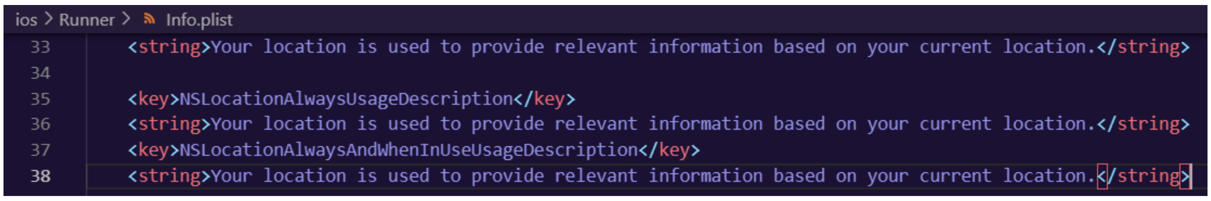
1. Open the **ios/Runner/Info.plist** file.
2. Add the necessary keys for location services:



```
ios > Runner > Info.plist
	<string>Main</string>
	<key>UISupportedInterfaceOrientations</key>
	<key>NSLocationWhenInUseUsageDescription</key>
	<string>Your location is used to provide relevant information based on your current location.</string>

	<key>NSLocationAlwaysUsageDescription</key>
	<string>Your location is used to provide relevant information based on your current location.</string>
```

The description strings is an explanation of why the app needs location access.

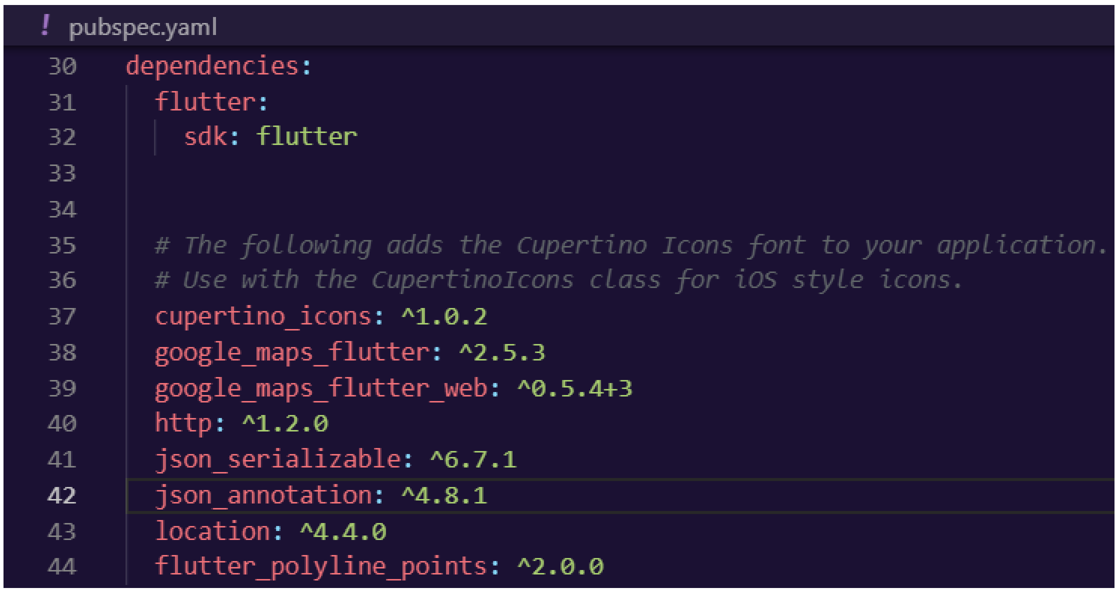
3. For the background location updates, add the following key:



ios > Runner > Info.plist

```xml
    <string>Your location is used to provide relevant information based on your current location.</string>

    <key>NSLocationAlwaysUsageDescription</key>
    <string>Your location is used to provide relevant information based on your current location.</string>
    <key>NSLocationAlwaysAndWhenInUseUsageDescription</key>
    <string>Your location is used to provide relevant information based on your current location.</string>
```

Once set up, the dependencies should look like this:

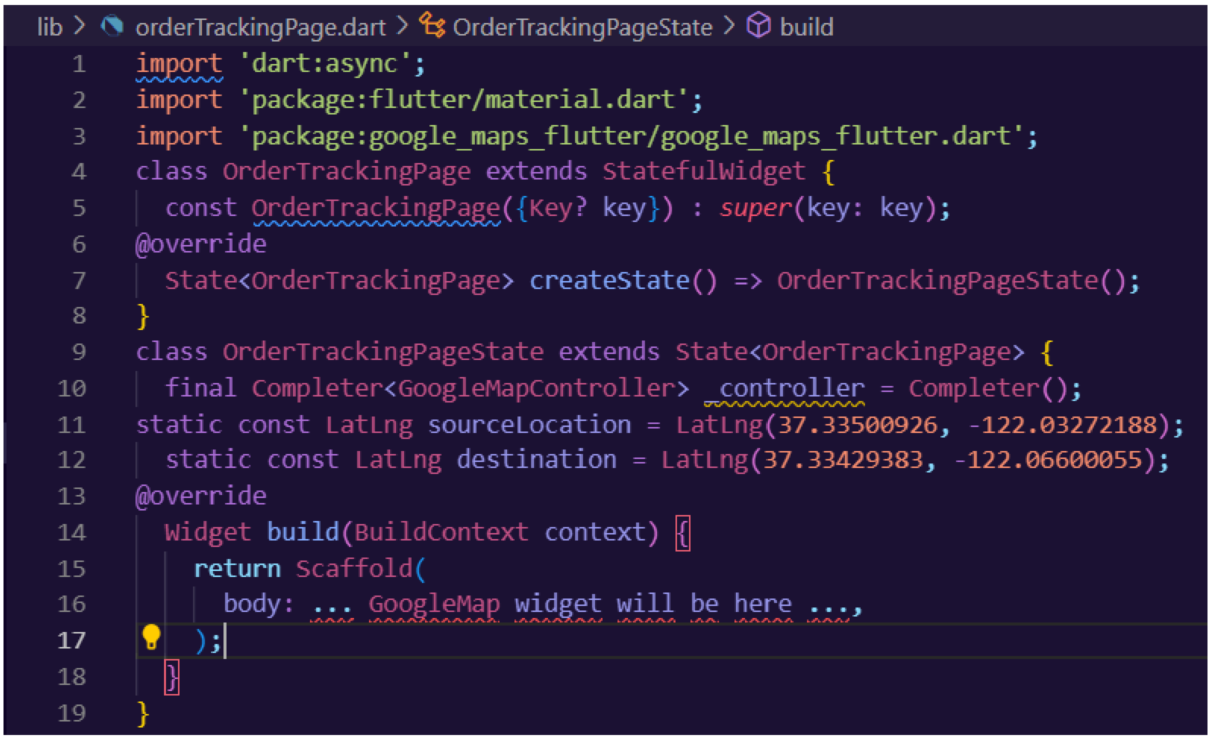


pubspec.yaml

```yaml
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
  google_maps_flutter: ^2.5.3
  google_maps_flutter_web: ^0.5.4+3
  http: ^1.2.0
  json_serializable: ^6.7.1
  json_annotation: ^4.8.1
  location: ^4.4.0
  flutter_polyline_points: ^2.0.0
```

Implement order tracking page

We will create a StatefulWidget called **OrderTrackingPage** with its corresponding State class, where we will import the required packages as well as some hardcoded source and destination location.

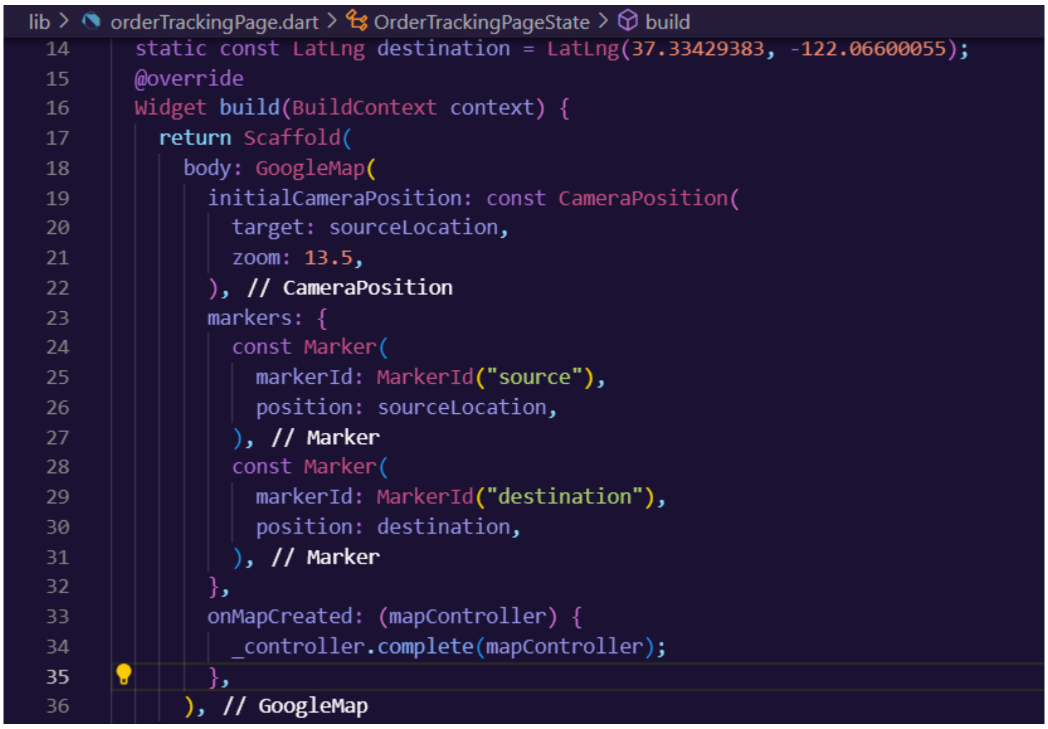


lib > orderTrackingPage.dart > OrderTrackingPageState > build

```dart
import 'dart:async';
import 'package:flutter/material.dart';
import 'package:google_maps_flutter/google_maps_flutter.dart';
class OrderTrackingPage extends StatefulWidget {
  const OrderTrackingPage({Key? key}) : super(key: key);
@override
  State<OrderTrackingPage> createState() => OrderTrackingPageState();
}
class OrderTrackingPageState extends State<OrderTrackingPage> {
  final Completer<GoogleMapController> _controller = Completer();
static const LatLng sourceLocation = LatLng(37.33500926, -122.03272188);
  static const LatLng destination = LatLng(37.33429383, -122.06600055);
@override
  Widget build(BuildContext context) {
    return Scaffold(
      body: ... GoogleMap widget will be here ...,
    );
  }
}
```

Next, we will create the GoogleMap widget and set the initialCameraPosition to the location of the source. The map needs to be zoomed in a bit, so set it to 13.5.

We need a marker/pin to understand the exact location. Define a marker and set its position to the source location. For the destination, add another marker/pin.



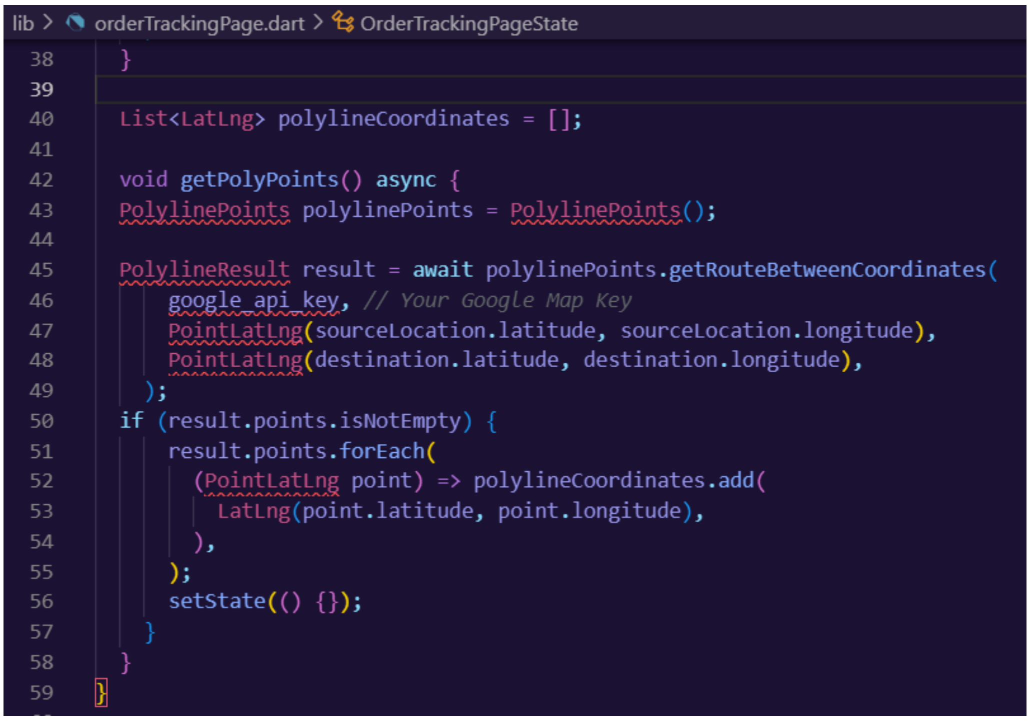
lib > orderTrackingPage.dart > OrderTrackingPageState > build

```
  static const LatLng destination = LatLng(37.33429383, -122.06600055);
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: GoogleMap(
        initialCameraPosition: const CameraPosition(
          target: sourceLocation,
          zoom: 13.5,
        ), // CameraPosition
        markers: {
          const Marker(
            markerId: MarkerId("source"),
            position: sourceLocation,
          ), // Marker
          const Marker(
            markerId: MarkerId("destination"),
            position: destination,
          ), // Marker
        },
        onMapCreated: (mapController) {
          _controller.complete(mapController);
        },
      ), // GoogleMap
```

Draw Route Direction

Next, we will try to draw a line from destination to source.

- Create an empty list called polylineCoordinates.
- Create an instance of PolylinePoints and an async function called getPolyPoints.

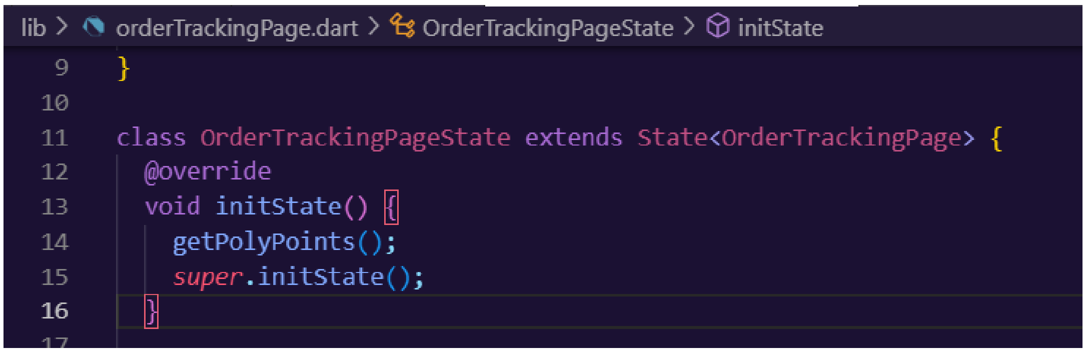


lib > orderTrackingPage.dart > OrderTrackingPageState

```
  }

  List<LatLng> polylineCoordinates = [];

  void getPolyPoints() async {
  PolylinePoints polylinePoints = PolylinePoints();

  PolylineResult result = await polylinePoints.getRouteBetweenCoordinates(
      google_api_key, // Your Google Map Key
      PointLatLng(sourceLocation.latitude, sourceLocation.longitude),
      PointLatLng(destination.latitude, destination.longitude),
    );
  if (result.points.isNotEmpty) {
      result.points.forEach(
        (PointLatLng point) => polylineCoordinates.add(
          LatLng(point.latitude, point.longitude),
        ),
      );
      setState(() {});
    }
  }
}
```

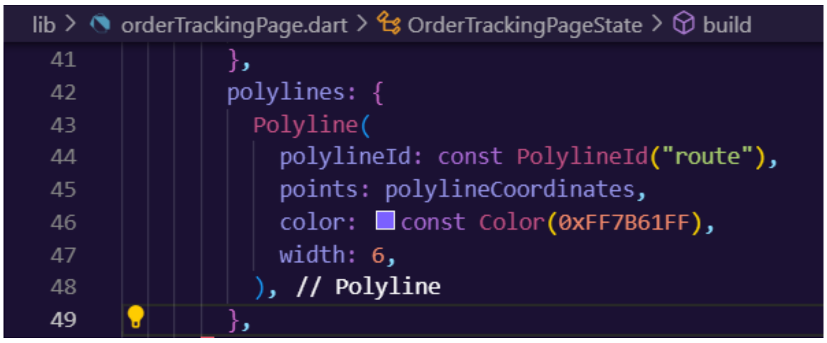
The method getRouteBetweenCoordinates returns the list of polyline points. The Google API key, source, and destination locations were required. If the points are not empty, we store them to polylineCoordinates.

On initState call getPolyPoints:


lib > orderTrackingPage.dart > OrderTrackingPageState > initState

```dart
}

class OrderTrackingPageState extends State<OrderTrackingPage> {
  @override
  void initState() {
    getPolyPoints();
    super.initState();
  }
```

Back to the GoogleMap widget, define the polylines:

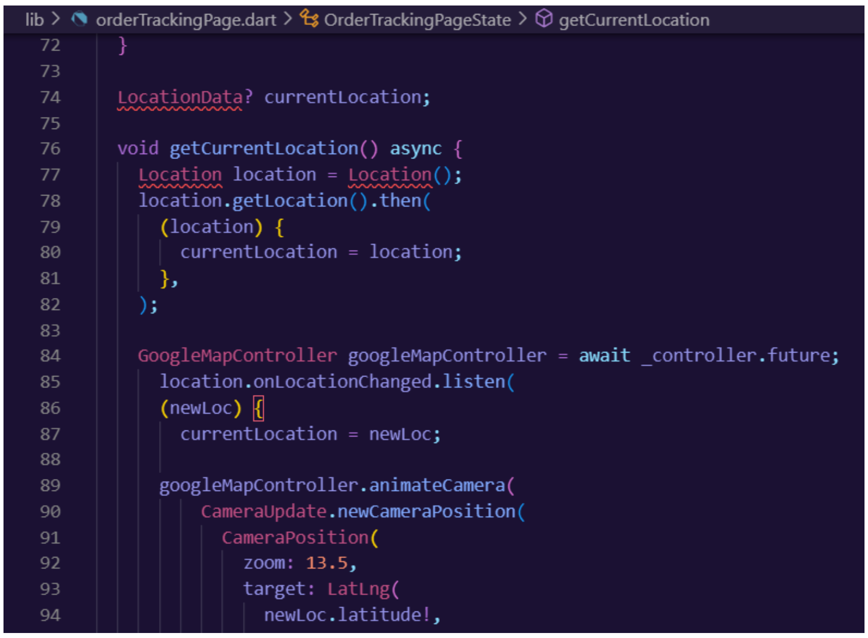

lib > orderTrackingPage.dart > OrderTrackingPageState > build

```dart
        },
        polylines: {
          Polyline(
            polylineId: const PolylineId("route"),
            points: polylineCoordinates,
            color: const Color(0xFF7B61FF),
            width: 6,
          ), // Polyline
        },
```

Real-time Location Updates on Map

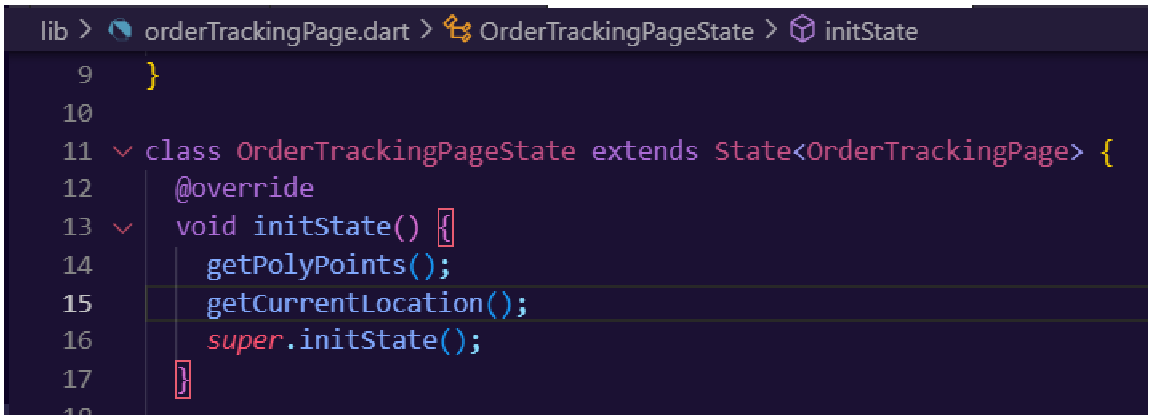
Now, we need the device's location.

To do so, we will create a nullable variable called currentLocation. Then a function called getCurrentLocation, Inside, creates an instance of Location. Once we get the location, set the current location to be equal to the location. On location change, update the current location. Make it visible to the map called setState.


lib > orderTrackingPage.dart > OrderTrackingPageState > getCurrentLocation

```dart
  }

  LocationData? currentLocation;

  void getCurrentLocation() async {
    Location location = Location();
    location.getLocation().then(
      (location) {
        currentLocation = location;
      },
    );

    GoogleMapController googleMapController = await _controller.future;
      location.onLocationChanged.listen(
      (newLoc) {
        currentLocation = newLoc;

      googleMapController.animateCamera(
          CameraUpdate.newCameraPosition(
            CameraPosition(
              zoom: 13.5,
              target: LatLng(
                newLoc.latitude!,
```

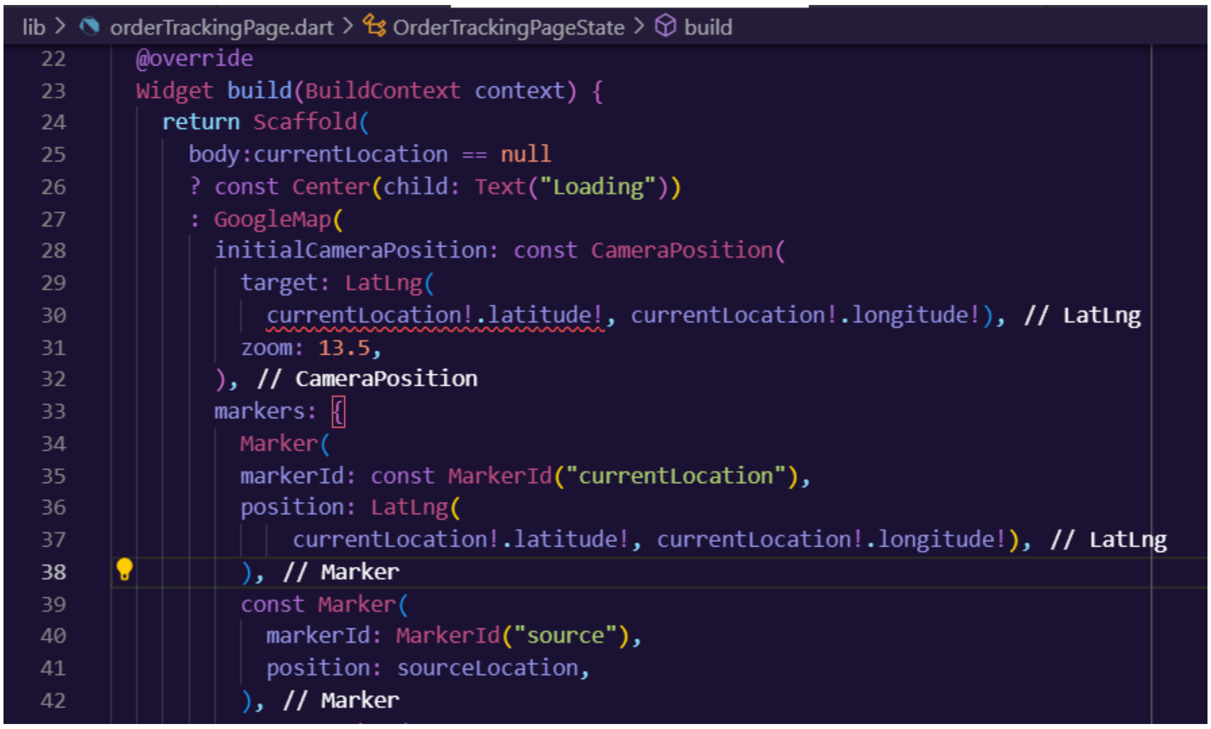
Make sure to call the getCurrentLocation on initState.



lib > orderTrackingPage.dart > OrderTrackingPageState > initState

```dart
}

class OrderTrackingPageState extends State<OrderTrackingPage> {
  @override
  void initState() {
    getPolyPoints();
    getCurrentLocation();
    super.initState();
  }
```

If the currentLocation is null, it shows a loading text. Also, add another marker/pin for the currentLocation as well as change the initial camera position to the current location.

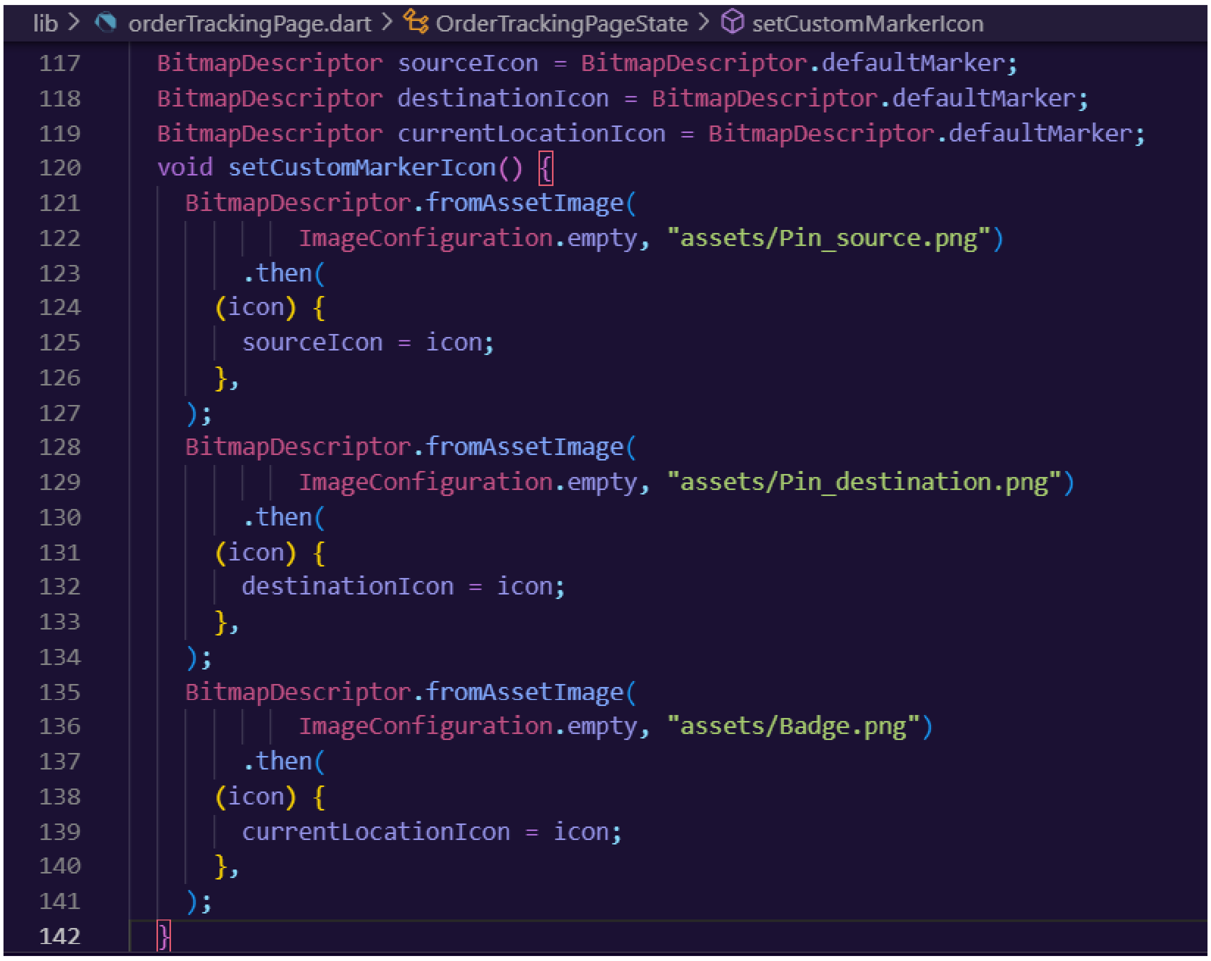


lib > orderTrackingPage.dart > OrderTrackingPageState > build

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    body:currentLocation == null
    ? const Center(child: Text("Loading"))
    : GoogleMap(
      initialCameraPosition: const CameraPosition(
        target: LatLng(
          currentLocation!.latitude!, currentLocation!.longitude!), // LatLng
        zoom: 13.5,
      ), // CameraPosition
      markers: {
        Marker(
        markerId: const MarkerId("currentLocation"),
        position: LatLng(
          currentLocation!.latitude!, currentLocation!.longitude!), // LatLng
        ), // Marker
        const Marker(
          markerId: MarkerId("source"),
          position: sourceLocation,
        ), // Marker
```

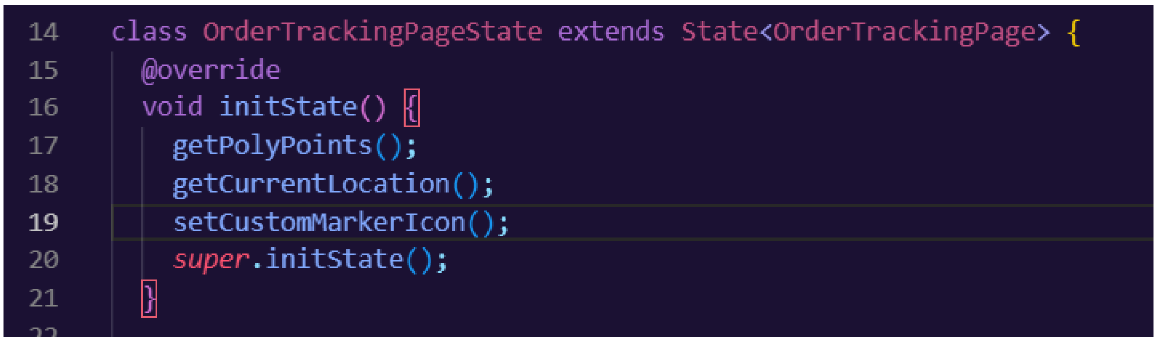
Add custom Marker/Pin

The source, destination, and current location icons are the same. We will use a custom marker/pin for them.



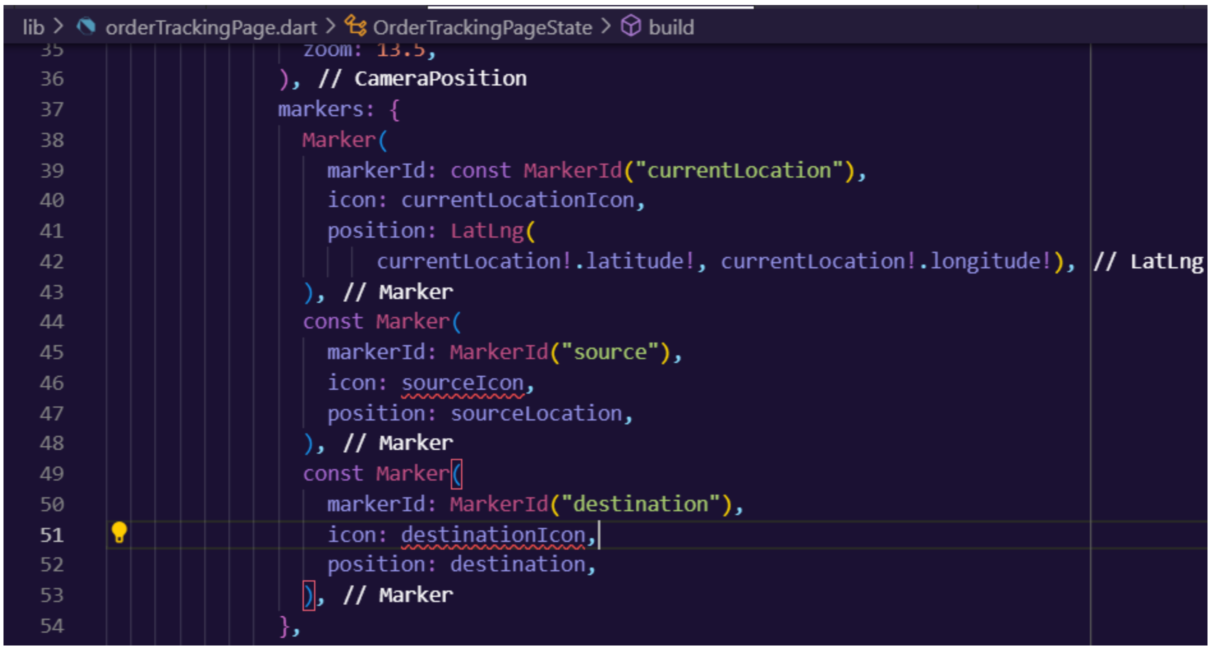
```
lib > orderTrackingPage.dart > OrderTrackingPageState > setCustomMarkerIcon
  BitmapDescriptor sourceIcon = BitmapDescriptor.defaultMarker;
  BitmapDescriptor destinationIcon = BitmapDescriptor.defaultMarker;
  BitmapDescriptor currentLocationIcon = BitmapDescriptor.defaultMarker;
  void setCustomMarkerIcon() {
    BitmapDescriptor.fromAssetImage(
            ImageConfiguration.empty, "assets/Pin_source.png")
        .then(
      (icon) {
        sourceIcon = icon;
      },
    );
    BitmapDescriptor.fromAssetImage(
            ImageConfiguration.empty, "assets/Pin_destination.png")
        .then(
      (icon) {
        destinationIcon = icon;
      },
    );
    BitmapDescriptor.fromAssetImage(
            ImageConfiguration.empty, "assets/Badge.png")
        .then(
      (icon) {
        currentLocationIcon = icon;
      },
    );
  }
```

Call setCustomMarkerIcon on initState



```
class OrderTrackingPageState extends State<OrderTrackingPage> {
  @override
  void initState() {
    getPolyPoints();
    getCurrentLocation();
    setCustomMarkerIcon();
    super.initState();
  }
```

The final touch, on the marker set icon.



lib > orderTrackingPage.dart > OrderTrackingPageState > build

```
                zoom: 13.5,
              ), // CameraPosition
              markers: {
                Marker(
                  markerId: const MarkerId("currentLocation"),
                  icon: currentLocationIcon,
                  position: LatLng(
                      currentLocation!.latitude!, currentLocation!.longitude!), // LatLng
                ), // Marker
                const Marker(
                  markerId: MarkerId("source"),
                  icon: sourceIcon,
                  position: sourceLocation,
                ), // Marker
                const Marker(
                  markerId: MarkerId("destination"),
                  icon: destinationIcon,
                  position: destination,
                ), // Marker
              },
```

Conclusion

This task showed that we can also customize the Google Map in a Flutter app. It helped us understand the logic of showing real time location from Google Map.
Unfortunately, due to the fact that we do not have API Key , we won't be able to test the app and how it works.