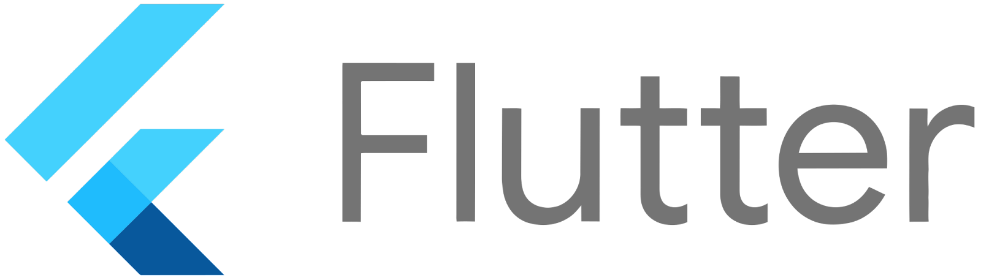
Rerac

1. Downloading Softwares

**Flutter**

Flutter is an open-source UI software development kit by Google. It is used to create cross-platform apps from a single codebase for Android, iOS, Linux, macOS, Windows, Google Fuchsia, and the web.

Flutter, which was first discussed in 2015, was released in May 2017.



**Figure 6.1 : Flutter Logo**

**Installation on macOS**

1. Flutter can be installed by following instruction in the link below:

**macOS install | Flutter**

|  |
| --- |
| <https://docs.flutter.dev/get-started/install/macos> |

The instructions given are:

1. Make sure that the system requirements are met.

* Operating Systems: macOS
* Disk Space: 2.8 GB (excluding disk space for IDE/tools).
* If you are installing on Apple Silicon Mac, make sure that the Rosetta 2 on your devices are up-to-date. Enter the command below to update it to the latest version. Password of your device will be required to install the updates.

|  |
| --- |
| % sudo softwareupdate --install-rosetta --agree-to-license |

1. Installing Flutter SDK

* In the link above, there will be two buttons under the section ‘Get the Flutter FDK’ for downloading Flutter SDK as shown below in Figure 10.1.



**Figure 6.2 : Screenshot of download buttons under ‘Get the Flutter SDK’ section in the official Flutter website.**

* Click the button according to what chip your Mac is using.
* Extract the file in the desired location. For example,

|  |
| --- |
| % cd ~/development  % unzip ~/Downloads/flutter\_macos\_arm64\_3.3.3-stable.zip |

Flutter SDK is now being extracted to your Mac. Below is the output on my Mac.

|  |
| --- |
| % cd ~/development  % unzip ~/Downloads/flutter\_macos\_arm64\_3.3.3-stable.zip  Archive:  /Users/admin/Downloads/flutter\_macos\_arm64\_3.3.3-stable.zip     creating: flutter/    inflating: flutter/CODE\_OF\_CONDUCT.md    inflating: flutter/.ci.yaml    inflating: flutter/LICENSE  **...**  finishing deferred symbolic links:    flutter/bin/cache/artifacts/engine/darwin-x64-profile/FlutterMacOS.framework/Resources -> Versions/Current/Resources  flutter/bin/cache/artifacts/engine/darwin-x64-profile/FlutterMacOS.framework/Versions/Current -> A    flutter/bin/cache/artifacts/engine/darwin-x64-profile/FlutterMacOS.framework/Headers -> Versions/Current/Headers    flutter/bin/cache/artifacts/engine/darwin-x64-profile/FlutterMacOS.framework/Modules -> Versions/Current/Modules    flutter/bin/cache/artifacts/engine/darwin-x64-profile/FlutterMacOS.framework/FlutterMacOS -> Versions/Current/FlutterMacOS    flutter/bin/cache/artifacts/engine/darwin-x64/FlutterMacOS.framework/Resources -> Versions/Current/Resources    flutter/bin/cache/artifacts/engine/darwin-x64/FlutterMacOS.framework/Versions/Current -> A    flutter/bin/cache/artifacts/engine/darwin-x64/FlutterMacOS.framework/Headers -> Versions/Current/Headers    flutter/bin/cache/artifacts/engine/darwin-x64/FlutterMacOS.framework/Modules -> Versions/Current/Modules    flutter/bin/cache/artifacts/engine/darwin-x64/FlutterMacOS.framework/FlutterMacOS -> Versions/Current/FlutterMacOS    flutter/bin/cache/artifacts/engine/darwin-x64-release/FlutterMacOS.framework/Resources -> Versions/Current/Resources  flutter/bin/cache/artifacts/engine/darwin-x64-release/FlutterMacOS.framework/Versions/Current -> A  flutter/bin/cache/artifacts/engine/darwin-x64-release/FlutterMacOS.framework/Headers -> Versions/Current/Headers  flutter/bin/cache/artifacts/engine/darwin-x64-release/FlutterMacOS.framework/Modules -> Versions/Current/Modules  flutter/bin/cache/artifacts/engine/darwin-x64-release/FlutterMacOS.framework/FlutterMacOS -> Versions/Current/FlutterMacOS |

1. Add the ‘flutter’ tool to your path using this command

|  |
| --- |
| % export PATH**=**"$PATH:`pwd`/flutter/bin" |

However, running this command will only temporarily add the path to your current terminal. Check out this link to permanently add the path to your Mac. (This is not required in this project)

**macOS install | Flutter**

|  |
| --- |
| <https://docs.flutter.dev/get-started/install/macos#update-your-path> |

Enter this command in the terminal to check for missing applications

|  |
| --- |
| % flutter doctor |

Below is the output on my personal laptop running on macOS

|  |
| --- |
| % flutter doctor    **╔════════════════════════════════════════════════════════════════════════════╗**  **║                 Welcome to Flutter! - https://flutter.dev                  ║**  **║                                                                            ║**  **║ The Flutter tool uses Google Analytics to anonymously report feature usage ║**  **║ statistics and basic crash reports. This data is used to help improve      ║**  **║ Flutter tools over time.                                                   ║**  **║                                                                            ║**  **║ Flutter tool analytics are not sent on the very first run. To disable      ║**  **║ reporting, type 'flutter config --no-analytics'. To display the current    ║**  **║ setting, type 'flutter config'. If you opt out of analytics, an opt-out    ║**  **║ event will be sent, and then no further information will be sent by the    ║**  **║ Flutter tool.                                                              ║**  **║                                                                            ║**  **║ By downloading the Flutter SDK, you agree to the Google Terms of Service.  ║**  **║ Note: The Google Privacy Policy describes how data is handled in this      ║**  **║ service.                                                                   ║**  **║                                                                            ║**  **║ Moreover, Flutter includes the Dart SDK, which may send usage metrics and  ║**  **║ crash reports to Google.                                                   ║**  **║                                                                            ║**  **║ Read about data we send with crash reports:                                ║**  **║ https://flutter.dev/docs/reference/crash-reporting                         ║**  **║                                                                            ║**  **║ See Google's privacy policy:                                               ║**  **║ https://policies.google.com/privacy                                        ║**  **╚════════════════════════════════════════════════════════════════════════════╝**    Running "flutter pub get" in flutter\_tools...                       8.5s    Doctor summary (to see all details, run flutter doctor -v):    [✓] Flutter (Channel stable, 3.3.3, on macOS 12.4 21F79 darwin-arm, locale en-GB)    [✗] Android toolchain - develop for Android devices  **✗ Unable to locate Android SDK.**  **Install Android Studio from: https://developer.android.com/studio/index.html**  **On first launch it will assist you in installing the Android SDK components.**  **(or visit https://flutter.dev/docs/get-started/install/macos#android-setup for detailed instructions).**  **If the Android SDK has been installed to a custom location, please use**  **`flutter config --android-sdk` to update to that location.**    [✗] Xcode - develop for iOS and macOS  **✗ Xcode installation is incomplete; a full installation is necessary for iOS development.**  **Download at: https://developer.apple.com/xcode/download/**  **Or install Xcode via the App Store.**  **Once installed, run:**  **sudo xcode-select --switch /Applications/Xcode.app/Contents/Developer**  **sudo xcodebuild -runFirstLaunch**  **✗ CocoaPods not installed.**  **CocoaPods is used to retrieve the iOS and macOS platform side's plugin code that responds to your plugin usage on the Dart side.**  **Without CocoaPods, plugins will not work on iOS or macOS.**  **For more info, see https://flutter.dev/platform-plugins**  **To install see https://guides.cocoapods.org/using/getting-started.html#installation for instructions.**    [✓] Chrome - develop for the web    [!] Android Studio (not installed)    [✓] Connected device (2 available)    [✓] HTTP Host Availability    ! Doctor found issues in 3 categories. |

The issues can be resolved by installing the missing applications which, in this case, are Android toolchain, Xcode, and Android Studio. Links below are for installing those applications.

|  |  |
| --- | --- |
| **Android toolchain** | Can be installed through Android Studio application |
| **Xcode** | <https://apps.apple.com/sg/app/xcode/id497799835?mt=12> |
| **Android Studio** | <https://developer.android.com/studio> |

When all are installed, the summary from running ‘flutter doctor’ will be shown below.

|  |
| --- |
| Doctor summary (to see all details, run flutter doctor -v):  [✓] Flutter (Channel stable, 3.3.3, on macOS 12.6 21G115 darwin-arm, locale en-GB)  [✓] Android toolchain - develop for Android devices (Android SDK version 33.0.0)  [✓] Xcode - develop for iOS and macOS (Xcode 14.0.1)  [✓] Chrome - develop for the web  [✓] Android Studio (version 2021.3)  [✓] Connected device (2 available)  [✓] HTTP Host Availability  • No issues found! |

**Installation on Windows**

1. Flutter can be installed by following instruction in the link below:

**Windows install | Flutter**

|  |
| --- |
| <https://docs.flutter.dev/get-started/install/windows> |

1. To install and run Flutter, your development environment must meet these minimum requirements:

* Operating Systems: Windows 10 or later (64-bit), x86-64 based.
* Disk Space: 1.64 GB (excluding disk space for IDE/tools).
* Tools: Flutter depends on these tools being available in your environment.
* [Windows PowerShell 5.0](https://docs.microsoft.com/en-us/powershell/scripting/install/installing-windows-powershell) or newer (this is pre-installed with Windows 10)

Install PowerShell using Winget (Recommended)

|  |
| --- |
| $ winget search Microsoft.PowerShell |

Output from my personal laptop:

|  |
| --- |
| Microsoft Windows [Version 10.0.22000.978]  (c) Microsoft Corporation. All rights reserved.  C:\Users\KumarDevadharshiniES>winget search Microsoft.PowerShell  The `msstore` source requires that you view the following agreements before using.  Terms of Transaction: https://aka.ms/microsoft-store-terms-of-transaction  The source requires the current machine's 2-letter geographic region to be sent to the backend service to function properly (ex. "US").  Do you agree to all the source agreements terms?  [Y] Yes  [N] No: Y  Name               Id                           Version Source  ---------------------------------------------------------------  PowerShell         Microsoft.PowerShell         7.2.6.0 winget  PowerShell Preview Microsoft.PowerShell.Preview 7.3.8.0 winget |

Install PowerShell or PowerShell Preview using the id parameter

|  |
| --- |
| winget install --id Microsoft.Powershell --source winget  winget install --id Microsoft.Powershell.Preview --source winget |

Output from my personal laptop:

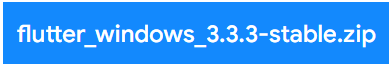
|  |
| --- |
| C:\Users\KumarDevadharshiniES>winget install --id Microsoft.Powershell --source winget  Found PowerShell [Microsoft.PowerShell] Version 7.2.6.0  This application is licensed to you by its owner.  Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.  Downloading https://github.com/PowerShell/PowerShell/releases/download/v7.2.6/PowerShell-7.2.6-win-x64.msi    ██████████████████████████████   102 MB /  102 MB  Successfully verified installer hash  Starting package install...  Successfully installed  C:\Users\KumarDevadharshiniES>winget install --id Microsoft.Powershell.Preview --source winget  Found PowerShell Preview [Microsoft.PowerShell.Preview] Version 7.3.8.0  This application is licensed to you by its owner.  Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.  Downloading https://github.com/PowerShell/PowerShell/releases/download/v7.3.0-preview.8/PowerShell-7.3.0-preview.8-win-x64.msi    ██████████████████████████████   102 MB /  102 MB  Successfully verified installer hash  Starting package install...  Successfully installed |

* Git for Windows 2.x, with the Use Git from the Windows Command Prompt option.

If Git for Windows is already installed, make sure you can run git commands from the command prompt or PowerShell.

1. Installing Flutter SDK

* Download the following installation bundle to get the latest stable release of the Flutter:



**Figure 6.3 : Screenshot of download button for Windows stable release**

* Extract the zip file and place the contained flutter in the desired installation location for the Flutter SDK
* Do not install flutter to a path that contains special characters or spaces
* Do not install flutter in a directory like C:\ Program Files\ that requires elevated privileges.
* If you don’t want to install a fixed version of the installation bundle, you can skip steps 1 and 2. Instead, get the source code from the [Flutter repo](https://github.com/flutter/flutter) on GitHub, and change branches or tags as needed. For example:

|  |
| --- |
| git clone https://github.com/flutter/flutter.git -b stable |

1. Update your Path

* If you wish to run Flutter commands in the regular Windows console, take these steps to add Flutter to the PATH environment variable:
* From the Start search bar, enter ‘env’ and select Edit environment variables for your account.
* Under User variables check if there is an entry called Path:
  + If the entry exists, append the full path to flutter\bin using ; as a separator from existing values.
  + If the entry doesn’t exist, create a new user variable named Path with the full path to flutter\bin as its value.

* You have to close and reopen any existing console windows for these changes to take effect.
* As of Flutter’s 1.19.0 dev release, the Flutter SDK contains the dart command alongside the flutter command so that you can more easily run Dart command-line programs. Downloading the Flutter SDK also downloads the compatible version of Dart, but if you’ve downloaded the Dart SDK separately, make sure that the Flutter version of ‘dart’ is first in your path, as the two versions might not be compatible. The following command tells you whether the flutter and dart commands originate from the same bin directory and are therefore compatible.

|  |
| --- |
| where flutter dart    C:\path-to-flutter-sdk\bin\flutter    C:\path-to-flutter-sdk\bin\flutter.bat    C:\path-to-dart-sdk\bin\dart.exe        :: this should go after `C:\path-to-flutter-sdk\bin\` commands    C:\path-to-flutter-sdk\bin\dart    C:\path-to-flutter-sdk\bin\dart.bat |

As shown above, the command dart from the Flutter SDK doesn’t come first. Update your path to use commands from ‘C:\path-to-flutter-sdk\bin\’  before commands from ‘C:\path-to-dart-sdk\bin\’ (in this case). After restarting your shell for the change to take effect, running the where command again should show that the flutter and dart commands from the same directory now come first.

|  |
| --- |
| where flutter dart    C:\dev\src\flutter\bin\flutter    C:\dev\src\flutter\bin\flutter.bat    C:\dev\src\flutter\bin\dart    C:\dev\src\flutter\bin\dart.bat    C:\dev\src\dart-sdk\bin\dart.exe |

However, if you are using PowerShell, in it ‘where’ is an alias of Where-Object command, so you need to use where.exe instead.

|  |
| --- |
| PS where.exe flutter dart |

To learn more about the dart command, run ‘dart -h’ from the command line, or see the dart tool page below:

**Dart run**

|  |
| --- |
| <https://dart.dev/tools/dart-run> |

1. Run ‘flutter doctor’

* From a console window that has the Flutter directory in the path (see above), run the following command to see if there are any platform dependencies you need to complete the setup:

|  |
| --- |
| $ flutter doctor |

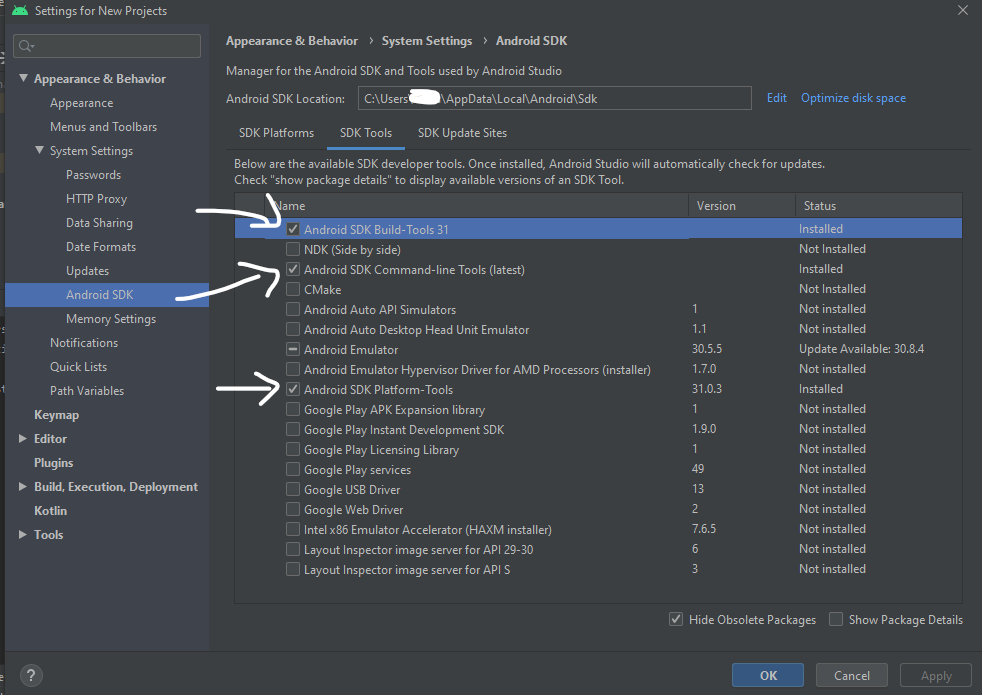
This command checks your environment and displays a report of the status of your Flutter installation. Check the output carefully for other software you might need to install or further tasks to perform (shown in below).

|  |
| --- |
| $ flutter doctor  Doctor summary (to see all details, run flutter doctor -v):  [✓] Flutter (Channel stable, 3.3.3, on on Microsoft Windows [Version 10.0.22000.978], locale en-SG)  [✓] Android toolchain - develop for Android devices (Android SDK version 33.0.0)  [✓] Chrome - develop for the web  [✓] Visual Studio - develop for Windows (Visual Studio Community 2019 16.10.4)  [✓] Android Studio (version 2021.3)  [✓] VS Code (version 1.70.1)  [✓] Connected device (3 available)  [✓] HTTP Host Availability  • No issues found! |

However, you might face the issue of ‘cmdline-tools component is missing’ as shown below:

|  |
| --- |
| $ flutter doctor  Doctor summary (to see all details, run flutter doctor -v):  [✓] Flutter (Channel stable, 3.3.3, on on Microsoft Windows [Version 10.0.22000.978], locale en-SG)  [!] Android toolchain - develop for Android devices (Android SDK version 33.0.0)  **✗ cmdline-tools component is missing**  **Run `path/to/sdkmanager –install “cmdline-tools;latest”`**  **See** [**https://developer.android/com/studio/command-line**](https://developer.android/com/studio/command-line) **for more details.**  **✗ Android license status unknown.**  **Run `flutter doctor -–android-licenses` to accept the SDK licenses.**  **See** [**https://flutter.dev/docs/get-started/install/windows#android-setup**](https://flutter.dev/docs/get-started/install/windows#android-setup) **for more details.**  [✓] Chrome - develop for the web  [✓] Visual Studio - develop for Windows (Visual Studio Community 2019 16.10.4)  [✓] Android Studio (version 2021.3)  [✓] VS Code (version 1.70.1)  [✓] Connected device (3 available)  [✓] HTTP Host Availability  ! Doctor found issues in 1 category |

The missing components can be resolved by installing Android SDK command-line tools (latest), Android Build Tools 31, Android SDK platform tools (latest version 31) in the Android studio application as shown below:



**Figure 6.4 : Screenshot of Android Studio’s System Settings with arrows pointing at essential components to be downloaded.**

After downloading those components:

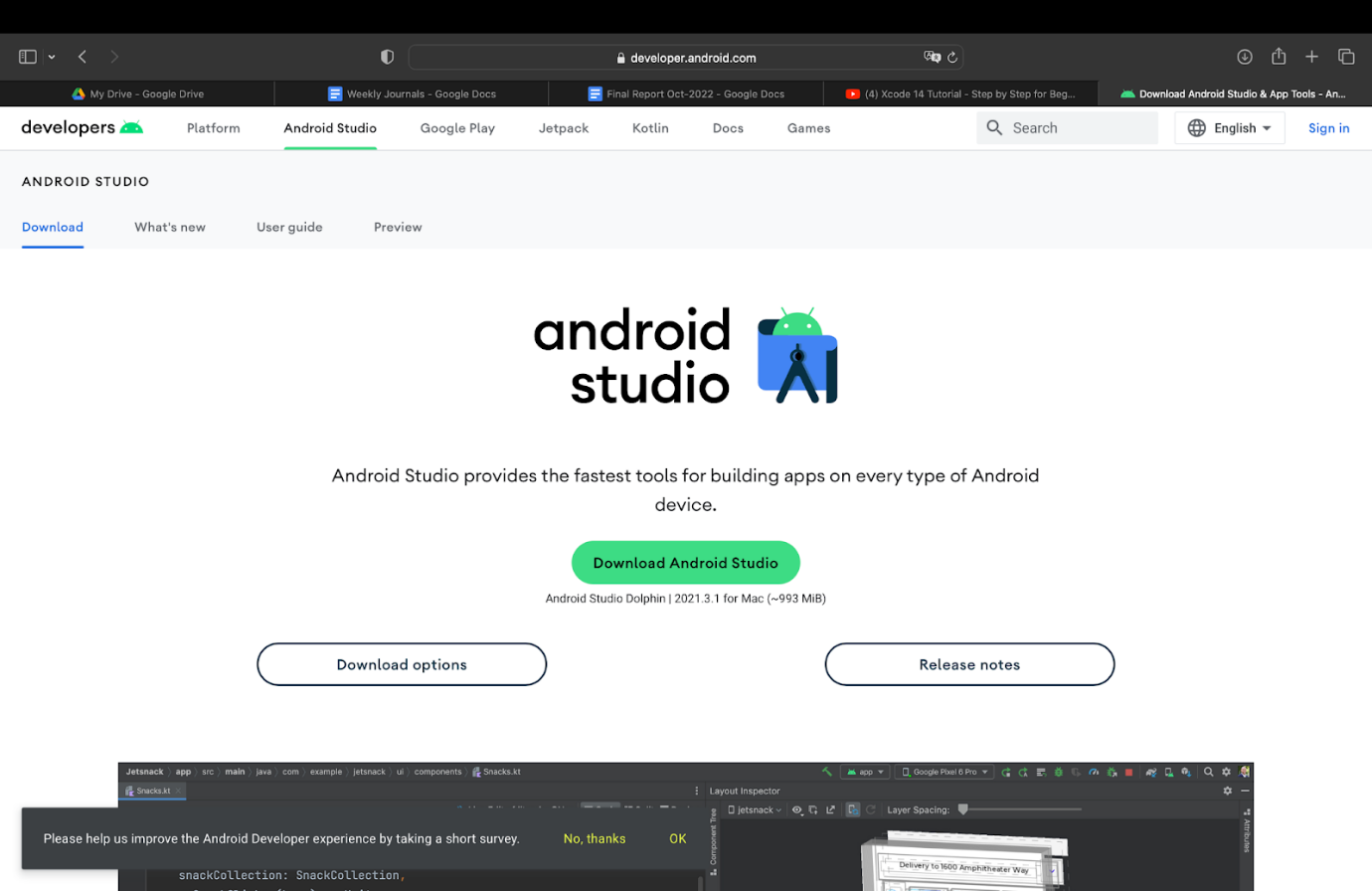
1. Run “flutter doctor –android licenses” to accept the terms and conditions
2. After that run “flutter doctor” again to check whether it has been ticked off or not

### Installation of Android Studio

The installation can be done through the link below:

|  |
| --- |
| <https://developer.android.com/studio> |

The link will detect the operating system your laptop is on and will give the correct button to download the application. Follow the instructions given accordingly and the application is now ready to use.



**Figure 6.7 : Download button under the Android Studio headings and its description**

## Google

Including Google Maps in your mobile app may really take your consumers to another dimension. This report includes instructions for using the official plugin for integrating Google Maps into your Flutter project.

* + 1. Creating a project in Google Cloud Platform

The first step is to start a new project in Google Developers Console. This is necessary since you will require APIs to incorporate Google Maps into your program.

1. Enabling the Maps API

Once the project is created, you’ll have to enable the Maps API SDK for both Android and iOS.

1. Click the Library menu on the left
2. Search for “Maps SDK”
3. Click Maps SDK for iOS and then click Enable
4. Click Maps SDK for Android and then click Enable

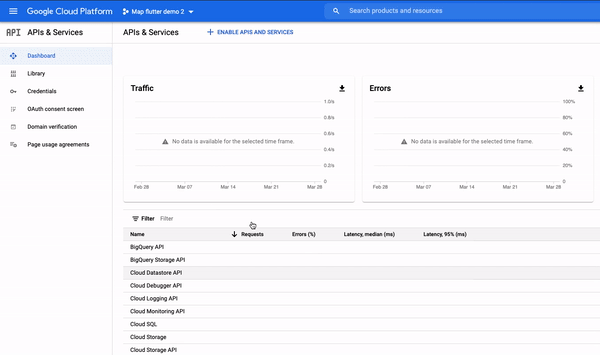


Figure 6.8: Generating Google API key

1. Generating and Restricting API keys

It should be noted that access to the Google Maps APIs is not free. As a result, you must generate a set of API keys for it and limit its access to minimise the possibility of unwanted usage. This is how you generate the keys.

1. Click on the Credentials menu on the left
2. Click the + CREATE CREDENTIALS button
3. Select the API key option

I would strongly advise putting some constraints in place after establishing the API keys. For example, if you only want to utilize the API keys for a map, you should restrict them to the Maps service.

There are two types of restrictions:

1. Application limitations allow you to specify which apps should have access to this key (i.e., Android or iOS). You can pick the relevant option to ensure that the key you generated is only valid for that platform.
2. API limits allow you to specify which services are available with this key. If you only wanted to use the map, choose Maps API from the options.

## Adding Google Maps Flutter plugin as a dependency

The easiest way to get started with Flutter is to use the flutter command line tool to create all the required code for a simple getting started experience.

Command to create project in Flutter

|  |
| --- |
| $ flutter create [project name] |

Then next you need to introduce the Google Maps flutter plugin by running the following command from the project directory.

Command line

|  |
| --- |
| $ flutter pub add google\_maps\_flutter |

## Configuring Android minSDK

To use Google Maps SDK on Android requires setting the minSDK to 21 and targetSDK to 31. Modify the android/app/build.gradle as follows.

|  |
| --- |
| android {      defaultConfig {          applicationId "com.example.google\_maps\_in\_flutter"          minSdkVersion 21                    // Set to 21          targetSdkVersion 31                // Set to 31          versionCode flutterVersionCode.toInteger()          versionName flutterVersionName      }  } |

## Adding an API key for an Android app

It all comes down to API keys. To integrate Google Maps in your Flutter app, create an API project with the Google Map Platforms by following the Map SDK for Android using API key and Map SDK for iOS using API key instructions. With your API keys in hand, follow the steps below to configure Android apps.

Edit the AndroidManifest.xml file in android/app/src/main to add an API key to the Android app. Inside the application node, add a single meta-data entry holding the API key established in the previous step.

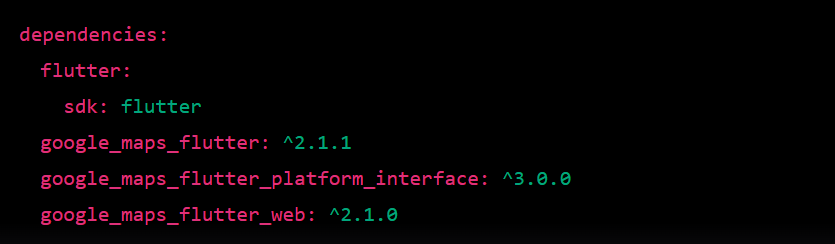
|  |
| --- |
| <manifest xmlns:android="http://schemas.android.com/apk/res/android"      package="com.example.google\_maps\_in\_flutter">      <application          android:label="google\_maps\_in\_flutter"          android:icon="@mipmap/ic\_launcher">          <!-- TODO: Add your Google Maps API key here -->          <meta-data android:name="com.google.android.geo.API\_KEY"                 android:value="YOUR-KEY-HERE"/>          <activity              android:name=".MainActivity"              android:launchMode="singleTop"              android:theme="@style/LaunchTheme"              android:configChanges="orientation|keyboardHidden|keyboard|screenSize|smallestScreenSize|locale|layoutDirection|fontScale|screenLayout|density|uiMode"              android:hardwareAccelerated="true"              android:windowSoftInputMode="adjustResize">              <meta-data                android:name="io.flutter.embedding.android.NormalTheme"                android:resource="@style/NormalTheme"                />              <meta-data                android:name="io.flutter.embedding.android.SplashScreenDrawable"                android:resource="@drawable/launch\_background"                />              <intent-filter>                  <action android:name="android.intent.action.MAIN"/>                  <category android:name="android.intent.category.LAUNCHER"/>              </intent-filter>          </activity>          <meta-data              android:name="flutterEmbedding"              android:value="2" />      </application>  </manifest> |

1. Google Maps

This section provided instructions on how to integrate Google Maps into a Flutter application.

Step 1: Add dependencies

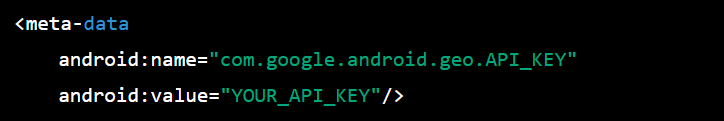
Open the pubspec.yaml file in your Flutter project and add the following dependencies:



Then, run the flutter pub get command to install the dependencies.

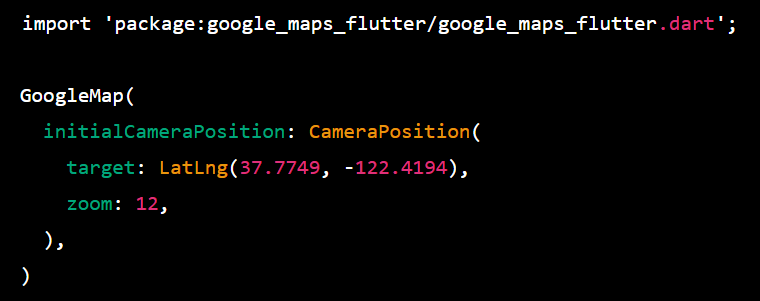
Step 2: Set up the Google Maps API key

To use Google Maps in your application, you need to provide your Google Maps API key in your project. Add the following code in your AndroidManifest.xml file:



Step 3: Add a GoogleMap widget to your screen

Open the Dart file where you want to display the map and add the following code:



This will display a map centered on San Francisco with a zoom level of 12. You can customize the location and zoom level to suit your needs.

Step 4: Run the application

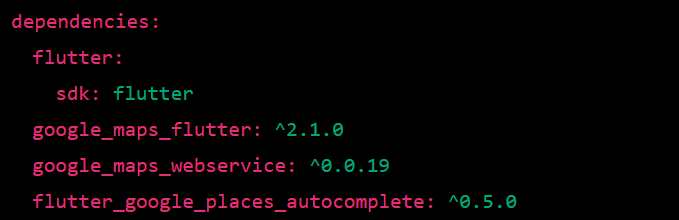
Run the flutter run command to launch the application on your device or emulator. You should see a map with the specified location displayed on the screen.

1. Google Places API and Autocomplete

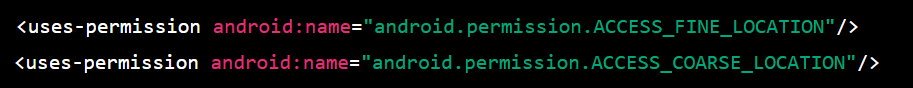
Google Places API and Autocomplete are powerful tools for developers to integrate location-based services into their applications. These tools allow users to search for places, such as restaurants, stores, and other points of interest, based on their location. Flutter, a popular open-source framework for building high-performance mobile applications, provides easy integration of these tools into mobile applications.

Implementation

Step 1: Add the necessary dependencies to the pubspec.yaml file. The following dependencies are required:



Step 2: In the AndroidManifest.xml file, add the following permissions:



Step 3: In the GCP project, enable the following APIs:

* Maps SDK for Android
* Places API

Step 4: Create the UI

Create a new SearchScreen widget that contains two TextFields, one for the starting location and the other for the ending location. Use the google\_maps\_flutter package to display a GoogleMap widget that shows the current location and the selected locations.

Step 5: Implement Autocomplete

Use the google\_maps\_webservice package to implement the Autocomplete feature. We will use the GooglePlace class to get Autocomplete predictions based on user input. We will also use the Details class to get detailed information about a place.

Step 6: Display the Autocomplete Suggestions

As the user types in the search field, we will use the google\_maps\_webservice package to fetch the Autocomplete predictions based on their input. We will use the ListView widget to display the predictions as a scrollable list of ListTiles.

Step 7: Get the Location Details

When the user selects a prediction from the list, we will use the Details class to get the detailed information about the place. We will display the location on the GoogleMap widget and set the corresponding TextFormField with the selected location's name.

Step 8: Handle Errors

In case of any error during the API request, we will display an error message to the user.

1. Geofencing

The Geofencing Service package is a Flutter plugin that provides geofencing capabilities for mobile applications. Geofencing allows applications to define a virtual perimeter around a physical location and track when a device enters or exits that area. This technology can be used to trigger actions based on a user's location, such as sending notifications, activating or deactivating features, and logging user data.

Features

The Geofencing Service package provides the following features:

1. Geofencing: The package provides the ability to create and manage geofences. A geofence is a virtual boundary around a geographical location. The package allows you to create a geofence by specifying the center coordinates and radius of the boundary. You can also customize the geofence's name, duration, and transition type.

2. Geofence Events: The package provides callbacks for geofence events such as when the device enters or exits a geofence, and when a geofence is added, removed or changed.

3. Background Processing: The package supports background processing of geofence events, even when the application is not running. This feature is important for triggering actions based on location changes, such as sending notifications or updating a user's location history.

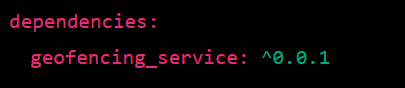
4. Error Handling: The package provides error handling for common geofencing issues such as permission errors and location services errors.

5. Cross-platform Support: The package is cross-platform and supports both Android and iOS.

Installation and Setup

To use the Geofencing Service package in your Flutter application, follow these steps:

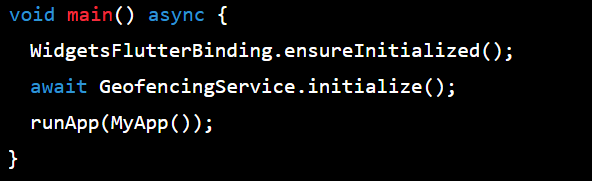
1. Add the package to your project's pubspec.yaml file:



2. Import the package in your Dart code:

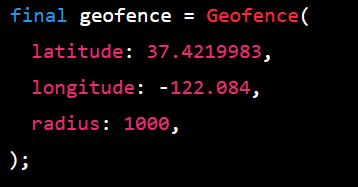


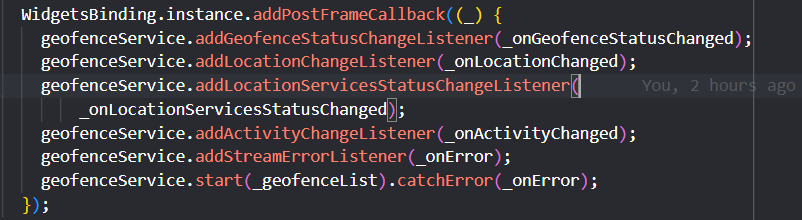
3. Initialize the package by calling the initialize() method in your application's main function:



To create a geofence using the Geofencing Service package, follow these steps:

1. Create a Geofence object by specifying the center coordinates and radius of the geofence:





The WidgetsBinding.instance.addPostFrameCallback method registers a callback that is executed after the framework finishes building the current frame. This ensures that the geofenceService is fully initialized before registering the listeners and starting geofencing.

The addGeofenceStatusChangeListener method registers a listener that receives updates when the geofence status changes, such as when the user enters or leaves a geofence.

The addLocationChangeListener method registers a listener that receives updates when the user's location changes.

The addLocationServicesStatusChangeListener method registers a listener that receives updates when the status of the device's location services changes.

The addActivityChangeListener method registers a listener that receives updates when the user's activity changes, such as when the user starts or stops walking or driving.

The addStreamErrorListener method registers a listener that receives updates when an error occurs in the geofencing stream.

The start method starts geofencing with the provided \_geofenceList, which is a list of Geofence objects that define the geofences to monitor. If an error occurs during geofencing, the \_onError callback is executed.

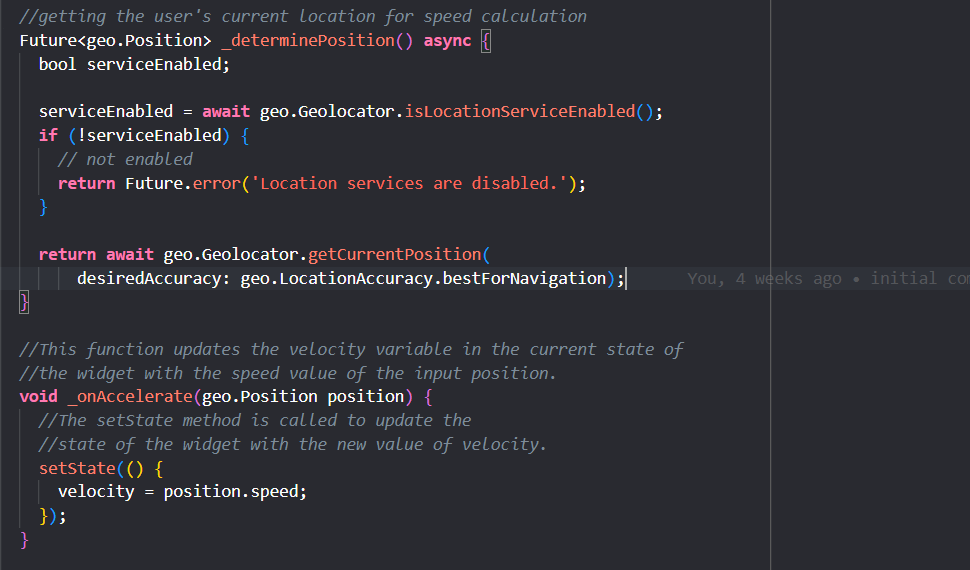
Overall, this code initializes and starts the geofencing service in a Flutter app and registers several listeners to receive updates about the user's location and activity. It is an essential part of any geofencing implementation and allows the app to provide location-based services and notifications to the user.

1. Speed

The addStreamErrorListener method registers a listener that receives updates when an error occurs in the geofencing stream.

This function uses this package:





The \_determinePosition() method is a Future that returns a geo.Position object after determining the current location of the device. It first checks whether location services are enabled using the isLocationServiceEnabled() method of the geo.Geolocator class. If location services are not enabled, the method returns a Future.error with a message indicating that location services are disabled. If location services are enabled, the method retrieves the current position using the getCurrentPosition() method of the geo.Geolocator class with a desiredAccuracy parameter set to LocationAccuracy.bestForNavigation.

The \_onAccelerate(geo.Position position) method is called when the device accelerates or changes velocity. It updates the state of the widget with the new velocity value retrieved from the position.speed property of the geo.Position object. The setState() method is used to update the state of the widget and trigger a rebuild of the UI.

1. Login Page

The login page contains two text fields for email and password, a "Forgot Password" button, a "Login" button, and a "New User? Create a new account" button.

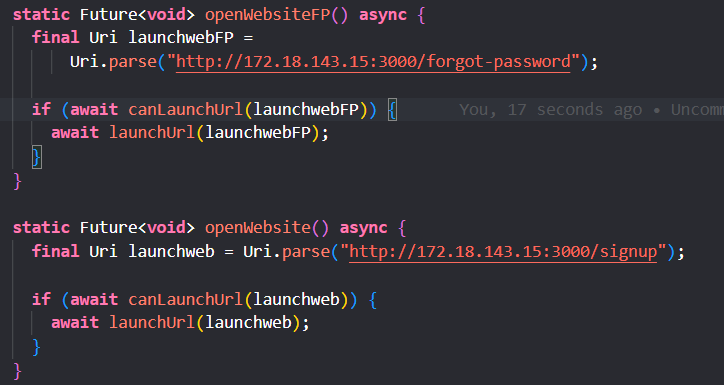
The login page's logic involves validating the user's email and password and navigating to the home page if they're valid. The getUsers() function retrieves a list of users from the backend and populates the userData list.

The validateEmail() function uses the Email\_validator package to check if the user's email address is valid. The validatePassword() function validates the user's email and password by comparing them to the values stored in userData.

If either the email or password is invalid, the \_errorText or \_errorText2 strings will have an error message. The login button will only navigate to the home page if both \_errorText and \_errorText2 are null, meaning both email and password fields are valid.

This file also imports several packages, including http, url\_launcher, flutter/material.dart, email\_validator, and local Dart files like NavBar.dart and globals.dart.

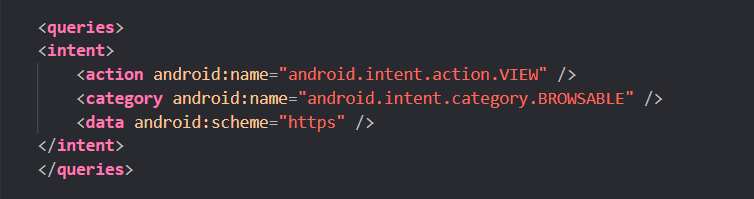
It also redirects the person to the website to sign up or access the forgot password function. It uses the url\_launcher package to do so.



The openWebsiteFP method opens the "forgot password" page of a website hosted at **http://172.18.143.15:3000**. It first creates a Uri object that represents the URL of the web page to open. Then it uses the canLaunchUrl function to check if the device has a web browser app installed that can open the URL. If the function returns true, meaning a web browser app is available, the launchUrl function is called to open the URL in the web browser.

The openWebsite method is similar to openWebsiteFP, but it opens the "signup" page of the same website instead of the "forgot password" page.

However, for the website to be able to show up when clicked on the button, there needs to be more changes made.



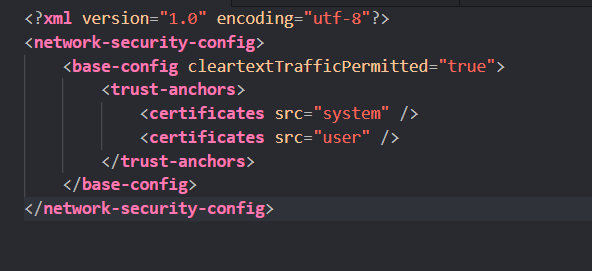
Include this code above application, at AndroidManifest.xml file which is under android.

This is an XML code snippet that defines an intent filter for an Android app. The filter specifies that the app can handle the VIEW action with a BROWSABLE category for URLs that use the HTTPS scheme.

In other words, the intent filter is saying that the app can handle requests to view web pages with URLs that use HTTPS.

Create a new file under \app\src\main\res\xml\ and name it as network\_security\_config.xml

Inside the xml file:



This is an XML configuration file for Android's Network Security Configuration. It specifies the security configuration for network connections made by the app.

The base-config element contains the base configuration for all network connections made by the app. The cleartextTrafficPermitted attribute set to true allows unencrypted HTTP traffic for all domains, which is not recommended from a security standpoint.

The trust-anchors element specifies the trusted root certificates for the app. The certificates element with src="system" indicates that the app trusts the system's root certificates, and src="user" indicates that the app trusts any additional certificates installed by the user.

Overall, this configuration file is allowing unencrypted HTTP traffic, which is not recommended. It's important to ensure that network connections are secured with HTTPS or other encryption methods to protect user data and prevent unauthorized access.

1. Notifications

'flutter\_local\_notifications,' enables developers to send local notifications in their Flutter applications. This technical report explains the process of importing and using the 'flutter\_local\_notifications' package in a Flutter project.

Step 1: Add 'flutter\_local\_notifications' Package to pubspec.yaml

To use the 'flutter\_local\_notifications' package in a Flutter project, we need to add it as a dependency in the pubspec.yaml file. We can add the following lines of code to the file:



Step 2: Install the Package

After adding the 'flutter\_local\_notifications' package to the pubspec.yaml file, we need to run the 'flutter pub get' command to install the package. We can run this command either from the terminal or by clicking on the 'Pub get' button in the Android Studio or VS Code editor.

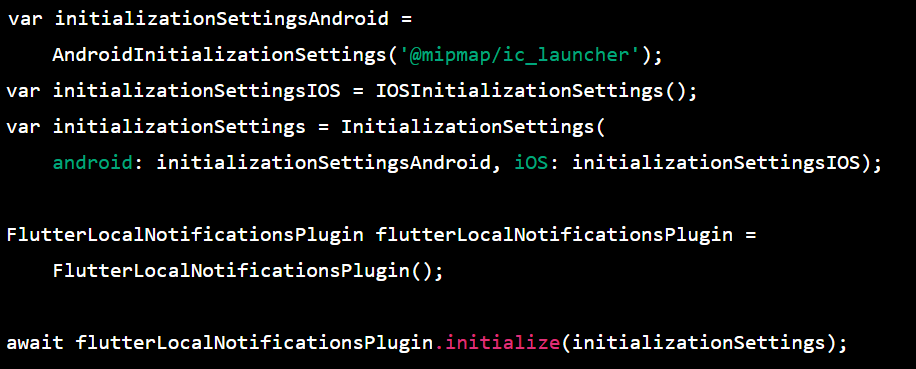
Step 3: Import the Package

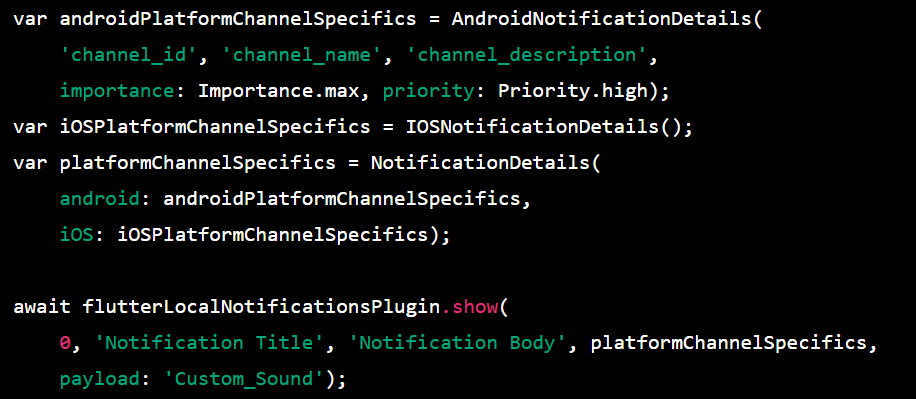
Once the package is installed, we can import it into our Dart code. We can do this by adding the following line of code at the top of our Dart file:



Step 4: Use the Package

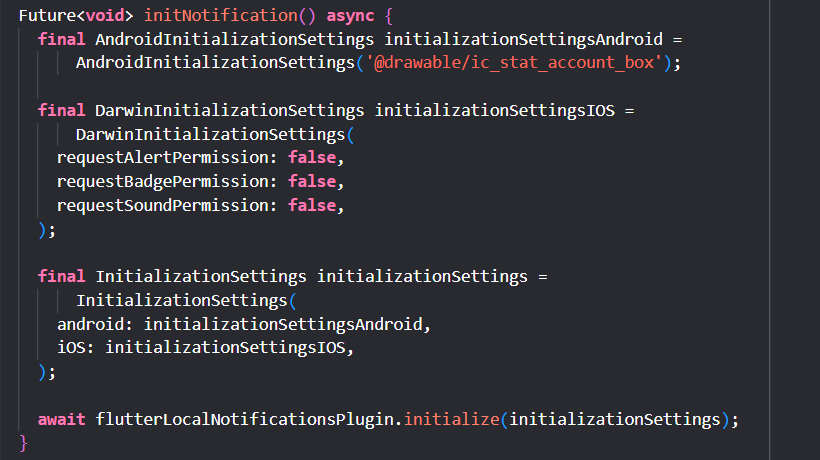
After importing the 'flutter\_local\_notifications' package, we can use its functionalities in our Flutter application. We can create an instance of the 'FlutterLocalNotificationsPlugin' class to use the package's functionalities. For example, to display a local notification, we can use the following code:

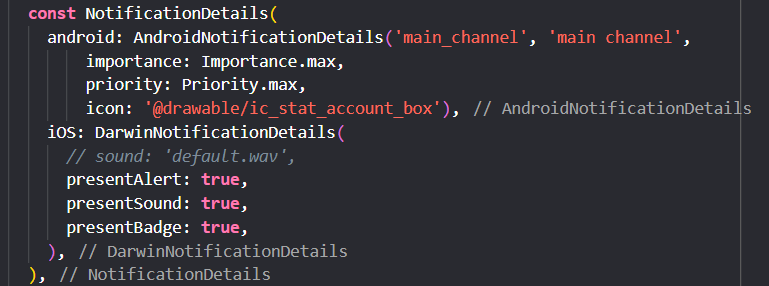




However, for this project we only did application that is compatible for android, iOSNotificationDetails() and iOSPlatformChannelSpecifics won’t work.

Instead, change it to DarwinNotificationDetails() and DarwinInitializationSettings().





For the notifications to work without an error, you need to initialise the timezones at the void initstate() function:



1. Location Information

The purpose of this code is to display various data related to the risk analysis of a specific location. This is achieved through a Flutter app that shows data on a sliding panel and a Google Map.

The code starts with importing the necessary packages required for the project, including http, Flutter, Google Maps, and sliding\_up\_panel. It then defines a StatefulWidget named Blk8 and its associated state class named Blk8State.

The stated location class has a variety of methods that are responsible for retrieving and displaying the data. These include \_getWaypoint, \_getRisk, and getColor.

The \_getWaypoint method retrieves waypoint data from a server and updates the state of the widget with the retrieved data. Similarly, the \_getRisk method retrieves risk data from the server and updates the state of the widget.

The getColor method takes a risk value as input and returns a color corresponding to the level of risk. If the risk value is "Low Risk", the color is green. If it is "Medium Risk", the color is orange. If it is "High Risk", the color is red.

The stated location class also contains a method for building the widget tree, named build. In this method, the \_getWaypoint and \_getRisk methods are called, and the retrieved data is stored in the waypointData and riskData variables, respectively. The blk8Risk variable is assigned the value of the risk value for the location, and the \_chartData variable is initialized to a List of RiskData objects.

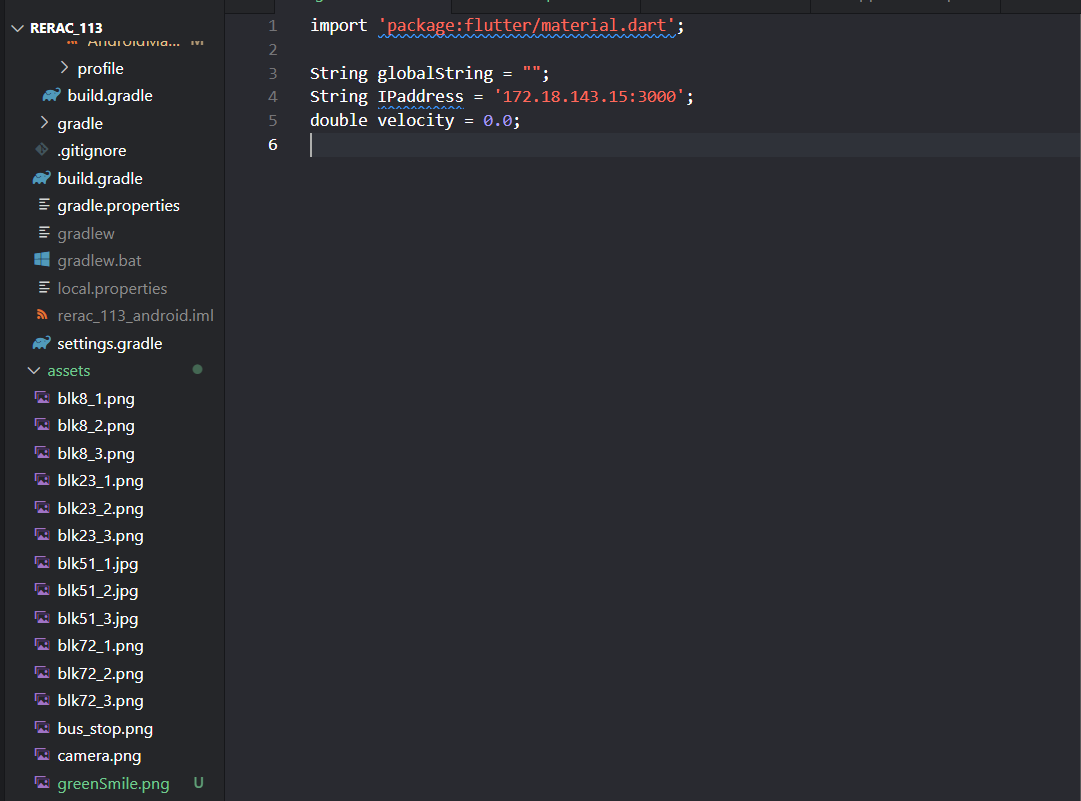
The build method returns a Scaffold widget that contains an AppBar and a SlidingUpPanel widget. The AppBar has a back button that navigates back to the Home screen. The SlidingUpPanel widget contains a panel that displays various data related to the risk analysis of the location.

The panel contains a Text widget that displays the location name, a Text widget that displays the current risk value, another Text widget that displays the average risk value for the day, an SfCartesianChart widget that displays a line chart showing hourly risk analysis, and an ImageSlideshow widget that displays images of the location.

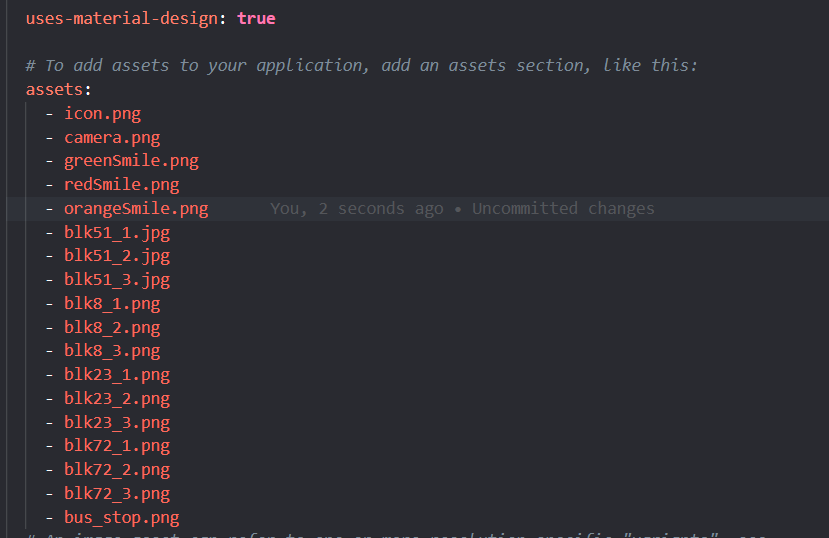
Finally, the panel is collapsed, and a Container widget is displayed instead. The color of the Container widget corresponds to the level of risk. If the risk value is between 1 and 3, the color is green. If it is between 4 and 7, the color is orange. If it is between 8 and 10, the color is red. If it is any other value, the color is grey.

1. Icons and Images

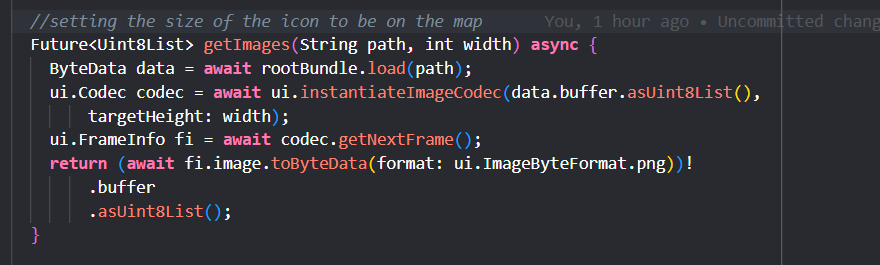
To be able to use images of our choice, need to create a folder ‘assets’ inside the workspace.



After this, you need to declare them under pubspec.yaml file.



This is a Flutter function that loads an image from a file located in the app's assets directory and resizes it to the specified width.



Here's a breakdown of what the function does:

It takes two parameters: path, which is the path of the image file in the app's assets directory, and width, which is the desired width of the resized image.

It loads the image file into memory using the rootBundle.load() method, which returns a ByteData object that represents the image file as a stream of bytes.

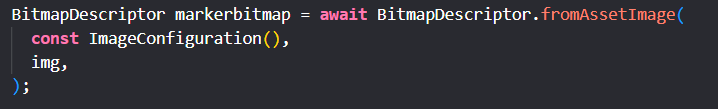
It creates an ui.Codec object by calling ui.instantiateImageCodec(), passing in the image bytes from the ByteData object and the desired targetHeight (which is calculated from the width parameter).

It calls codec.getNextFrame() to decode the first frame of the image and retrieve a ui.FrameInfo object.

It calls fi.image.toByteData() to convert the decoded image into a byte array in PNG format and returns it as a Uint8List.

Overall, this function loads an image from the app's assets directory, resizes it to the desired width, and returns it as a byte array in PNG format.

This code uses the BitmapDescriptor.fromAssetImage method to load an image asset and create a BitmapDescriptor object from it. A BitmapDescriptor is used to represent an image that can be used as a marker icon in a Google Maps widget.



The method takes two parameters:

ImageConfiguration: this is an optional parameter that can be used to specify various configuration options for the image, such as the device pixel ratio and the locale.

img: this is the path to the image asset that is being loaded.

The method returns a Future<BitmapDescriptor> which resolves to a BitmapDescriptor object once the image asset has been loaded. This BitmapDescriptor object can then be used to create a marker icon in a Google Maps widget.

Overall, this code is used to load an image asset and create a marker icon from it.