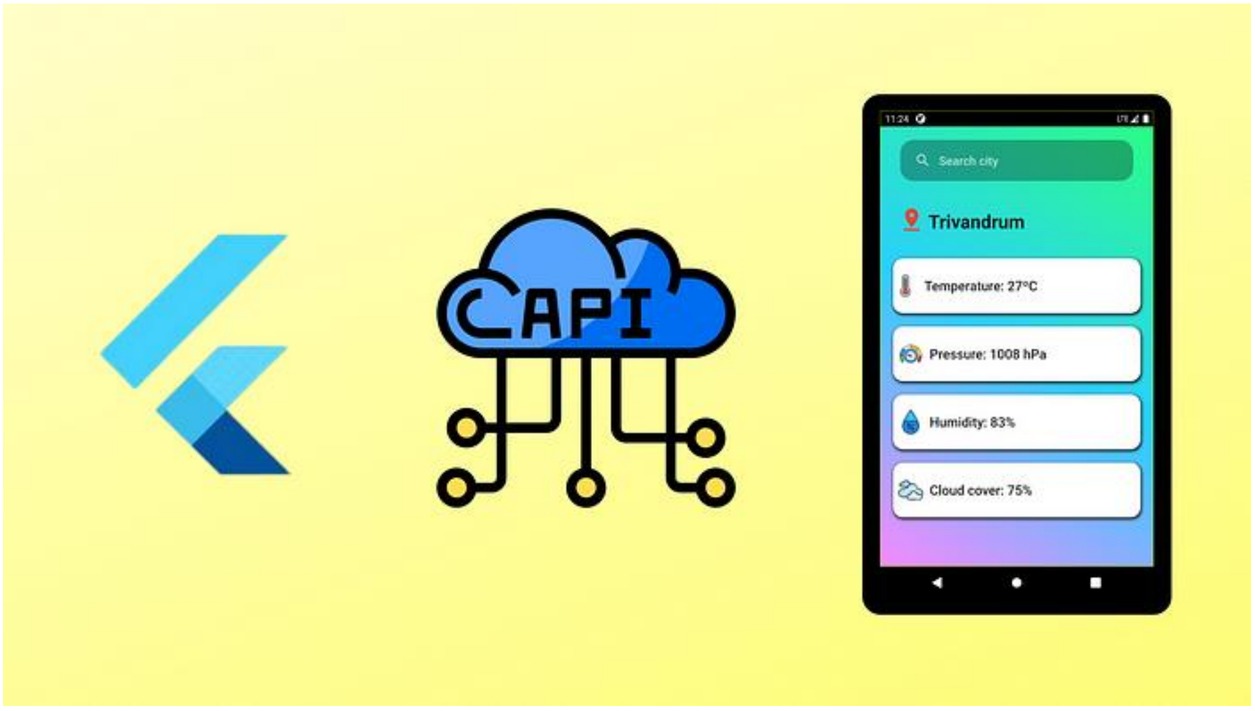
Weather app using API integration in Flutter



Ever wanted to create a real life mobile application. Here's your chance. Today we will be creating a complete application from scratch which enables us to observe the weather conditions using OpenWeatherMap API and Flutter.

Flutter is Google's UI toolkit for building beautiful, natively compiled applications for mobile, web, desktop, and embedded devices from a single codebase. OpenWeatherMap is an online service, owned by OpenWeather Ltd, that provides global weather data via API, including current weather data, forecasts, nowcasts and historical weather data for any geographical location.

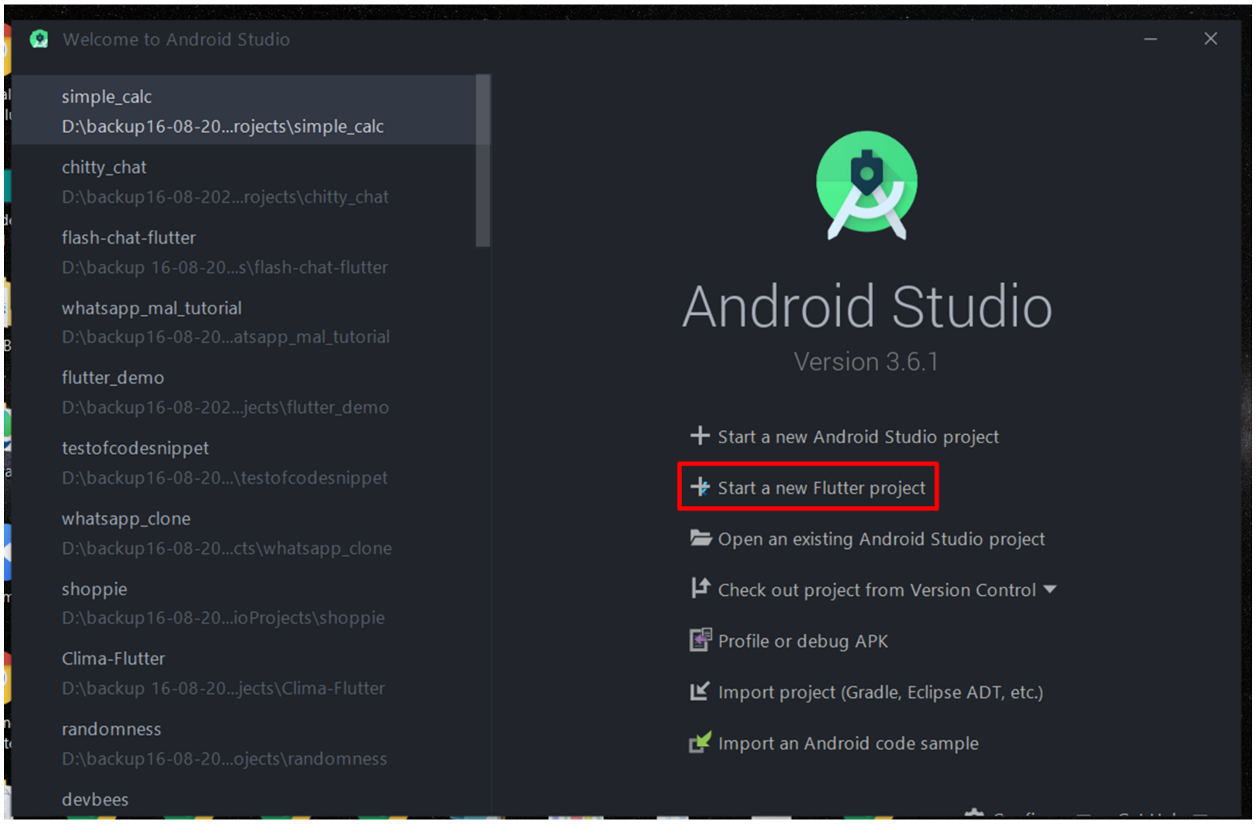
Now that we have a brief idea about Flutter and OpenWeatherMap, let's dive right in.

Video Tutorial

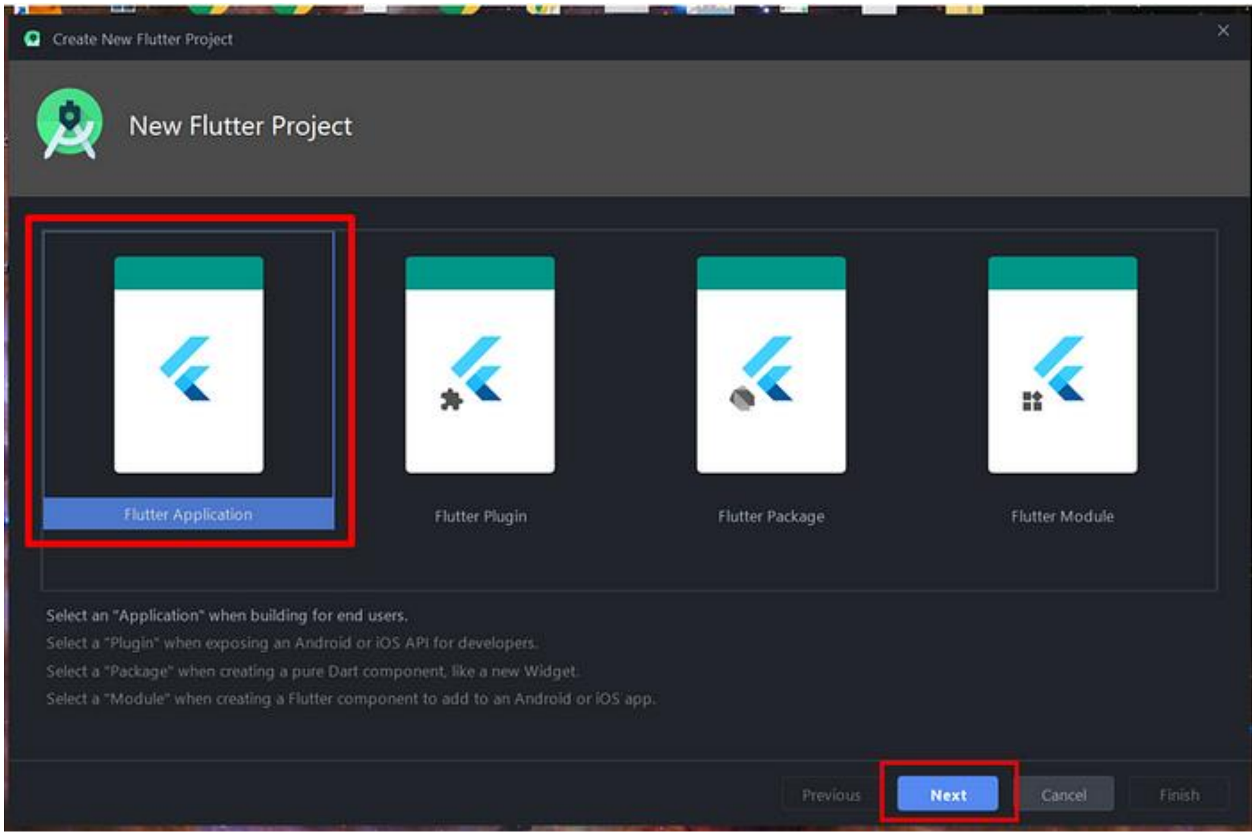
I have also uploaded a video tutorial in YouTube. Make sure to check it out and hit the like and subscribe button if you guys found it useful.

Creating a New Flutter Project

Open up android studio(or visual studio) to create a new flutter project(If you don't have flutter installed then follow steps in this video and proceed). Choose Start a new flutter project option s

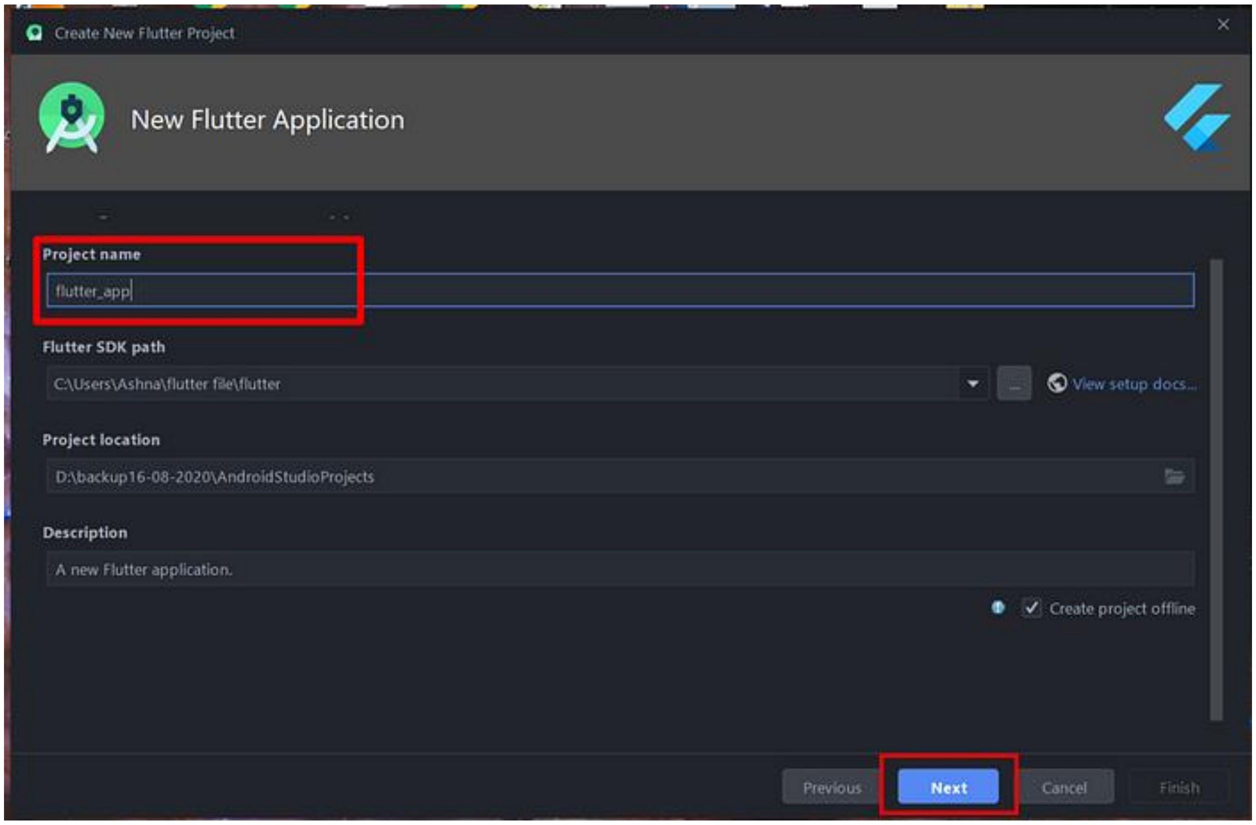

Welcome to Android Studio
simple_calc
D:\backup16-08-20...rojects\simple_calc
chitty_chat
D:\backup16-08-202...rojects\chitty_chat
flash-chat-flutter
D:\backup 16-08-20...s\flash-chat-flutter
whatsapp_mal_tutorial
D:\backup16-08-20...atsapp_mal_tutorial
flutter_demo
D:\backup16-08-202...jects\flutter_demo
testofcodesnippet
D:\backup16-08-20...\testofcodesnippet
whatsapp_clone
D:\backup16-08-20...cts\whatsapp_clone
shoppie
D:\backup16-08-20...ioProjects\shoppie
Clima-Flutter
D:\backup 16-08-20...jects\Clima-Flutter
randomness
D:\backup16-08-20...ojects\randomness
devbees
Android Studio
Version 3.6.1
Start a new Android Studio project
Start a new Flutter project
Open an existing Android Studio project
Check out project from Version Control
Profile or debug APK
Import project (Gradle, Eclipse ADT, etc.)
Import an Android code sample

Then a new pop up window comes where we will select flutter application and hit the next button



Select an "Application" when building for end users.
Select a "Plugin" when exposing an Android or iOS API for developers.
Select a "Package" when creating a pure Dart component, like a new Widget.
Select a "Module" when creating a Flutter component to add to an Android or iOS app.

Next give your application a name and then again hit next.


Create New Flutter Project

New Flutter Application

Project name
flutter_app

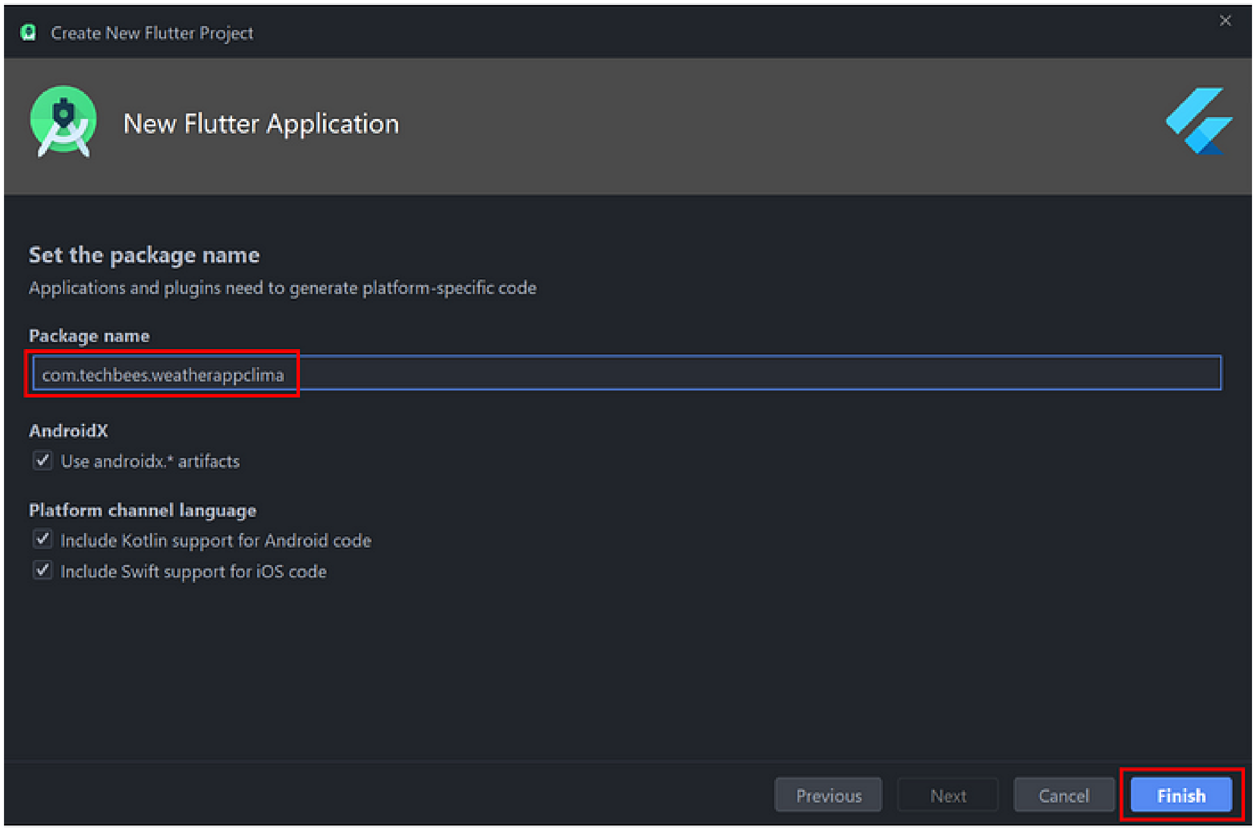
Flutter SDK path
C:\Users\Ashna\flutter file\flutter

Project location
D:\backup16-08-2020\AndroidStudioProjects

Description
A new Flutter application.

✓ Create project offline

Once the above steps are done, the package name comes up. This is the unique id by which the application will be known once published. Then hit on finish to create your flutter project.


Create New Flutter Project

New Flutter Application

Set the package name

Applications and plugins need to generate platform-specific code

Package name: com.techbees.weatherappclima

AndroidX
- [✓] Use androidx.* artifacts

Platform channel language
- [✓] Include Kotlin support for Android code
- [✓] Include Swift support for iOS code

Once that is done we will have our main.dart file loaded, along with the necessary files, with the default flutter start project, which is a counter app.

Fetching the packages

Flutter provides various packages for simplifying the app building process. Here, we make use of a few packages like http and geolocator. The http package aids in establishing a connection between our Flutter app with the internet and the geocoding package aids in obtaining the location data. We can incorporate the packages into our app by providing the package name with version in the pubspec.yaml in our flutter project, that is the configuration file.

```
dependencies:
  geolocator:
  http:
```

Here I have not specified the version number of the packages. This enables to obtain the latest compatible versions of the dependencies on running pub get command. In order to see the exact version obtained you can refer the pubspec.lock file. For this project I have used version 8.0.5 of geolocator package and version 0.13.3 of http package.

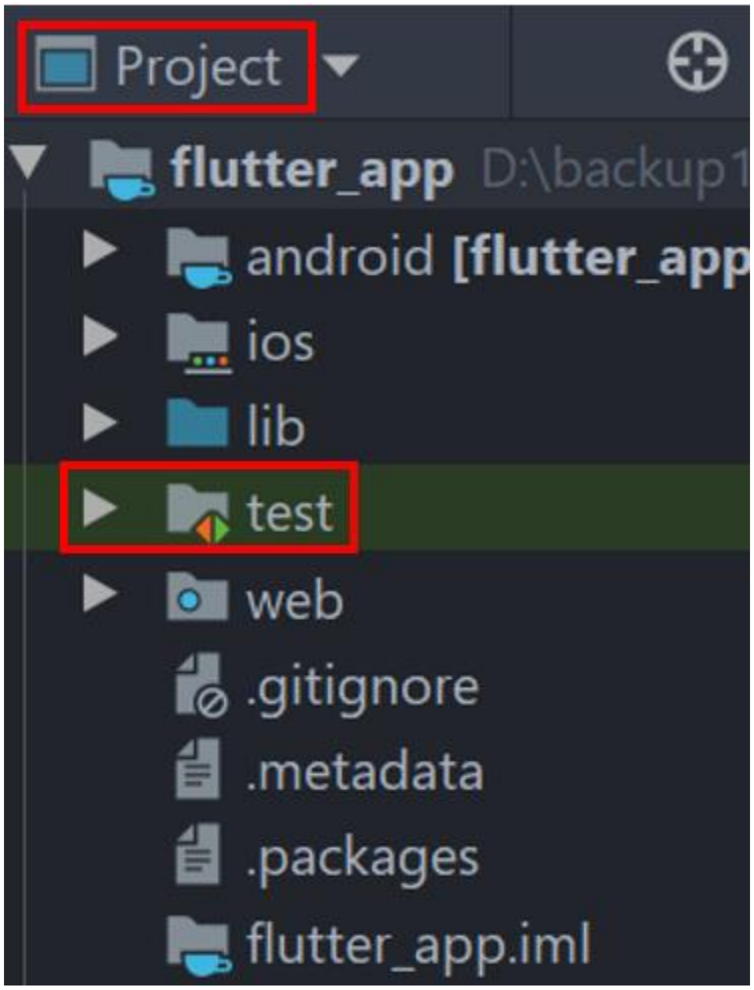
We will be implementing the code as functions. So our aim is to display the current location weather conditions on opening the app and then to get the current weather conditions of any other locations or city.

In order to achieve this initially we will have to get the current location's latitude and longitude. We can achieve this with the help of geolocator package. However in order to use the package there are certain configurations for android as well as iOS platforms which can be referred in the **readme** section of geolocator package.

Once the configurations are done we can proceed to get the current location latitude and longitude.

Building the App

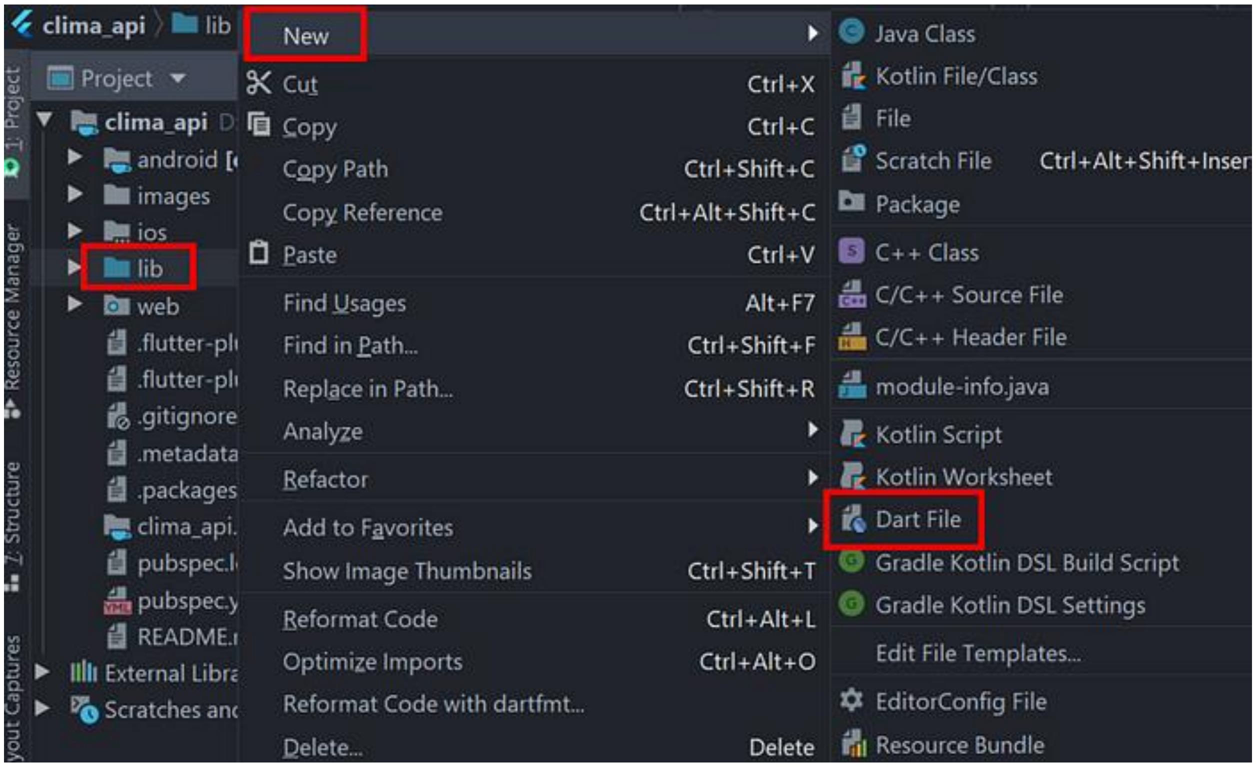
In the default main.dart file we will be removing the unnecessary code and keep the necessary code only. Here we will be working with *MaterialApp* instead of *MyApp*. Therefore we delete the *MyApp* references in *main.dart* and **delete** the **test file** under projects section.



Once the MyApp references are deleted and replaced with MaterialApp() as the root widget, the main.dart looks like below

```
import 'package:flutter/material.dart';void main() {
  runApp(
    MaterialApp(),
 );
}
```

Now create a new dart file by right clicking on **lib >New>Dart File**



Inside this dart file import the material.dart package. We will now create a Stateful widget to build our app. The shortcut to create one is to simply type ***stful*** and we will get a skeletal code of the Stateful widget. Replace the YourWidgetName with your own custom names.

```
class YourWidgetName extends StatefulWidget {
 @override
 _YourWidgetNameState createState() => _YourWidgetNameState();
}

class _YourWidgetNameState extends State<YourWidgetName> {
 @override
 Widget build(BuildContext context) {
  return Container();
 }
}
```

Now once that is done, import this dart file in main.dart and set the home property of MaterialApp() to the name given for the Stateful widget created. The home property is used to display the starting screen in an app when the app involves only a single screen. Here I have given the name as HomeScreen() and the dart file name as homescreen.dart.

```
import 'package:flutter/material.dart';
import 'homescreen.dart';

void main() {
 runApp(
  MaterialApp(
   debugShowCheckedModeBanner: false,
   home: HomeScreen(),
   theme: ThemeData(
    primaryColor: Colors.white,
    accentColor: Colors.white,
   ),
  ),
 );
}
```

Apart from the home property of MaterialApp we will also be specifying the debugShowCheckedModeBanner property to false so the debug banner won't be visible and also setting the primary and accent theme colors to white using the theme property.

Step 1: Getting Current location co-ordinates

It is inside the homescreen.dart we will be obtaining the current location latitude and longitude. The latitude and longitude is not displayed in the final version of the app so we will just try to print the values in terminal.

For this first we have to import the geolocator package in homescreen.dart.

```
import 'package:geolocator/geolocator.dart';
```

Then we add a asynchronous function as given below

```
getCurrentLocation() async {
  var p = await Geolocator.getCurrentPosition(
   desiredAccuracy: LocationAccuracy.low,
   forceAndroidLocationManager: true,
  );
  if (p != null) {
   print('Lat:${p.latitude}, Long:${p.longitude}');

  } else {
   print('Data unavailable');
  }
 }
```

When the getCurrentLocation() method of Geolocator package is called it returns a Position value(comprising of latitude, longitude, accuracy, altitude, heading etc) is returned. Since the time in obtaining the Position value is not known and can be obtained at any point in the future we use await keyword. This means the function works asynchronously from the remaining executions.

The accuracy of the position can set by the desiredAccuracy property and the location manager of android can be set by the forceAndroidLocationManager property. The returned Position value consists of the details of the current location details of the device. To obtain the latitude and longitude, the corresponding parameters of the Position value is called and printed accordingly.

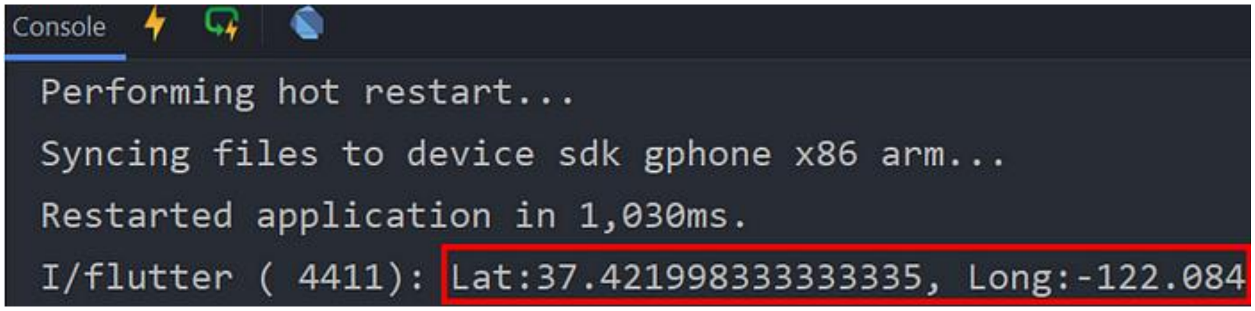
This function is then called in the initState() method of the Stateful() widget as shown

```
@override
void initState() {
  // TODO: implement initState
  super.initState();
  getCurrentLocation();
}
```

The build method is returned with a simple Scaffold() wrapped in SafeArea() as shown.

```
Widget build(BuildContext context) {
  return SafeArea(
    child: Scaffold(),
  );
}
```

With this run the application and we will obtain the latitude and longitude of the current location.

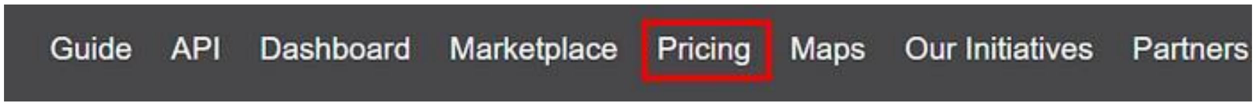


```
Performing hot restart...
Syncing files to device sdk gphone x86 arm...
Restarted application in 1,030ms.
I/flutter ( 4411): Lat:37.421998333333335, Long:-122.084
```

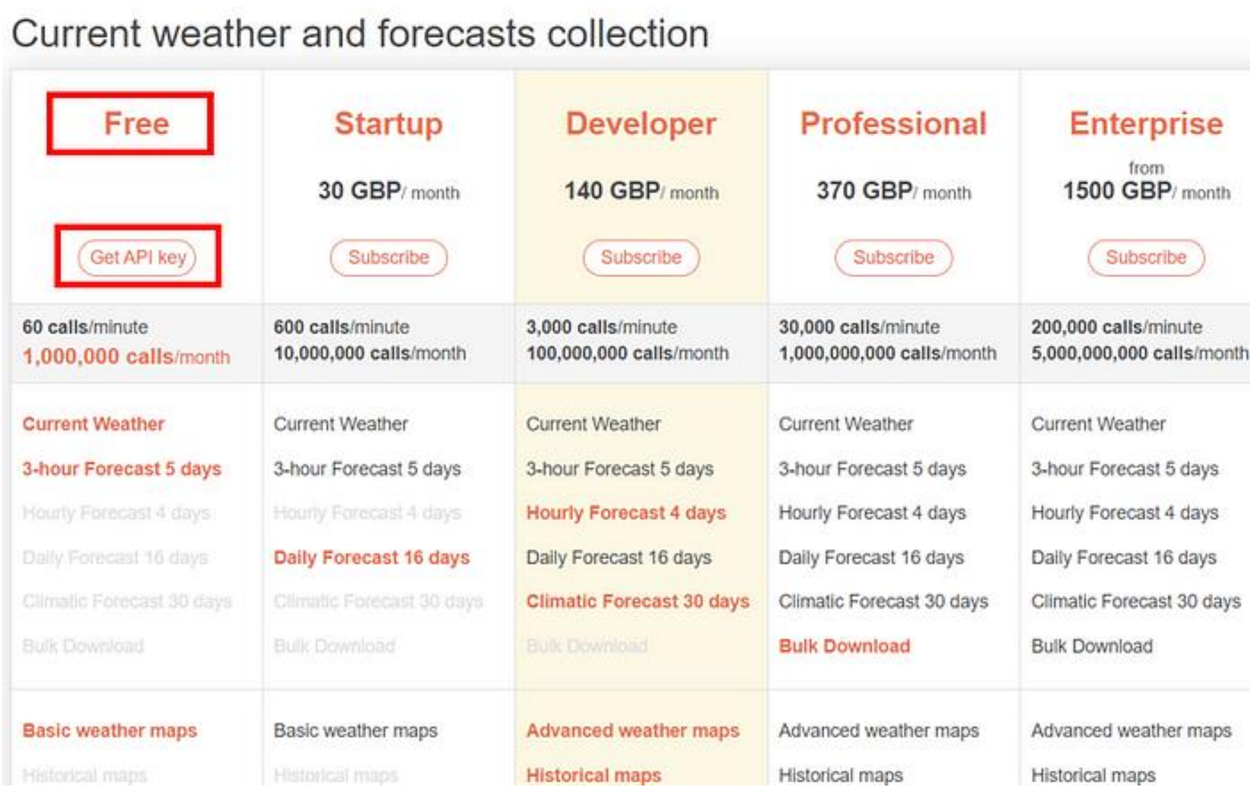
Note: Here the latitude and longitude are that of the emulator's which is set as default.

STEP 2: Getting API key from OpenWeatherMap

For getting the real-time weather data we will be making use of [OpenWeatherMap]() API. To access the API data we require an API key. In order to do that, choose the Pricing section from the top menu.



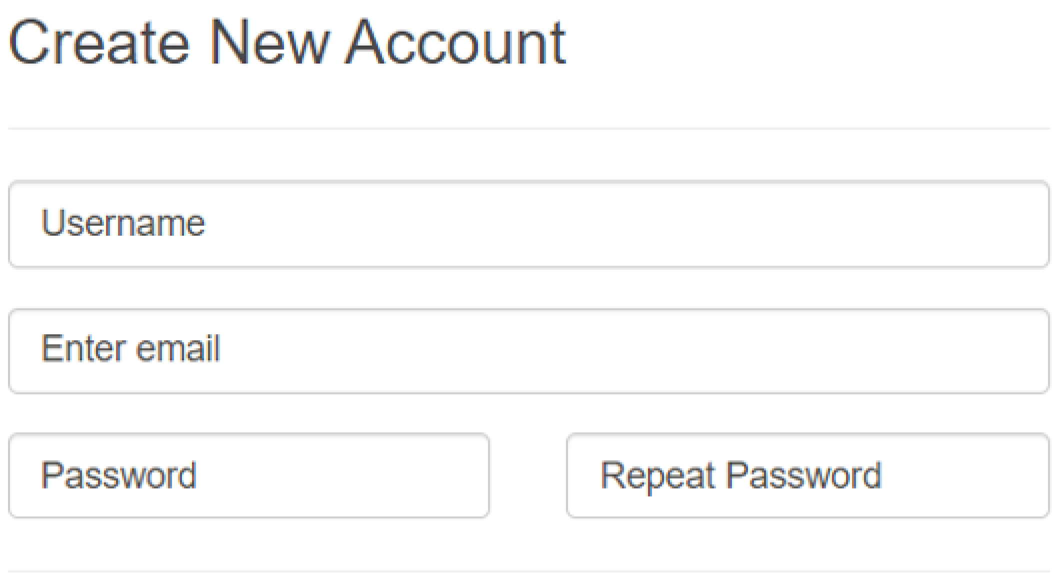
Then scroll down till you see the various plans for current weather. Choose the Free plan and click on the Get API key option.



Current weather and forecasts collection

Free	Startup	Developer	Professional	Enterprise
	30 GBP/ month	140 GBP/ month	370 GBP/ month	from 1500 GBP/ month
Get API key	Subscribe	Subscribe	Subscribe	Subscribe
60 calls/minute 1,000,000 calls/month	600 calls/minute 10,000,000 calls/month	3,000 calls/minute 100,000,000 calls/month	30,000 calls/minute 1,000,000,000 calls/month	200,000 calls/minute 5,000,000,000 calls/month
Current Weather	Current Weather	Current Weather	Current Weather	Current Weather
3-hour Forecast 5 days	3-hour Forecast 5 days	3-hour Forecast 5 days	3-hour Forecast 5 days	3-hour Forecast 5 days
Hourly Forecast 4 days	Hourly Forecast 4 days	Hourly Forecast 4 days	Hourly Forecast 4 days	Hourly Forecast 4 days
Daily Forecast 16 days	Daily Forecast 16 days	Daily Forecast 16 days	Daily Forecast 16 days	Daily Forecast 16 days
Climatic Forecast 30 days	Climatic Forecast 30 days	Climatic Forecast 30 days	Climatic Forecast 30 days	Climatic Forecast 30 days
Bulk Download	Bulk Download	Bulk Download	Bulk Download	Bulk Download
Basic weather maps	Basic weather maps	Advanced weather maps	Advanced weather maps	Advanced weather maps
Historical maps	Historical maps	Historical maps	Historical maps	Historical maps

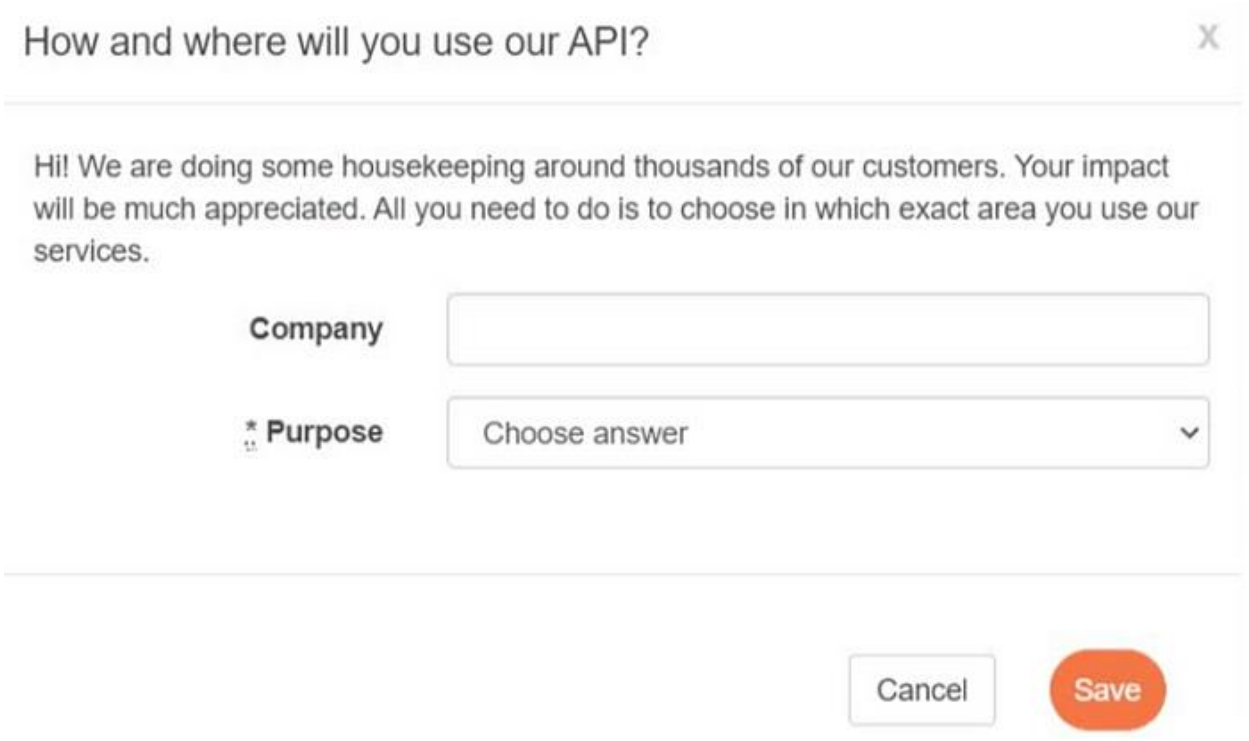
Once that is done, a new window comes up where you have to create an account. Provide the necessary information and create your account.



Create New Account

Username	
Enter email	
Password	Repeat Password

Once the account is created, a pop-up will come up asking about the purpose for which we intend to use our API key.



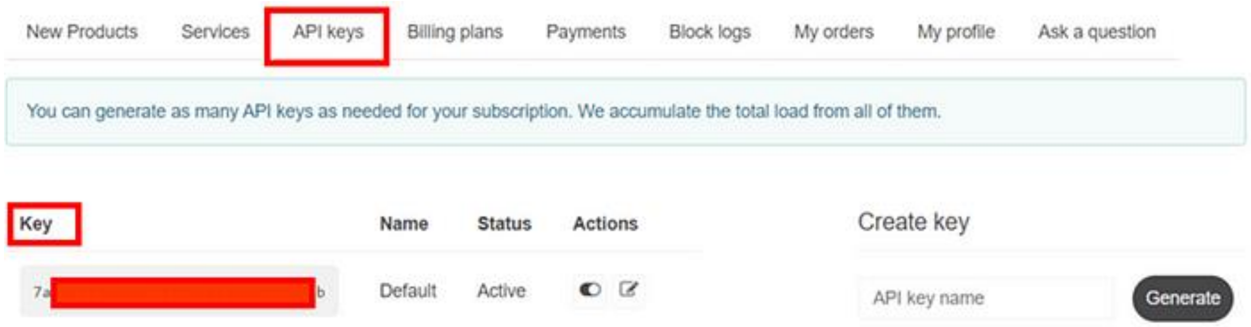
How and where will you use our API?

Hi! We are doing some housekeeping around thousands of our customers. Your impact will be much appreciated. All you need to do is to choose in which exact area you use our services.

Company	
* Purpose	Choose answer

Here specifying the company is optional while specifying the purpose is mandatory. If you don't know which purpose to choose you can go with Education/Science. Then click save.

Now in your dashboard go to the API keys section. Here you will find all your API keys under the Key section. Note that the API key here is unique to you and you shouldn't share it with anyone else.



You can generate as many API keys as needed for your subscription. We accumulate the total load from all of them.

Key	Name	Status	Actions
7a[redacted]b	Default	Active	

Once you get your API key you can proceed to call the API to get the weather data. There are various formats of calling the API and how the response will be, all of which can be referred here. For our purpose we will be calling the API via the latitude longitude format and the city name format as given below

API call by latitude longitude format
https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={API key}**API call by city name format**
https://api.openweathermap.org/data/2.5/weather?q={city name}&appid={API key}

As both API calls have same API key and domain, we will store those values in a new dart file named constants under lib folder. For creating the file refer the Building App section above. In the created constants.dart file add the following lines of code.

```
const String domain = "https://api.openweathermap.org/data/2.5/weather?";
const String apiKey = "PASTE YOUR API KEY HERE";
```

Now we will try obtaining the weather data which will be a JSON response.

STEP 3: Getting weather data of Current location

After successfully completing the above steps, we will try obtaining the weather data. For this inside the homescreen.dart we will write a function to first connect our app to the internet and then obtain weather data. This is where we will be implementing http package. So first we import the package as http so it becomes easier to access the different fields in the package

```
import 'package:http/http.dart' as http;
import 'constants.dart' as k;
import 'dart:convert';
```

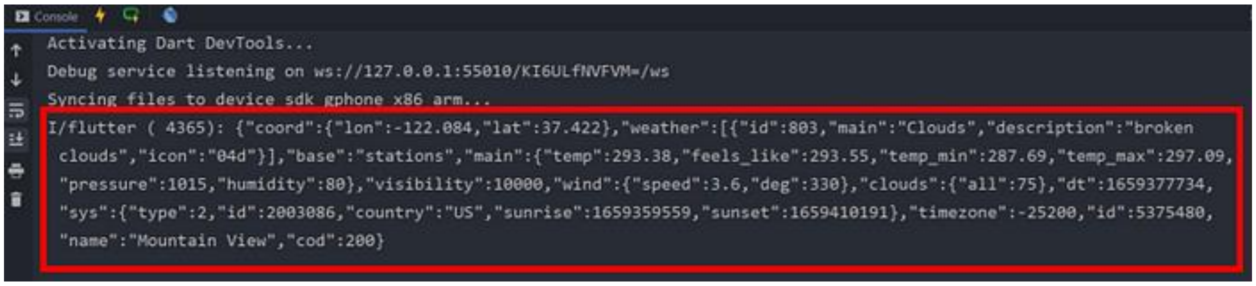
Similarly we also import the constants.dart where we copied the API key and domain link. Along with it as we are dealing with getting a single JSON response for each call, we use the convert library of dart for decoding the JSON response we obtain.

API call by latitude longitude format
https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={API key}

Next we will write the function to connect our app to internet. This function will also be asynchronous as it is returning a future. Here we first create a Client object so we don’t have to open and close ports every time we call the get method. Then we provide our URI (A URI is a character sequence that helps identify a logical or physical resource connected to the internet) address which is the API call format providing the necessary fields.

```
getCurrentCityWeather(Position position) async {
  var client = http.Client();
  var uri =
     '${k.domain}lat=${position.latitude}&lon=${position.longitude}&appid=${k.apiKey}';
  var url = Uri.parse(uri);
  var response = await client.get(url);
  if (response.statusCode == 200) {
    var data = response.body;
    print(data);
   } else {
    print(response.statusCode);
  }
}
```

After that the URI is converted into URL address(It is the location of the URI in the internet). Then using the client field, the get method which fetches the data from the URL is called. This value is then stored in the response variable. The status code of the response determines the result of the data fetching process. If the code is 200, it means that the data is fetched successfully. The JSON data can be obtained from the body of response.



```
Activating Dart DevTools...
Debug service listening on ws://127.0.0.1:55010/KI6ULfNVFVM=/ws
Syncing files to device sdk gphone x86 arm...
I/flutter ( 4365): {"coord":{"lon":-122.084,"lat":37.422},"weather":[{"id":803,"main":"Clouds","description":"broken
 clouds","icon":"04d"}],"base":"stations","main":{"temp":293.38,"feels_like":293.55,"temp_min":287.69,"temp_max":297.09,
 "pressure":1015,"humidity":80},"visibility":10000,"wind":{"speed":3.6,"deg":330},"clouds":{"all":75},"dt":1659377734,
 "sys":{"type":2,"id":2003086,"country":"US","sunrise":1659359559,"sunset":1659410191},"timezone":-25200,"id":5375480,
 "name":"Mountain View","cod":200}
```

The above code when made to run will print the weather data we receive from the API call. In the body we can see various parameters like temperature, pressure, humidity, visibility, wind speed etc.

STEP 4: Getting weather data of various cities

Similar to STEP 3 we will implement getting the weather data of a particular city based on the city name. Here the only difference is instead of providing latitude and longitude in the API call we provide the city name.

API call by city name format
https://api.openweathermap.org/data/2.5/weather?q={city name}&appid={API key}

So the same function with modification in function name, parameter and uri is provided to obtain the weather data.

```
getCityWeather(String cityname) async {
   var client = http.Client();
   var uri = '${k.domain}q=$cityname&appid=${k.apiKey}';
   var url = Uri.parse(uri);
   var response = await client.get(url);
   if (response.statusCode == 200) {
     var data = response.body;
     var decodeData = json.decode(data);
     print(data);

   } else {
     print(response.statusCode);
   }
 }
```

The above code also yields a similar result as in STEP 3 but with the location (city) we have specified.

STEP 5: Completing the Build

Now that we have obtained the weather data and location data, we will now proceed to building the UI of the app. So here we will be using cards on a gradient background to display weather conditions.

For that we first provide the body of the Scaffold() which is a Container() covering the entire screen and which has a gradient filling. Here I have used gradients from *Gradient Backgrounds* which has an immense collection of background gradients along with the corresponding colour codes. The following is the gradient I have chosen.

The code for obtaining complete background gradient is as shown below. Here we will set the resizeToAvoidBottomInset property of Scaffold() widget to false so that resizing of the widgets when the keyboard pops up is avoided.

```
SafeArea(
 child: Scaffold(
  resizeToAvoidBottomInset: false,
  body: Container(
   width: double.infinity,
   height: double.infinity,
   padding: EdgeInsets.all(20),
   decoration: BoxDecoration(
    gradient: LinearGradient(
     colors: [
      Color(0xffFA8BFF),
      Color(0xff2BD2FF),
      Color(0xff2BFF88),
          ],
          begin: Alignment.bottomLeft,
          end: Alignment.topRight,
        ),
      ),
    ),
  ),
);
```

Then we will provide a few variables to store values of temperature, pressure, humidity, cloud cover, city name and the state of the data that is whether the data is fetched and ready to be used by the app. These variables are declared inside the Stateful widget before the initState method.

```
bool isLoaded = false;
num temp;
num press;
num hum;
num cover;
String cityname = '';
```

Next we will add a Visibility widget as the child of the Container widget. So only when there is data will the weather data is displayed else it shows a loading indicator. For this the value of isLoaded is dynamically changed in the functions using setState method. The code is as given below.

```
Container(
  child: Visibility(
    visible: isLoaded,
    child: Column(),
    replacement: Center(
      child: CircularProgressIndicator(),
    ),
  ),
),
```

Now to store the weather data into the variables we will provide another function to update the variables. This function will have the decoded JSON data as the parameter and based on the value of it, the parameter values are set.

```
updateUI(var decodedData) {
  setState(() {
    if (decodedData == null) {
      temp = 0;
      press = 0;
      hum = 0;
      cover = 0;
      cityname = 'Not available';
    } else {
      temp = decodedData['main']['temp'] - 273;
      press = decodedData['main']['pressure'];
      hum = decodedData['main']['humidity'];
      cover = decodedData['clouds']['all'];
      cityname = decodedData['name'];
    }
  });
}
```

This function will be called in both the API calling functions if the status code of received data is 200

```
getCurrentCityWeather(Position position) async {
  //Networking code
  if (response.statusCode == 200) {
    var data = response.body;
    var decodeData = json.decode(data);
    updateUI(decodeData);
    setState(() {
      isLoaded = true;
    });
  } else {
    print(response.statusCode);
  }
}
```

The code highlighted in bold is added to the function. It basically decodes the JSON data and calls the updateUI method which sets the value of the variables to the corresponding weather data. Along with that, the varibles representing the state of data obtained is also updated throughout the build with the help of setState method. Similarly for the function using API call with city name, the same code is added.

```
getCityWeather(String cityname) async {
    //Networking code
    if (response.statusCode == 200) {
      var data = response.body;
      var decodeData = json.decode(data);
      updateUI(decodeData);
      setState(() {
        isLoaded = true;
      });
    } else {
      print(response.statusCode);
    }
  }
```

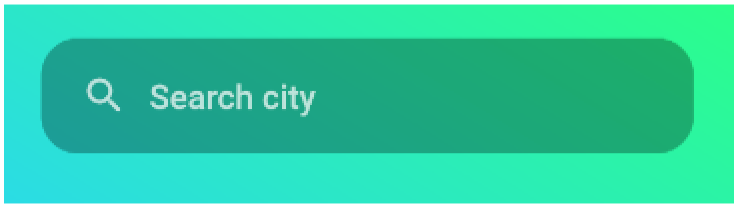
Inside the column of Visibilty widget we will add a TextFormField as the first child. For the controller of this TextFormField a controller is also provided, which is declared along with the variables.

```
TextEditingController controller = TextEditingController();
```

The code of the TextFormField is as given below. Note that we have set the value of isLoaded to false once the city name is entered. This provides a loading indicator to the user while fetching the data, so the user will know there is some process going on.

```
child: Column(
 children: [
  Container(
   width: MediaQuery.of(context).size.width * 0.85,
   height: MediaQuery.of(context).size.height * 0.09,
   padding: EdgeInsets.symmetric(
    horizontal: 10,
   ),
   decoration: BoxDecoration(
    color: Colors.black.withOpacity(0.3),
    borderRadius: BorderRadius.all(
     Radius.circular(
      20,
     ),
    ),
   ),
   child: Center(
    child: TextFormField(
     onFieldSubmitted: (String s) {
      setState(() {
       cityname = s;
       getCityWeather(s);
       isLoaded = false;
       controller.clear();
      });
     },
     controller: controller,
     cursorColor: Colors.white,
     style: TextStyle(
       fontSize: 20,
       fontWeight: FontWeight.w600,
       color: Colors.white),
     decoration: InputDecoration(
      hintText: 'Search city',
      hintStyle: TextStyle(
       fontSize: 18,
       color: Colors.white.withOpacity(0.7),
       fontWeight: FontWeight.w600,
      ),
      prefixIcon: Icon(
       Icons.search_rounded,
       size: 25,
       color: Colors.white.withOpacity(0.7),
      ),
      border: InputBorder.none,
     ),
    ),
   ),
  ),
 ],
),
```

As the width cannot be specified in a TextFormField, it is wrapped inside a container.



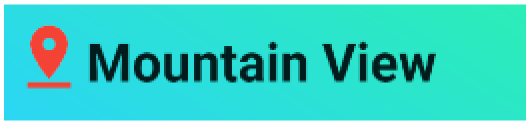
Afterwards in order for minimum memory usage, we will dispose the controller in the dispose method of Stateful widget as shown below.

```
@override
void dispose() {
 // TODO: implement dispose
 controller.dispose();
 super.dispose();
}
```

Next we will add the next child of the Column() which is a SizedBox widget for adding necessary space between the components.

```
SizedBox(
 height: 30,
),
```

This is followed by the City name data.



It is implemented by use of a Row() widget wrapped with a Padding widget. It comprises of an Icon and a Text and the code is as given below

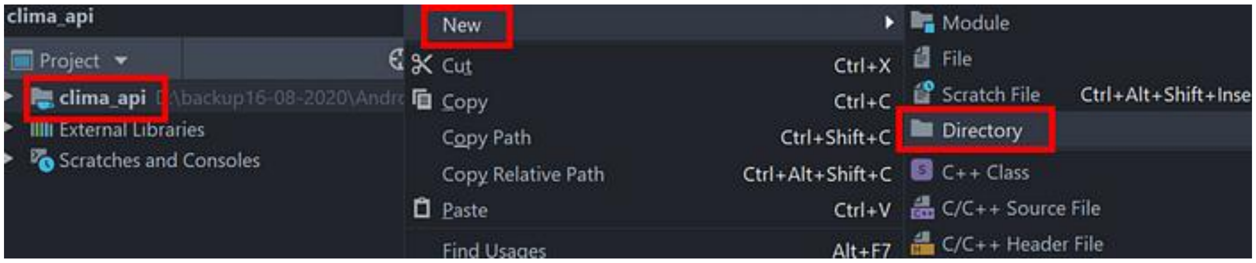
```
Padding(
 padding: const EdgeInsets.all(8.0),
 child: Row(
  crossAxisAlignment: CrossAxisAlignment.end,
  children: [
   Icon(
    Icons.pin_drop,
    color: Colors.red,
    size: 40,
   ),
   Text(
    cityname,
    overflow: TextOverflow.ellipsis,
    style: TextStyle(
     fontSize: 28,
     fontWeight: FontWeight.bold,
    ),
   )
  ],
 ),
),
```

This is again followed by a SizedBox widget

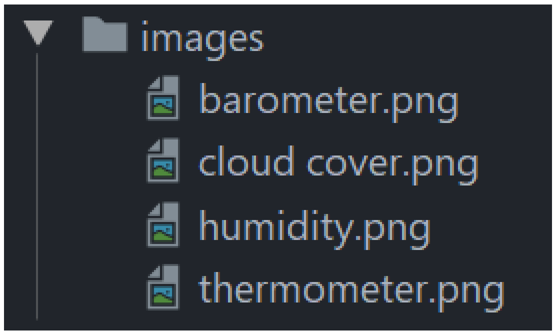
```
SizedBox(
 height: 20,
),
```

Next comes the weather data section displayed as cards. Here I have used Container for creating custom cards. For the data I have used Image and Text widgets.

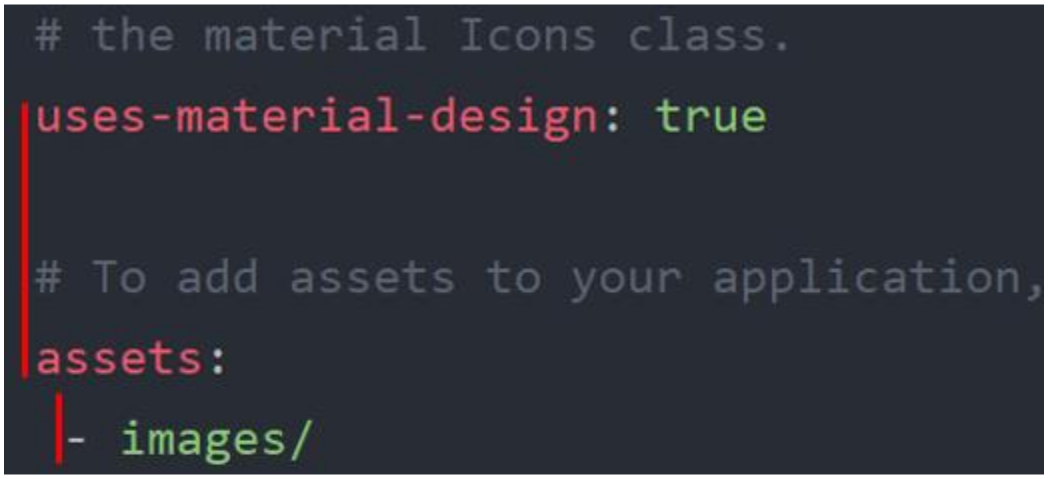
Note that the images used here are from the project file itself. So in order to do that we will have to configure those images. For this first create a new directory in the project folder under project_name>New>Directory



Name the new directory as images. Then once the directory is created add the necessary images.

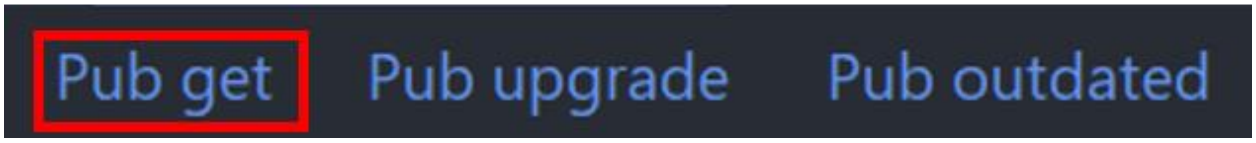

```
images
    barometer.png
    cloud cover.png
    humidity.png
    thermometer.png
```

Afterwards, go to your pubspec.yaml file. Scroll down to the asset section where images are incorporated. Uncomment the lines and make sure the spacing is correct as shown below. The below code includes all the files under images directory.


```
# the material Icons class.
uses-material-design: true

# To add assets to your application,
assets:
  - images/
```

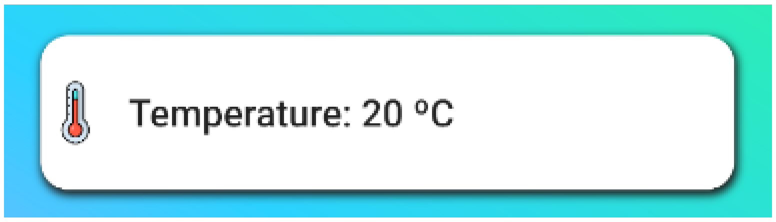
Then run pub get command at the top right corner.



Once that is done, the images will be configured in the app. Now in homescreen.dart we will add the code for creating the card display.

```
Container(
 width: double.infinity,
 height: MediaQuery.of(context).size.height * 0.12,
 margin: EdgeInsets.symmetric(
  vertical: 10,
 ),
 decoration: BoxDecoration(
  borderRadius: BorderRadius.all(
   Radius.circular(15),
  ),
  color: Colors.white,
  boxShadow: [
   BoxShadow(
    color: Colors.grey.shade900,
    offset: Offset(1, 2),
    blurRadius: 3,
    spreadRadius: 1,
   )
  ],
 ),
 child: Row(
  children: [
   Image(
    image: AssetImage('images/thermometer.png'),
    width: MediaQuery.of(context).size.width * 0.09,
   ),
   SizedBox(
    width: 10,
   ),
   Text(
    'Temperature: ${temp?.toInt()} ºC',
    style: TextStyle(
      fontSize: 20, fontWeight: FontWeight.w600),
   )
  ],
 ),
),
```

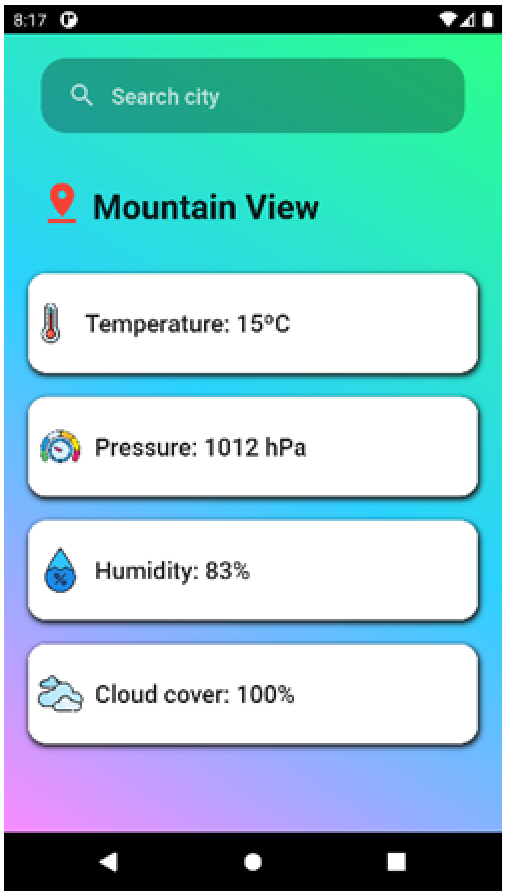
Once the above code is added stop and cold start the app for the configuration changes to be included. When the app is loaded we will get a card display as shown below


Temperature: 20 ºC

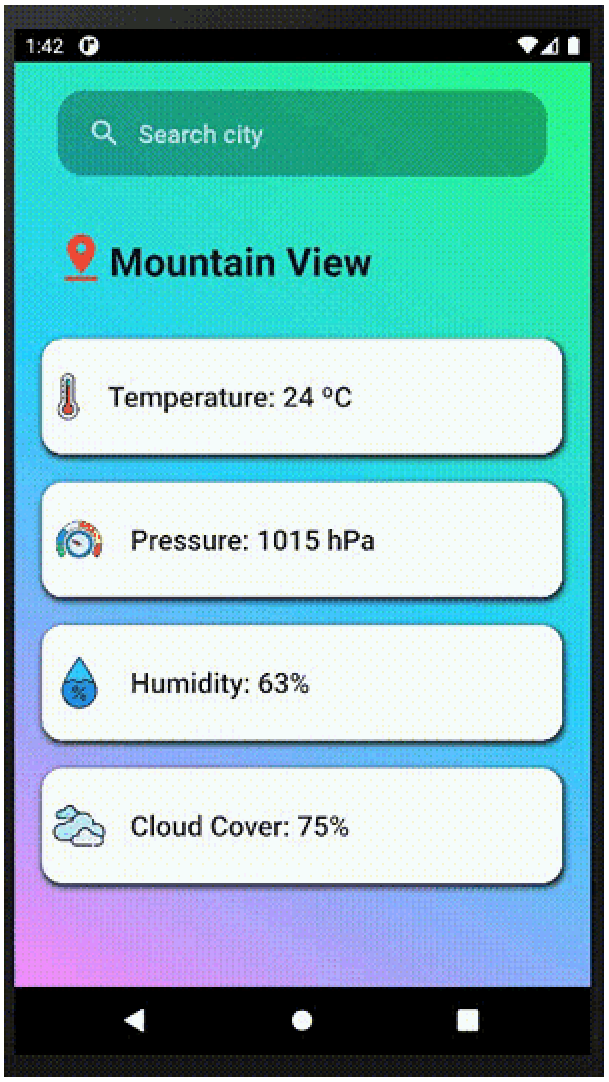
Again 3 more cards are added by replacing the image and text for the respective weather parameters. For the next 3 card displays, the images are wrapped with a padding on all sides.

```
Padding(
 padding: const EdgeInsets.all(8.0),
 child: Image(
  image: AssetImage('images/barometer.png'),
  width: MediaQuery.of(context).size.width * 0.09,
 ),
),
```

Once that is done we will get an output as shown below



The final working demo of the app is as shown below



Mountain View

Temperature: 24 °C

Pressure: 1015 hPa

Humidity: 63%

Cloud Cover: 75%