

**JOBSHEET 12**  
**“Penggunaan API”**

**Mata Kuliah [Pemrograman Mobile]**

Dosen Pengampu: Ade Ismail, S.Kom., M.TI.



Nama : Wardanadira Pinkan Dwiyuwanda  
NIM : 2241760116  
Kelas : SIB-3D  
No Absen : 26

**PROGRAM STUDI SISTEM INFORMASI BISNIS**  
**JURUSAN TEKNOLOGI INFORMASI**  
**POLITEKNIK NEGERI MALANG**  
**2024**

# WEATHER APP USING API INTEGRATION IN FLUTTER

Link github : <https://github.com/WardanadiraPinkan/PEMROGRAMAN-MOBILE>

## FETCHING THE PACKAGES

### Langkah 1: Menginstal paket http pada terminal VS CODE

Dengan mengetik “flutter pub add http” pada terminal VS CODE”

```
PS C:\Users\Wardanadira Pinkan\OneDrive\Documents\SEMESTER 5\PEMR_MOB_SIB\project12\flutter_api> flutter pub add http
Resolving dependencies...
Downloading packages... (1.0s)
  async 2.11.0 (2.12.0 available)
  boolean_selector 2.1.1 (2.1.2 available)
  characters 1.3.0 (1.3.1 available)
  clock 1.1.1 (1.1.2 available)
  collection 1.18.0 (1.19.1 available)
  fake_async 1.3.1 (1.3.2 available)
  flutter_lints 4.0.0 (5.0.0 available)
+ http 1.2.2
+ http_parser 4.0.2 (4.1.1 available)
  leak_tracker 10.0.5 (10.0.8 available)
  leak_tracker_flutter_testing 3.0.5 (3.0.9 available)
  lints 4.0.0 (5.1.0 available)
  matcher 0.12.16+1 (0.12.17 available)
  material_color_utilities 0.11.1 (0.12.0 available)
  meta 1.15.0 (1.16.0 available)
  path 1.9.0 (1.9.1 available)
```

Fungsi paket http pada Flutter digunakan untuk [HTTP Request] dengan sebagai berikut:

1. **GET Request** [mengambil data dari server/API]  
Mengambil data JSON dari server untuk mendapatkan data seperti cuaca, berita, dan produk dari API itu sendiri.
2. **POST Request** [mengirim data ke server]  
Mengirim data ke server dengan mengisi formular, mendaftarkan pengguna, dan mengunggah data.
3. **DELETE Request** [menghapus data di server]  
Menghapus data tertentu dari server dengan menghapus catatan pengguna.
4. **PUT atau PATCH Request** [mengubah data di server]  
Memperbarui data tertentu pada server dengan memperbarui informasi profil pengguna.

### Langkah 2: Menginstal paket geolocator pada terminal VS CODE

```
PS C:\Users\Wardanadira Pinkan\OneDrive\Documents\SEMESTER 5\PEMR_MOB_SIB\project12\flutter_api> flutter pub add geolocator
Resolving dependencies...
Downloading packages... (1.9s)
  async 2.11.0 (2.12.0 available)
  boolean_selector 2.1.1 (2.1.2 available)
  characters 1.3.0 (1.3.1 available)
  clock 1.1.1 (1.1.2 available)
  collection 1.18.0 (1.19.1 available)
+ crypto 3.0.6
+ fake_async 1.3.1 (1.3.2 available)
+ fixnum 1.1.1
+ flutter_lints 4.0.0 (5.0.0 available)
+ flutter_web_plugins 0.0.0 from sdk flutter
+ geolocator 13.0.2
+ geolocator_android 4.6.1
+ geolocator_apple 2.3.8+1
+ geolocator_platform_interface 4.2.4
+ geolocator_web 4.1.1
+ geolocator_windows 0.2.3
```

### Langkah 3: Menginstal dan mengelola depedensi pada terminal VS CODE

```
PS C:\Users\Wardanadira Pinkan\OneDrive\Documents\SEMESTER 5\PEMR_MOB_SIB\project12\flutter_api> flutter pub get
Resolving dependencies...
Downloading packages...
async 2.11.0 (2.12.0 available)
boolean_selector 2.1.1 (2.1.2 available)
characters 1.3.0 (1.3.1 available)
clock 1.1.1 (1.1.2 available)
collection 1.18.0 (1.19.1 available)
fake_async 1.3.1 (1.3.2 available)
flutter_lints 4.0.0 (5.0.0 available)
http_parser 4.0.2 (4.1.1 available)
leak_tracker 10.0.5 (10.0.8 available)
leak_tracker_flutter_testing 3.0.5 (3.0.9 available)
lints 4.0.0 (5.1.0 available)
matcher 0.12.16+1 (0.12.17 available)
material_color_utilities 0.11.1 (0.12.0 available)
meta 1.15.0 (1.16.0 available)
path 1.9.0 (1.9.1 available)
stack_trace 1.11.1 (1.12.0 available)
```

### Langkah 4: Menambahkan Kode berikut ke file [gradle.properties]

```
android.useAndroidX=true
android.enableJetifier=true
```

### Langkah 5: Menyeting compileSdk Versionfile [android/app/build.gradle] dengan mengisi 34

```
android [
    namespace = "com.example.flutter_api"
    compileSdk = 34 // Set compileSdkVersion
    ndkVersion = flutter.ndkVersion
```

### Langkah 6:

- Menambahkan izin ACCESS\_COARSE\_LOCATION atau ACCESS\_FINE\_LOCATION ke android
- Membuka file AndroidManifest.xml [terletak di bawah android/app/src/main]
- Menambahkan salah satu dari dua baris berikut sebagai turunan langsung dari <manifest>tag (saat mengonfigurasi kedua izin).
- ACCESS\_FINE\_LOCATION digunakan oleh plugin geolocator

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Izin Lokasi -->
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Dimulai dari android 10, perlu menambahkan ACCESS\_BACKGROUND\_LOCATION disamping ACCESS\_COARSE\_LOCATION atau ACCESS\_FINE\_LOCATION jika ingin terus menerima pembaruan bahkan Ketika aplikasi masih berjalan di latar belakang

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Izin Lokasi -->
    <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" />
```

## BUILDING THE APP

### Langkah 1: file main.dart default

Menghapus kode yang tidak diperlukan dan menyimpan kode yang diperlukan saja. Dengan bekerja pada MaterialApp, bukan MyApp. Oleh karena itu, perlu menghapus referensi MyApp pada file main.dart dan menghapus file pengujian di bagian proyek. Setelah referensi MyApp dihapus dan diganti dengan MaterialApp() sebagai widget root. Maka tampilan isi kode program pada file main.dart seeperti ini:

```
lib > main.dart > main
      Run | Debug | Profile
1 import 'package:flutter/material.dart';void main() {
2 runApp(
3   MaterialApp(),
4 );
5 }
```

### Langkah 2: Mengimpor paket material.dart.

Membuat widget StatefulWidget untuk membangun aplikasi yang akan kita buat.

```
lib > homescreen.dart > ...
1926 import 'package:flutter/material.dart';
1927 import 'package:http/http.dart' as http;
1928 import 'dart:convert';
1929
1930 class HomeScreen extends StatefulWidget {
1931   @override
1932   _HomeScreenState createState() => _HomeScreenState();
1933 }
1934
```

### Langkah 3: Mengimpor paket material.dart.

Mengimpor berkas dart ini di main.dart dan tetapkan properti home MaterialApp() ke nama yang diberikan untuk widget StatefulWidget yang dibuat. Properti home digunakan untuk menampilkan layar awal dalam suatu aplikasi saat aplikasi hanya melibatkan satu layar. Di sini saya telah memberi nama sebagai HomeScreen() dan nama berkas dart sebagai homescreen.dart

```
lib > main.dart > main
1 import 'package:flutter/material.dart';
2 import 'homescreen.dart';
      Run | Debug | Profile
3 void main() {
4 runApp(
5   MaterialApp(
6     debugShowCheckedModeBanner: false,
7     home: HomeScreen(),
8     theme: ThemeData(
9       primaryColor: Colors.white,
10      hintColor: Colors.white,
11    ), // ThemeData
12  ), // MaterialApp
13 )
14 }
```

#### Langkah 4:

1. Di dalam homescreen.dart kita akan mendapatkan lintang dan bujur lokasi saat ini. Lintang dan bujur tidak ditampilkan dalam versi akhir aplikasi, jadi kita akan mencoba mencetak nilai-nilai di terminal. Untuk ini, pertama-tama kita harus mengimpor paket geolocator di homescreen.dart.

```
lib > homescreen.dart > ...
1 import 'package:flutter/material.dart';
2 import 'package:geolocator/geolocator.dart';
3
4 class HomeScreen extends StatefulWidget {
5   @override
6   _YourWidgetName createState() => _YourWidgetName();
7 }
8 class _YourWidgetName extends State< HomeScreen> {
9   @override
10  Widget build(BuildContext context) {
11    return Container();
12  }
13  getCurrentLocation() async {
14    var p = await Geolocator.getCurrentPosition(
15      desiredAccuracy: LocationAccuracy.low,
16      forceAndroidLocationManager: true,
17    );
18    if (p != null) {
19      print('Lat: ${p.latitude}, Long: ${p.longitude}');
20    } else {
21      print('Data unavailable');
22    }
23  }
24}
25
26}
```

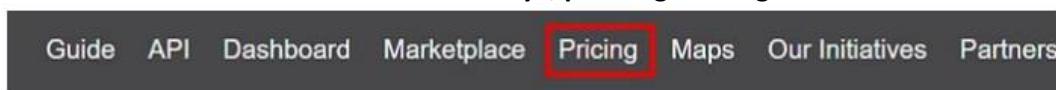
Ketika metode getCurrentLocation() dari paket Geolocator dipanggil, metode tersebut akan mengembalikan nilai Posisi (yang terdiri dari lintang, bujur, akurasi, ketinggian, arah, dll.). Karena waktu untuk mendapatkan nilai Posisi tidak diketahui dan dapat diperoleh kapan saja di masa mendatang, kita menggunakan kata kunci await. Ini berarti fungsi tersebut bekerja secara asinkron dari eksekusi yang tersisa. Akurasi posisi dapat diatur oleh properti desirableAccuracy dan pengelola lokasi android dapat diatur oleh properti forceAndroidLocationManager. Nilai Posisi yang dikembalikan terdiri dari detail lokasi perangkat saat ini. Untuk mendapatkan garis lintang dan garis bujur, parameter yang sesuai dari nilai Posisi dipanggil dan dicetak sesuai dengan itu. Fungsi ini kemudian dipanggil dalam metode initState() dari widget StatefulWidget() seperti yang ditunjukkan

```
28 // Step 1: Getting Current location co-ordinates
29 import 'package:flutter/material.dart';
30 import 'package:geolocator/geolocator.dart';
31
32 class HomeScreen extends StatefulWidget {
33   @override
34   _HomeScreenState createState() => _HomeScreenState();
35 }
36
37 class _HomeScreenState extends State<HomeScreen> {
38   @override
39   void initState() {
40     // TODO: implement initState
41     super.initState();
42     getCurrentLocation();
43   }
44
45   Future<void> getCurrentLocation() async {
46     try {
47       var position = await Geolocator.getCurrentPosition(
48         desiredAccuracy: LocationAccuracy.low,
49         forceAndroidLocationManager: true,
50       );
51       if (position != null) {
52         print('Lat: ${position.latitude}, Long: ${position.longitude}');
53       } else {
54         print('Data unavailable');
55       }
56     } catch (e) {
57       print('Error: $e');
58     }
59   }
60
61   @override
62   Widget build(BuildContext context) {
63     return SafeArea(
64       child: Scaffold(
65         // Scaffold
66       ), // SafeArea
67     );
68 }
```

Dengan ini jalankan aplikasinya dan kita akan mendapatkan lintang dan bujur lokasi kita saat ini.

```
Launching lib\main.dart on Windows in debug mode...
Nuget.exe not found, trying to download or use cached version.
✓ Built build\windows\x64\runner\Debug\flutter_api.exe
Connecting to VM Service at ws://127.0.0.1:51666/V2dUt2tnEBs=/ws
Connected to the VM Service.
flutter: Lat: -7.949007338600296, long: 112.62040465007573
```

2. Mendapatkan kunci API dari OpenWeatherMap Untuk mendapatkan data cuaca terkini, kami akan menggunakan API OpenWeatherMap. Untuk mengakses data API, kami memerlukan kunci API. Untuk melakukannya, pilih bagian Harga dari menu atas.



Kemudian scroll ke bawah hingga Anda melihat berbagai rencana untuk cuaca terkini. Pilih paket Gratis dan klik opsi Dapatkan kunci API.

Free	Startup	Developer	Professional	Enterprise
	30 GBP/month	140 GBP/month	370 GBP/month	from 1500 GBP/month
<a href="#">Get API key</a>	<a href="#">Subscribe</a>	<a href="#">Subscribe</a>	<a href="#">Subscribe</a>	<a href="#">Subscribe</a>
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
<a href="#">Current Weather</a>	<a href="#">Current Weather</a>	<a href="#">Current Weather</a>	<a href="#">Current Weather</a>	<a href="#">Current Weather</a>
<a href="#">3-hour Forecast 5 days</a>	<a href="#">3-hour Forecast 5 days</a>	<a href="#">3-hour Forecast 5 days</a>	<a href="#">3-hour Forecast 5 days</a>	<a href="#">3-hour Forecast 5 days</a>
<a href="#">Hourly Forecast 4 days</a>	<a href="#">Hourly Forecast 4 days</a>	<a href="#">Hourly Forecast 4 days</a>	<a href="#">Hourly Forecast 4 days</a>	<a href="#">Hourly Forecast 4 days</a>
<a href="#">Daily Forecast 16 days</a>	<a href="#">Daily Forecast 16 days</a>	<a href="#">Daily Forecast 16 days</a>	<a href="#">Daily Forecast 16 days</a>	<a href="#">Daily Forecast 16 days</a>
<a href="#">Climatic Forecast 30 days</a>	<a href="#">Climatic Forecast 30 days</a>	<a href="#">Climatic Forecast 30 days</a>	<a href="#">Climatic Forecast 30 days</a>	<a href="#">Climatic Forecast 30 days</a>
<a href="#">Bulk Download</a>	<a href="#">Bulk Download</a>	<a href="#">Bulk Download</a>	<a href="#">Bulk Download</a>	<a href="#">Bulk Download</a>
<a href="#">Basic weather maps</a> Historical maps	<a href="#">Basic weather maps</a> Historical maps	<a href="#">Advanced weather maps</a> Historical maps	<a href="#">Advanced weather maps</a> Historical maps	<a href="#">Advanced weather maps</a> Historical maps

Setelah selesai, akan muncul jendela baru yang mengharuskan Anda membuat akun. Berikan informasi yang diperlukan dan buat akun Anda

## Create New Account

Setelah akun dibuat, akan muncul pop-up yang menanyakan tujuan penggunaan kunci API kita

## How and where will you use our API?

X

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

Cancel

Save

Di sini, menentukan perusahaan bersifat opsional, sedangkan menentukan tujuan bersifat wajib. Jika Anda tidak tahu tujuan mana yang harus dipilih, Anda dapat memilih Pendidikan/Sains. Lalu, klik simpan.

Sekarang, di dasbor Anda, buka bagian kunci API. Di sini, Anda akan menemukan semua kunci API di bawah bagian Kunci. Harap dicatat bahwa kunci API di sini bersifat unik bagi Anda dan Anda tidak boleh membagikannya kepada orang lain.

Key	Name	Status	Actions	Create key
[REDACTED]	Default	Active		<input type="text" value="API key name"/>

Setelah Anda mendapatkan kunci API, Anda dapat melanjutkan untuk memanggil API guna mendapatkan data cuaca. Ada berbagai format pemanggilan API dan bagaimana responsnya, yang semuanya dapat dirujuk di sini. Untuk tujuan kita, kita akan memanggil API melalui format lintang bujur dan format nama kota seperti yang diberikan di bawah ini

### **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}>

The screenshot shows a Postman request to the OpenWeatherMap API. The URL is `https://api.openweathermap.org/data/2.5/weather?lat=44.34&lon=10.99&appid=2d6c883d7710b79c4a300d43b8376de2`. The request method is GET. The query parameters are: lat (44.34), lon (10.99), and appid (2d6c883d7710b79c4a300d43b8376de2). The response is a 200 OK status with a response time of 50 ms and a body size of 849 B. The response body is a JSON object:

```
1 {  
2   "coord": {  
3     "lon": 10.99,  
4     "lat": 44.34  
5   },  
6   "weather": [  
7     {  
8       "id": 804,  
9       "main": "Clouds",  
10      "description": "overcast clouds",  
11      "icon": "8dd"  
12    }  
13  ]  
14}
```

Karena kedua panggilan API memiliki kunci dan domain API yang sama, kami akan menyimpan nilai-nilai tersebut dalam file dart baru bernama constants di bawah folder lib. Untuk membuat file, lihat bagian Membangun Aplikasi di atas. Dalam file

constants.dart yang dibuat, tambahkan baris kode berikut.

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

Sekarang kami akan mencoba memperoleh data cuaca yang akan menjadi respons JSON.

```
lib >  contains.dart > ...  
1 const String domain = "https://api.openweathermap.org/data/2.5/weather?";  
2 const String apiKey = 'REDACTED'; // API Key Anda
```

**3. Mendapatkan data cuaca dari lokasi saat ini** Setelah berhasil menyelesaikan langkah-langkah di atas, kita akan mencoba mendapatkan data cuaca. Untuk ini di dalam homescreen.dart kita akan menulis fungsi untuk menghubungkan aplikasi kita ke internet terlebih dahulu, lalu memperoleh data cuaca. Di sinilah kita akan mengimplementasikan paket http. Jadi pertama-tama kita mengimpor paket sebagai http sehingga menjadi lebih mudah untuk mengakses berbagai bidang dalam paket

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

Demikian pula kita juga mengimpor constants.dart tempat kita menyalin kunci API dan tautan domain. Bersamaan dengan itu, saat kita berurusan dengan mendapatkan satu respons JSON untuk setiap panggilan, kita menggunakan pustaka konversi dart untuk mendekode respons JSON yang kita peroleh.

Panggilan API dengan format lintang bujur

<https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={A PI key}>

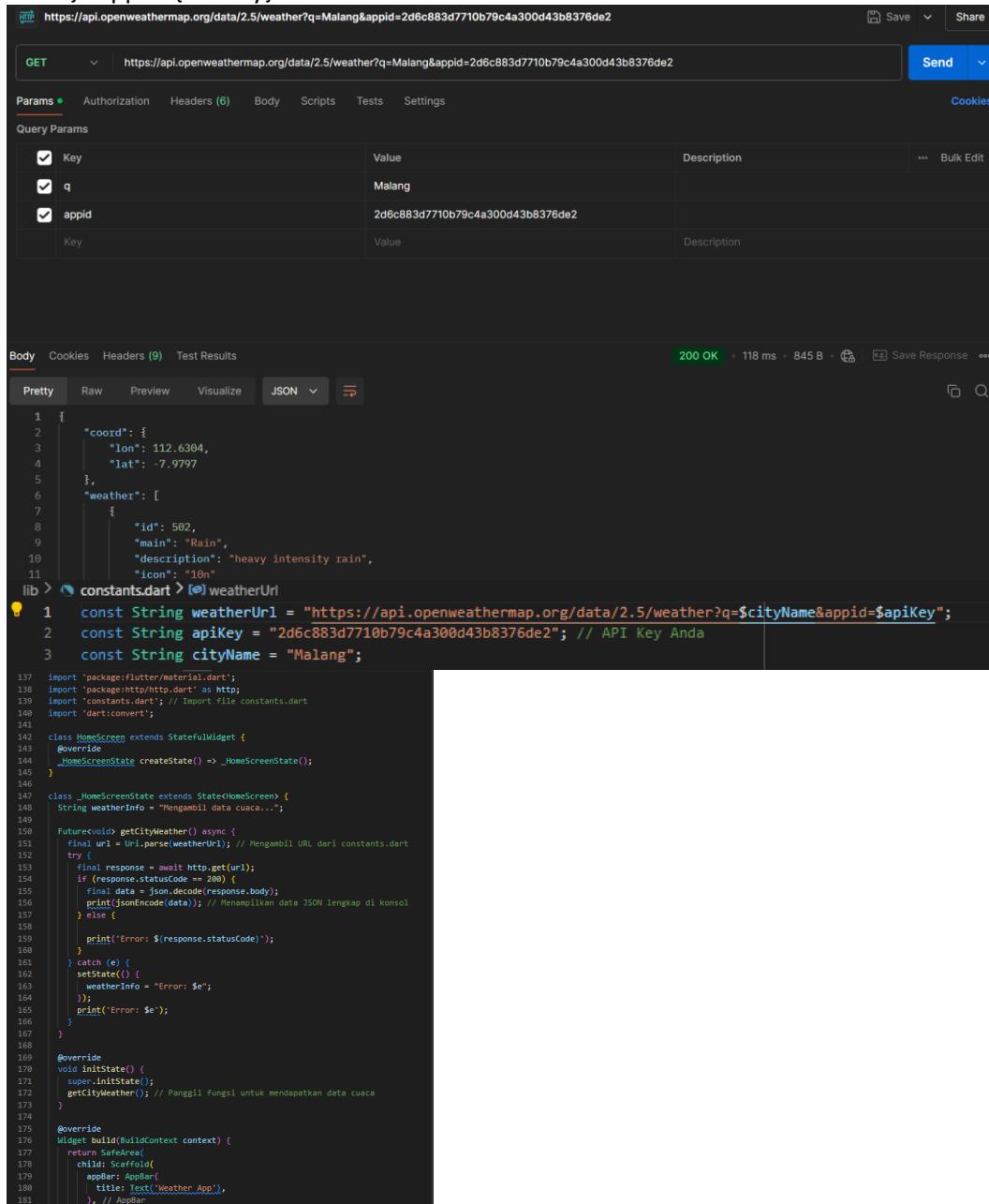
Selanjutnya kita akan menulis fungsi untuk menghubungkan aplikasi kita ke internet. Fungsi ini juga akan asynchronous karena mengembalikan masa depan. di sini kita pertama-tama membuat objek Klien sehingga kita tidak perlu membuka dan menutup port setiap kali kita memanggil metode get. Kemudian kita menyediakan alamat URI (URI adalah urutan karakter yang membantu mengidentifikasi sumber daya logis atau fisik yang terhubung ke internet) yang merupakan format panggilan API yang menyediakan bidang yang diperlukan.

```
77 class HomeScreen extends StatefulWidget {
78   @override
79   _HomeScreenState createState() => _HomeScreenState();
80 }
81
82 class _HomeScreenState extends State<HomeScreen> {
83   @override
84   void initState() {
85     super.initState();
86     getCurrentLocation();
87   }
88
89   Future<void> getCurrentLocation() async {
90     try {
91       var position = await Geolocator.getCurrentPosition(
92         desiredAccuracy: LocationAccuracy.low,
93         forceAndroidLocationManager: true,
94       );
95       if (position != null) {
96         print('Lat: ${position.latitude}, Long: ${position.longitude}');
97         fetchWeather(position.latitude, position.longitude);
98       } else {
99         print('Data unavailable');
100     }
101   } catch (e) {
102     print('Error: $e');
103   }
104 }
105
106 Future<void> fetchWeather(double lat, double lon) async {
107   final url = '${k.domain}lat=$lat&lon=$lon&appid=${k.apikey}&units=metric';
108   try {
109     final response = await http.get(Uri.parse(url));
110     if (response.statusCode == 200) {
111       final data = json.decode(response.body);
112       print('Weather: ${data['weather'][0]['description']}');
113     } else {
114       print('Error: ${response.statusCode}');
115     }
116   } catch (e) {
117     print('Error: $e');
118   }
119 }
120
121 @override
122 Widget build(BuildContext context) {
123   return SafeArea(
124     child: Scaffold(
125       appBar: AppBar(
126         title: Text('Weather App'),
127       ), // AppBar
128       body: Center(
129         child: Text('Mengambil Cuaca...'),
130       ), // Center
131     ), // Scaffold
132   ); // SafeArea
133 }
134 }
```

Connecting to VM Service at ws://127.0.0.1:54004/mLijwyIpqfU=/ws  
Connected to the VM Service.  
flutter: Lat: -7.949086938702308, Long: 112.62029253302583  
flutter: Weather: heavy intensity rain

**4. Mendapatkan data cuaca dari berbagai kota Mirip dengan LANGKAH 3, kami akan menerapkan pengambilan data cuaca dari kota tertentu berdasarkan nama kota. Di sini, satu-satunya perbedaan adalah alih memberikan garis lintang dan garis bujur dalam panggilan API, kami memberikan nama kota**

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



```

https://api.openweathermap.org/data/2.5/weather?q=Malang&appid=2d6c883d7710b79c4a300d43b8376de2
GET https://api.openweathermap.org/data/2.5/weather?q=Malang&appid=2d6c883d7710b79c4a300d43b8376de2
Params Authorization Headers (6) Body Scripts Tests Settings Cookies
Query Params
  Key Value Description Bulk Edit
  q Malang
  appid 2d6c883d7710b79c4a300d43b8376de2
  Key Value Description
  Body Cookies Headers (9) Test Results
  200 OK 118 ms 845 B Save Response ...
  Pretty Raw Preview Visualize JSON
  1 {
  2   "coord": {
  3     "lon": 112.6304,
  4     "lat": -7.9797
  5   },
  6   "weather": [
  7     {
  8       "id": 502,
  9       "main": "Rain",
 10       "description": "heavy intensity rain",
 11       "icon": "10n"
 12     }
 13   ],
 14   "base": "cmc-safex",
 15   "main": {
 16     "temp": 28.8,
 17     "feels_like": 28.8,
 18     "temp_min": 28.8,
 19     "temp_max": 28.8,
 20     "pressure": 1012,
 21     "humidity": 100
 22   },
 23   "wind": {
 24     "speed": 4.12,
 25     "deg": 115
 26   },
 27   "clouds": {
 28     "all": 100
 29   },
 30   "dt": 1571425200,
 31   "sys": {
 32     "type": 1,
 33     "id": 1852,
 34     "country": "ID",
 35     "sunrise": 1571418781,
 36     "sunset": 1571425081
 37   },
 38   "timezone": 3600,
 39   "id": 1438147,
 40   "name": "Malang"
 41 }
 42
 43 > constants.dart > weatherUrl
 44 1 const String weatherUrl = "https://api.openweathermap.org/data/2.5/weather?q=$cityName&appid=$apiKey";
 45 2 const String apiKey = "2d6c883d7710b79c4a300d43b8376de2"; // API Key Anda
 46 3 const String cityName = "Malang";
 47
 48 import 'package:flutter/material.dart';
 49 import 'package:http/http.dart' as http;
 50 import 'constants.dart'; // Import file constants.dart
 51 import 'dart:convert';
 52
 53 class HomeScreen extends StatefulWidget {
 54   @override
 55   _HomeScreenState createState() => _HomeScreenState();
 56 }
 57
 58 class _HomeScreenState extends State<HomeScreen> {
 59   String weatherInfo = "Mengambil data cuaca...";
 60
 61   Future<void> getCityWeather() async {
 62     final url = Uri.parse(weatherUrl); // Mengambil URL dari constants.dart
 63     final response = await http.get(url);
 64     if (response.statusCode == 200) {
 65       final data = json.decode(response.body);
 66       print(jsonEncode(data)); // Menampilkan data JSON lengkap di konsol
 67     } else {
 68       print("Error: ${response.statusCode}");
 69     }
 70   } catch (e) {
 71     setState(() {
 72       weatherInfo = "Error: $e";
 73     });
 74     print("Error: $e");
 75   }
 76 }
 77
 78 @override
 79 void initState() {
 80   super.initState();
 81   getCityWeather(); // Panggil fungsi untuk mendapatkan data cuaca
 82 }
 83
 84 @override
 85 Widget build(BuildContext context) {
 86   return SafeArea(
 87     child: Scaffold(
 88       appBar: AppBar(
 89         title: Text("Weather App"),
 90       ), // AppBar
 91     ),
 92   );
 93 }
 94
 95 
```

5. Menyelesaikan Pembuatan Sekarang setelah kita memperoleh data cuaca dan data lokasi, kita akan melanjutkan untuk membuat UI aplikasi. Jadi di sini kita akan menggunakan kartu pada latar belakang gradien untuk menampilkan kondisi cuaca. Untuk itu, pertama-tama kita menyediakan badan Scaffold() yang merupakan Container() yang menutupi seluruh layar dan yang memiliki isian gradien. Di sini saya telah menggunakan gradien dari Gradient Backgrounds yang memiliki koleksi besar gradien latar belakang beserta kode warna yang sesuai. Berikut ini adalah gradien yang saya pilih.

Kode untuk memperoleh gradien latar belakang lengkap seperti yang ditunjukkan di bawah ini. Di sini kita akan menyetel properti resizeToAvoidBottomInset dari widget Scaffold() ke false sehingga perubahan ukuran widget saat keyboard muncul dapat dihindari.

```

198 import 'package:flutter/material.dart';
199 import 'package:http/http.dart' as http;
200 import 'constants.dart'; // Import file constants.dart
201 import 'dart:convert';
202
203 class HomeScreen extends StatefulWidget {
204   @override
205   _HomeScreenState createState() => _HomeScreenState();
206 }
207
208 class _HomeScreenState extends State<HomeScreen> {
209
210   Future<void> getCityWeather() async {
211     final url = Uri.parse(weatherUrl); // Mengambil URL dari constants.dart
212     try {
213       final response = await http.get(url);
214       if (response.statusCode == 200) {
215         final data = json.decode(response.body);
216         print(jsonEncode(data)); // Menampilkan data JSON lengkap di konsol
217       } else {
218         print('Error: ${response.statusCode}');
219       }
220     } catch (e) {
221       setState(() {
222         | "Error: $e";
223       });
224       print('Error: $e');
225     }
226   }
227
228   @override
229   void initState() {
230     super.initState();
231     getCityWeather(); // Panggil fungsi untuk mendapatkan data cuaca
232   }
233
234   @override
235   Widget build(BuildContext context) {
236     return SafeArea(
237       child: Scaffold(
238
239         body: Container(
240           decoration: BoxDecoration(
241             gradient: LinearGradient(
242               begin: Alignment.topCenter,
243               end: Alignment.bottomCenter,
244               colors: [
245                 | Color(0xFFBF43F), // Warna awal gradasi (#8ff43f)
246                 | Color(0xFF16A085), // Warna akhir gradasi (#16A085)
247               ],
248             ), // LinearGradient
249           ), // BoxDecoration
250           child: Center(
251             child: Padding(
252               padding: const EdgeInsets.all(16.0),
253               child: Text("Mengambil data cuaca...", // Mengambil data cuaca...
254                 style: TextStyle(
255                   | fontSize: 18,
256                   | color: Colors.white, // Mengatur warna teks menjadi putih
257                 ), // TextStyle
258                 textAlign: TextAlign.center,
259               ), // Text
260             ), // Padding
261           ), // Center
262         ), // Container
263       ), // Scaffold
264     ); // SafeArea
265   }
266 }

```

Kemudian kami akan menyediakan beberapa variabel untuk menyimpan nilai suhu, tekanan, kelembaban, tutupan awan, nama kota dan status data, yaitu apakah data tersebut diambil dan siap digunakan oleh aplikasi. Variabel-varibel ini dideklarasikan di dalam widget Stateful sebelum metode initState.

```

class _HomeScreenState extends State<HomeScreen> {
  bool isLoading = false;
  num temp = 0;
  num press = 0;
  num hum = 0;
  num cover = 0;
  String cityname = '';

```

```

    temp = data['main']['temp']; // Suhu
    press = data['main']['pressure']; // Tekanan
    hum = data['main']['humidity']; // Kelembapan
    cover = data['clouds']['all']; // Tutupan awan
    cityname = data['name']; // Nama kota
    isLoading = true; // Menandakan data sudah siap
);

```

Kota: Malang

Suhu: 296.4°C

Tekanan: 1012 hPa

Kelembapan: 97%

Tutupan Awan: 98%

Selanjutnya kita akan menambahkan widget Visibilitas sebagai anak dari widget Kontainer. Jadi hanya saat ada data data cuaca akan ditampilkan, selain itu akan ditampilkan indikator pemuatan. Untuk ini nilai isLoading diubah secara dinamis dalam fungsi menggunakan metode setState. Kodenya seperti yang diberikan di bawah ini.

```

child: Visibility(
  visible: isLoading,
  child: Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      Text(
        "Kota: $cityname",
        style: TextStyle(
          fontSize: 18,
          color: Colors.white,
        ), // TextStyle
      ),
      SizedBox(height: 8),
      Text(
        "Suhu: ${temp.toStringAsFixed(1)}°C",
        style: TextStyle(
          fontSize: 18,
          color: Colors.white,
        ), // TextStyle
      ),
      SizedBox(height: 8),
      Text(
        "Tekanan: ${press} hPa",
        style: TextStyle(
          fontSize: 18,
          color: Colors.white,
        ), // TextStyle
      ),
      SizedBox(height: 8),
      Text(
        "Kelembapan: ${hum}%",
        style: TextStyle(
          fontSize: 18,
          color: Colors.white,
        ), // TextStyle
      ),
      SizedBox(height: 8),
      Text(
        "Tutupan Awan: ${cover}%",
        style: TextStyle(
          fontSize: 18,
          color: Colors.white,
        ), // TextStyle
      ),
    ],
  ),
);

```

```

        replacement: Center(
            child: CircularProgressIndicator(
                color: Colors.white,
            ), // CircularProgressIndicator
        ), // Center
    ), // Visibility
), // Padding
), // Center
), // Container
), // Scaffold
); // SafeArea
}
}

```

Sekarang, untuk menyimpan data cuaca ke dalam variabel, kita akan menyediakan fungsi lain untuk memperbarui variabel. Fungsi ini akan memiliki data JSON yang didekode sebagai parameter dan berdasarkan nilainya, nilai parameter ditetapkan.

```

void updateUI(var decodedData) {
    setState(() {
        if (decodedData == null) {
            temp = 0;
            press = 0;
            hum = 0;
            cover = 0;
            cityname = 'Not available';
            isLoaded = false;
        } else {
            temp = decodedData['main']['temp'] - 273; // Konversi Kelvin ke Celsius
            press = decodedData['main'][['pressure']];
            hum = decodedData['main'][['humidity']];
            cover = decodedData['clouds'][['all']];
            cityname = decodedData['name'];
            isLoaded = true;
        }
    });
}

```

Fungsi ini akan dipanggil di kedua fungsi pemanggil API jika kode status data yang diterima adalah 200

```

Future<void> getCityWeather() async {
    final url = Uri.parse(weatherUrl); // Mengambil URL dari constants.dart
    try {
        final response = await http.get(url);
        if (response.statusCode == 200) {
            final data = json.decode(response.body);
            updateUI(data); // Memperbarui variabel dengan data JSON
            print(jsonEncode(data)); // Menampilkan data JSON lengkap di konsol
        } else {
            setState(() {
                isLoaded = false;
            });
            print('Error: ${response.statusCode}');
        }
    } catch (e) {
        setState(() {
            isLoaded = false;
        });
        print('Error: $e');
    }
}

```

Kode yang disorot dengan huruf tebal ditambahkan ke fungsi. Pada dasarnya, kode tersebut mendekode data JSON dan memanggil metode updateUI yang menetapkan nilai variabel ke data cuaca terkait. Bersamaan dengan itu, variabel yang mewakili status data yang diperoleh juga diperbarui di seluruh proses dengan bantuan metode setState. Demikian pula untuk fungsi yang menggunakan panggilan API dengan nama kota, kode yang sama ditambahkan.

```

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);
}
}

```

```
Q | Search city
```

Di dalam kolom widget `Visibility`, kita akan menambahkan `TextField` sebagai anak pertama. Untuk pengontrol `TextField` ini, pengontrol juga disediakan, yang dideklarasikan bersama dengan variabel.

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

Kode `TextField` seperti yang diberikan di bawah ini. Perhatikan bahwa kami telah menetapkan nilai `isLoaded` menjadi `false` setelah nama kota dimasukkan. Ini memberikan indikator pemuatan kepada pengguna saat mengambil data, sehingga pengguna akan tahu ada beberapa proses yang sedang berlangsung.

```
child: Column(
  mainAxisAlignment: MainAxisAlignment.center,
  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),
        ),
      ),
    ),
  ],
),
```

Karena lebar tidak dapat ditentukan dalam `TextField`, lebarnya dibungkus di dalam wadah.

```
Q | Search city
```

Berikutnya kita akan menambahkan anak berikutnya dari `Column()` yang merupakan widget `SizedBox` untuk menambahkan ruang yang diperlukan di antara komponen-komponen.

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

Hal ini diikuti oleh data nama Kota.



Hal ini diimplementasikan dengan menggunakan widget `Row()` yang dibungkus dengan widget `Padding`. Widget ini terdiri dari Ikon dan Teks dan kodennya seperti yang diberikan di bawah ini

```

Padding(
  padding: const EdgeInsets.all(8.0),
  child: Row(
    crossAxisAlignment: CrossAxisAlignment.end,
    children: [
      Icon(
        Icons.pin_drop,
        color: Colors.red,
        size: 40,
      ),
      SizedBox(width: 8), // Memberikan jarak antara ikon dan teks
      Text(
        cityname,
        overflow: TextOverflow.ellipsis,
        style: TextStyle(
          fontSize: 28,
          fontWeight: FontWeight.bold,
          color: Colors.white,
        ),
      ),
    ],
  ),
),
SizedBox(height: 20) // Memberikan jarak tambahan

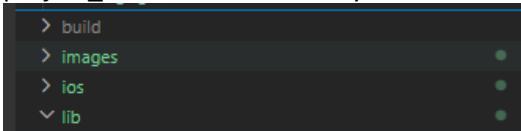
```

Hal ini diikuti lagi oleh widget `SizedBox`

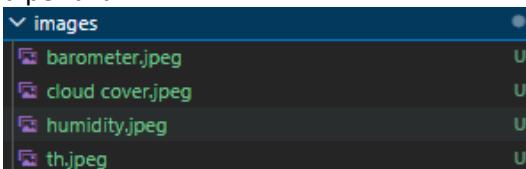
```
SizedBox(height: 20),
```

Berikutnya adalah bagian data cuaca yang ditampilkan sebagai kartu. Di sini saya menggunakan Container untuk membuat kartu kustom. Untuk data, saya menggunakan widget Gambar dan Teks.

Perhatikan bahwa gambar yang digunakan di sini berasal dari berkas proyek itu sendiri. Jadi untuk melakukannya, kita harus mengonfigurasi gambar tersebut. Untuk ini, pertama-tama buat direktori baru di folder proyek di bawah  
`project_name>New>Directory`



Beri nama direktori baru sebagai image. Setelah direktori dibuat, tambahkan gambar yang diperlukan



Setelah itu, buka file pubspec.yaml Anda. Gulir ke bawah ke bagian asset tempat gambar dimasukkan. Hapus komentar pada baris dan pastikan spasi sudah benar seperti yang ditunjukkan di bawah ini. Kode di bawah ini menyertakan semua file di bawah direktori gambar.

```
uses-material-design: true
assets:
- images/
```

Lalu jalankan perintah `pub get` di sudut kanan atas.

```
PS C:\Users\Werdanindra Ppinkan\OneDrive\Documents\SEMESTER 5\PEMR_MOB_SIB\project12\flutter_api> flutter pub get
Resolving dependencies... (1.3s)
Downloading packages...
async 2.11.0 (2.12.0 available)
boolean 3.1.0 (3.1.1 available)
characters 1.3.0 (1.3.1 available)
clock 1.1.1 (1.1.2 available)
collection 1.18.0 (1.19.1 available)
fake_async 1.3.1 (1.3.2 available)
flutter_lints 4.0.0 (5.0.0 available)
http_parser 4.1.1 (4.1.2 available)
lazycache 0.0.5 (0.0.6 available)
leak_tracker_flutter_testing 3.0.5 (3.0.9 available)
lints 4.0.0 (5.1.0 available)
matcher 0.12.16+1 (0.12.17 available)
material_color_utilities 0.11.1 (0.12.0 available)
meta 1.15.0 (1.16.0 available)
path 1.9.0 (1.9.1 available)
stack_trace 1.11.1 (1.12.0 available)
```

Setelah selesai, gambar akan dikonfigurasikan di aplikasi. Sekarang di homescreen.dart kita akan menambahkan kode untuk membuat tampilan kartu.

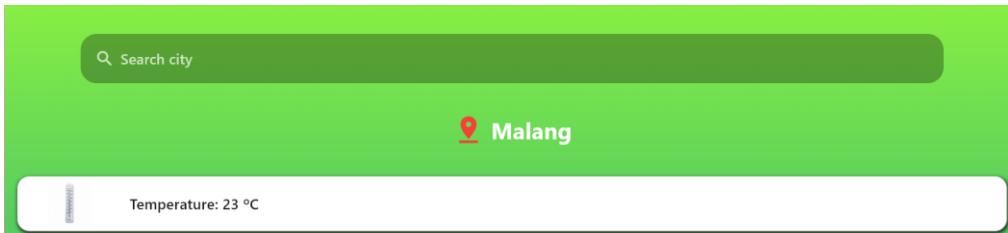
```

Widget buildWeatherCard(String title, String value, String imagePath) {
  return Container(
    width: double.infinity,
    height: MediaQuery.of(context).size.height * 0.10, // Sesuaikan tinggi kartu
    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,
        ), // BoxShadow
      ],
    ), // BoxDecoration
    child: Row(
      children: [
        Padding(
          padding: const EdgeInsets.all(8.0),
          child: Image(
            image: AssetImage('images/th.jpeg'),
            width: MediaQuery.of(context).size.width * 0.09,
          ), // Image
        ), // Padding
        SizedBox(width: 10),
        Text(
          '$title: $value',
          style: TextStyle(fontSize: 18, fontWeight: FontWeight.w600),
        ), // Text
      ],
    ), // Row
  ); // Container
}

```

Setelah kode di atas ditambahkan, hentikan dan mulai aplikasi secara otomatis agar perubahan konfigurasi dapat disertakan. Saat aplikasi dimuat, kita akan mendapatkan tampilan kartu seperti yang ditunjukkan di bawah ini.

Sekali lagi 3 kartu ditambahkan dengan mengganti gambar dan teks untuk parameter cuaca masing-masing. Untuk tampilan 3 kartu berikutnya, gambar dibungkus dengan bantalan di semua sisi.



Sekali lagi 3 kartu ditambahkan dengan mengganti gambar dan teks untuk parameter cuaca masing-masing. Untuk tampilan 3 kartu berikutnya, gambar dibungkus dengan bantalan di semua sisi.

