

LEARNING FLETTER

Flutter adalah SDK aplikasi Mobile dari Google yang digunakan untuk membangun aplikasi native iOS, Android, Desktop (Windows, Linux, macOS), dan Web dari satu source code. Saat membangun aplikasi dengan Flutter, semuanya berkaitan dengan Widgets, yaitu sebuah blok yang digunakan untuk membangun aplikasi Flutter. Widget merupakan elemen struktural yang dilengkapi dengan banyak fungsi khusus untuk material design, dan widget baru dapat digunakan dengan cara menyusun widget yang sudah ada. Proses menggabungkan widget disebut komposisi. Antarmuka pengguna aplikasi terdiri dari banyak widget sederhana yang memiliki tugasnya masing-masing. Itulah sebabnya mengapa pengembang Flutter cenderung memandang aplikasi Flutter mereka sebagai pohon widget.

Flutter adalah framework open source untuk membuat aplikasi mobile berkualitas tinggi dan performa tinggi pada sistem operasi mobile seperti Android dan iOS. Framework ini menyediakan SDK yang sederhana, efisien, dan mudah dipahami untuk membuat aplikasi mobile menggunakan bahasa pemrograman Dart milik Google. Tutorial ini membahas dasar-dasar framework Flutter, proses instalasi SDK Flutter, cara menyiapkan Android Studio untuk mengembangkan

aplikasi berbasis Flutter, desain struktur Flutter, serta cara mengembangkan berbagai jenis aplikasi seluler menggunakan framework Flutter.

Kenapa memilih flutter

1. *High-Performance App*: Aplikasi yang dikembangkan menggunakan Flutter sangat ekspresif dan memiliki antarmuka pengguna yang fleksibel. Pengembangan yang cepat berkat fitur hot reloading membuat aplikasi menjadi hidup dan ekspresif dengan memberikan fitur-fitur yang diinginkan untuk end user experience yang mirip dengan aplikasi asli.
2. *Expressive and Flexible UI*: Flutter memungkinkan pengembang untuk membangun aplikasi yang indah dengan mudah menggunakan widget material pre-build. Meskipun banyak widget telah ada, tetapi Flutter memungkinkan penggunaan widget yang fleksibel.
3. *Fast Development & Hot Reloading*: Hot Reloading mengacu pada penyisipan versi baru dari file yang diedit saat runtime sambil menjalankan aplikasi.

Kelebihan dan Kekurangan dari Flutter

Kelebihan:

- Flutter menggunakan satu source code tunggal, yaitu Dart, untuk kedua platform, Android dan iOS, yang merupakan bahasa yang sederhana dan menjamin keamanan tipe data.

- Baik bahasa Flutter maupun komunitasnya sedang berkembang dengan pesat dan merilis fitur, widget, serta add-on baru secara teratur.
- Flutter memiliki set widget-nya sendiri, daripada menggunakan widget yang disediakan oleh sistem operasi host. Hal ini berarti pengguna dapat memberikan model pengenalan gestur sendiri dan memiliki kontrol yang lebih besar atas rendering yang akurat atau penyesuaian widget.
- Hot-reloading menjadi game changer dalam produktivitas proses development. Ini memberikan efek yang hidup pada aplikasi yang sedang dikembangkan, sehingga membuat seluruh siklus pengembangan lebih menarik bagi pengembang UI / UX yang menggunakan Flutter.
- Flutter tidak terikat pada ROM sehubungan dengan sistem widget. Jadi, meningkatkan portabilitasnya pada berbagai versi Android dan dengan demikian, mengurangi ketergantungannya pada platform host.
- Dart dan Flutter menyatu secara erat untuk mengoptimalkan Virtual Machine (VM) dart untuk ponsel yang khusus dibutuhkan oleh Flutter.
- Flutter merupakan pemain terkemuka dalam bidang pengembangan aplikasi lintas platform dengan dukungan komunitas yang luar biasa.

Kekurangan:

- Sebenarnya, tidak ada kekurangan pada Flutter karena tidak ada kerangka kerja lain yang seefektif dan sekompleks Flutter. Meskipun jika harus mencantumkan, akan berkaitan dengan bahasa

pemrograman Dart karena ketika mengonversi dart ke JavaScript ada beberapa bug yang harus diperbaiki, dart tidak memiliki kerangka kerja untuk backend, dan sebagainya.

Arsitektur Flutter

Arsitektur aplikasi adalah suatu konsep atau rancangan struktural yang mengatur bagaimana komponen-komponen pada aplikasi saling berinteraksi, berkomunikasi, dan terorganisasi secara sistematis. Arsitektur aplikasi bertujuan untuk memastikan bahwa aplikasi dapat berjalan dengan baik, mudah dikembangkan, diuji, dipelihara, dan ditingkatkan.

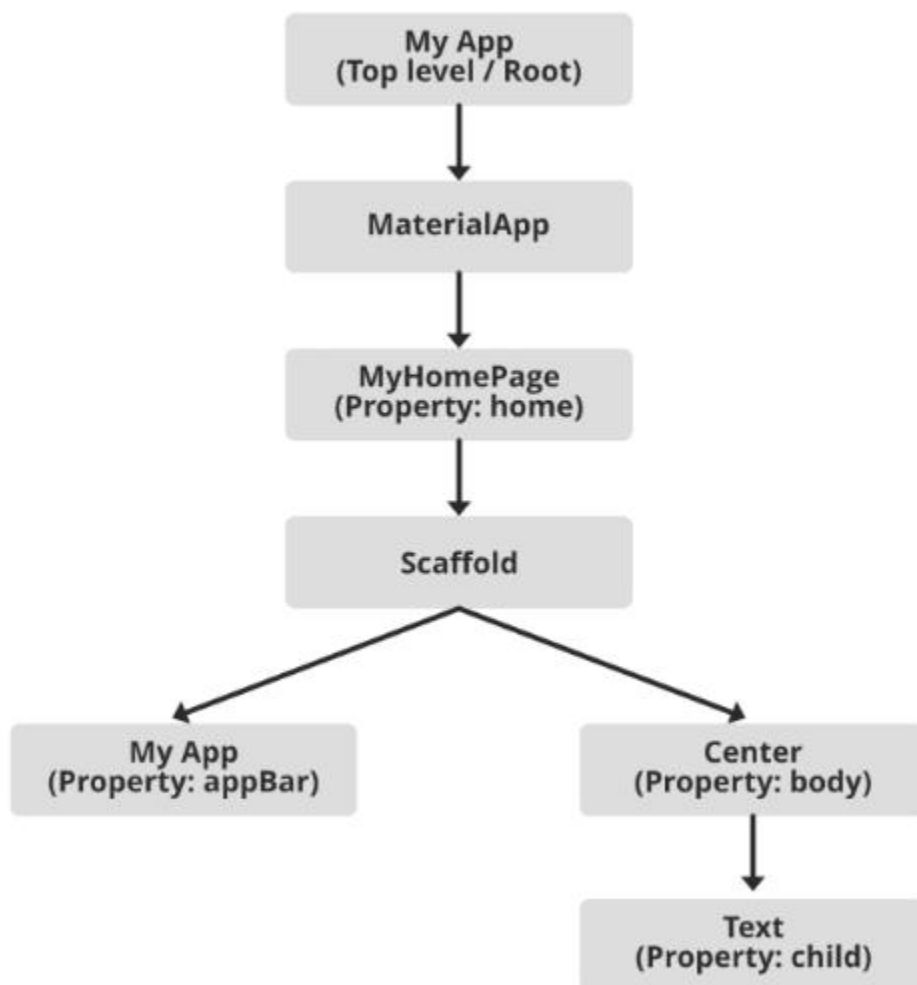
Arsitektur aplikasi Flutter terdiri dari :

1. Widget
2. Gesture
3. Concept of State
4. Layer

Widget :

Widget adalah komponen utama dalam aplikasi Flutter. Widget berfungsi sebagai antarmuka pengguna (UI) yang digunakan untuk berinteraksi dengan aplikasi. Aplikasi Flutter itu sendiri terdiri dari kombinasi beberapa widget. Pada aplikasi standar yang dikembangkan menggunakan Flutter, root widget akan mendefinisikan struktur dari aplikasi, diikuti oleh Material App widget

yang menempatkan komponen internalnya di tempat. Di sinilah properti UI dan aplikasi itu sendiri ditetapkan. Material App memiliki widget Scaffold yang terdiri dari komponen yang terlihat (widget) dari aplikasi. Scaffold memiliki dua properti utama, yaitu body dan appBar. Properti ini menampung semua child widget dan di sinilah semua propertinya didefinisikan. Diagram di bawah ini menunjukkan hierarki dari aplikasi Flutter:



Di dalam Scaffold, biasanya terdapat widget appBar, yang seperti namanya menentukan appBar dari aplikasi. Scaffold juga memiliki body di mana semua widget komponen ditempatkan. Ini adalah tempat di mana properti-widget ini diatur. Semua widget ini digabungkan untuk membentuk halaman utama aplikasi itu sendiri. Widget Center memiliki properti Child, yang merujuk pada konten sebenarnya dan dibangun menggunakan widget Text.

Gesture :

Untuk melakukan interaksi fisik dengan aplikasi flutter, semua bentuk gerakan telah ditentukan sebelumnya dan dapat dilakukan melalui **Gesture-Detectors**, yaitu widget yang tidak terlihat namun digunakan untuk memproses interaksi tersebut. Gerakan tersebut mencakup mengetuk, menarik, menggeser, dan lain sebagainya. Dengan menggunakan fitur-fitur ini secara kreatif, pengalaman pengguna dari aplikasi dapat ditingkatkan dengan membuatnya melakukan tindakan yang diinginkan dengan mudah.

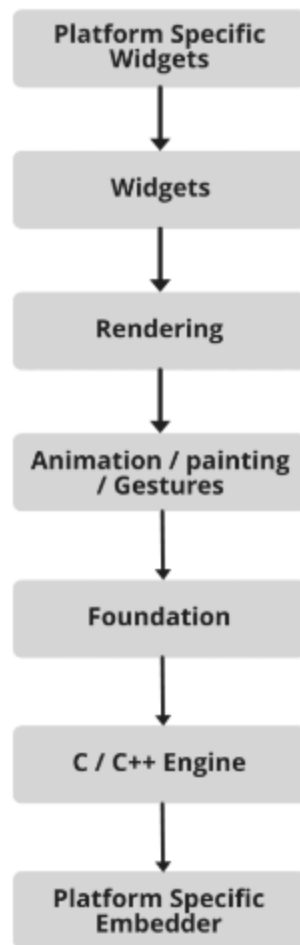
Konsep dari State :

Jika Anda pernah bekerja dengan menggunakan React-js, Anda mungkin sudah familiar dengan konsep state. State hanyalah sebuah object data. Flutter juga bekerja di area yang sama. Untuk mengelola state dalam aplikasi Flutter, digunakan Stateful-Widget. Mirip dengan konsep state dalam React-js, widget-widget tertentu akan di-render ulang ketika state berubah. Hal ini juga menghindari rendering ulang seluruh aplikasi setiap kali state dari widget berubah.

Arsitektur aplikasi Flutter atau framework Flutter secara umum terdiri dari kombinasi widget kecil dan besar yang berinteraksi untuk membangun aplikasi. Semua lapisan di dalamnya penting untuk desain dan fungsinya. Meskipun membangun aplikasi di flutter terlihat sederhana, sebenarnya memiliki komponen yang sama kompleks di inti aplikasinya.

Layer :

Framework Flutter memiliki kategori yang biasanya dikategorikan berdasarkan tingkat kompleksitasnya dan membentuk hierarki yang menurun. Kategori ini dikenal sebagai lapisan (layer) dan dibangun satu per satu. Lapisan paling atas terdiri dari widget yang khusus untuk sistem operasi perangkat seperti Android atau iOS. Lapisan kedua terdiri dari widget asli Flutter, termasuk komponen-komponen UI struktural, detektor gerakan, state management, dan lain sebagainya. Lapisan ketiga adalah tempat di mana semua penggambaran UI dan state terjadi dan mencakup semua komponen yang terlihat dari aplikasi Flutter. Lapisan berikutnya terdiri dari animasi yang digunakan dalam transisi, aliran gambar, dan gerakan. Diagram di bawah ini memberikan gambaran yang sama:



Install Flutter

Bahan Persiapan Tools :

1. Android Studio : <https://developer.android.com/studio>
2. Flutter SDK for windows:
<https://docs.flutter.dev/get-started/install/windows/desktop>
3. Git : <https://git-scm.com/downloads>
4. Java development Kit JDK :
<https://www.oracle.com/in/java/technologies/downloads/#jdk23-windows>

Menginstal Flutter dan mengatur Android Studio untuk menjalankan Aplikasi Flutter. Android Studio merupakan salah satu Integrated Development Environment (IDE) yang populer dan dikembangkan oleh Google sendiri untuk membuat aplikasi Android lintas platform. Untuk mengintegrasikan Flutter ke dalam Android Studio, pertama-tama Anda perlu menginstal Android Studio versi 3.0 atau yang lebih baru, karena menyediakan pengalaman IDE terintegrasi untuk Flutter. Untuk informasi lebih lanjut tentang penginstalan Android Studio, Anda dapat merujuk ke dokumentasi resmi Android Studio

Install Plugin Flutter dan Dart :

Setelah berhasil menginstal Android Studio, langkah selanjutnya adalah menginstal plugin Flutter dan Dart. Berikut adalah langkah-langkah untuk menginstal plugin tersebut:

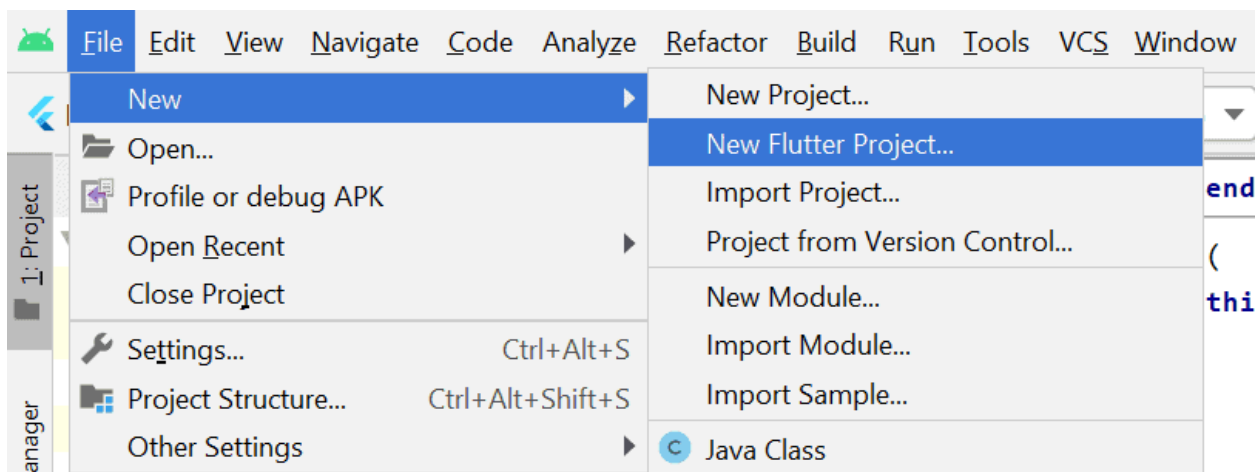
1. Buka Android Studio.
2. Buka Preferensi Plugin. Untuk macOS, Anda dapat membukanya dengan cara klik Preferences > Plugins. Untuk Windows dan Linux, klik File > Settings > Plugin.
3. Pilih Marketplace, kemudian cari dan pilih plugin Flutter dan klik Install.
4. Setelah plugin Flutter terinstal, klik Yes ketika diminta untuk menginstal plugin Dart.
5. Setelah proses penginstalan selesai, restart Android Studio.

Dengan mengikuti langkah-langkah di atas, Anda dapat mengintegrasikan Flutter ke dalam Android Studio dan mulai membuat aplikasi lintas platform yang menarik dan inovatif.

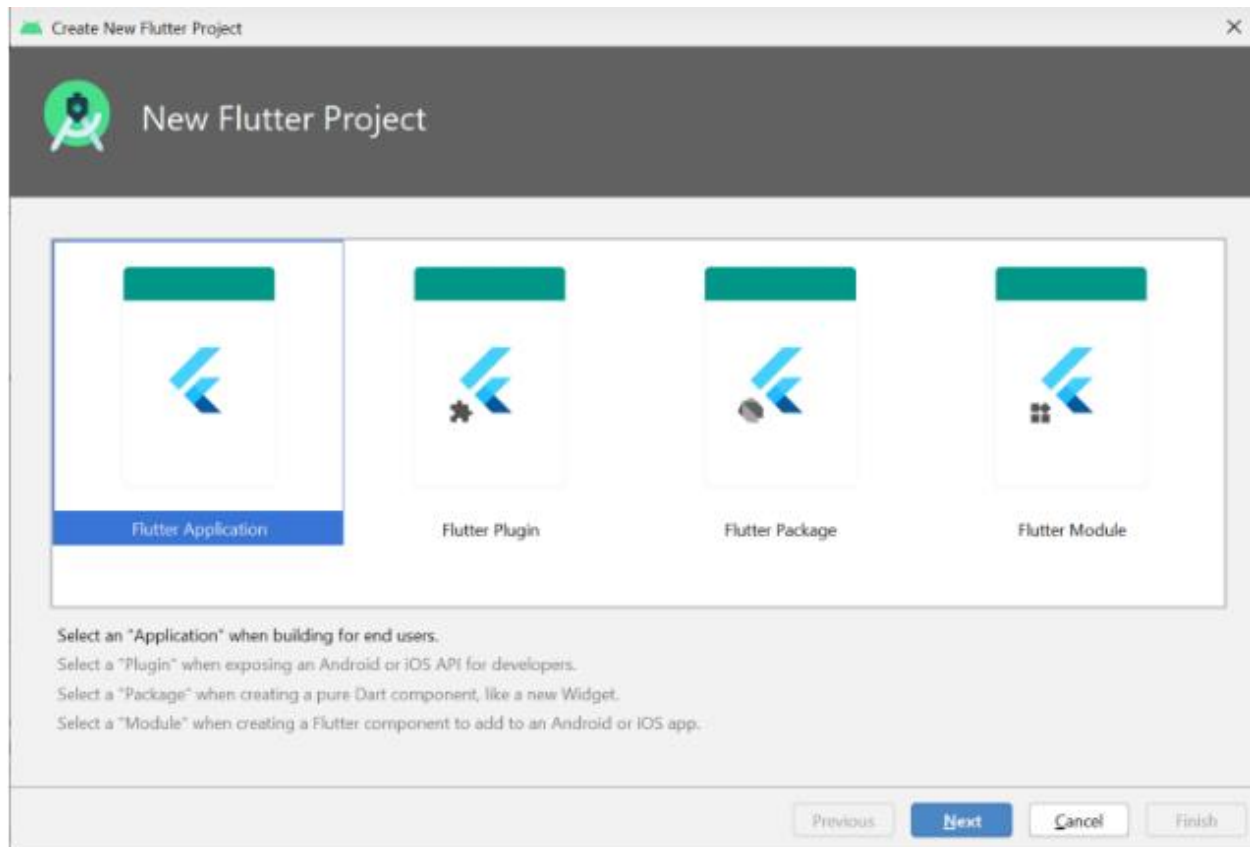
Membuat Project Baru:

Setelah menginstal plugin Dart dan Flutter, buatlah aplikasi flutter untuk memeriksa apakah berfungsi dengan baik atau tidak. Untuk melakukannya, ikuti langkah-langkah berikut:

Langkah 1: Buka IDE dan pilih "New Flutter Project".



Langkah 2: Pilih jenis proyek Flutter sebagai aplikasi. Kemudian klik tombol Berikutnya.



Langkah 3: Verifikasi jalur Flutter SDK menunjukkan lokasi SDK (jika kolom teks kosong, pilih "Install SDK...").

Langkah 4: Masukkan nama proyek (misalnya, myapp), lalu klik Next.

Create New Flutter Project

New Flutter Application

Project name

flutter_app1

Flutter SDK path

C:\flutter

... Install SDK...

Project location

C:\Users\ymsaur\AndroidStudioProjects

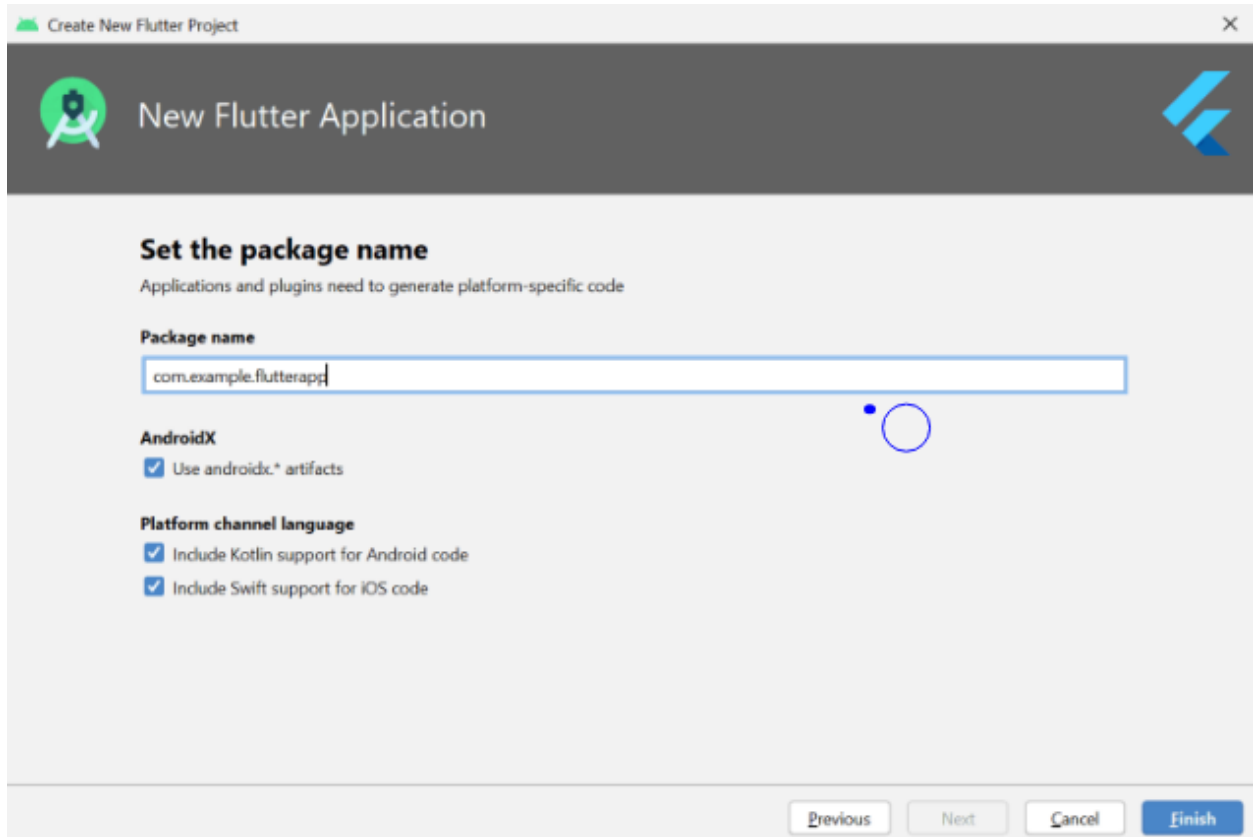
Description

A new Flutter application.

☐ Create project offline

Previous Next Cancel Finish

Langkah 5: Klik Finish.



Langkah 6: Tunggu hingga Android Studio selesai menginstal SDK dan membuat proyek.

Catatan: Ketika membuat aplikasi Flutter baru, beberapa plugin IDE Flutter meminta untuk memberikan nama domain perusahaan secara terbalik, seperti contoh `com.example`. Nama domain perusahaan dan nama proyek akan digunakan bersama sebagai nama paket untuk Android (ID Bundle untuk iOS) saat aplikasi dirilis. Jika Anda berpikir bahwa aplikasi mungkin akan dirilis, sebaiknya tetapkan nama paket sekarang. Nama paket tidak dapat diubah setelah aplikasi dirilis, oleh karena itu buatlah nama yang unik.

Langkah-langkah di atas membuat direktori proyek Flutter bernama flutter_app yang berisi aplikasi demo sederhana yang menggunakan Komponen Material.

Menjalankan Aplikasi :

Untuk menjalankan aplikasi, ikuti langkah-langkah berikut:

Langkah 1: Temukan toolbar utama Android Studio.

Langkah 2: Pada pemilih target, pilih perangkat Android yang akan digunakan untuk menjalankan aplikasi. Jika tidak tersedia, buat AVD baru dengan memilih Tools > Android > AVD Manager. Untuk informasi lebih lengkap, lihat bagaimana mengelola AVD.

Langkah 3: Klik ikon run pada toolbar atau pilih menu Run > Run.

Setelah proses pembangunan aplikasi selesai, aplikasi starter akan muncul pada perangkat Anda.

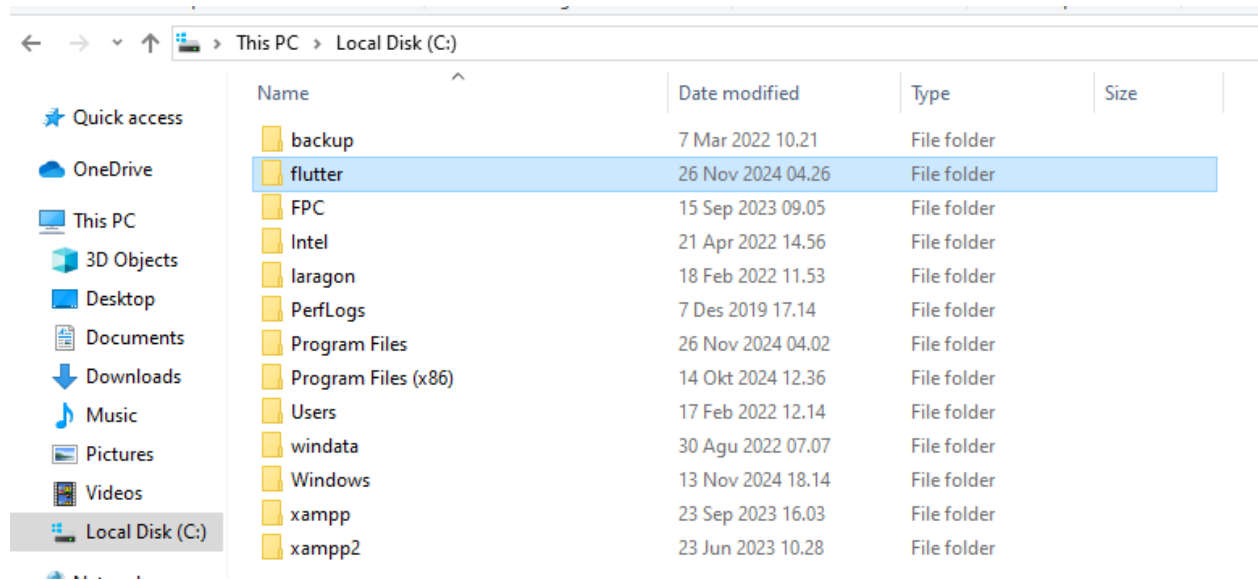


Cara Membuat Project Flutter di Visual Studio Code :

- **Langkah 1: Menyiapkan Flutter SDK**

1. Unduh SDK terbaru di sini: [Flutter SDK Archive](#)
2. Ekstrak file zip dan letakkan folder 'flutter' yang berisi di direktori yang diinginkan.

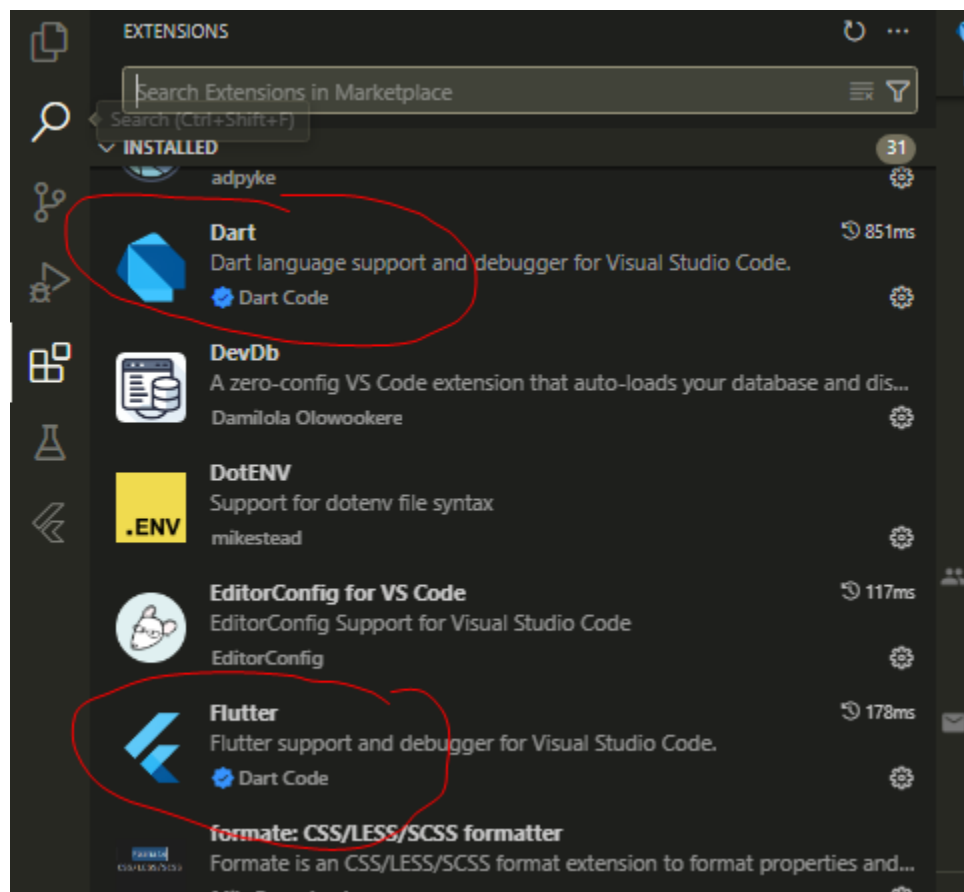
Catatan: Disarankan untuk tidak menginstal Flutter di direktori yang memerlukan hak admin, seperti 'C:\Program Files'.



- **Langkah 2:** Tambahkan Flutter ke **PATH**: Meskipun tidak diperlukan, disarankan untuk mengatur variabel **PATH** agar Flutter mudah diakses dari mana saja di sistem.
 1. Untuk mengatur variabel PATH, buka **Edit environment variables for your account** di Control Panel. Anda dapat mencari pengaturan ini di bilah pencarian.
 2. Di bawah user variable, periksa apakah ada entri yang disebut PATH:
 - Jika sudah ada, tambahkan PATH baru ke 'flutter\bin'.
 - Jika entri tersebut belum ada, buat entri baru bernama PATH, dan kemudian tambahkan lokasi lengkap ke 'flutter\bin'.
 3. Restart ulang Windows setelah mengatur variabel PATH agar dapat berfungsi.
- **Langkah 3:** Siapkan **Android Studio**: Android Studio secara otomatis mengunduh alat pengembangan yang diperlukan agar Flutter dapat berfungsi dengan Android.
 1. Unduh Android Studio di sini: [Android Studio](#)
 2. Mulai Android Studio dan ikuti wizard SDK Manager untuk mengunduh semua alat build yang diperlukan.

- **Langkah 4: Siapkan Visual Studio Code:** Visual Studio Code (atau VS Code) adalah code editor ringan yang dapat digunakan dalam pengembangan Flutter. Dalam artikel ini, VS Code digunakan sebagai code editor karena lebih ringan dan memiliki fitur minimal yang diperlukan.

1. Unduh dan instal VS Code: Visual Studio Code
2. Untuk membantu dalam pengembangan, dua ekstensi atau plugin pada VS Code harus dipasang. Instal plugin Flutter dan Dart dari Tab Ekstensi VS Code. Anda dapat merujuk: [Setting Up VS Code Extensions for Flutter](#)



3. Pasang Git Bash: Meskipun bersifat opsional, disarankan untuk digunakan sebagai command prompt. Git Bash menyediakan banyak perintah unix yang umum

digunakan untuk beberapa tugas cepat. Anda dapat mengunduh Git Bash untuk Windows di sini: Git.

4. Jalankan Flutter Doctor: Flutter doctor adalah alat bawaan oleh Flutter yang dapat digunakan untuk memeriksa status instalasi Flutter. Setelah mengatur variabel PATH, Anda dapat membuka Command Prompt dan mengeksekusi perintah “flutter doctor”.

```
MINGW64; c:/Users/Home-PC
Home-PC@DESKTOP-1A6NIAI MINGW64 ~
$ flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.24.5, on Microsoft Windows [Version 10.0.19045.5131], locale id-ID)
[✓] Windows Version (Installed version of Windows is version 10 or higher)
[!] Android toolchain - develop for Android devices (Android SDK version 32.1.0-rc1)
    X cmdline-tools component is missing
      Run 'path/to/sdkmanager --install "cmdline-tools;latest"'
      See https://developer.android.com/studio/command-line for more details.
    X Android license status unknown.
      Run 'flutter doctor --android-licenses' to accept the SDK licenses.
      See https://flutter.dev/to/windows-android-setup for more details.
[✓] Chrome - develop for the web
[X] Visual Studio - develop Windows apps
    X Visual Studio not installed; this is necessary to develop Windows apps.
      Download at https://visualstudio.microsoft.com/downloads/.
      Please install the "Desktop development with C++" workload, including all
of its default components
[✓] Android Studio (version 2021.1)
[✓] VS Code (version 1.95.3)
[✓] Connected device (3 available)
[✓] Network resources

! Doctor found issues in 2 categories.

Home-PC@DESKTOP-1A6NIAI MINGW64 ~
$ |
```

```
Home-PC@DESKTOP-1A6NIAI MINGW64 ~
$ flutter doctor --version
Flutter 3.24.5 • channel stable • https://github.com/flutter/flutter.git
Framework • revision dec2ee5c1f (12 days ago) • 2024-11-13 11:13:06 -0800
Engine • revision a18df97ca5
Tools • Dart 3.5.4 • DevTools 2.37.3

Home-PC@DESKTOP-1A6NIAI MINGW64 ~
```

Membuat Proyek Kosong

Arahkanlah ke tempat di mana Anda ingin membuat proyek Anda. Kemudian, bukalah command prompt. Anda juga dapat menggunakan konteks “Git Bash here” dengan mengklik kanan untuk membuka Git Bash di lokasi tersebut. Setelah itu, ketiklah perintah untuk membuat proyek baru:

```
PS C:\Users\Home-PC\Videos\mobile> flutter create latihan1
Creating project latihan1...
Resolving dependencies in `latihan1`... (2.1s)
Downloading packages...
Got dependencies in `latihan1`.
Wrote 129 files.

All done!
You can find general documentation for Flutter at: https://docs.flutter.dev/
Detailed API documentation is available at: https://api.flutter.dev/
If you prefer video documentation, consider: https://www.youtube.com/c/flutterdev

In order to run your application, type:

$ cd latihan1
Your application code is in latihan1\lib\main.dart.

The configured version of Java detected may conflict with the Android Gradle Plugin (AGP) version in your ne

[RECOMMENDED] If so, to keep the default AGP version 8.1.0, make
sure to download a compatible Java version
(Java 17 <= compatible Java version < Java 21).
You may configure this compatible Java version by running:
`flutter config --jdk-dir=<JDK_DIRECTORY>`
Note that this is a global configuration for Flutter.
```

Buka folder ini di VS Code. Kamu bisa klik kanan dan menggunakan menu konteks untuk membuka langsung di VS Code, atau mulai VS Code terlebih dahulu dan kemudian buka folder ini sebagai proyek.

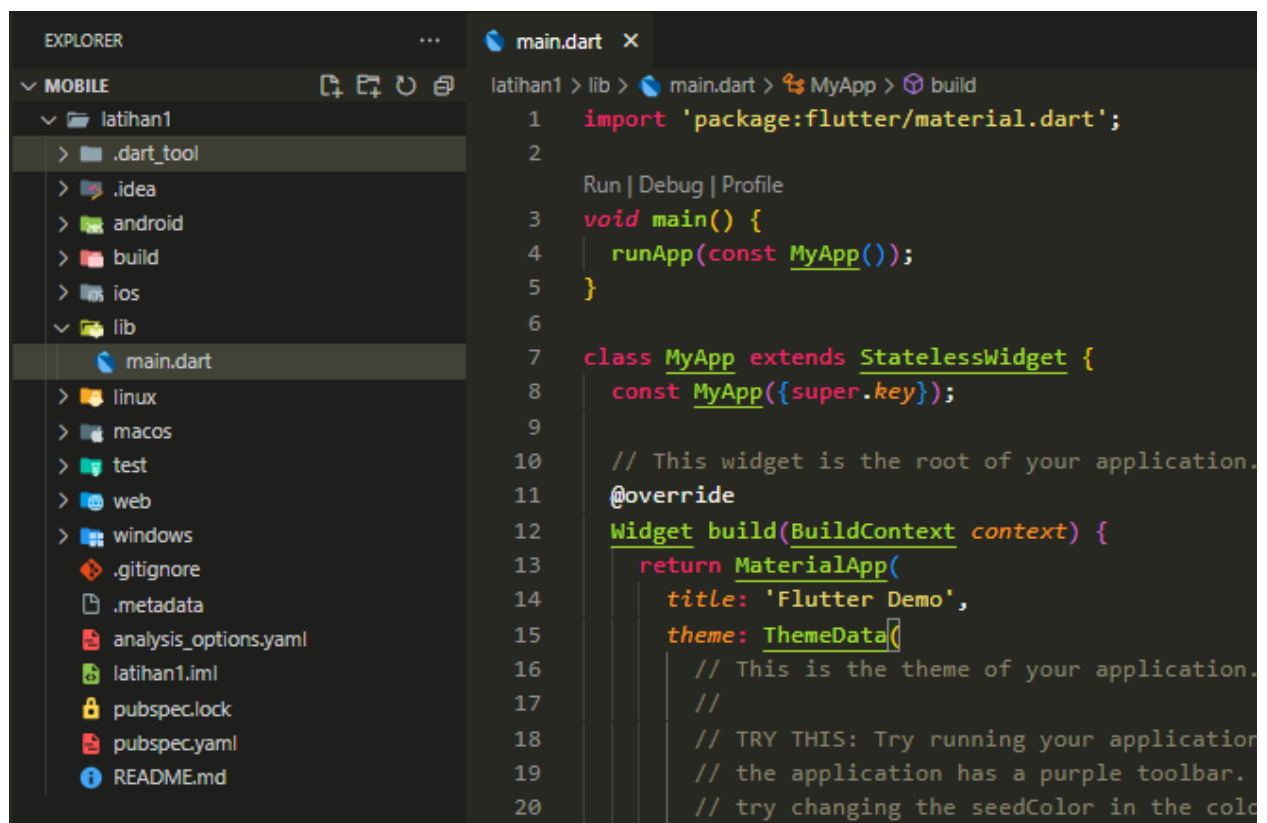
```

PS C:\Users\Home-PC\Videos\mobile> cd latihan1
PS C:\Users\Home-PC\Videos\mobile\latihan1> flutter run
Connected devices:
Windows (desktop) • windows • windows-x64 • Microsoft Windows [Version 10.0.19045.5131]
Chrome (web) • chrome • web-javascript • Google Chrome 131.0.6778.86
Edge (web) • edge • web-javascript • Microsoft Edge 131.0.2903.63
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (or "q" to quit): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome... 44,5s
This app is linked to the debug service: ws://127.0.0.1:62298/Ek3uiilegtc=/ws
Debug service listening on ws://127.0.0.1:62298/Ek3uiilegtc=/ws

compatible Java/AGP versions.

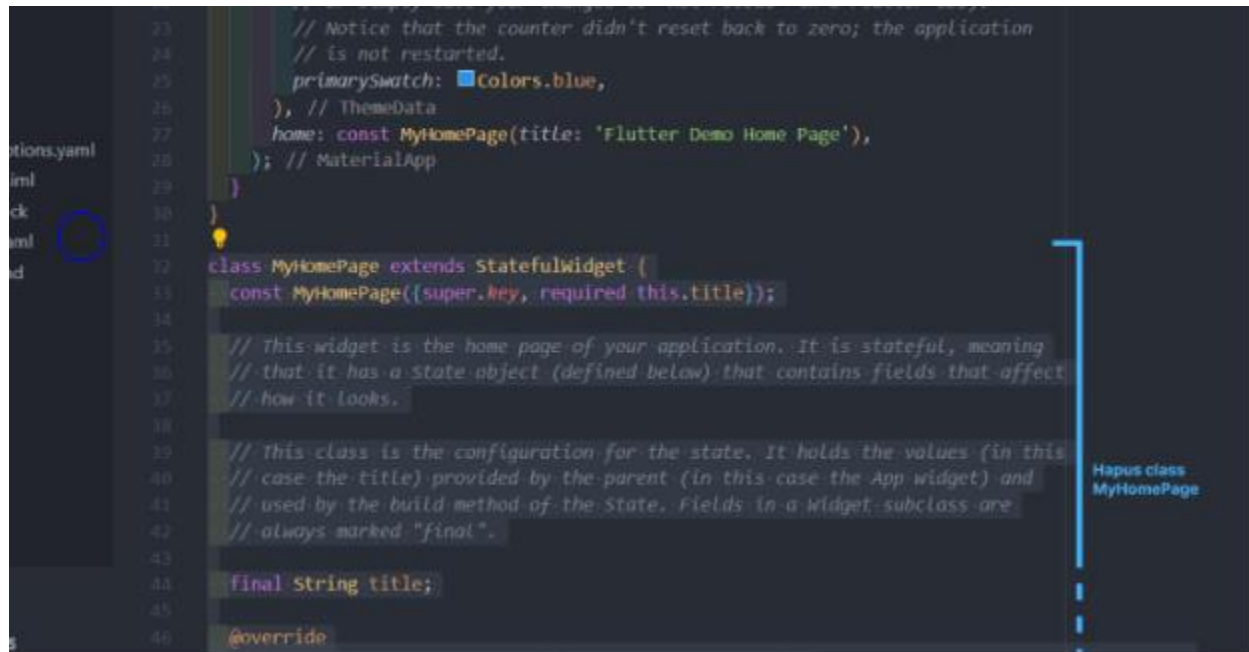
```

Ada sebuah panel besar di sebelah kiri yang menampilkan semua file dan folder, disebut Explorer Panel. Kamu bisa menuju folder 'lib' dan memilih file 'main.dart'. File ini merupakan titik awal dari mana aplikasi akan mulai dijalankan.



Menjalankan Aplikasi Halo Flutter

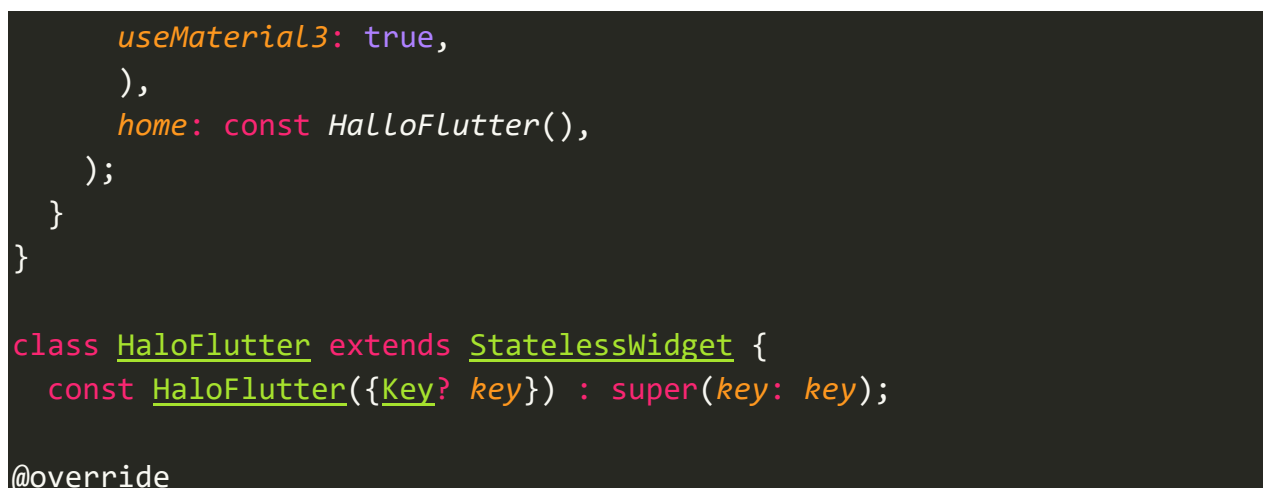
Hapuslah Widget MyHomePage.



The screenshot shows a code editor with a dark theme. On the left, a file explorer shows 'options.yaml', 'lib', 'main.dart', and 'assets'. The 'main.dart' file is open, showing lines 23 to 46. The code defines a StatefulWidget named MyHomePage. A blue bracket on the right side of the code, spanning from line 23 to line 46, indicates the selection of the entire MyHomePage class. A tooltip next to the bracket says 'Hapus class MyHomePage'. The code is as follows:

```
23 // Notice that the counter didn't reset back to zero; the application
24 // is not restarted.
25 primarySwatch: Colors.blue,
26 }, // ThemeData
27 home: const MyHomePage(title: 'Flutter Demo Home Page'),
28 ); // MaterialApp
29 }
30
31
32 class MyHomePage extends StatefulWidget {
33   const MyHomePage({super.key, required this.title});
34
35   // This widget is the home page of your application. It is stateful, meaning
36   // that it has a state object (defined below) that contains fields that affect
37   // how it looks.
38
39   // This class is the configuration for the state. It holds the values (in this
40   // case the title) provided by the parent (in this case the App widget) and
41   // used by the build method of the State. Fields in a Widget subclass are
42   // always marked "final".
43
44   final String title;
45
46   @override
```

Buatlah sebuah Stateless Widget baru dan beri nama HaloFlutter. Stateless Widget digunakan untuk mendefinisikan Widget yang tidak perlu menangani perubahan pada keadaan internalnya. Jenis Widget ini biasanya digunakan untuk membangun komponen yang setelah digambar, tidak perlu diperbarui lagi.



The screenshot shows a code editor with a dark theme. The code defines a StatelessWidget named HaloFlutter. The code is as follows:

```
useMaterial3: true,
),
home: const HalloFlutter(),
);
}
}

class HaloFlutter extends StatelessWidget {
  const HaloFlutter({Key? key}) : super(key: key);

  @override
```

```
Widget build(BuildContext context) {
  return Container(

  );
}
```

```
26      ), // ThemeData
27      home: const HaloFlutter(),
28    ); // MaterialApp
29  }
30 }
31
32 class HaloFlutter extends StatelessWidget {
33   const HaloFlutter({super.key});
34
35   @override
36   Widget build(BuildContext context) {
37     return Container();
38   }
39 }
```

Buat widget baru dengan nama HaloFlutter

Gantilah widget Container dengan widget Scaffold. Scaffold akan mengimplementasikan struktur tata letak visual dasar desain material. Widget ini menyediakan API untuk menampilkan drawer, appbar, dan konten aplikasi. Properti body pada Scaffold akan digunakan untuk menampilkan konten aplikasi.

```
class HaloFlutter extends StatelessWidget {
  const HaloFlutter({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return const Scaffold(
```

```
);  
}  
}
```

Deklarasikan Widget **Container** di badan Scaffold.

Widget Container adalah widget berguna yang menggabungkan widget painting, positioning, dan sizing. Anda dapat membungkus widget apa pun dengan Container dan mengontrol properti yang disebutkan di atas.

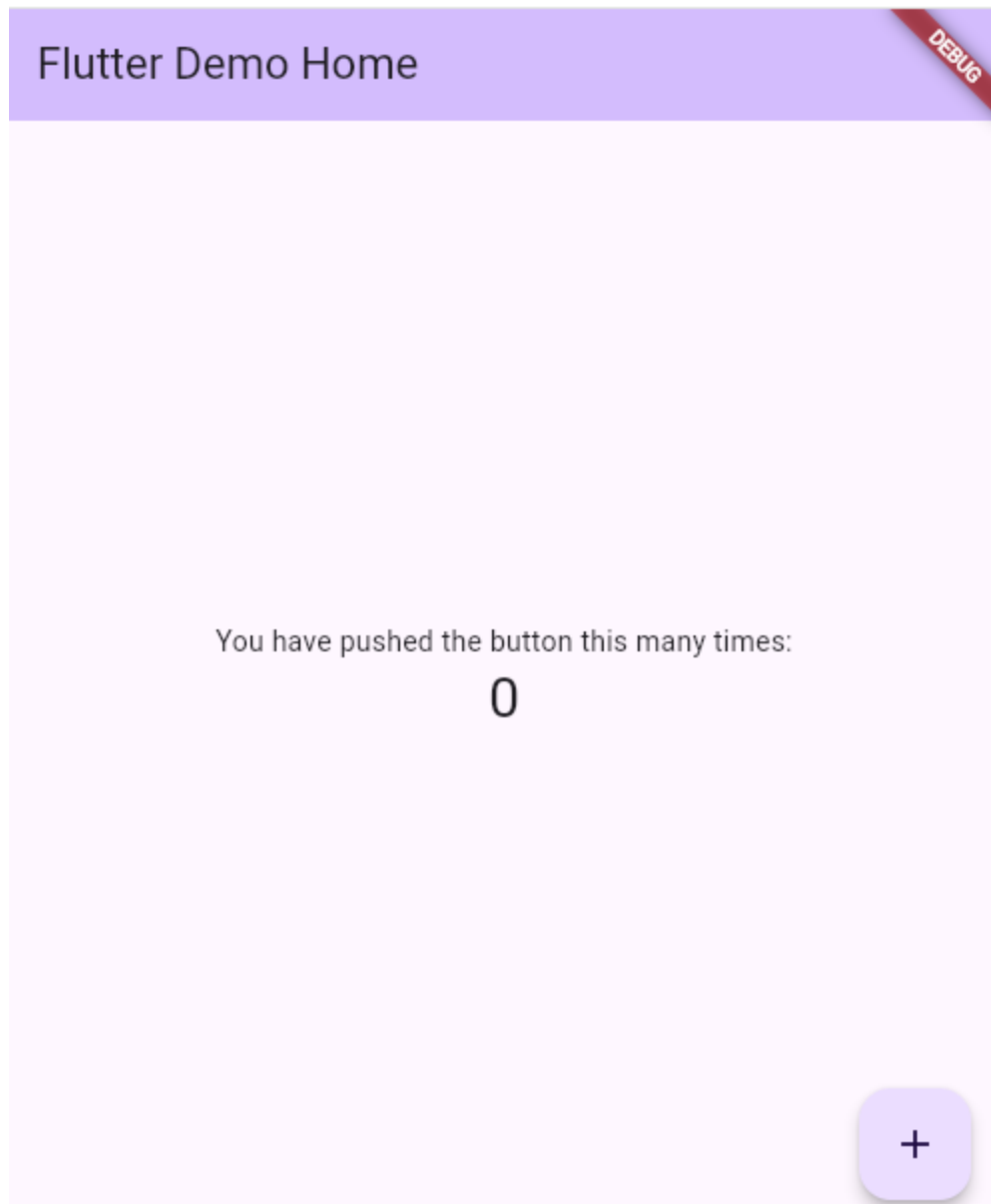
```
class HaloFlutter extends StatelessWidget {  
  const HaloFlutter({Key? key}) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    return const Scaffold(  
      body: Container()  
    );  
  }  
}
```

Widget Container memiliki properti alignment yang dapat membantu dalam memposisikan Widget ke tengah layar. Untuk mengatur alignment, gunakan class Alignment.

Pada properti children dari Widget Container, deklarasikan Widget Teks. Widget Teks digunakan untuk menampilkan dan mengelola teks. Setelah membuat Widget Teks, masukkan 'Halo Flutter' di antara tanda kutip tunggal. Apapun yang dimasukkan di antara tanda kutip tunggal akan ditampilkan oleh Widget Teks.



Hasil



Mengenal Flutter Widget :

Flutter adalah toolkit UI milik Google untuk membuat aplikasi yang terkompilasi secara native untuk platform iOS dan Android dari satu source code. Untuk membangun aplikasi dengan Flutter, kita harus memulainya dengan menggunakan widget sebagai blok bangunan utama. Widget

digunakan untuk menggambarkan tampilan apa yang harus ditampilkan sesuai dengan konfigurasi dan statusnya saat ini. Beberapa contoh widget yang tersedia di Flutter adalah widget text, widget row, widget column, widget container, dan masih banyak lagi.

Widget adalah setiap elemen pada layar aplikasi Flutter. Tampilan layar sepenuhnya bergantung pada pilihan dan urutan widget yang digunakan untuk membangun aplikasi. Struktur kode aplikasi disusun dalam bentuk pohon widget.

Kategori Widget

Ada 14 kategori utama di mana widget flutter dibagi. Mereka terutama dipisahkan berdasarkan fungsionalitas yang mereka berikan dalam aplikasi flutter.

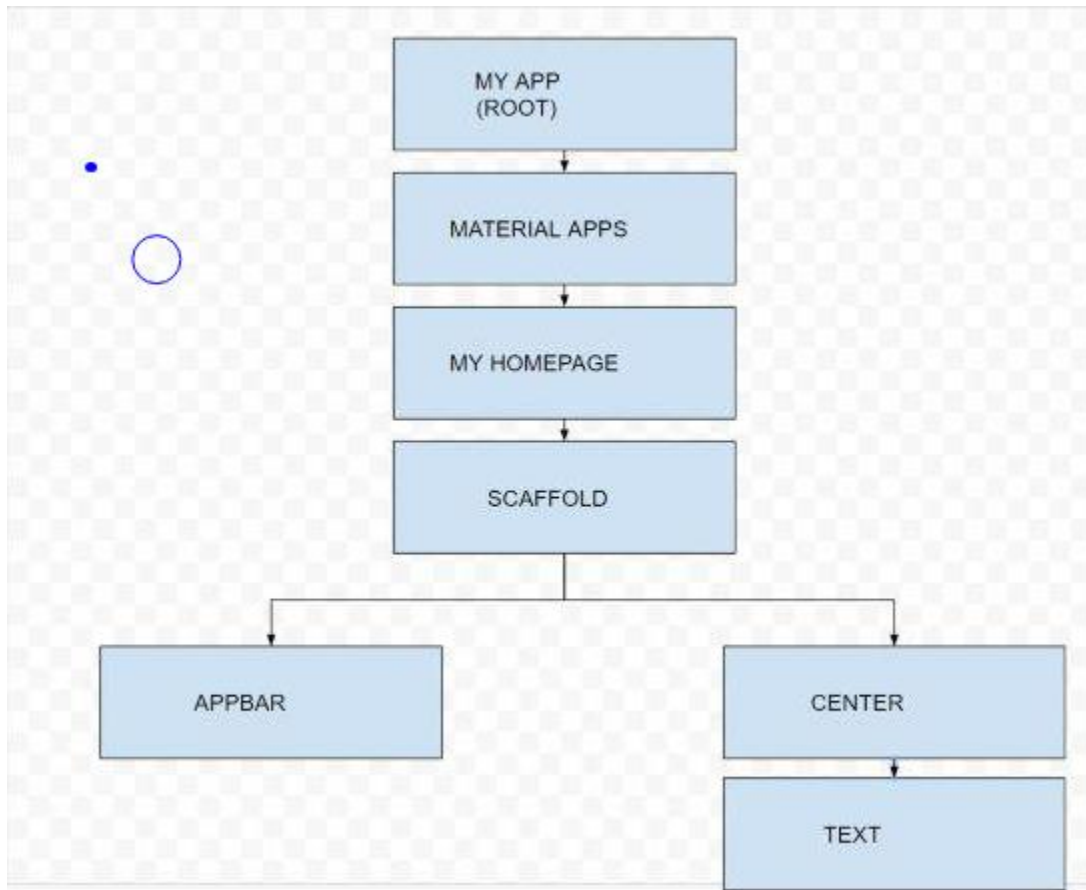
1. Accessibility: Ini adalah kumpulan widget yang membuat aplikasi flutter lebih mudah diakses.
2. Animation dan Motion: Widget ini menambahkan animasi ke widget lain.
3. Assets, Images, dan Icons: Widget ini bertanggung jawab atas aset seperti tampilan gambar dan menunjukkan ikon.
4. Async: Ini menyediakan fungsionalitas async dalam aplikasi flutter.

5. Basics: Ini adalah kumpulan widget yang benar-benar diperlukan untuk pengembangan aplikasi flutter dasar.
6. Cupertino: Ini adalah widget yang dirancang untuk iOS.
7. Input: Kumpulan widget ini menyediakan fungsionalitas input dalam aplikasi flutter.
8. Interaction Models: Widget ini digunakan untuk mengelola interaksi pengguna dan mengarahkan pengguna ke tampilan yang berbeda dalam aplikasi.
9. Layout: Kumpulan widget ini membantu dalam menempatkan widget lain di layar sesuai kebutuhan.
10. Material Components: Ini adalah kumpulan widget yang terutama mengikuti desain material oleh Google.
11. Painting dan effects: Ini adalah kumpulan widget yang mengaplikasikan perubahan visual pada widget anak mereka tanpa mengubah tata letak atau bentuk mereka.
12. Scrolling: Ini menyediakan kemampuan scroll ke sekelompok widget lain yang tidak dapat di-scroll secara default.
13. Styling: Ini berkaitan dengan tema, responsivitas, dan ukuran aplikasi.
14. Text: Ini menampilkan teks.

Tipe Widget

Secara umum ada dua jenis widget di flutter:

1. Stateless Widget
2. Stateful Widget



Contoh: Layout Tree dari layar aplikasi dasar menggunakan Stateless Widgets:

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

// fungsi untuk mentrigger proses build aplikasi flutter
void main() => runApp(const NextgenTutorial());

class NextgenTutorial extends StatelessWidget {
  const NextgenTutorial({Key? key}) : super(key: key);
```

```

@override
Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
      backgroundColor: Colors.lightBlue,
      appBar: AppBar(
        backgroundColor: Colors.blue,
        title: const Text("Nextgen Tutorial"),
      ), // AppBar
      body: Container(
        child: const Center(
          child: Text("Halo guys!!"),
        ), // Center
      ), // Container
    ), // Scaffold
  ); // MaterialApp
}
}

```

Contoh: Layout Tree dari layar aplikasi dasar yang menggunakan Stateful Widgets. Ini juga menghasilkan hasil yang sama seperti kode di atas.

```

import 'package:flutter/material.dart';

// fungsi untuk mentrigger proses build aplikasi flutter
void main() => runApp(const NextgenTutorial());

class NextgenTutorial extends StatelessWidget {
  const NextgenTutorial({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        backgroundColor: Colors.lightBlue,
        appBar: AppBar(

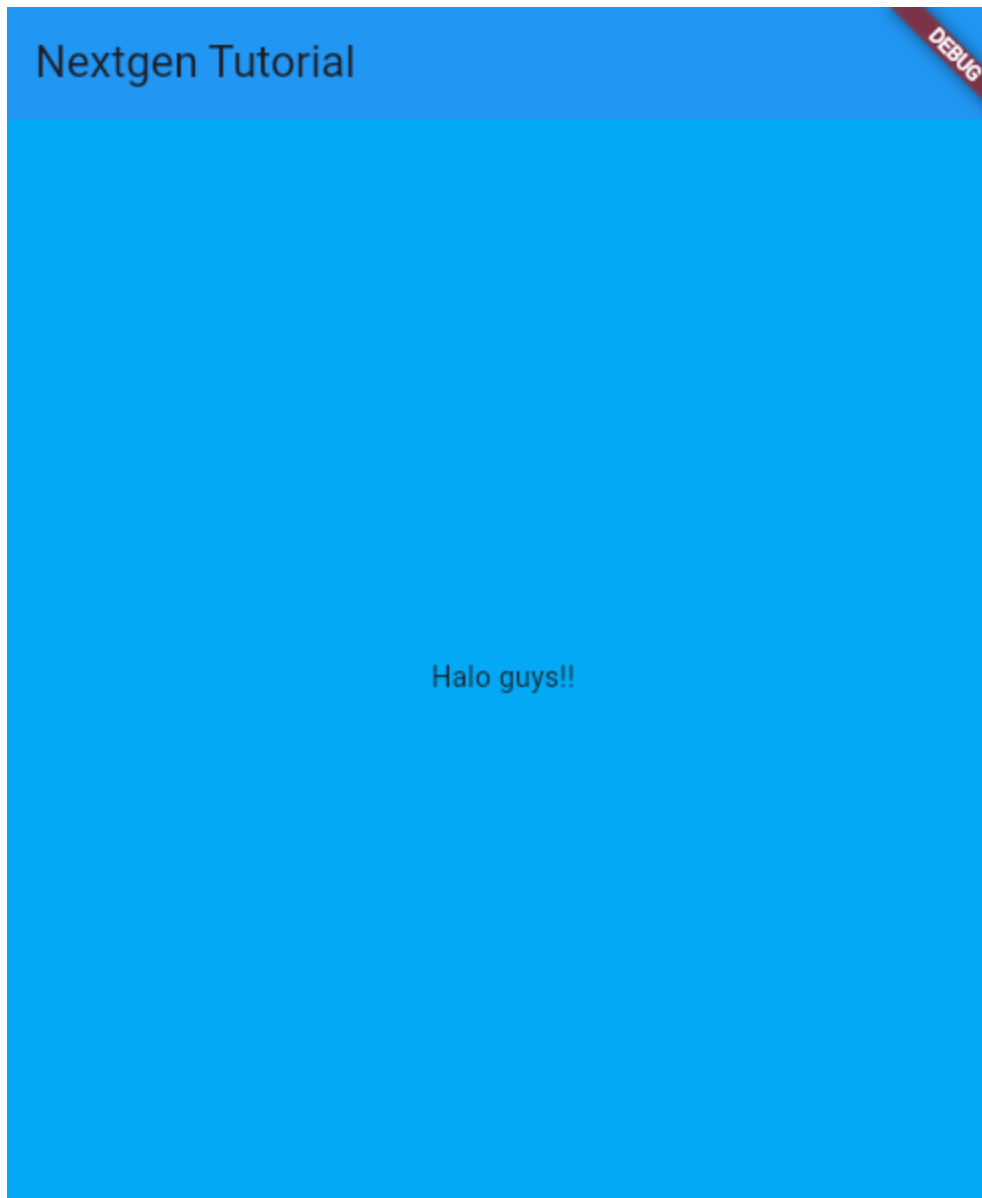
```

```
        backgroundColor: Colors.blue,  
        title: const Text("Nextgen Tutorial"),  
      ), // AppBar  
      body: Container(  
        child: const Center(  
          child: Text("Halo guys!!"),  
        ), // Center  
      ), // Container  
    ), // Scaffold  
  ); // MaterialApp  
}
```

Deskripsi widget yang digunakan adalah sebagai berikut:

- Scaffold – Menerapkan struktur tata letak visual desain material dasar.
- App-Bar – Untuk membuat bilah di bagian atas layar.
- Teks Untuk menulis text pada layar.
- Container – Wadah untuk memuat widget lainnya.
- Center – Untuk memberikan perataan tengah ke widget lain.

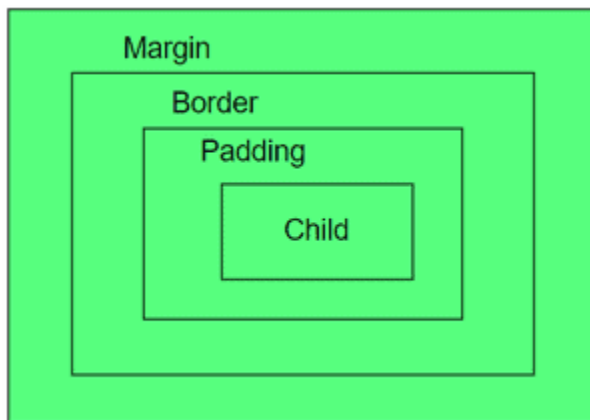
Hasil



FLUTTER CONTAINER

Class **Container** di Flutter adalah widget yang mudah digunakan karena menggabungkan fungsi penempatan dan penyesuaian ukuran umum dari widget. Dengan menggunakan class Container, kita dapat menempatkan satu

atau lebih widget di layar sesuai dengan kebutuhan. Secara sederhana, Container mirip seperti sebuah kotak untuk menyimpan konten. Setiap elemen Container dasar memiliki **margin**, yang memisahkan elemen tersebut dari konten lainnya. Selain itu, kita dapat memberikan **border** pada Container dengan bentuk yang berbeda seperti persegi panjang, melengkung, dan lainnya. Container juga menyediakan **padding** untuk setiap child-nya dan menerapkan batasan tambahan pada lebar dan tinggi setelah padding diterapkan (jika salah satu di antaranya tidak null).



Constructor dari Class Container

Constructor adalah metode khusus yang digunakan untuk membuat object dari suatu class. Dalam Flutter, class container memiliki beberapa jenis constructor yang dapat digunakan untuk membuat object dengan mudah.

Sintaks :

```
Container({Key key,  
          AlignmentGeometry alignment,  
          EdgeInsetsGeometry padding,  
          Color color,  
          Decoration decoration,
```



```
Decoration foregroundDecoration,  
double width,  
double height,  
BoxConstraints constraints,  
EdgeInsetsGeometry margin,  
Matrix4 transform,  
Widget child,  
Clip clipBehavior: Clip.none});
```

Properti dari Class Container

Dalam Flutter, class Container digunakan untuk menampung widget lain dan memberikan kontrol atas tata letak mereka di dalamnya. Properti pada class Container sangat penting karena memungkinkan developer untuk mengatur tampilan widget dengan lebih presisi.

1. child

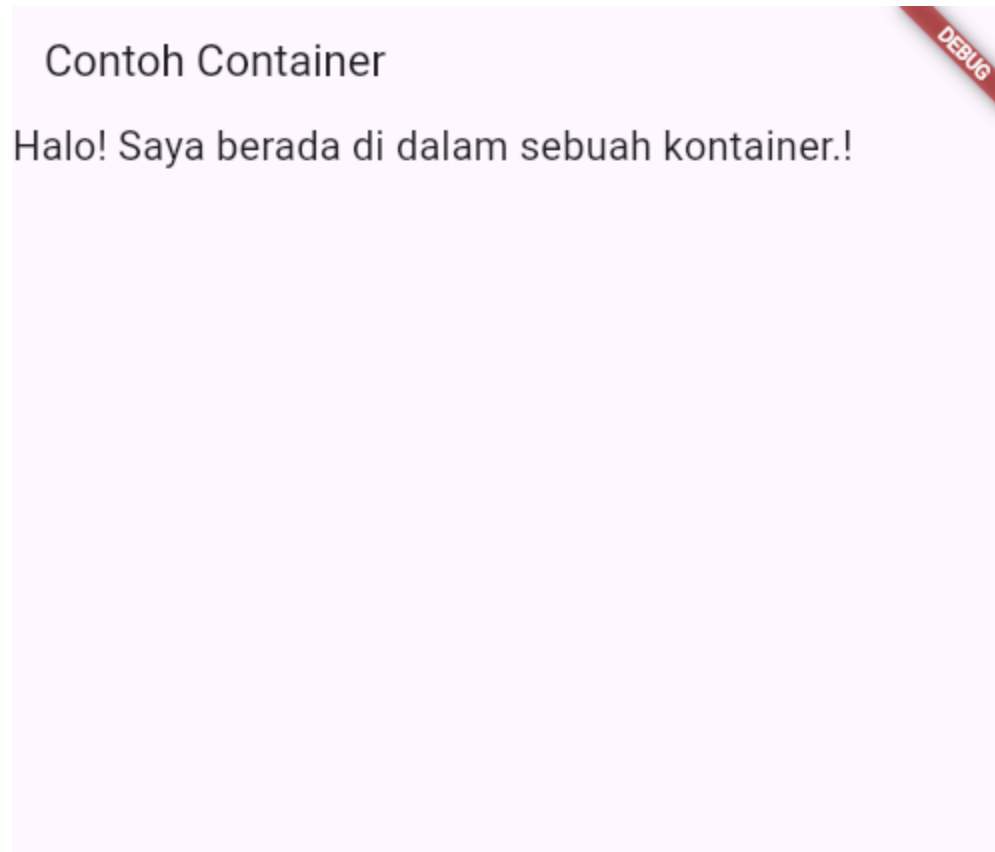
Class container memiliki properti 'child:' yang berfungsi untuk menyimpan anak-anaknya. Class child bisa berupa widget apa saja. Sebagai contohnya, kita dapat menggunakan widget teks sebagai salah satu child.

```
import 'package:flutter/material.dart';  
  
void main() => runApp(const MyApp());  
  
class MyApp extends StatelessWidget {  
  const MyApp({Key? key}) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      home: Scaffold(  

```

```
    appBar: AppBar(
      title: const Text("Contoh Container"),
    ),
    body: Container(
      child: const Text("Halo! Saya berada di dalam sebuah
kontainer.!",
        style: TextStyle(fontSize: 20)),
    ),
  );
}
```

Hasil



2. color

Properti color digunakan untuk menentukan warna latar belakang pada semua container. Dengan memanfaatkan warna latar belakang ini, kita dapat memvisualisasikan posisi container.

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

void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          backgroundColor: Colors.blue,
          title: const Text("Contoh Container"),
        ),
        body: Container(
          color: Colors.orange,
          child: const Text("Halo! Saya berada di dalam sebuah
kontainer!",
            style: TextStyle(fontSize: 20)),
        ),
      ),
    );
  }
}
```

Hasil :

Contoh Container

DEBUG

Halo! Saya berada di dalam sebuah kontainer!

3. height dan width

Secara default, class container mengambil ruang yang diperlukan oleh elemen anaknya. Namun, kita juga dapat menentukan tinggi dan lebar dari container sesuai dengan kebutuhan yang ada.

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

void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

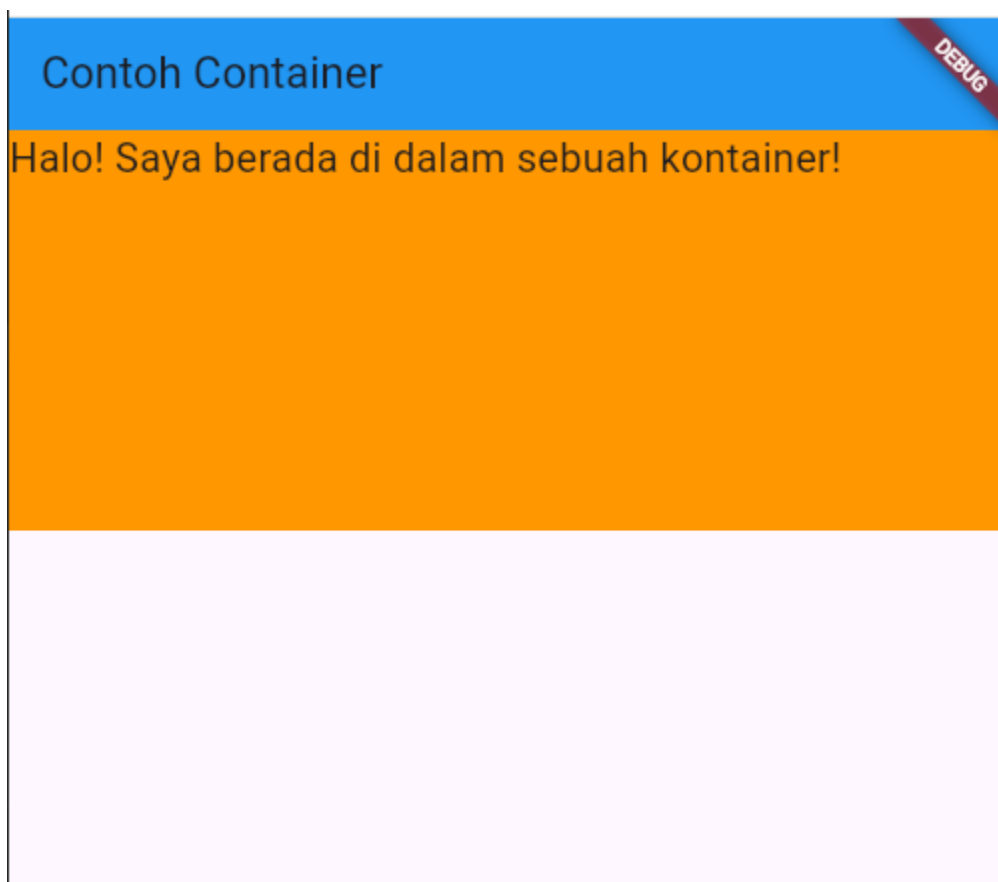
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          backgroundColor: Colors.blue,
```

```

        title: const Text("Contoh Container"),
    ),
    body: Container(
        height: 200,
        width: double.infinity,
        color: Colors.orange,
        child: const Text("Halo! Saya berada di dalam sebuah
kontainer!",
            style: TextStyle(fontSize: 20)),
    ),
);
}
}

```

Hasil :



4. margin

Margin digunakan untuk memberikan jarak atau spasi di sekitar suatu container. Anda dapat melihat ada ruang kosong di sekeliling container tersebut. Untuk menentukan margin, Anda bisa menggunakan `EdgeInsets.geometry`. Dengan menggunakan metode `.all()`, margin akan diterapkan di keempat sisi (atas, bawah, kiri, dan kanan) dengan ukuran yang sama.

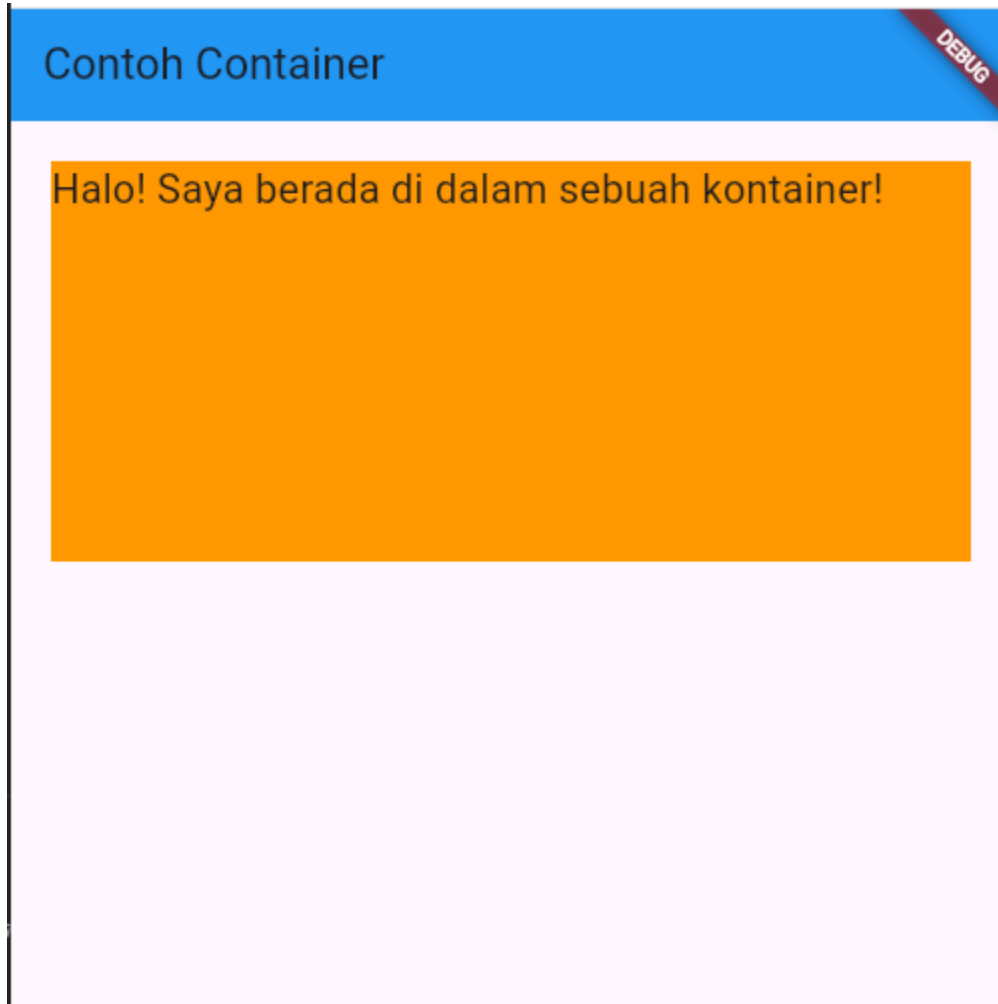
```
import 'package:flutter/material.dart';

void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          backgroundColor: Colors.blue,
          title: const Text("Contoh Container"),
        ),
        body: Container(
          height: 200,
          width: double.infinity,
          color: Colors.orange,
          margin: const EdgeInsets.all(20),
          child: const Text("Halo! Saya berada di dalam sebuah
kontainer!",
            style: TextStyle(fontSize: 20)),
        ),
      ),
    );
  }
}
```

Hasil:



5. padding

Padding digunakan untuk memberikan ruang antara batas kotak dan elemen-elemennya. Perhatikan contoh di bawah ini dengan menambahkan properti padding akan terdapat jarak yang cukup antara batas dan teks.

```
import 'package:flutter/material.dart';  
  
void main() => runApp(const MyApp());
```

```

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          backgroundColor: Colors.blue,
          title: const Text("Contoh Container"),
        ),
        body: Container(
          height: 200,
          width: double.infinity,
          color: Colors.orange,
          margin: const EdgeInsets.all(20),
          padding: const EdgeInsets.all(30),
          child: const Text("Halo! Saya berada di dalam sebuah
kontainer!",
                        style: TextStyle(fontSize: 20)),
        ),
      ),
    );
  }
}

```

Hasil :

Contoh Container

DEBUG

Halo! Saya berada di dalam sebuah kontainer!

6. alignment

Alignment digunakan untuk memposisikan elemen anak di dalam wadah. Kita dapat memposisikan dengan berbagai cara: bottom, bottom center, left, right, dan sebagainya. Di sini, elemen child akan diselaraskan dengan posisi tengah bawah.

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

void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
```

```

return MaterialApp(
  home: Scaffold(
    appBar: AppBar(
      title: const Text("Contoh Container"),
    ),
    body: Container(
      height: 200,
      width: double.infinity,
      color: Colors.orange,
      margin: const EdgeInsets.all(20),
      alignment: Alignment.bottomCenter,
      padding: const EdgeInsets.all(30),
      child: const Text("Halo! Saya berada di dalam sebuah
kontainer!",
        style: TextStyle(fontSize: 20)),
    ),
  ),
);
}

```

Hasil :

Contoh Container

DEBUG

Halo! Saya berada di dalam sebuah kontainer!

7. decoration

Properti decoration digunakan untuk menghias kotak, seperti memberikan bingkai pada kotak tersebut. Properti ini akan melukis di belakang elemen yang ada di dalam kotak. Sedangkan dekorasi depan akan melukis di bagian depan elemen tersebut. Mari kita tambahkan border pada kotak. Namun, kita tidak dapat memberikan warna dan border dengan warna secara bersamaan.

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

void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
```

```

Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.blues,
        title: const Text("Contoh Container"),
      ),
      body: Container(
        height: 200,
        width: double.infinity,
        margin: const EdgeInsets.all(20),
        alignment: Alignment.center,
        padding: const EdgeInsets.all(30),
        decoration: BoxDecoration(
          border: Border.all(color: Colors.black, width: 3),
        ),
        child: const Text("Halo! Saya berada di dalam sebuah
kontainer!",
          style: TextStyle(fontSize: 20)),
      ),
    ),
  );
}

```

Hasil :

Contoh Container

DEBUG

Halo! Saya berada di dalam sebuah kontainer!

8. transform

Properti dari container ini membantu kita untuk memutar container tersebut. Kita dapat memutar container pada sumbu apapun, di sini kita sedang memutar pada sumbu z.

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

void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
```

```

Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.blue,
        title: const Text("Contoh Container"),
      ),
      body: Container(
        height: 200,
        width: double.infinity,
        margin: const EdgeInsets.all(20),
        alignment: Alignment.center,
        padding: const EdgeInsets.all(30),
        transform: Matrix4.rotationZ(0.1),
        decoration: BoxDecoration(
          border: Border.all(color: Colors.black, width: 3),
          color: Colors.orange,
        ),
        child: const Text("Halo! Saya berada di dalam sebuah
kontainer!",
          style: TextStyle(fontSize: 20)),
      ),
    ),
  );
}

```

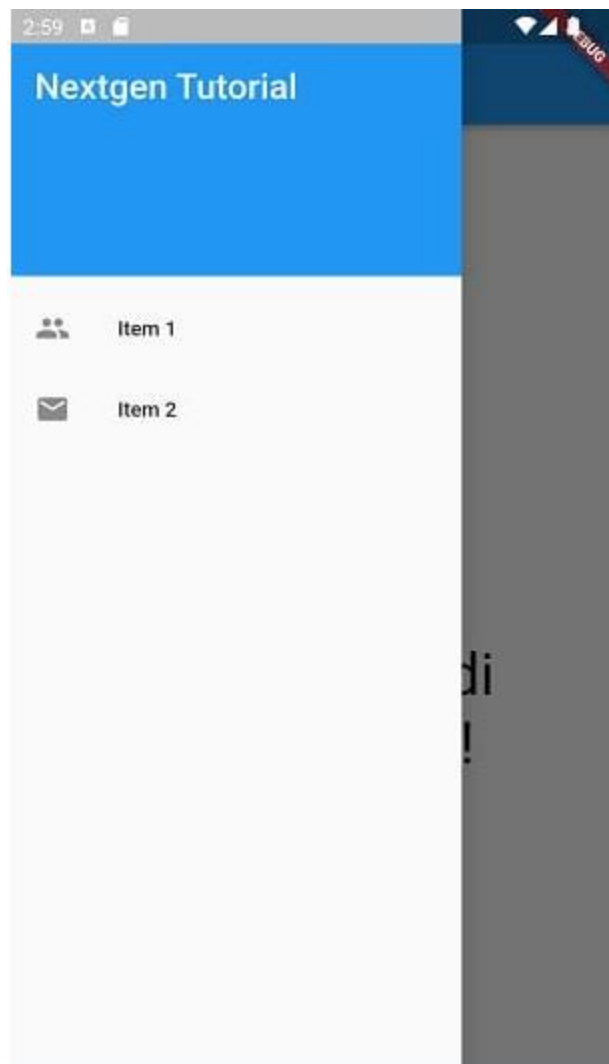
Hasil :

Contoh Container

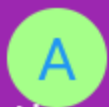
DEBUG

Halo! Saya berada di dalam sebuah kontainer!

Flutter Scaffold



Flutter Drawer



Ahmadi,M.KOM

ahmadi@uniska-bjm.ac.id



My Profile



My Course



Go Premium



Saved Videos



Edit Profile



LogOut

DEBUG

yang tidak terlihat.

```

1 import 'package:flutter/material.dart';
2
3 // Fungsi ini memulai proses build dari aplikasi flutter.
4 void main() => runApp(const MyApp());
5
6 class MyApp extends StatelessWidget {
7   final appTitle = 'Flutter Drawer Demo';
8
9   const MyApp({Key? key}) : super(key: key);
10
11   @override
12   Widget build(BuildContext context) {
13     return MaterialApp(
14       title: appTitle,
15       home: MyHomePage(title: appTitle),
16     ); // MaterialApp
17   }
18 }
19
20 class MyHomePage extends StatelessWidget {
21   final String title;
22
23   const MyHomePage({Key? key, required this.title}) : super(key: key);
24
25   @override
26   Widget build(BuildContext context) {
27     return Scaffold(
28       appBar: AppBar(
29         title: Text(title),
30         backgroundColor: Colors.purple,
31       ),
32       body: const Center(
33         child: Text(
34           'Drawer adalah layar samping yang tidak terlihat.',
35           textAlign: TextAlign.center,
36           style: TextStyle(fontSize: 20.0),
37         ),
38       ),
39       drawer: Drawer(
40         child: ListView(
41           padding: const EdgeInsets.all(0),
42           children: [
43             const DrawerHeader(
44               decoration: BoxDecoration(
45                 color: Colors.purple,
46               ), //BoxDecoration
47             UserAccountsDrawerHeader(
48               decoration: BoxDecoration(color: Colors.purple),
49               accountName: Text(
50                 "Ahmadi,M.KOM",
51                 style: TextStyle(fontSize: 18),
52               ),
53               accountEmail: Text("ahmadi@uniska-bjm.ac.id"),
54               currentAccountPictureSize: Size.square(50),
55               currentAccountPicture: CircleAvatar(
56                 backgroundColor: Color.fromARGB(255, 165, 255, 137),
57                 child: Text(
58                   "A",
59                   style: TextStyle(fontSize: 30.0, color: Colors.blue),
60                 ), //Text
61               ), //CircleAvatar
62             ), //UserAccountDrawerHeader
63             ListTile(
64               leading: const Icon(Icons.person),
65               title: const Text(' My Profile '),
66               onTap: () {
67                 Navigator.pop(context);
68               },
69             ),
70             ListTile(
71               leading: const Icon(Icons.book),
72               title: const Text(' My course '),
73               onTap: () {
74                 Navigator.pop(context);
75               },
76             ),
77             ListTile(
78               leading: const Icon(Icons.workspace_premium),
79               title: const Text(' Go Premium '),
80               onTap: () {
81                 Navigator.pop(context);
82               },
83             ),
84             ListTile(
85               leading: const Icon(Icons.video_label),
86               title: const Text(' Saved Videos '),
87               onTap: () {
88                 Navigator.pop(context);
89               },
90             ),
91             ListTile(
92               leading: const Icon(Icons.edit),
93               title: const Text(' Edit Profile '),
94               onTap: () {
95                 Navigator.pop(context);
96               },
97             ),
98             ListTile(
99               leading: const Icon(Icons.logout),
100              title: const Text('LogOut'),
101              onTap: () {
102                Navigator.pop(context);
103              },
104            ),
105          ],
106        ), //Drawer
107      );
108    }
109  }
110 }

```