

VERSION 1.0

JULI, 2023



PEMROGRAMAN MOBILE

INTRODUCE DART AND FLUTTER – MODUL 1 MATERI

TIM PENYUSUN:

- DIDIH RIZKI CHANDRANEGERA, S.KOM., M.KOM.
- MUHAMMAD ZULFIQOR LILHAQ
- RIYAN PUTRA FIRJATULLAH

PRESENTED BY: LAB. INFORMATIKA UNIVERSITAS MUHAMMADIYAH MALANG

CAPAIAN PEMBELAJARAN PRAKTIKUM

1. Mahasiswa dapat memahami konsep dasar Framework Flutter.
 2. Mahasiswa dapat memahami Widget pada Framework Flutter.
 3. Mahasiswa dapat menjalankan project pertama menggunakan Framework Flutter.
 4. Mahasiswa dapat membuat User Interface sederhana menggunakan Framework Flutter.
-

SUB CAPAIAN PEMBELAJARAN PRAKTIKUM

1. Mahasiswa dapat mengimplementasikan Framework Flutter untuk membangun User Interface berdasarkan referensi yang sudah ada
-

KEBUTUHAN HARDWARE & SOFTWARE

1. PC/Laptop
 2. IDE Android Studio/ Visual Studio Code
 3. Flutter SDK: <https://docs.flutter.dev/release/archive?tab=windows>
-

MATERI POKOK

Dart

Adalah bahasa pemrograman yang dioptimalkan untuk pengembangan aplikasi yang cepat di berbagai platform. Tujuannya adalah untuk menyediakan bahasa pemrograman yang paling produktif untuk pengembangan multi-platform, yang dikombinasikan dengan platform runtime eksekusi yang fleksibel untuk kerangka kerja aplikasi [dart](#). Berikut adalah beberapa poin penting tentang Dart:

1. Sintaks yang Ekspresif:

Sintaks Dart mirip dengan bahasa pemrograman lain seperti Java atau JavaScript, membuatnya mudah dipelajari oleh pengembang yang sudah memiliki pengalaman dengan bahasa-bahasa tersebut. Dart menggunakan gaya pemrograman berorientasi objek dan mendukung konsep seperti kelas, pewarisan, polimorfisme, dan enkapsulasi.

2. Bahasa Typing Statis Opsional:

Dart mendukung tipe data statis yang dapat dideklarasikan secara eksplisit, tetapi juga memungkinkan tipe data dinamis yang secara otomatis disimpulkan oleh kompiler. Fitur ini memungkinkan pengembang untuk memilih antara pendekatan typing statis atau dinamis, tergantung pada kebutuhan proyek.

3. Dart Virtual Machine (VM):

Dart awalnya dirancang untuk dijalankan pada Dart VM, mesin virtual yang dioptimalkan untuk menjalankan kode Dart. Dart VM digunakan dalam lingkungan server-side dan desktop. Namun, dengan diperkenalkannya Flutter, Dart juga dapat dikompilasi menjadi kode mesin (native) untuk berjalan pada platform Android, iOS, web, dan desktop.

4. Asinkron dan Await:

Dart memiliki dukungan asli untuk pemrograman asinkron. Ini memungkinkan pengembang untuk melakukan operasi asinkron seperti pemanggilan jaringan atau akses ke sistem file tanpa memblokir eksekusi program. Dart menggunakan kata kunci "async" dan "await" untuk menangani pemrograman asinkron dengan mudah dan menghindari callback hell.

Dart digunakan sebagai bahasa utama untuk mengembangkan aplikasi Flutter, kerangka kerja populer untuk membangun antarmuka pengguna lintas platform. Namun, Dart juga dapat digunakan secara mandiri untuk mengembangkan aplikasi web, server, atau desktop. Dengan kelebihan-kelebihan yang ditawarkannya, Dart menjadi bahasa yang populer bagi pengembang yang mencari alternatif yang efisien dan produktif dalam pengembangan aplikasi modern.

Flutter

Adalah kerangka kerja (framework) UI mobile gratis dan open-source yang dibuat oleh Google dan dirilis pada bulan Mei 2017. Flutter memungkinkan Anda membuat multi-platform aplikasi dengan hanya menggunakan satu kode sumber. Ini berarti Anda dapat menggunakan satu bahasa pemrograman dan satu kode sumber untuk membuat berbagai aplikasi berbeda [freecodecamp-flutter](#). Berikut adalah beberapa poin penting tentang Flutter:

1. Bahasa Pemrograman:

Flutter menggunakan bahasa pemrograman Dart, yang juga dikembangkan oleh Google. Dart adalah bahasa yang efisien, dengan sintaks yang mirip dengan bahasa pemrograman lain seperti Java atau JavaScript. Flutter dan Dart saling mendukung dan menyediakan alat pengembangan yang kaya untuk mempercepat proses pengembangan aplikasi.

2. Single Codebase:

Salah satu keuntungan utama Flutter adalah kemampuannya untuk membuat aplikasi dengan kode basis tunggal. Artinya, Anda dapat mengembangkan aplikasi untuk platform Android dan iOS tanpa perlu menulis ulang kode dari awal. Hal ini menghemat waktu dan upaya pengembangan.

3. Antarmuka Pengguna Responsif:

Flutter menggunakan widget sebagai elemen dasar dalam membangun antarmuka pengguna. Widget merupakan komponen yang dapat dikombinasikan dan disusun secara hierarki untuk membentuk tampilan aplikasi. Flutter memiliki banyak widget bawaan dan juga mendukung pembuatan widget kustom. Dengan widget, pengembang dapat membuat antarmuka yang responsif dan menarik dengan

mudah.

4. Kinerja Tinggi:

Salah satu keunggulan Flutter adalah kinerja yang tinggi. Kode Flutter dikompilasi menjadi kode mesin, bukan diinterpretasikan, yang menghasilkan performa yang cepat dan responsif. Flutter juga menggunakan teknologi bernama "Skia" untuk menggambar antarmuka pengguna, yang memberikan kecepatan dan efisiensi visual yang tinggi.

5. Hot Reload:

Fitur "Hot Reload" di Flutter memungkinkan pengembang untuk melihat perubahan yang dilakukan pada kode secara langsung, tanpa harus mengulangi proses kompilasi atau me-restart aplikasi. Hal ini mempercepat siklus pengembangan dan memungkinkan pengembang untuk eksperimen dengan cepat.

6. Komunitas yang Aktif:

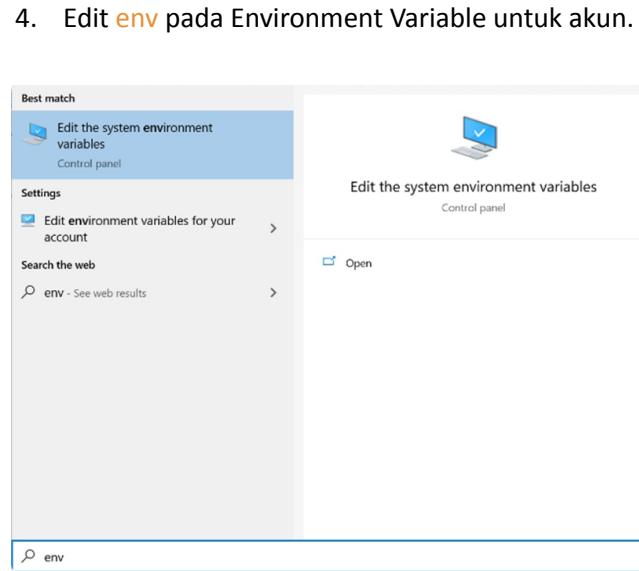
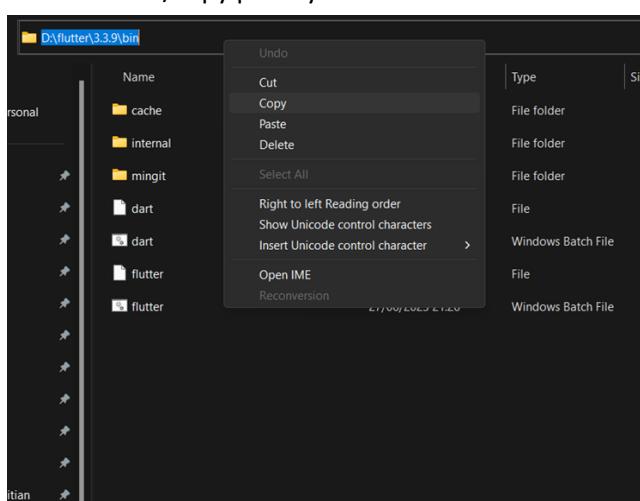
Flutter memiliki komunitas pengembang yang besar dan aktif, yang terus berkontribusi dengan paket dan library tambahan untuk memperluas fungsionalitas Flutter. Komunitas ini juga memberikan dukungan dan sumber daya yang berharga bagi pengembang baru.

Installation

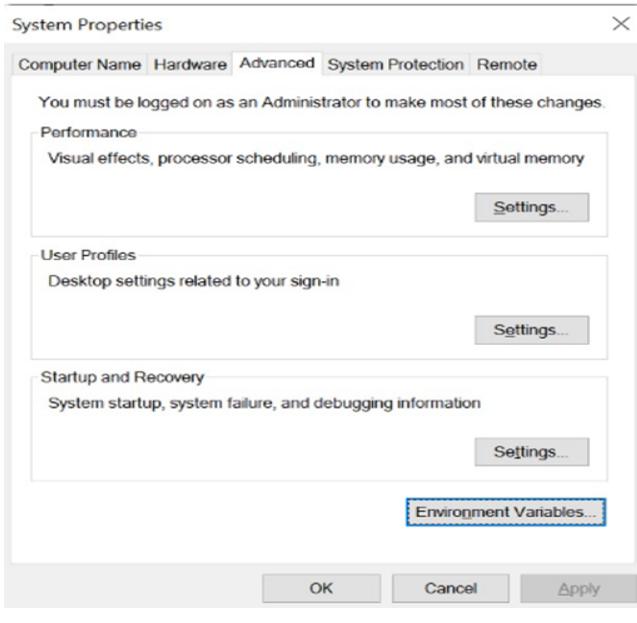
Terdapat 4 sistem operasi yang mendukung penggunaan framework flutter yaitu **windows**, **macos**, **linux**, dan **chrome os**. Setiap sistem operasi memiliki cara instalasi flutter yang berbeda-beda [install flutter](#).

Windows

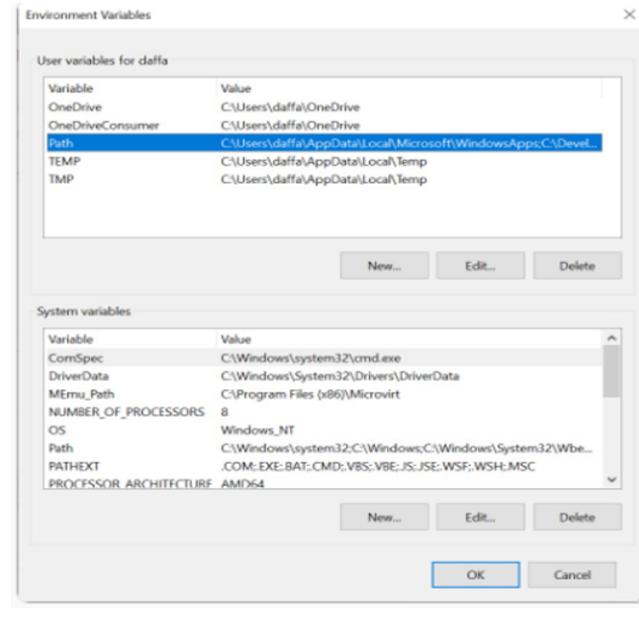
1. Unduh Flutter SDK versi 3.10.0 [instalasi flutter](#).
2. Ekstrak berkas zip pada lokasi contoh **C:\Development**, jangan extract pada **C:\Program Files** atau directory lain yang membutuhkan hak administrator.
3. Pergi ke folder **bin** Flutter yang telah diekstrak lalu salin/copy pathnya.
4. Edit **env** pada Environment Variable untuk akun.



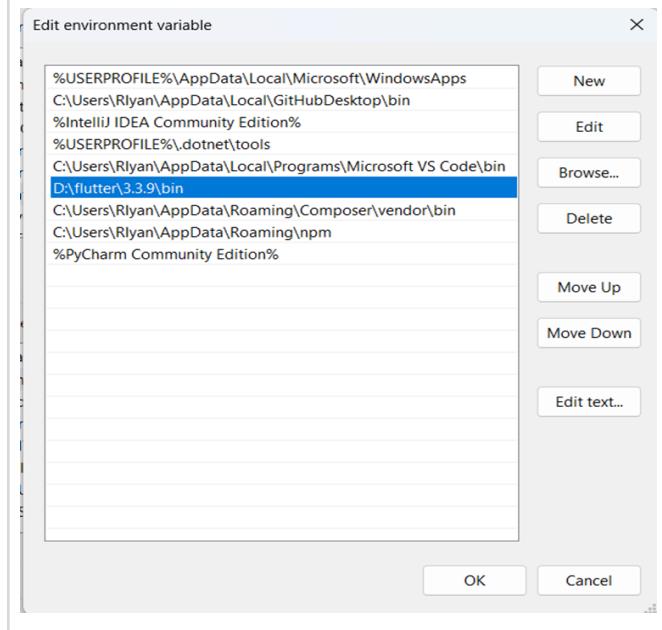
5. Pilih Environment Variables



6. Pilih **Path** lalu edit, jika tidak ada maka buat baru dengan nama variabel **Path**.



7. Klik new dan Paste link **Path** bin Flutter tadi. Lalu klik Ok.



8. Install [android studio](#).

Linux

1. Download snap store [snap store](#).
2. Install flutter [snap-flutter](#).

```
sudo snap install flutter --classic
```

3. Check instalasi

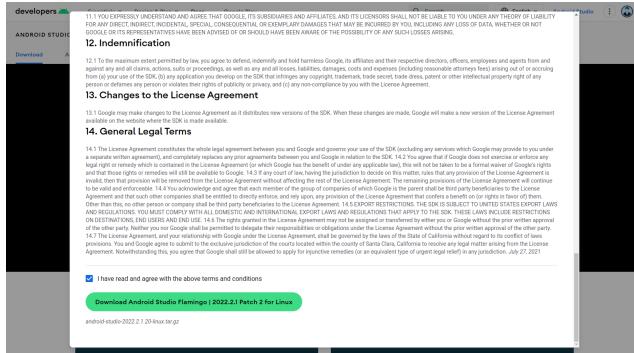
```
flutter doctor
```

4. Install [android studio](#)

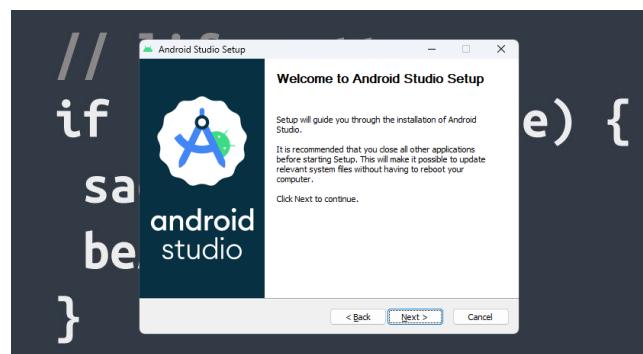
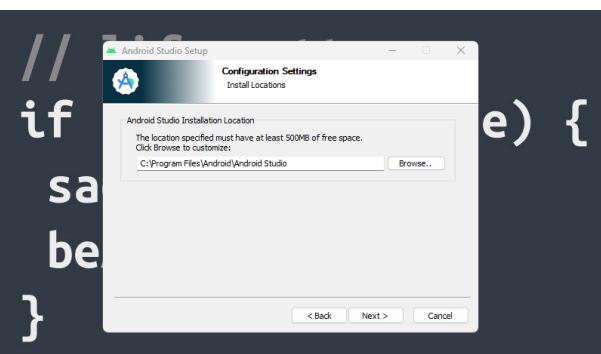
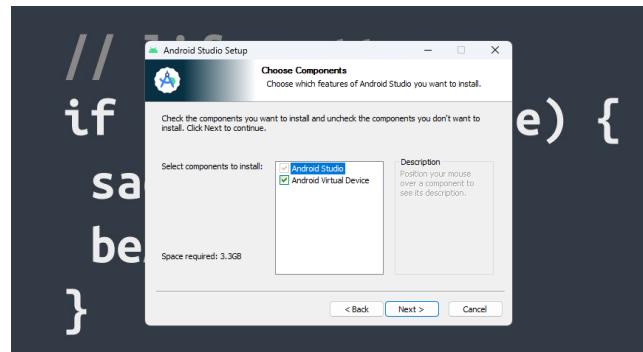
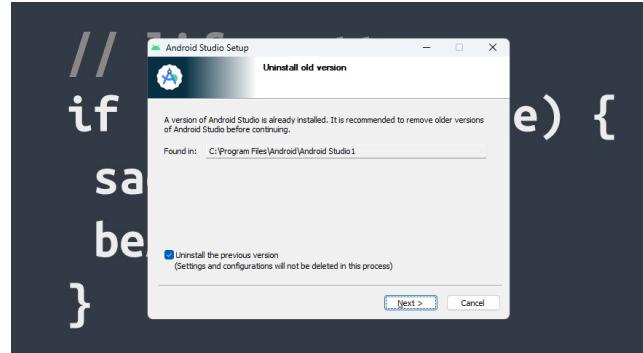
Android Studio

Digunakan flutter untuk mendevelop aplikasi berbasis android. Terdapat beberapa tahapan yang perlu dilakukan mulai dari instalasi sampai dengan setup emulator. Berikut tahapannya dibaca dari **kiri ke kanan**

1. Download android studio [developer-android-studio](#)



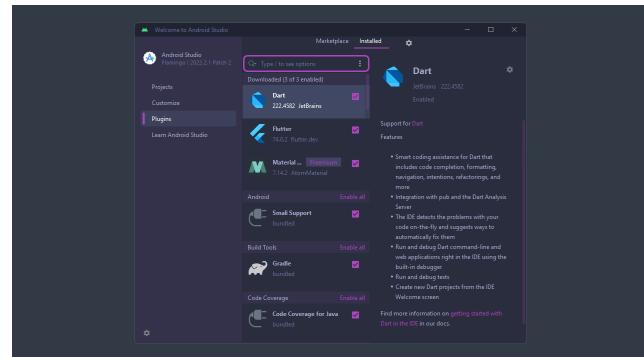
2. Lakukan Instalasi kurang lebih sebagai berikut



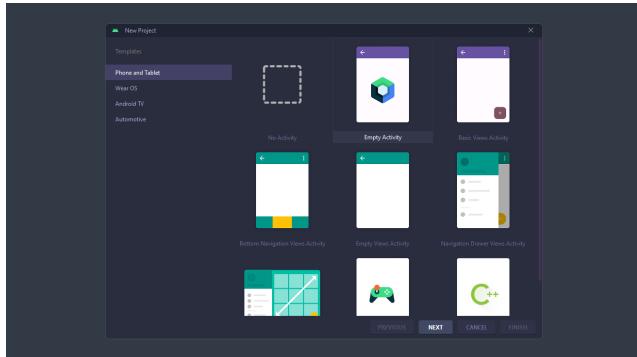
3. Jalankan studio untuk setup



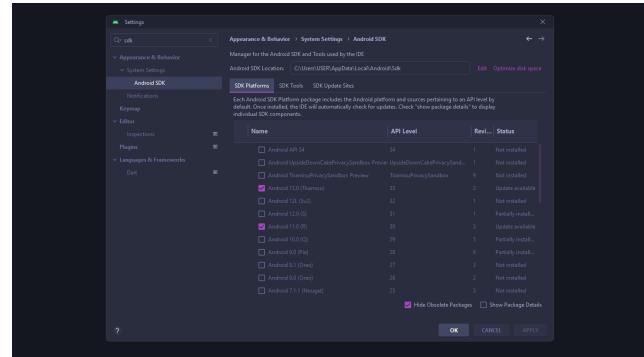
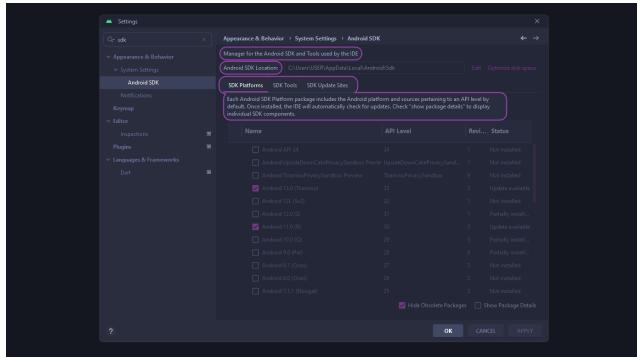
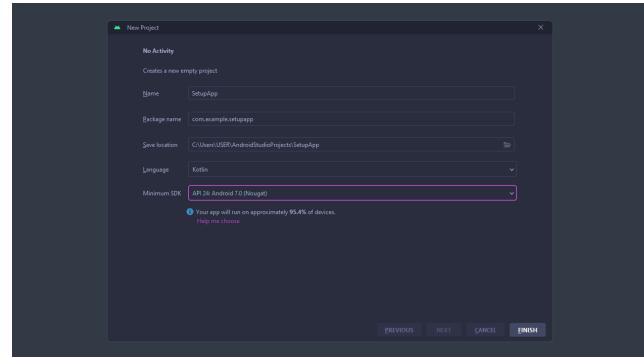
4. Install plugin untuk flutter dan dart



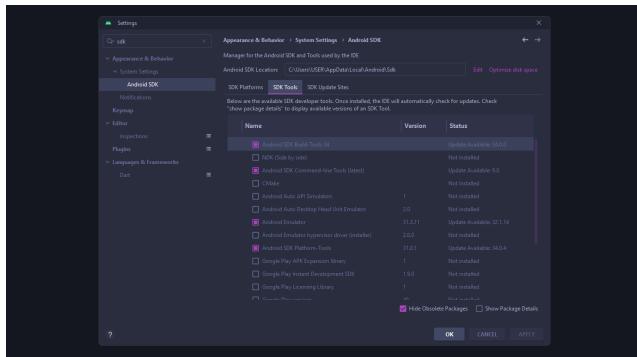
5. Buat project baru dengan non activity



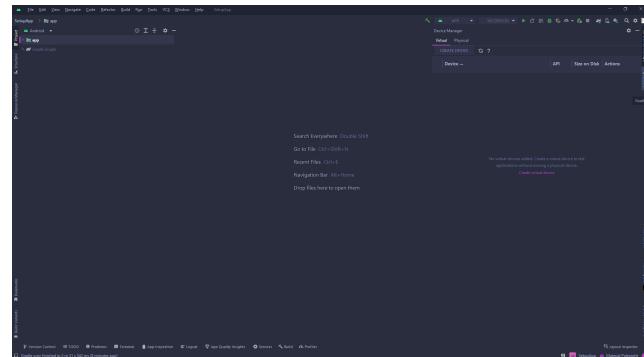
6. Project ini digunakan untuk setup

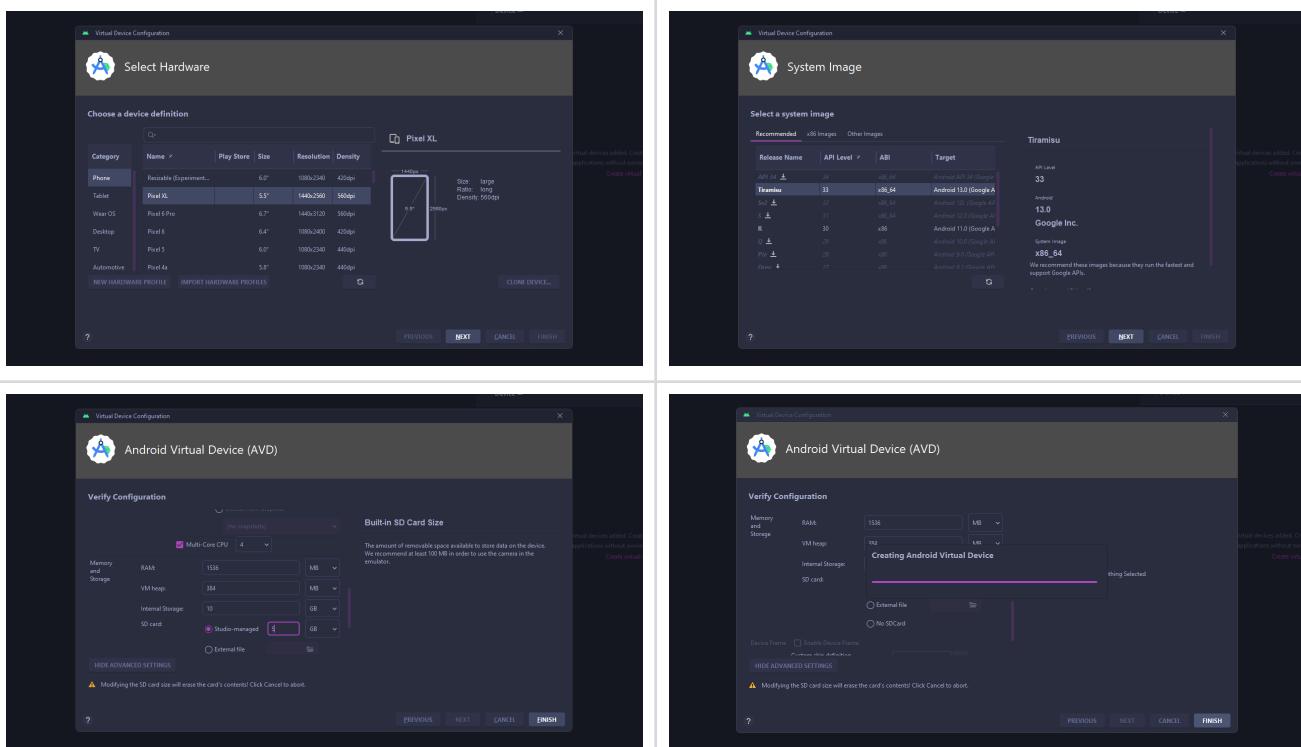


7. Apply atau oke untuk menginstal checklist

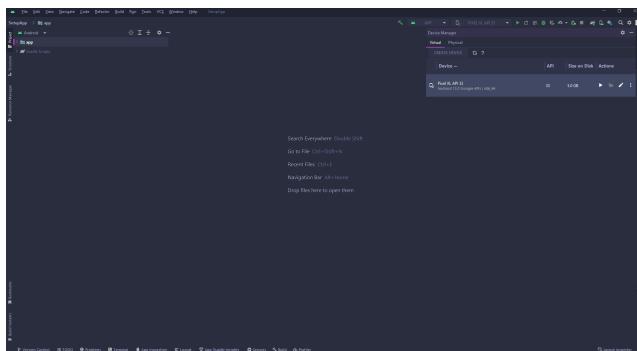


8. Click create devices untuk membuat emulator

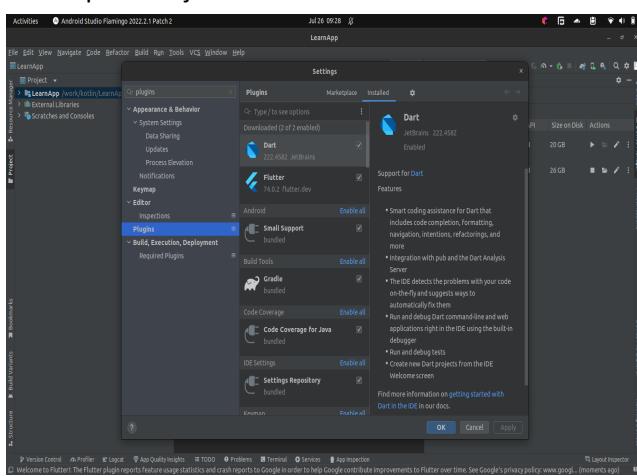




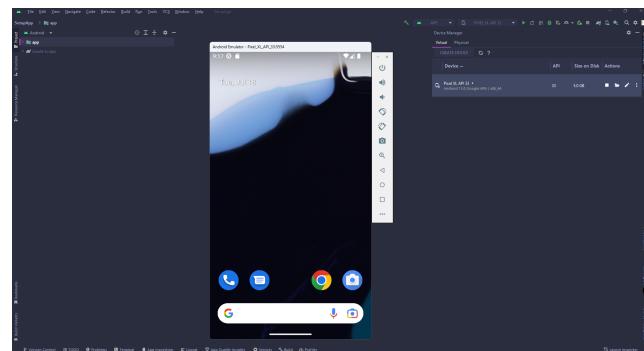
9. Tekan tombol play untuk menjalankan emulator



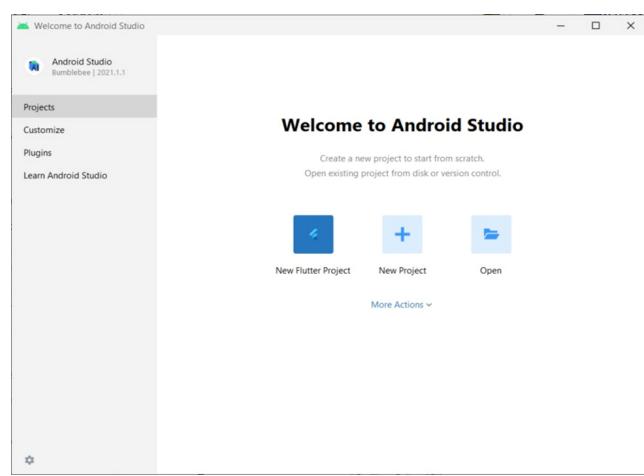
11. Android studio membutuhkan 2 plugins untuk dapat menjalankan flutter



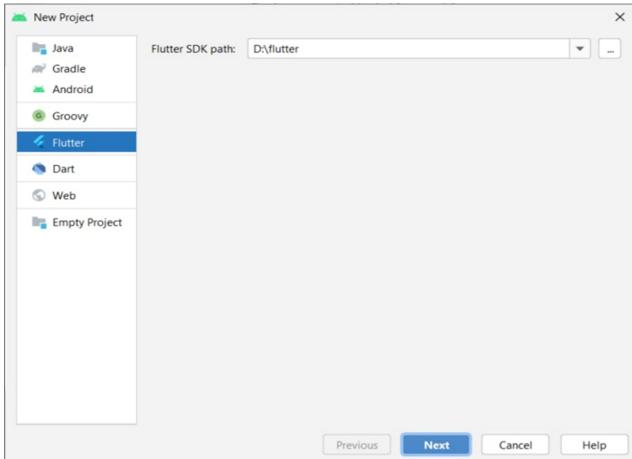
10. Emulator sudah dapat dipakai melalui IDE maupun Text Editor lainnya



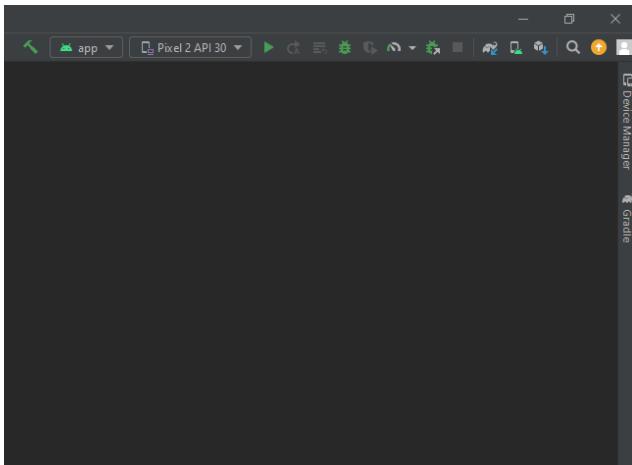
12. Untuk membuat project Flutter, Pilih New Project Flutter



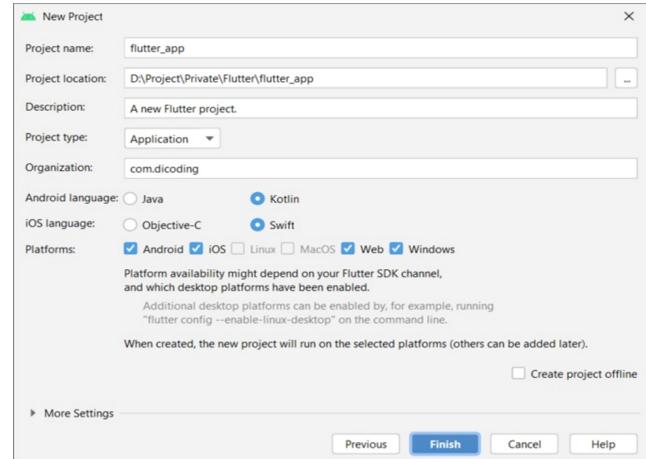
13. Pilih **Flutter SDK path** yang tersedia. Jika path tidak tersedia, klik tombol titik tiga untuk cari direktori Flutter SDK. Kemudian klik tombol **Next**.



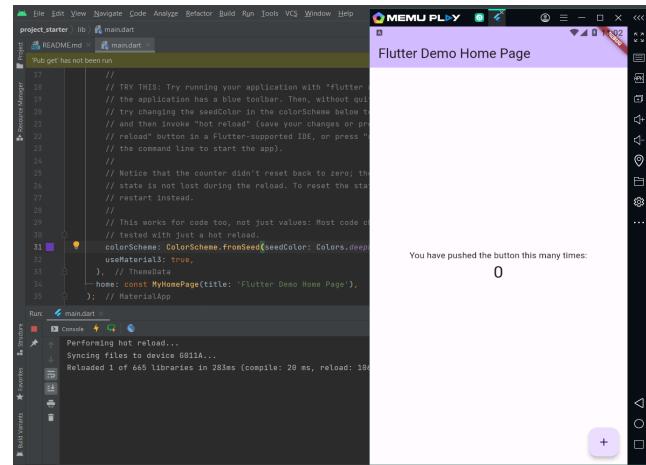
15. Pada pojok kanan atas, temukan toolbar seperti berikut. Kemudian klik tombol **run**. Pastikan sebelum klik tombol **run**, emulator sudah berjalan.



14. Masukkan nama proyek yang ingin dibuat pada **Project name**. Tentukan dimana proyek Anda disimpan pada **Project location**. Pilih Application pada kolom **Project type** untuk membuat aplikasi Flutter.



16. Setelah klik tombol run, tunggu sampai build aplikasi selesai. Setelah build aplikasi selesai pada emulator anda akan melihat **project template/starter**.



11. Agree **android license** untuk flutter

```
flutter doctor --android-licenses
```

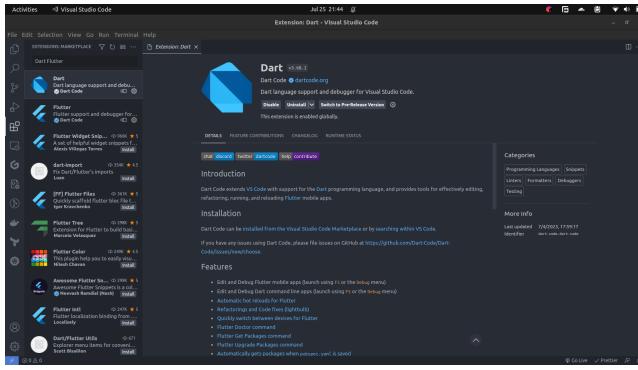
Visual Studio Code

Untuk menjalankan flutter pada vscode dibutuhkan 2 extension yang harus di install [vscode flutter project](#).

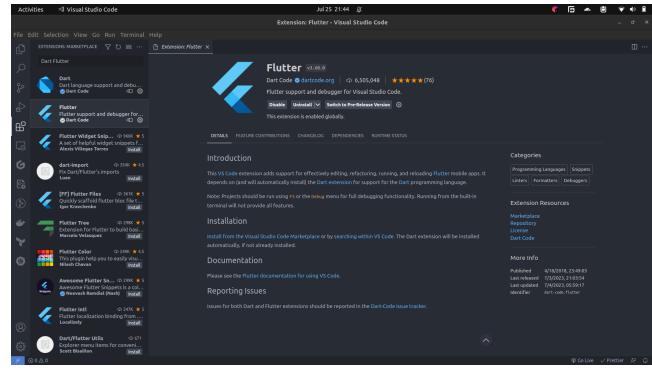
- Pastikan sudah agree **android license** untuk flutter

flutter doctor --android-licenses

2. Install Dart extension



3. Install Flutter extension



4. Buka command line dan create project flutter

Flutter create nama_aplikasi

5. tekan **ctrl+shift+p** lalu ketik **select device**

The screenshot shows the VS Code terminal with the command 'Flutter: Select Device' entered. A dropdown menu titled 'Select Device' lists available devices: 'Linux desktop', 'Chrome web', 'Start Galaxy Nexus API 33 mobile emulator', 'Start Pixel XL API 33 mobile emulator (cold boot)', 'Start Pixel XL API 33 mobile emulator (cold boot)', and '+ Create Android emulator'. The user has selected 'Start Pixel XL API 33 mobile emulator (cold boot)'.

```
Jul 26 08:08 ⚡
main.dart - module_app - Visual Studio Code

custom.widget.dart:1,0 >select device
Flutter: Select Device
Available Devices
Linux desktop
Chrome web
Start Galaxy Nexus API 33 mobile emulator
Start Pixel XL API 33 mobile emulator (cold boot)
Start Pixel XL API 33 mobile emulator (cold boot)
+ Create Android emulator
```

6. Pilih salah satu device yang diinginkan

The screenshot shows the VS Code terminal with the command 'Flutter: Select Device' entered. A dropdown menu titled 'Select a device to use' lists available devices: 'Linux desktop', 'Chrome web', 'Start Galaxy Nexus API 33 mobile emulator', 'Start Pixel XL API 33 mobile emulator (cold boot)', 'Start Pixel XL API 33 mobile emulator (cold boot)', and '+ Create Android emulator'. The user has selected 'Start Pixel XL API 33 mobile emulator (cold boot)'.

```
Jul 26 08:21 ⚡
main.dart - module_app - Visual Studio Code

custom.widget.dart:1,0 >Select a device to use
Available Devices
Linux desktop
Chrome web
Start Galaxy Nexus API 33 mobile emulator
Start Pixel XL API 33 mobile emulator (cold boot)
Start Pixel XL API 33 mobile emulator (cold boot)
+ Create Android emulator
```

7. Setelah terbuka, tekan **ctrl+f5** atau klik **run** diatas main

The screenshot shows the VS Code interface with the 'main.dart' file open. The code defines a simple StatelessWidget named 'MyApp'. The 'build' method sets the system UI mode to 'immersiveSticky'. The 'runApp' method runs the app on the 'Pixel_XL_API_33' emulator. The emulator window shows the 'Flutter Demo Home Page' with a blue background and white text.

```
Jul 26 08:29 ⚡
main.dart - module_app - Visual Studio Code

lib/main.dart
main.dart
You, 26 minutes ago | 1 author (You)
// ignore_for_file: unnecessary_string_interpolations

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    SystemChrome.setSystemUIOverlayStyle(
      SystemUiOverlayStyle(
        statusBarColor: Colors.blue,
        systemNavigationBarColor: Colors.white,
        systemNavigationBarInvertColor: true,
        systemNavigationBarIconColor: Colors.white,
        systemNavigationBarDividerColor: Colors.white,
        systemUiOverlayStyle: SystemUiOverlayStyle(
          statusBarColor: Colors.blue,
          systemNavigationBarColor: Colors.white,
          systemNavigationBarInvertColor: true,
          systemNavigationBarIconColor: Colors.white,
          systemNavigationBarDividerColor: Colors.white,
        ),
      ),
    );
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: Scaffold(
        appBar: AppBar(
          title: Text('Flutter Demo Home Page'),
        ),
        body: Center(
          child: Text('Hello World!'),
        ),
      ),
    );
  }
}
```

8. Tunggu sampai aplikasi berjalan

The screenshot shows the VS Code interface with the 'main.dart' file open. The code is identical to the previous screenshot. The 'Run' command is being executed, indicated by the status bar message 'Launching lib/main.dart on sdk gphone64_x86 4 in debug mode...'. The emulator window shows the 'Flutter Demo Home Page' with a blue background and white text.

```
Jul 26 08:03 ⚡
main.dart - example_app - Visual Studio Code

lib/main.dart
You, 26 minutes ago | 1 author (You)
// ignore_for_file: unnecessary_string_interpolations

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    SystemChrome.setSystemUIOverlayStyle(
      SystemUiOverlayStyle(
        statusBarColor: Colors.blue,
        systemNavigationBarColor: Colors.white,
        systemNavigationBarInvertColor: true,
        systemNavigationBarIconColor: Colors.white,
        systemNavigationBarDividerColor: Colors.white,
        systemUiOverlayStyle: SystemUiOverlayStyle(
          statusBarColor: Colors.blue,
          systemNavigationBarColor: Colors.white,
          systemNavigationBarInvertColor: true,
          systemNavigationBarIconColor: Colors.white,
          systemNavigationBarDividerColor: Colors.white,
        ),
      ),
    );
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: Scaffold(
        appBar: AppBar(
          title: Text('Flutter Demo Home Page'),
        ),
        body: Center(
          child: Text('Hello World!'),
        ),
      ),
    );
  }
}
```

9. Setelah itu aplikasi sudah bisa di develop sesuai kebutuhan

Syntax

Pada Dart syntax seperti variable dan operator yang digunakan tidak jauh berbeda dengan yang digunakan oleh bahasa pemrograman lain [dart variables](#).

Variable

Variabel bisa dibayangkan sebagai sebuah kotak atau wadah yang menyimpan nilai. Di dalam komputer variabel ini disimpan di dalam memori komputer. Setiap variabel memiliki nama yang dapat kita panggil dan gunakan. Pada Dart kita mendefinisikan sebuah variabel dengan keyword **var**. Perhatikan contoh berikut.

```
void main() {
  var message= 'Hello World!';
  print(message);
}
```

Perhatikan tanda sama dengan (=) di atas. Simbol tersebut dikenal dengan **assignment operator**. Kode di atas berarti kita memasukkan nilai ‘Hello World!’ ke dalam sebuah kotak atau variabel yang bernama **message**. Proses assignment nilai ke variabel ini disebut inisialisasi. Selain itu, kita juga bisa memasukkan nilai numerik atau angka, cukup ganti nilai ‘Hello World!’ ke angka seperti **50**.

Selain inisialisasi, kita juga dapat melakukan deklarasi terlebih dahulu contohnya seperti ini.

```
void main() {
  var size;
  size= 50;
  print(size);
}
```

Perlu diingat bahwa ketika kita mendeklarasikan variabel dengan **var**, Dart akan secara otomatis menentukan tipe datanya, contohnya seperti berikut:

```
var message= 'Hello World!';      //String
var size= 50;                      //integers
```

Kita tetap bisa mendeklarasikan tipe data variabel secara eksplisit untuk menghindari kebingungan dan memudahkan proses *debugging*.

```
String message= 'Hello World!';
int size= 50;
```

Operators

Operators yang digunakan sama dengan bahasa pemrograman lain. Contoh operator yang digunakan antara lain:

Operator Aritmatika

Operator	Deskripsi
+	Penjumlahan
-	Pengurangan
*	Perkalian
/	Pembagian
~/	Pembagian, mengembalikan nilai int
%	Modulo atau sisa hasil bagi

Operator Perbandingan

Operator	Deskripsi
==	Sama dengan
!=	Tidak sama dengan
>	Lebih dari
<	Kurang dari
>=	Lebih dari sama dengan
<=	Kurang dari sama dengan

Operator Logika

Operator	Deskripsi
	OR
&&	AND
!	NOT

Cascade Notation

Operator cascade notation adalah fitur yang memungkinkan kita untuk memanggil metode atau mengakses property dari objek tanpa harus mengulang-ulang menyebutkan nama objek tersebut. Dengan menggunakan

operator `..`, kita dapat melakukan sejumlah tindakan pada objek yang sama dalam satu baris kode. Perhatikan contoh kode berikut:

```
class Person {  
    String name = " ";  
    int age = 0;  
  
    void setName(String name) {  
        this.name = name;  
    }  
  
    void setAge(int age) {  
        this.age = age;  
    }  
  
    void printInfo() {  
        print('Name: $name, Age: $age');  
    }  
}  
  
void main() {  
    Person person = Person();  
  
    // Tanpa menggunakan cascade notation  
    person.setName('John');  
    person.setAge(30);  
    person.printInfo(); // Output: Name: John, Age: 30  
  
    // Menggunakan cascade notation  
    person..setName('Alice')..setAge(25)..printInfo(); // Output: Name: Alice, Age: 25  
}
```

Pada contoh di atas, kita memiliki kelas `Person` dengan beberapa metode dan properti. Tanpa menggunakan cascade notation, kita harus memanggil metode `setName`, `setAge`, dan `printInfo` secara terpisah untuk mengatur dan mencetak informasi. Namun, dengan menggunakan cascade notation (operator `..`), kita dapat memanggil metode-metode tersebut berurutan pada objek `person` dalam satu baris kode.

Function

Penggunaan function pada dart flutter dapat dilakukan dengan beberapa cara [function](#). Mulai dari main function [educative-main function](#), type function [geeks-type function](#), output function , dan type parameters [sarunw-param function](#).

Main

Fungsi `main()` adalah fungsi teratas di Dart yang bertanggung jawab untuk semua jenis eksekusi program. Fungsi ini sangat penting dan krusial sama seperti `main()` function pada bahasa **C, Java, Go**, dll. Dalam sebuah program, fungsi `main()` hanya dapat digunakan satu kali.

```
main(){
  var name = "Riyan";
  String language = "Dart";
  print("Hai $name. Welcome to $language");
}
```

Type

1. Fungsi tanpa argumen dan tanpa return

```
namaFunction(){
  print("Function Tanpa Argument dan Return");
}
```

2. Fungsi tanpa argumen namun memiliki return [Output](#)

```
TypeData namaFunction(){
  print("Function Tanpa Argumen namun memiliki Return");
  return ValueTypeData;
}
```

3. Fungsi berargumen namun tanpa return

```
namaFunction(TypeArgument argument1){
  print("Function ber $argument1 namun tanpa Return");
}
```

4. Fungsi berargumen dan memiliki return [Output](#)

```
TypeData namaFunction(TypeArgument argument1){
    print("Function ber $argument1 namun dan memiliki Return");
    return argument1.toTypeData();
}
```

Output

Keluaran dari sebuah function disesuaikan dengan kebutuhan. Berikut beberapa opsi yang dapat dipilih sebagai output function.

<code>String outFunc() { return "string"; }</code>	String	<code>String text = outFunc();</code>
<code>int outFunc() { return 1; }</code>	int	<code>int angka = outFunc();</code>
<code>double outFunc() { return 1.2; }</code>	double	<code>double decimal = outFunc();</code>
<code>bool outFunc() { return false; }</code>	bool	<code>bool isConfused = outFunc();</code>
<code>TypeData? outFunc(){ return null; }</code>	TypeData?	<code>if (outFunc() != null) print("tidak kosong");</code>
<code>dynamic outFunc(){ return "string"; }</code>	dynamic	<code>String sebuahText = outFunc() as String;</code>
<code>List<TypeData> outFunc() { return ["string 1", "string 2"]; }</code>	List<TypeData>	<code>List<TypeData> listData = outFunc();</code>

<pre>Future outFunc() async { return await Future.delayed(Duration(seconds: 1), () { return Random().nextInt(1).toString(); }); }</pre>	<p style="text-align: center;">Future</p>	<pre>void main() async { print(await outFunc()); }</pre>
<pre>Future<int> outFunc() async { return await Future.delayed(const Duration(seconds: 1), () { return Random().nextInt(1); }); }</pre>	<p style="text-align: center;">Future<int></p>	<pre>void main() async { int number = await outFunc(); }</pre>
<pre>Object outFunc() { return const Object(username: "Lorem Ipsum", title: "Flutter Developer", image: "https://i.pravatar.cc/300",); }</pre>	<p style="text-align: center;">Object</p>	<pre>void main() async { Object number = outFunc(); }</pre>
<pre>Future<Object> outFunc() async { return await Future.delayed(const Duration(seconds: 1), () { return const Object(username: "Lorem Ipsum", title: "Flutter Developer", image: "https://i.pravatar.cc/300",); }); }</pre>	<p style="text-align: center;">Future<Object></p>	<pre>void main() async { Object number = await outFunc(); }</pre>
<pre>Future<TypeData?> outFunc() async { return await Future.delayed(const Duration(seconds: 1), () { return null; }); }</pre>	<p style="text-align: center;">Future<TypeData?></p>	<pre>if (await outFunc() != null) print("tidak kosong");</pre>

TypeData	semua tipe data
TypeData?	semua tipe data yang dapat nullable
Dynamic	tipe data dinamis
List<TypeData>	list dari sebuah tipe data
Future	berjalan secara asynchronous
Future<TypeData>	berjalan secara asynchronous dengan output tipe data specific
Future<TypeData?>	berjalan secara asynchronous dengan output tipe data specific yang bisa nullable
Object	tipe data object yang telah dibuat
Future<Object>	berjalan secara asynchronous dengan output tipe data object

Semua tipe data diatas dapat dikreasikan antara satu dengan yang lain tergantung kreatifitas masing-masing. Yang perlu diingat dalam mengkreasikan tipe data adalah perbaiki semua error-error yang muncul.

Parameters

1. Required positional parameters

Semua parameters menjadi wajib diisi dengan pengisian berurut sesuai tipe data

```
namaFunction(String params1, int params2, bool params3) {
    print("$params1 $params2 $params3");
}

void main() async {
    print(namaFunction("", 1, false));
}
```

2. Required Named parameters

Parameter wajib diisi namun urutan tidak mempengaruhi kesesuaian tipe data. Input dilakukan dengan menggunakan nama parameternya.

```
namaFunction({required String params1, required int params2, required bool params3}) {
    print("$params1 $params2 $params3");
}

void main() async {
    print(namaFunction(params3: false, params1: "", params2: 1));
}
```

3. Optional Named parameters

Parameter datap nullable diisi dan urutan tidak mempengaruhi kesesuaian tipe data. Input dilakukan dengan menggunakan nama parameternya.

```
namaFunction({String? params1, required int params2, bool? params3}) {  
    print("$params1 $params2 $params3");  
}  
  
void main() async {  
    print(namaFunction(params2: 1));  
}
```

4. Optional positional parameters

Semua parameters menjadi dapat nullable namun untuk pengisian harus berurut sesuai tipe data

```
namaFunction([String? params1, int? params2, bool? params3]) {  
    print("$params1 $params2 $params3");  
}  
  
void main() async {  
    print(namaFunction("1"));  
}
```

Beberapa metode parameter dapat digabungkan selagi tidak menimbulkan error dan selagi memang diperlukan.

Features

Terdapat banyak fitur yang disediakan dart dan flutter untuk mempermudah proses development dan menangani kasus yang kompleks [features](#) [javatpoint-dart features](#) [educative-flutter features](#) [dart syntax](#).

Null Safety

Null safety adalah fitur variabel dalam Dart yang dapat menampung nilai null atau tanpa nilai (null or absence of value). Null safety bertujuan untuk mengurangi risiko bug yang disebabkan oleh kesalahan null, yang merupakan penyebab umum dari error dan crash dalam aplikasi.

```
String dataValuable = null; // Compile error
String? dataNullable = null;
```

Selain dapat menghindari **error compiling** karena value null, fitur ini juga dapat berguna pada sebuah function.

```
main() {
    // pass data yang dapat null
    String? dataNullable;
    dataOptional(dataNullable);

    // pass data! yang harus bervalue
    String? dataValuable = "value";
    dataRequired(dataValuable!);
}

void dataOptional(String? data) {
    if (data == null) {
        print('Data nullable bisa jadi kosong');
    } else {
        print('Data nullable bisa jadi memiliki $data');
    }
}

void dataRequired(String data) {
    print('Data required pasti memiliki $data');
}
```

dataOptional	String?	Variabel yang dapat tidak diisi atau diisi dengan value null
dataRequired	String	Variabel yang harus diisi dan tidak boleh bernilai null

dataOptional!	String	Variabel optional yang ditegaskan sebagai pasti bernilai
---------------	--------	---

required	nullable
----------	----------

dataOptional menggunakan tanda ! / seru (dataOptional!) berfungsi untuk menegaskan bahwa **dataOptional!** **dipastikan** memiliki nilai dan **bukan null** lagi.

Async Await

Adalah fitur yang digunakan untuk melakukan pemrograman asinkron yaitu mengeksekusi tugas yang mungkin memerlukan waktu untuk selesai, seperti operasi jaringan, pembacaan file, atau operasi database, tanpa menghentikan eksekusi aplikasi.

```
void main() async {
  print("Memulai Aplikasi");

  // Memanggil fungsi fetchDataFromNetwork dan menunggu hasilnya dengan await
  String dataFromNetwork = await fetchDataFromNetwork();
  print("menampilkan: $dataFromNetwork");

  // Memanggil fungsi fetchDataFromLocalStorage dan menunggu hasilnya dengan await
  String dataFromLocal = await fetchDataFromLocalStorage();
  print("menampilkan: $dataFromLocal");
}

Future<String> fetchDataFromNetwork() async {
  await Future.delayed(const Duration(seconds: 2));
  return "Data dari penyimpanan network";
}

Future<String> fetchDataFromLocalStorage() async {
  await Future.delayed(const Duration(seconds: 2));
  return "Data dari penyimpanan lokal";
}
```

async	Fitur pada function agar dapat dijalankan secara bersamaan dengan tugas lain
await	Fitur menunggu yang hanya bisa dipakai pada function async

Ternary

Fitur untuk menulis singkat ekspresi kondisional dalam Flutter agar memeriksa suatu kondisi dan mengembalikan sebuah nilai dari kondisi yang sesuai. Ternary ditulis dalam sebuah baris dan memiliki dua atau lebih opsi kondisi sehingga dapat membantu mengurangi jumlah kode untuk ekspresi kondisional.

```
var kondisi ? nilai_jika_benar : nilai_jika_salah
```

```
void main() async {
    int age = 25;

    // Menggunakan Ternary untuk menentukan pesan berdasarkan kondisi
    String message = age >= 23 ? "Anda kelahiran tahun 2000an" : "Anda kelahiran tahun 1000an";

    print(message); // Output: "Anda kelahiran tahun 2000an"
}
```

age >= 23	Sebuah kondisi yang diharapkan
(tanda tanya) ?	Untuk menyatakan kondisi
(titik dua) :	Untuk memisahkan opsi pada ternary
message	Variabel yang menampung hasil dari suatu kondisi

Widget

Widget digunakan untuk membangun UI, widget ditulis secara declarative untuk mendeskripsikan bentuk tampilan yang dibuat. Flutter memiliki widget dasar yang sering dipakai, diantaranya adalah sebagai berikut [basic widget](#):

StatelessWidget

Sebuah widget untuk mendeskripsikan bagian-bagian user interface yang hanya dapat menampilkan informasi konfigurasi object dan BuildContext widget itu sendiri [stateless widget](#). Artinya stateless tidak dapat mengubah data-data didalamnya secara langsung.

```
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Stateless Widget',
      home: StatelessPage(dataStateless: 10),
    );
  }
}

class StatelessPage extends StatelessWidget {
  StatelessPage({super.key, required this.dataStateless});

  int dataStateless;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Text(
          "Data Stateless = $dataStateless"
        ),
      ),
      floatingActionButton: FloatingActionButton(
        child: const Icon(Icons.add),
        onPressed: () {
          // data tidak akan berubah secara langsung
          // bila menggunakan StatelessWidget
          dataStateless += 1;
        },
      ),
    );
  }
}
```

Data Stateless = 10



StatefulWidget

Widget untuk mendeskripsikan bagian-bagian user interface yang memiliki status bisa berubah. StatefulWidget dapat mengubah data-data didalamnya secara dinamis [stateful widget](#). Untuk mengubah data cukup dengan memanggil `setState(() {});`

```
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Stateful Widget',
      home: StatefulWidget(dataStateful: 10),
    );
  }
}

class StatefulWidget extends StatefulWidget {
  StatefulPage({super.key, required this.dataStateful});
  int dataStateful;

  @override
  State< StatefulWidget > createState() => _ StatefulWidgetState();
}

class _ StatefulWidgetState extends State< StatefulWidget > {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Text(
          "Data Stateful = ${widget.dataStateful}",
          style: const TextStyle(fontSize: 30),
        ),
      ),
      floatingActionButton: FloatingActionButton(
        child: const Icon(Icons.add),
        onPressed: () {
          widget.dataStateful += 1;
          setState(() {});
        },
      ),
    );
  }
}
```

Data Stateful = 15

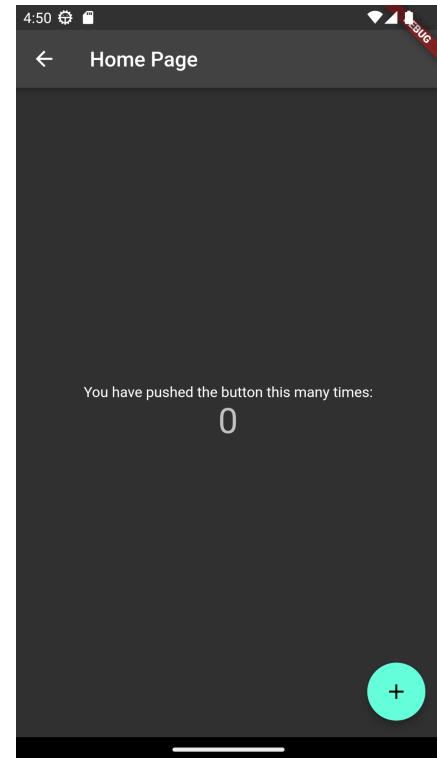


MaterialApp

Widget utama yang diinisialisasi di awal agar dapat menggunakan material design aplikasi. MaterialApp akan membungkus material widget lainnya dan membuat aplikasi menjadi interaktif mengikuti panduan Material Design [materialapp_geeksforgeeks-materialapp](#).

```
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'MaterialApp Demo',
      theme: ThemeData.light(),
      themeMode: ThemeMode.system,
      darkTheme: ThemeData.dark(),
      initialRoute: "/home",
      routes: <String, WidgetBuilder>{
        '/home': (context) => const MyHomePage(title: 'Home Page'),
        '/about': (context) => const AboutPage(),
      },
      home: const MyHomePage(title: 'Home Page'),
    );
  }
}
```



title	String	Description yang dipakai device untuk identifikasi aplikasi.
theme	ThemeData?	Untuk setup light theme mode
darkTheme	ThemeData?	Untuk setup dark theme mode
themeMode	ThemeData?	Setup yang dipakai aplikasi saat: light (terang), dark (gelap), system (mengikuti sistem)
initialRoute	String?	Screen pertama dari aplikasi
routes	Map<String, WidgetBuilder>	Kumpulan screen atau routes yang ada pada aplikasi

required

nullable

Scaffold

Widget Desain Material dasar untuk memvisualisasi tata letak. Serta dapat menampilkan drawers and bottom sheets [scaffold](#).

```
int _count = 0;

@Override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Sample Code'),
    ),
    body: Center(child: Text('You have pressed the button $_count times.')),
    floatingActionButton: FloatingActionButton(
      onPressed: () => setState(() => _count++),
      child: const Icon(Icons.add),
    ),
  );
}
```

Sample Code

You have pressed the button 0 times.



appBar	PreferredSize Widget?	AppBar menampilkan widget toolbar, leading, title, actions, dan widget apapun pada bagian bawahnya
body	Widget	Badan tampilan pada widget Scaffold yang terletak dibawah AppBar
floatingActionButton	Widget?	Button mengambang di atas konten yang sering digunakan pada Scaffold
backgroundColor	Color?	Warna background scaffold
drawer	Widget?	Panel menu yang tampil di samping body yang dapat muncul atau hide saat ditekan

required

nullable

Column

Widget yang menampilkan childrennya secara vertical array. Column tidak dapat di scroll dan akan error bila children melewati space tersedia [column](#).

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: SizedBox(
      width: double.infinity,
      height: double.infinity,
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        crossAxisAlignment: CrossAxisAlignment.center,
        verticalDirection: VerticalDirection.down,
        children: const [
          Text("Children urutan 1"),
          Text("Children urutan 2"),
          Text("Children urutan 3"),
        ],
      ),
    );
}
```

Children urutan 1
Children urutan 2
Children urutan 3

children	List<Widget>	Kumpulan list widget yang akan ditampilkan berurut secara vertikal pada Column
mainAxisAlignment	MainAxisAlignment	Align vertical pada column: <code>center</code> , <code>start</code> , <code>end</code> , <code>spaceAround</code> , <code>spaceBetween</code> , <code>spaceEvenly</code>
crossAxisAlignment	CrossAxisAlignment	Align horizontal pada column: <code>center</code> , <code>start</code> , <code>end</code>
verticalDirection	VerticalDirection	Urutan peletakan children: <code>up</code> , <code>down</code>

required

nullable

Row

Widget yang menampilkan childrennya secara horizontal array. Row tidak dapat di scroll dan akan error bila children melewati space tersedia [row](#).

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: SizedBox(
      width: double.infinity,
      height: double.infinity,
      child: Row(
        mainAxisAlignment: MainAxisAlignment.spaceAround,
        crossAxisAlignment: CrossAxisAlignment.center,
        verticalDirection: VerticalDirection.down,
        children: const [
          Text("1"),
          Text("2"),
          Text("3"),
          Text("4"),
          Text("5"),
        ],
      ),
    );
}
```

1 2 3 4 5

children	List<Widget>	Kumpulan list widget yang akan ditampilkan berurut secara horizontal pada Row
mainAxisAlignment	MainAxisAlignment	Align horizontal pada row: center , start , end , spaceAround , spaceBetween , spaceEvenly
crossAxisAlignment	CrossAxisAlignment	Align vertical pada row: center , start , end
verticalDirection	VerticalDirection	Urutan peletakan children: up , down

required

nullable

Container

Adalah sebuah convenience widget yang dapat mengkombinasi pewarnaan, penentuan posisi, ukuran, dan bentuk widget [geeks-container container](#).

```
String dataContainer = """Di dalam Container
Di dalam Container
Di dalam Container
Di dalam Container""";
```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Container(
          alignment: Alignment.center,
          color: Colors.blue[100],
          child: Text(dataContainer),
        ),
      ],
    ),
  );
}
```

Didalam Container
Didalam Container
Didalam Container
Didalam Container

child	Widget?	Sebuah widget di dalam container
width	Double?	Set ukuran lebar container secara langsung
height	Double?	Set ukuran tinggi container secara langsung
alignment	AlignmentGeometry?	Align specific pada container: <code>topLeft</code> , <code>topCenter</code> , <code>topRight</code> , <code>centerLeft</code> , <code>center</code> , <code>centerRight</code> , <code>bottomLeft</code> , <code>bottomCenter</code> , <code>bottomRight</code> ,
color	Color?	Warna container yang sudah memiliki ukuran, baik secara langsung maupun tidak

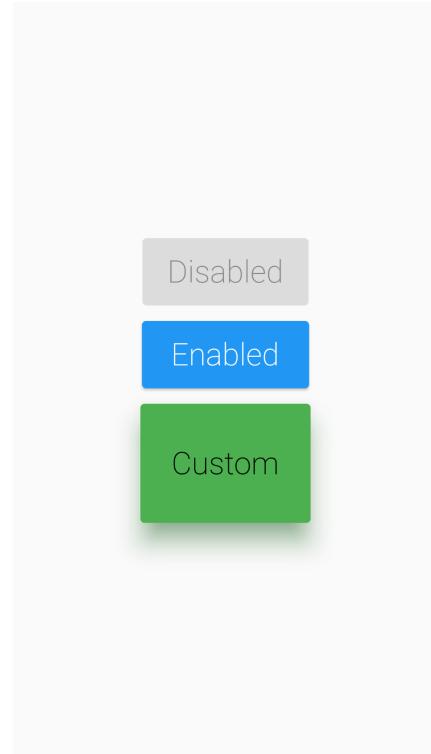
required

nullable

ElevatedButton

Adalah sebuah widget button standar yang memiliki text dan style serta komponen pendukung lainnya [elevated button](#).

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: SizedBox(
      width: double.infinity,
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        mainAxisSize: MainAxisSize.max,
        children: [
          const ElevatedButton(
            onPressed: null,
            child: Text('Disabled'),
          ),
          ElevatedButton(
            style: ElevatedButton.styleFrom(),
            onPressed: () {},
            child: const Text('Enabled'),
          ),
          ElevatedButton(
            style: ElevatedButton.styleFrom(
              elevation: 20,
              backgroundColor: Colors.green,
              shadowColor: Colors.green,
              foregroundColor: Colors.black,
              padding: const EdgeInsets.symmetric(
                horizontal: 18,
                vertical: 40,
              ),
            ),
            onPressed: () {},
            child: const Text('Custom'),
          ),
        ],
      ),
    );
}
```



onPressed	void Function()?	Melakukan aksi ketika tombol diketuk atau diaktifkan.
style	ButtonStyle?	Style dari tombol

child	Widget?	Sebuah widget di dalam elevatedbutton
-------	---------	---------------------------------------

Setiap style button memiliki banyak atribut yang dapat dipakai, beberapa diantaranya dapat dilihat pada tabel berikut

elevation	double?	Angka bayangan berdasarkan ketinggiannya
backgroundColor	Color?	Warna dari background button
shadowColor	Color?	Warna dari shadow button
foregroundColor	Color?	Warna effect saat button ditekan
padding	EdgeInsetsGeometry?	Ruang kosong pada button

required

nullable

Terdapat beberapa style yang dapat dipakai untuk sebuah button widget [button-style](#). Dibawah ini setiap button dapat menggunakan style button yang lainnya.

Elevated	ElevatedButton	ElevatedButton.styleFrom
Filled	FilledButton	FilledButton.styleFrom
Outlined	OutlinedButton	OutlinedButton.styleFrom
Text	TextButton	TextButton.styleFrom

Image

Adalah widget untuk menampilkan gambar yang memiliki beberapa metode penggunaan berdasarkan format input nya [image](#).

Network

Penggunaan widget image dengan menggunakan inputan url gambar di internet [image network](#).

@override

```
Widget build(BuildContext context) {  
    return Scaffold(  
        body: SizedBox(  
            width: double.infinity,  
            child: Column(  
                mainAxisAlignment: MainAxisAlignment.center,  
                mainAxisSize: MainAxisSize.max,  
                children: [  
                    Container(  
                        margin: const EdgeInsets.only(bottom: 10),  
                        child: const Text(  
                            "Image network",  
                            style: TextStyle(fontSize: 30),  
                        ),  
                    ),  
                    SizedBox(  
                        height: 250,  
                        width: 250,  
                        child: Image.network("https://picsum.photos/500"),  
                    ),  
                ],  
            ),  
        );  
}
```

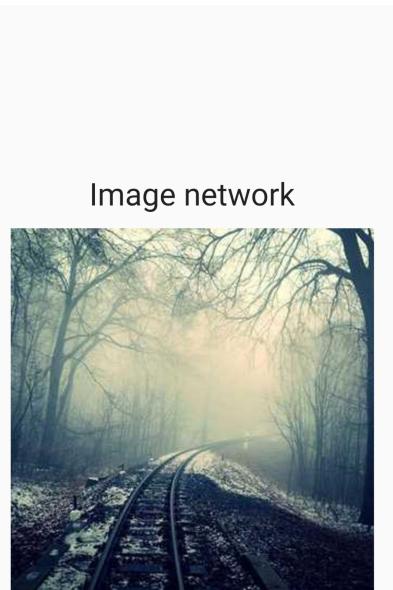


Image network

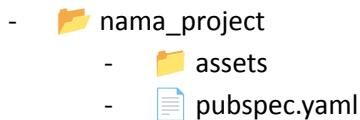
Asset

Penggunaan widget image dengan menggunakan inputan path directory gambar di project [image-asset](#). Sebelum dapat menggunakan widget ini kita harus melakukan beberapa setup terlebih dahulu.

1. Taruh image pada sebuah folder di project



2. Buka file [pubspec.yaml](#) pada project



3. Ubah dari code **sebelum** (kiri) menjadi **sesudah** (kanan) lalu **save**.

flutter: <pre># The following line ensures that the Material Icons font is # included with your application, so that you can use the icons in # the material Icons class. uses-material-design: true # To add assets to your application, add an assets section, like this: # assets: # - images/a_dot_burr.jpeg # - images/a_dot_ham.jpeg</pre>	flutter: <pre>uses-material-design: true</pre> assets: <pre>- assets/images/</pre>
---	---

4. Selanjutnya semua assets pada **images** sudah dapat digunakan.

```

@Override
Widget build(BuildContext context) {
  return Scaffold(
    body: SizedBox(
      width: double.infinity,
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Container(
            margin: const EdgeInsets.only(bottom: 10),
            child: const Text(
              "Image asset",
              style: TextStyle(fontSize: 30),
            ),
          ),
          SizedBox(
            height: 250,
            width: 250,
            child: Image.asset(
              "assets/images/nature.jpg",
              fit: BoxFit.cover,
            ),
          ),
        ],
      ),
    );
}

```



Image asset

fit	BoxFit?	Pengalokasian gambar pada sebuah ruangan: cover, fill, contain, none, fitHeight, fitWidth
-----	---------	---

required

nullable

File

Penggunaan widget image dengan menggunakan file image yang ada pada smartphone [image file](#). Sebelum dapat menggunakan widget ini kita harus menginstall sebuah library terlebih dahulu [image-picker](#).

1. Buka file [pubspec.yaml](#) pada project

-  nama_project
 -  pubspec.yaml

2. Ubah dari code [sebelum](#) (kiri) menjadi [sesudah](#) (kanan) lalu [save](#).

```
dependencies:  
  flutter:  
    sdk: flutter
```

```
dependencies:  
  flutter:  
    sdk: flutter  
  
  # media  
  image_picker: ^1.0.1
```

3. Setelah library terinstall kita sudah dapat mengambil image dari smartphone

```
File? image;
```

```
Future getImage() async {  
  final ImagePicker picker = ImagePicker();  
  
  final XFile? imageX = await picker.pickImage(source:  
    ImageSource.gallery);  
  if (imageX == null) return;  
  
  image = File(imageX.path);  
  setState(() {});  
}
```

Image file

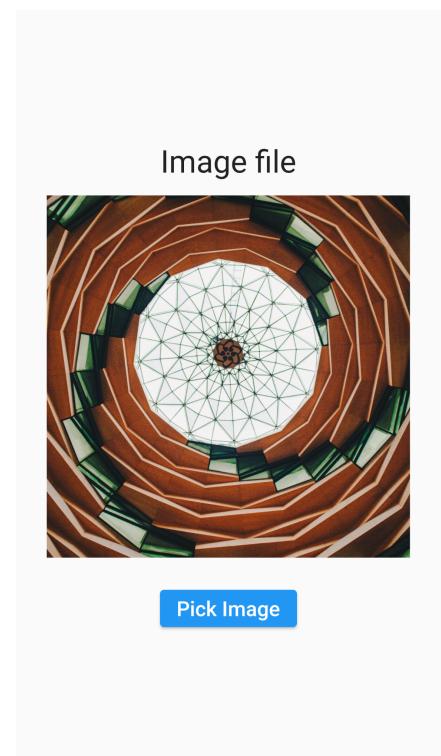
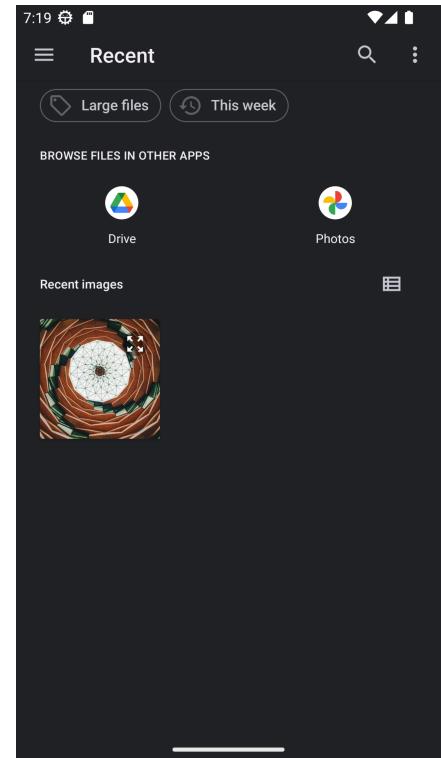


[Pick Image](#)

```

@Override
Widget build(BuildContext context) {
  return Scaffold(
    body: SizedBox(
      width: double.infinity,
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        mainAxisSize: MainAxisSize.max,
        children: [
          // title
          Container(
            margin: const EdgeInsets.only(bottom: 15),
            child: const Text(
              "Image file",
              style: TextStyle(fontSize: 30),
            ),
          ),
          // image file
          if (image != null) ...[
            Container(
              height: 350,
              width: 350,
              margin: const EdgeInsets.only(bottom: 25),
              child: Image.file(image!, fit: BoxFit.fitWidth),
            ),
          ] else ...[
            Container(
              height: 350,
              width: 350,
              padding: const EdgeInsets.all(100),
              margin: const EdgeInsets.only(bottom: 25),
              child: const CircularProgressIndicator(),
            ),
          ],
          // button
          ElevatedButton(
            style: ElevatedButton.styleFrom(),
            onPressed: () async {
              await getImage();
            },
            child: const Text(
              'Pick Image',
              style: TextStyle(fontSize: 20),
            ),
          ),
        ],
      ),
    ),
  );
}

```



```
  },
);
}
```

<code>if (image != null) ... // widget] else ...[// widget ,</code>	List<Widget>	If else atau percabangan pada sebuah List<Widget> atau children
<code>getImage()</code>	Future<dynamic>	Function yang dibuat untuk mengambil image dari smartphone
<code>image</code>	File?	Menampung file image setelah diambil dari smartphone
<code>picker</code>	ImagePicker	Library flutter.dev untuk mengambil image atau asset dari smartphone
<code>CircularProgressIndicator</code>	Widget	Loading indicator untuk menampilkan loading proses

required

nullable

Text

Widget teks menampilkan serangkaian teks string dengan menerapkan sebuah style yang diinginkan [text](#).

```
class _StatefulWidget extends StatefulWidget {
  String dataText = """Text Widget Line 1
Text Widget Line 2
Text Widget Line 3
Text Widget Line 4""";
```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Center(
      child: Text(
        dataText,
        maxLines: 2,
        textAlign: TextAlign.center,
        style: const TextStyle(
          fontSize: 40,
          color: Colors.white,
          decoration: TextDecoration.underline,
          backgroundColor: Colors.red,
          fontWeight: FontWeight.w600,
          fontStyle: FontStyle.italic,
          decorationStyle: TextDecorationStyle.dashed,
          fontFamily: "ebGaramond",
        ),
      ),
    ),
  );
}
```



Text Widget Line 1
Text Widget Line 2

maxLines	int?	Jumlah maximal line text yang ditampilkan
textAlign	TextAlign?	Align text atau perataan: left , right , justify , center , start , end
fontSize	double?	Size dari sebuah font
color	Color?	Warna font yang dipakai

decoration	TextDecoratio n?	Dekorasi pada text: <code>lineThrough</code> , <code>overline</code> , <code>underline</code>
backgroundColor	Color?	Background color pada dari text
fontWeight	FontWeight?	Ketebalan text atau tingkat bold: <code>normal</code> , <code>bold</code> , <code>w100</code> , <code>w200</code> , <code>w300</code> , <code>w400</code> , <code>w500</code> , <code>w600</code> , <code>w700</code> , <code>w800</code> , <code>w900</code>

required**nullable**

Untuk mengubah fontFamily harus terdapat data font family yang bisa diakses, salah satu caranya dengan menambahkan data font ke sebuah asset folder [beads-text](#).

1. Tambahkan pada folder `/assets/fonts` sebuah font yang sudah didownload [google-fonts](#)

- `nama_project`
 - `assets`
 - `fonds`
 - `eb_garamond.ttf`

2. Buka file `pubspec.yaml` pada project

- `nama_project`
 - `pubspec.yaml`

3. Ubah dari code **sebelum** (kiri) menjadi **sesudah** (kanan) lalu **save**.

```
# example:  
#fonts:  
# - family: Schyler  
#   fonts:  
#     - asset: fonts/Schyler-Regular.ttf  
#     - asset: fonts/Schyler-Italic.ttf  
#       style: italic
```

```
fonts:  
- family: ebGaramond  
  fonts:  
    - asset: assets/fonts/eb_garamond.ttf
```

4. Setelah itu string “ebGaramond” dapat digunakan sebagai fontFamily pada `style: const TextStyle(fontFamily: "ebGaramond")`

Grid

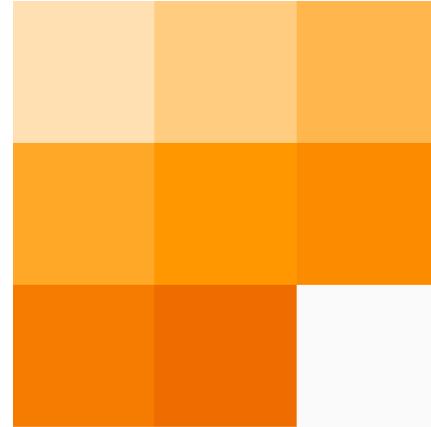
Widget yang menata children atau list widget agar berurut dan dapat langsung mengambil space kosong dari horizontal dan vertikal [ninja-gridview](#).

GridView

Standar widget untuk tampilan grid

`@override`

```
Widget build(BuildContext context) {
  return Scaffold(
    body: GridView(
      physics: const BouncingScrollPhysics(parent:
        AlwaysScrollableScrollPhysics()),
      gridDelegate: const SliverGridDelegateWithMaxCrossAxisExtent(
        maxCrossAxisExtent: 200,
      ),
      children: [
        Container(color: Colors.orange[100]),
        Container(color: Colors.orange[200]),
        Container(color: Colors.orange[300]),
        Container(color: Colors.orange[400]),
        Container(color: Colors.orange[500]),
        Container(color: Colors.orange[600]),
        Container(color: Colors.orange[700]),
        Container(color: Colors.orange[800]),
      ],
    ),
  );
}
```



gridDelegate	SliverGridDel egate	Delegasi yang mengontrol tata letak anak dalam
--------------	------------------------	--

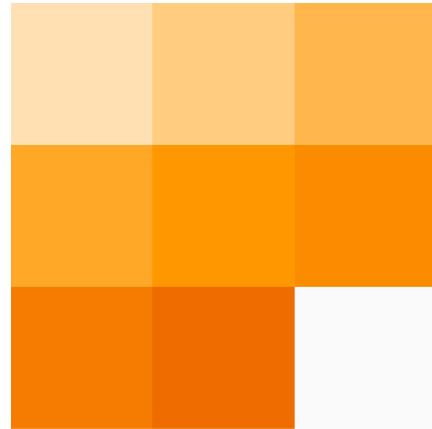
required

nullable

GridView.Count

Widget grid yang memiliki angka wajib untuk pembagian cros space atau horizontal space

```
@override  
Widget build(BuildContext context) {  
  return Scaffold(  
    body: GridView.count(  
      physics: const BouncingScrollPhysics(parent:  
        AlwaysScrollableScrollPhysics()),  
      crossAxisCount: 3,  
      children: [  
        Container(color: Colors.orange[100]),  
        Container(color: Colors.orange[200]),  
        Container(color: Colors.orange[300]),  
        Container(color: Colors.orange[400]),  
        Container(color: Colors.orange[500]),  
        Container(color: Colors.orange[600]),  
        Container(color: Colors.orange[700]),  
        Container(color: Colors.orange[800]),  
      ],  
    ),  
  );  
}
```



crossAxisCount	int	Angka wajib untuk pembagian cros space atau horizontal space
----------------	-----	--

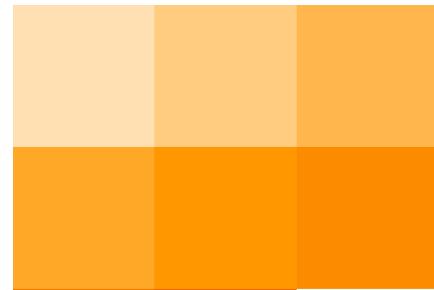
required

nullable

GridView.Builder

Widget grid yang memiliki angka wajib untuk pembagian cros space atau horizontal space

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: GridView.builder(
      physics: const BouncingScrollPhysics(parent:
        AlwaysScrollableScrollPhysics()),
      gridDelegate: const SliverGridDelegateWithMaxCrossAxisExtent(
        maxCrossAxisExtent: 200,
      ),
      itemCount: 8,
      itemBuilder: (context, index) {
        final indexColor = (index + 1) * 100;
        return Container(color: Colors.orange[indexColor]);
      },
    ),
  );
}
```



ItemCount	int	Banyak widget yang di return atau dibuild
itemBuilder	Widget? Function(Buil dContext, int)	Widget yang akan di return atau dibuid
gridDelegate	SliverGridDel egate	Delegasi yang mengontrol tata letak anak dalam

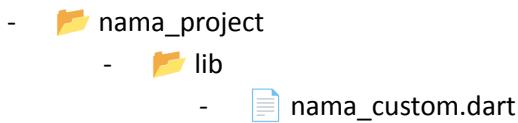
required

nullable

Custom Widget

Widget gabungan yang dibuat untuk mempermudah, menyeragamkan dan memperbagus user interface.

1. Buat file `nama_custom.dart` pada project



```
class BorderBox extends StatelessWidget {  
    BorderBox({  
        super.key,  
        this.height,  
        this.width,  
        this.margin,  
        this.padding,  
        this.color,  
        this.border,  
        this.borderRadius,  
        this.gradient,  
        this.boxShadow,  
        this.alignment,  
        this.child,  
    });  
  
    final double? height;  
    final double? width;  
    final EdgeInsetsGeometry? margin;  
    final EdgeInsetsGeometry? padding;  
    final Color? color;  
    BoxBorder? border;  
    final BorderRadiusGeometry? borderRadius;  
    final Gradient? gradient;  
    final List<BoxShadow>? boxShadow;  
    final AlignmentGeometry? alignment;  
    final Widget? child;  
  
    @override  
    Widget build(BuildContext context) {  
        border ??= Border.all(color: Colors.black, width: 0.5);  
  
        return Container(  
            height: height,  
            width: width,  
            margin: margin,  
            padding: padding,  
            decoration: BoxDecoration(  
                color: color,  
            ),  
        );  
    }  
}
```

```

border: border,
borderRadius: borderRadius,
gradient: gradient,
boxShadow: boxShadow,
),
alignment: alignment,
child: child,
);
}
}

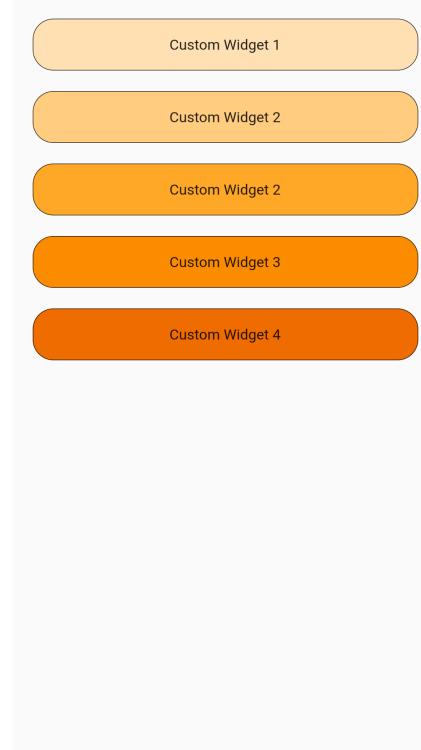
```

2. Gunakan custom widget sesuai kebutuhan

```

@Override
Widget build(BuildContext context) {
  return Scaffold(
    body: ListView(
      physics: const BouncingScrollPhysics(parent:
AlwaysScrollableScrollPhysics()),
      padding: const EdgeInsets.only(top: 20, left: 20, right: 20),
      children: [
        BorderBox(
          height: 50,
          margin: const EdgeInsets.only(bottom: 20),
          alignment: Alignment.center,
          color: Colors.orange[100],
          borderRadius: BorderRadius.circular(20),
          child: const Text("Custom Widget 1"),
        ),
        BorderBox(
          height: 50,
          margin: const EdgeInsets.only(bottom: 20),
          alignment: Alignment.center,
          color: Colors.orange[200],
          borderRadius: BorderRadius.circular(20),
          child: const Text("Custom Widget 2"),
        ),
        BorderBox(
          height: 50,
          margin: const EdgeInsets.only(bottom: 20),
          alignment: Alignment.center,
          color: Colors.orange[400],
          borderRadius: BorderRadius.circular(20),
          child: const Text("Custom Widget 2"),
        ),
        BorderBox(
          height: 50,

```



```

margin: const EdgeInsets.only(bottom: 20),
alignment: Alignment.center,
color: Colors.orange[600],
borderRadius: BorderRadius.circular(20),
child: const Text("Custom Widget 3"),
),
BorderBox(
height: 50,
margin: const EdgeInsets.only(bottom: 20),
alignment: Alignment.center,
color: Colors.orange[800],
borderRadius: BorderRadius.circular(20),
child: const Text("Custom Widget 4"),
),
],
),
);
}
}

```

itemCount	int	Banyak widget yang di return atau dibuild
itemBuilder	Widget? Function(Buil dContext, int)	Widget yang akan di return atau dibuid
gridDelegate	SliverGridDel egate	Delegasi yang mengontrol tata letak anak dalam

required

nullable

Custom widget juga dapat dikombinasikan dengan teknik refactoring, seperti [extract method](#), [extract variable](#), [extract widget](#), dll.

Navigation

Pada saat membangun sebuah aplikasi kita akan membuat banyak sekali screen dan kita akan berpindah dari satu screen ke screen lainnya. Pada Flutter kita akan menggunakan sebuah class bernama **Navigator**. Dengan **Navigator** ini kita akan berpindah dari satu screen ke screen lainnya. Berikut ini contohnya:

Codingan FirstScreen:

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('First Screen'),
      ),
      body: Center(
        child: ElevatedButton(
          child: const Text('Pindah Screen'),
          onPressed: () {
            Navigator.push(context, MaterialPageRoute(builder: (context) {
              return const SecondScreen();
            }));
          },
        ),
      );
    }
}
```

Perhatikan kode berikut di FirstScreen:

```
Navigator.push(context, MaterialPageRoute(builder: (context) {
  return const SecondScreen();
}));
```

Untuk berpindah ke screen kedua kita akan menggunakan sebuah method **Navigator.push**. Pada kode di atas **Navigator.push** memiliki dua parameter. Pertama ialah **context** dan yang kedua **Route**. Parameter **context** ini merupakan variabel **BuildContext** yang ada pada method build. Parameter **route** berguna untuk menentukan tujuan ke mana kita akan berpindah screen. Route tersebut kita isikan dengan **MaterialPageRoute** yang didalamnya terdapat builder yang nantinya akan diisi dengan tujuan screen-nya.

Codingan SecondScreen:

```
class SecondScreen extends StatelessWidget {  
  const SecondScreen({Key? key}) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: const Text('Second Screen'),  
      ),  
      body: Center(  
        child: OutlinedButton(  
          child: const Text('Kembali'),  
          onPressed: () {  
            Navigator.pop(context);  
          },  
        ),  
      ),  
    );  
  }  
}
```

Perhatikan code berikut pada SecondScreen:

```
Navigator.pop(context);
```

Setelah dapat berpindah ke screen lain maka kita dapat menggunakan `Navigator.pop` untuk kembali ke screen sebelumnya. Pada `Navigator.pop` kita hanya cukup menambahkan parameter `context` yang merupakan variabel dari method build.

Object

Class object pada dart flutter tidak jauh berbeda dengan java. Salah satu cara inisialisasi dapat dilakukan dengan menggunakan konstruktor.

- Buat file **model.dart**

- **nama_project**
 - **lib**
 - **main.dart**
 - **model.dart**

- Tuliskan code class pada **model.dart**

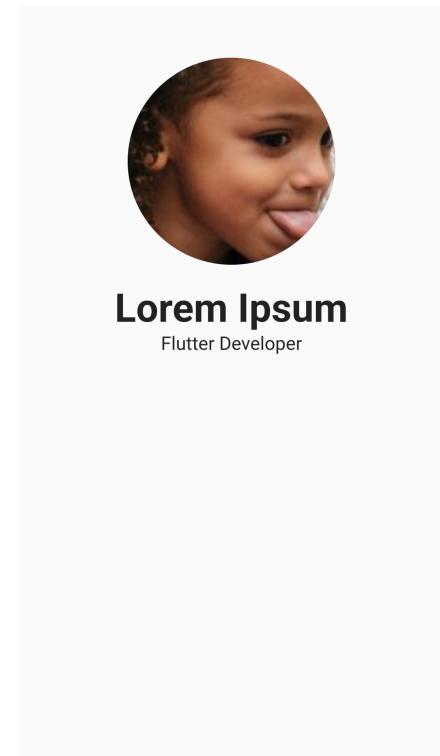
```
class User {
    final String username;
    final String title;
    final String image;

    const User({
        required this.username,
        required this.title,
        required this.image,
    });
}
```

- Selanjutnya model user sudah dapat dipakai

```
class _StatefulWidget extends State<StatefulWidget> {
    final user = const User(
        username: "Lorem Ipsum",
        title: "Flutter Developer",
        image: "https://i.pravatar.cc/300",
    );

    @override
    Widget build(BuildContext context) {
        return Scaffold(
            body: ListView(
                physics: const BouncingScrollPhysics(parent:
                    AlwaysScrollableScrollPhysics()),
                padding: const EdgeInsets.only(top: 50),
                children: [
                    Container(
                        margin: const EdgeInsets.only(bottom: 20),
                        child: CircleAvatar(
                            radius: 100,
                            backgroundImage: NetworkImage("${user.image}"),
                        ),
                    ),
                ],
            ),
        );
    }
}
```



```
        ),  
        ),  
        Container(  
            alignment: Alignment.center,  
            child: Text(  
                "${user.username}",  
                style: const TextStyle(  
                    fontSize: 38,  
                    fontWeight: FontWeight.bold,  
                ),  
                ),  
            ),  
        ),  
        Container(  
            margin: const EdgeInsets.only(bottom: 10),  
            alignment: Alignment.center,  
            child: Text(  
                "${user.title}",  
                style: const TextStyle(fontSize: 18),  
            ),  
            ),  
        ],  
    ),  
);  
}  
}  
}
```

user	User	Object yang digunakan perlu diinisialisasi terlebih dahulu sebelum dapat digunakan
------	------	--

required

nullable

Inisialisasi object pada contoh ini menggunakan metode manual, kedepan object akan diinisialisasi dengan metode parsing **JSON** dan metode **API**

Link Github

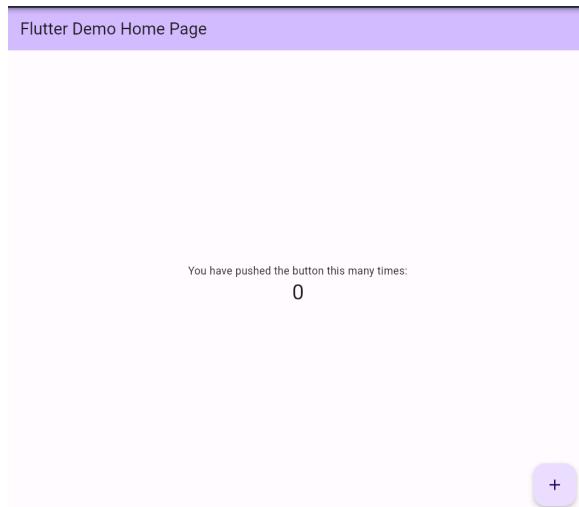
Berikut merupakan link github yang akan digunakan pada modul pemrograman mobile:

[Link Github](#)

KEGIATAN PRAKTIKUM

A. Instalasi Flutter

1. Lakukan instalasi Flutter dan software yang diperlukan seperti IDE di perangkat kalian masing-masing
2. Setelah melakukan instalasi, buatlah 1 project Flutter dan jalankan project tersebut
3. Ketika membuat project Flutter untuk pertama kali, biasanya sudah terdapat template yang sudah disediakan, jadi bisa langsung di jalankan saja. Ketika dijalankan hasil nya akan seperti berikut:



B. Merubah Tampilan

1. Setelah menjalankan project Flutter dan muncul tampilan seperti di atas, ubahlah tampilannya seperti warna, teks, icon, dan lain-lain. Selain itu, bisa juga menambahkan image atau menambahkan teks lagi di tengahnya. Jadi bisa dikreasikan sendiri.
2. Opsi tambahan selain merubah tampilan yaitu implementasikan Function seperti mengubah function untuk counter nya atau menambah suatu function baru, lalu Class atau halaman baru dengan menerapkan stateless atau statefull Widget.
3. Konfirmasi ke asisten dan jelaskan bagian mana saja yang diubah atau ditambahkan

RUBRIK PENILAIAN MODUL 1 MATERI

Bobot Penilaian Modul 1 Materi (20%)

Berhasil melakukan instalasi Flutter dan menjalankan project	60
Berhasil mengubah tampilan aplikasi template project	40
Total	100

