

VERSION 1.0

JUNE 28, 2024



PEMROGRAMAN MOBILE

Modul 0 - Setup Flutter Environment & Introducing Flutter

Tim Penyusun:

Ali Sofyan, S.kom, M.Kom

Ahmad Naufal
Marchanda Balqis

LAB. INFORMATIKA
UNIVERSITAS MUHAMMADIYAH MALANG

PEMROGRAMAN MOBILE

PERSIAPAN MATERI

Mahasiswa diharapkan mempelajari materi sebelum mengerjakan tugas, materi yang tercakup antara lain:

1. Setup Flutter Environment
2. Framework Flutter
3. Bahasa Pemrograman Dart
4. Android Studio

TUJUAN

Mahasiswa mampu untuk melakukan setup framework Flutter, membuat android emulator, membuat *project* awal Flutter, dan mengetahui struktur *project* Flutter.

TARGET MODUL

Mahasiswa mampu memahami:

1. Melakukan setup Flutter Environment
2. Apa itu Framework Flutter
3. Membuat android emulator
4. Membuat Project Awal
5. Memahami Struktur Flutter

PERSIAPAN SOFTWARE/APLIKASI

1. Laptop/PC
2. IDE Android Studio/ Visual Studio Code
3. Flutter SDK: [Flutter 3.16.9](#)

REFERENSI MATERI

1. [Flutter Fundamental Playlist](#)
2. [Flutter Basic Widgets](#)
3. [Write First Flutter App](#)
4. [Bahasa Pemrograman Dart](#)

MATERI POKOK

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, pengembangan 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 (seperti python), sehingga menghasilkan performa yang cepat dan responsif. Flutter juga menggunakan teknologi bernama "Skia" untuk menggambarkan antarmuka pengguna, memberikan kecepatan dan efisiensi visual yang tinggi

5. Hot Reload:

Fitur "Hot Reload" di Flutter memungkinkan pengembangan 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 package dan library tambahan untuk memperluas fungsionalitas Flutter. Komunitas ini juga memberikan dukungan dan number daya yang berharga bagi pengembang baru.

DART

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 point 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. Asynchron 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.

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. Kalian bisa membaca dokumentasi [install flutter](#).

- **Requirement**

Flutter memiliki minimum dan recommended requirement yang harus kalian penuhi dan bisa kalian liat pada gambar di bawah ini.

Hardware requirements

Your Windows Flutter development environment must meet the following minimal hardware requirements.

Requirement	Minimum	Recommended
x86_64 CPU Cores	4	8
Memory in GB	8	16
Display resolution in pixels	WXGA (1366 x 768)	FHD (1920 x 1080)
Free disk space in GB	11.0	60.0

Jika, kalian memiliki spek laptop/PC yang tidak mencukupi, teman-teman tidak perlu risau karena kalian bisa menggunakan text editor online bernama [Project IDX](#). Platform yang di buat oleh google berbasis cloud yang menyediakan template mulai dari Web App, Backend, Mobile, AI & ML, Database, hingga Misc.

- **Project IDX**

Jika kalian menggunakan project IDX, kalian tidak perlu melakukan instalasi apapun dan cukup mengunjungi [Project IDX](#) dan melakukan registrasi akun. Untuk membuat project awal Flutter menggunakan **Project IDX** bisa kalian lihat pada video tutorial dibawah ini:

FLUTTER SETUP for PROJECT IDX

- **Windows**

Silahkan untuk menonton video instalasi Flutter dan ikuti langkah-langkahnya. Pastikan untuk mendownload [Flutter 3.16.9](#) . Jika saat melakukan instalasi terdapat kendala atau error, sangat di anjurkan untuk menanyakannya di grup praktikum angkatan 2022.

FLUTTER INSTALLATION for WINDOWS

- **MacOS**

Untuk teman teman yang menggunakan macOS, sangat di anjurkan untuk mendownload [Flutter 3.16.9](#) dan mengikuti tutorial dari video di bawah ini:

FLUTTER INSTALLATION for MAC

- **Create Flutter Application**

Untuk membuat Flutter project application, terdapat berbagai macam cara. Namun, pada modul ini akan memberitahukan 3 cara bagaimana membuat Flutter project application menggunakan Android Studio, Command Prompt, dan VS-Code. Kalian bisa menonton video tutorial di bawah ini untuk mengetahui bagaimana cara membuat sebuah **Project Flutter**.

[CREATE FLUTTER FIRST APP](#)

DART SYNTAX

Sisi positif dari bahasa pemrograman Dart salah satunya adalah sintaksis seperti variable dan operator yang digunakan tidak jauh berbeda dengan bahasa pemrograman lainnya.

- **Variabel**

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

```
var message = "Hello World"
===== atau =====
void main () {
  var message = "Hello World"
  print(message);
}
```

Selain mendefinisikan variable dengan **var**, Dart juga memiliki 4 jenis variables. Diantaranya yaitu:

1) Null Safety

Null safety memastikan bahwa variabel tidak boleh null kecuali secara eksplisit dinyatakan bisa null. Ini membantu mengurangi bug yang disebabkan oleh null reference.

```
int? nullableInt; // nullable, bisa diisi dengan nilai null, di tandai dengan tanda tanya '?'
int nonNullableInt = 10; // non-nullable, tidak bisa diisi dengan nilai null
void main () {
  nullableInt = null; // valid
  nonNullableInt = null; // error
}
```

2) Default Value

Dalam Dart, variabel yang tidak diinisialisasi secara **otomatis** memiliki nilai **default null**. Untuk variabel yang memiliki tipe **non-nullable**, kita harus menginisialisasinya sebelum digunakan.

```
void main () {
    int? nullableInt; // default value adalah null
    int nonNullableInt = 10; // default value nya adalah 10, karena di inisialisasikan 10
    print(nullableInt); // prints null
}
```

3) Late Variables

Variabel late memungkinkan deklarasi variabel non-nullable tanpa harus langsung diinisialisasi, tetapi menjamin bahwa variabel tersebut akan diinisialisasi sebelum digunakan.

```
late String description;
void main () {
    description = 'This is a late variable'
    print(description); // prints 'This is a late variable'
}
```

4) Final and Const

- final variabel hanya dapat diinisialisasi sekali, tetapi nilainya bisa ditentukan saat runtime.
- const variabel bersifat compile-time constant, artinya nilai tersebut harus sudah diketahui pada waktu kompilasi.

```
late String description;
void main () {
    final currentTime = DateTime.now(); // can be set only once at runtime
    const pi = 3.14; // must be a compile-time constant
    // currentTime = DateTime.now(); // error: final variable can only be set once
    // pi = 3.14159; // error: const variable cannot be reassigned
    print(currentTime);
    print(pi);
}
```

Setiap jenis variabel ini memiliki kegunaannya masing-masing dan membantu dalam penulisan kode Dart yang aman dan efisien. Untuk lebih lengkapnya kalian bisa membaca dokumentasi [Dart Variables](#).

- **Data Type**

Dart memiliki beberapa tipe bawaan (built-in types) yang digunakan untuk berbagai kebutuhan pemrograman. Berikut adalah penjelasan singkat tentang tipe-tipe tersebut:

Number	<ul style="list-style-type: none"> • int: Tipe data integer untuk bilangan bulat. • double: Tipe data floating-point untuk bilangan desimal.
String	Tipe data untuk teks atau urutan karakter. Didefinisikan dengan tanda kutip tunggal atau ganda.
Boolean	Tipe data untuk nilai benar atau salah.
List	Koleksi yang terurut dari elemen yang dapat diakses menggunakan indeks.
Sets	Koleksi yang tidak terurut dari elemen unik.
Map	Koleksi pasangan kunci-nilai. Kunci harus unik.
Runes	Tipe data untuk merepresentasikan karakter Unicode. String di Dart adalah urutan unit kode UTF-16, dan Rune memungkinkan penggunaan karakter Unicode.
Symbols	Tipe data untuk merepresentasikan nama program yang tidak ambigu, biasanya digunakan dalam refleksi.

Setiap tipe bawaan ini memiliki karakteristik dan penggunaan yang spesifik, memungkinkan Dart untuk menangani berbagai jenis data dengan efisien dan aman. Agar lebih memahami tipe data yang ada pada bahasa Dart kalian bisa membaca dokumentasi [Dart Data Types](#).

- **Operators**

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

Operator Aritmatika		Operator Perbandingan	
Operator	Deskripsi	Operator	Deskripsi
+	Penjumlahan	==	Sama dengan
-	Pengurangan	!=	Tidak sama dengan
*	Perkalian	>	Lebih dari
/	Pembagian	<	Kurang dari
~/	Pembagian, mengembalikan nilai int	>=	Lebih dari sama dengan
%	Modulos atau sisa hasil bagi	<=	Kurang dari sama dengan

Operator Logika	
Operator	Deskripsi
	OR
&&	AND
!	NOT

- **Functions**

Penggunaan function pada dart flutter dapat dilakukan dengan beberapa cara [function](#). Mulai dari **main function**, **type function & type parameters**, **output function**, dan **async await function**.

- 1) **Main Function**

Main Function adalah titik awal dari sebuah aplikasi Dart atau Flutter. Ketika aplikasi dijalankan, eksekusi dimulai dari fungsi **main**.

- 2) **Type Function & Type Parameters**

Fungsi **tipe** dan **parameter** tipe digunakan untuk mendefinisikan fungsi yang bekerja dengan tipe data generik. Ini memungkinkan pembuatan fungsi yang lebih fleksibel dan dapat digunakan dengan berbagai tipe data.

- 3) **Output Function**

Fungsi **output** adalah fungsi yang **mengembalikan nilai**. Dalam Dart, setiap fungsi yang mengembalikan nilai memiliki tipe kembalian yang ditentukan.

- 4) **Async await function**

Fungsi **async** dan **await** digunakan untuk menangani operasi asynchronous dalam Dart dan Flutter. Fungsi yang dideklarasikan dengan **async** akan selalu mengembalikan Future.

Untuk lebih memahami fungsi-fungsi apa saja yang ada di Dart dan bagaimana contoh penulisannya, teman teman bisa membaca dokumentasi [function](#) atau [Function Flutter](#).

- **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
```

WIDGET

Widget adalah elemen dasar dalam Flutter yang digunakan untuk membangun antarmuka pengguna (UI). Setiap elemen visual, seperti teks, tombol, gambar, dan layout, adalah sebuah widget. Flutter menyediakan berbagai widget bawaan yang dapat dikombinasikan dan disesuaikan untuk membuat UI yang kompleks dan interaktif. Untuk penjelasan yang lebih detail dan syntax code nya teman teman bisa lihat di [Widget Flutter Docs](#) atau [Widget catalog](#)

Stateless Widget

[Stateless Widget](#) digunakan untuk mendeskripsikan bagian-bagian User Interface yang hanya menampilkan informasi konfigurasi object dan BuildContext. Stateless Widget adalah jenis widget di Flutter yang tidak berubah (immutable). Artinya, begitu widget ini dibuat, tampilannya tidak akan berubah meskipun ada perubahan data.

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Stateless Widget',
      debugShowCheckedModeBanner: false,
      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",
          style: const TextStyle(fontSize: 24),
        ),
      ),
      floatingActionButton: FloatingActionButton(
        backgroundColor: Colors.blue,
        child: const Icon(Icons.add),
        onPressed: () {
          //data tidak akan berubah
          dataStateless += 1;
        },
      ),
    );
  }
}
```

Stateful Widget

Stateful Widget digunakan untuk mendeskripsikan bagian-bagian User Interface yang dapat berubah. Stateful Widget mempunyai sifat dinamis, oleh karena itu data dapat berubah dengan adanya interaksi user atau ketika terjadi perubahan data. Untuk mengubah data dapat dilakukan dengan memanggil `setState(() {});`

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

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

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

  int dataStateful;

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

class _ StatefulWidget extends State< StatefulWidget > {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Text(
          "Data Stateful = ${widget.dataStateful}",
          style: const TextStyle(fontSize: 24),
        ),
      ),
      floatingActionButton: FloatingActionButton(
        backgroundColor: Colors.blue,
        child: const Icon(Icons.add),
        onPressed: () {
          //data akan berubah
          widget.dataStateful += 1;
          setState(() {});
        },
      ),
    );
  }
}
```

MaterialApp

[MaterialApp](#) merupakan widget pembungkus material widget lainnya dan membuat aplikasi menjadi interaktif mengikuti panduan Material Design [geeksforgeeks-materialapp](#).

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData.light(),
      themeMode: ThemeMode.system,
      darkTheme: ThemeData.dark(),
      home: const MyHomePage(title: 'Home Page'),
      debugShowCheckedModeBanner: false,
    );
  }
}
```

title	string	Deskripsi untuk identifikasi aplikasi
theme	ThemeData?	Untuk setup light theme mode
themeMode	ThemeData?	Setup yang dipakai aplikasi saat: light (terang), dark (gelap), system (mengikuti sistem)
darkTheme	ThemeData?	Untuk setup dark theme mode

required nullable

Scaffold

[Scaffold](#) merupakan widget design material dasar untuk memvisualisasi tata letak serta menampilkan drawers dan bottom sheets.

```
class _MyHomePageState extends State<MyHomePage> {
  int _count = 0;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            const Text(
              'You have pushed the button this many times:',
            ),
            Text(

```

```

        '$_count',
        style: Theme.of(context).textTheme.headlineMedium,
    ),
],
),
),
floatingActionButton: FloatingActionButton(
    onPressed: () => setState(() => _count++),
    child: const Icon(Icons.add),
),
);
}
}

```

appBar	PreferredSizeWidget?	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

required nullable

Column

[Column](#) merupakan widget yang digunakan untuk mengatur dan menampilkan children secara vertikal array. Column menampilkan widget children secara berurutan dari atas ke bawah.

```

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

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

```

```
    );
}
```

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

required nullable

Row

[Row](#) merupakan widget yang digunakan untuk mengatur dan menampilkan children secara horizontal array. Row menampilkan widget children secara berurutan dari kiri ke kanan.

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

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

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

required nullable

Container

[Container](#) merupakan convenience widget yang dapat mengkombinasikan pewarnaan, penentuan posisi, ukuran, dan bentuk widget [geeksforgeeks-container](#).

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

  final 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,
              style: TextStyle(fontSize: 16.0),
            ),
          ),
        ],
      ),
    );
  }
}
```

child	Widget?	Widget di dalam container
width	Double?	Setup ukuran lebar container
height	Double?	Setup ukuran tinggi container
alignment	AlignmentGeometry?	Align specific pada container: topLeft, topCenter, topRight, centerLeft, center, centerRight, bottomLeft, bottomCenter, bottomRight

color	Color?	Warna untuk container yang sudah memiliki ukuran
required	nullable	

Elevated**Button**

[Elevated button](#) merupakan widget button yang memiliki text dan style serta komponen pendukung lainnya.

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            ElevatedButton(
              onPressed: null,
              child: Text('Disabled'),
            ),
            ElevatedButton(
              onPressed: () {},
              child: const Text('Enabled'),
            ),
            ElevatedButton(
              style: ElevatedButton.styleFrom(
                foregroundColor: Colors.black, backgroundColor: Colors.green, elevation: 10,
                padding: const EdgeInsets.symmetric(
                  horizontal: 18,
                  vertical: 20,
                ),
              ),
              onPressed: () {},
              child: const Text('Custom'),
            ),
          ],
        ),
      );
  }
}
```

onPressed	voidFunction()	Melakukan aksi ketika tombol diketuk atau digunakan
style	ButtonStyle?	Style yang digunakan untuk tombol
child	Widget?	Widget di dalam elevatedButton

required nullable

Setiap button style memiliki banyak atribut yang dapat digunakan, beberapa diantaranya seperti pada tabel berikut

elevation	double?	Digunakan untuk menentukan seberapa tinggi bayangan widget tersebut dari permukaan di bawahnya.
foregroundColor	Color?	Warna effect saat button ditekan
backgroundColor	Color?	Warna background button
shadowColor	Color?	Warna shadow button
padding	EdgeInsetsGeometry?	Ruang kosong pada button

required nullable

Adapun beberapa [button style](#) yang dapat digunakan untuk button widget.

Type	Implementasi Flutter
Elevated	ElevatedButton
Filled	FilledButton
Filled Tonal	FilledButton.tonal
Outlined	OutlinedButton

Image

[Image](#) merupakan widget yang digunakan untuk menampilkan gambar dengan beberapa metode penggunaan berdasarkan format inputnya.

Network

[Image network](#) merupakan penggunaan widget image dengan menggunakan inputan url gambar dari internet.

```

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SizedBox(
        width: double.infinity,
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          crossAxisAlignment: CrossAxisAlignment.center,
          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://shorturl.at/xMMT4",
                fit: BoxFit.cover,
              ),
            ),
            ],
          ),
        );
      );
}

```

Image network



fit	BoxFit?	Digunakan untuk memastikan gambar menyesuaikan dengan ukuran kotak yang ditentukan tanpa mengubah rasio aspeknya
-----	---------	--

required nullable

Asset

[Image asset](#) merupakan penggunaan widget image dengan menggunakan inputan path directory gambar pada project. Untuk menggunakan widget ini ada beberapa setup yang harus dilakukan terlebih dahulu.

1. Taruh image pada sebuah folder dalam project
 - nama_project
 - assets
 - images
 - foto.jpg

2. Buka file `pubspec.yaml` pada project

- nama_project
 - assets
 - pubspec.yaml

3. Tambahkan path direktori gambar pada `pubspec.yaml`. Ubah dari code sebelum (kiri) menjadi sesudah (kanan) lalu save.

flutter:

```
# 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
```

flutter:

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

4. Assets pada images sudah dapat digunakan

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SizedBox(
        width: double.infinity,
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          crossAxisAlignment: CrossAxisAlignment.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(
                "lib/assets/images/foto.jpg",
                fit: BoxFit.cover,
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

Image asset



```
    );
}
```

fit	BoxFit?	Digunakan untuk memastikan gambar menyesuaikan dengan ukuran kotak yang ditentukan tanpa mengubah rasio aspeknya
-----	---------	--

required nullable

Image Picker

[Image picker](#) merupakan penggunaan widget image dengan menggunakan [file image](#) yang ada pada smartphone. Untuk menggunakan widget ini kita harus menginstall sebuah library terlebih dahulu.

1. Buka file `pubspec.yaml` pada project flutter
 - nama_project
 - pubspec.yaml
2. Tambahkan dependensi `image_picker` pada `pubspec.yaml` lalu save

```
dependencies:
  flutter:
    sdk: flutter

  image_picker: ^1.0.1
```

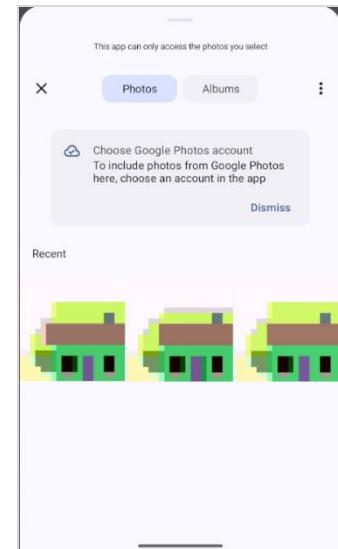
3. Buka terminal lalu jalankan perintah `flutter pub get` untuk menginstall library
4. Setelah library terinstall, maka image sudah bisa diambil dari smartphone

```
class _MyHomePageState extends State<MyHomePage> {
  File? image;

  Future getImage() async {
    final ImagePicker picker = ImagePicker();
    final XFile? imageX = await picker.pickImage(
      source: ImageSource.gallery);
    if (imageX == null) return;
    setState(() {
      image = File(imageX.path);
    });
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SizedBox(
        width: double.infinity,
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          crossAxisAlignment: CrossAxisAlignment.center,
          children: [
            Container(
              margin: const EdgeInsets.only(bottom: 15),
              child: const Text(
```



```
        "Image file",
        style: TextStyle(fontSize: 30),
    ),
),
if (image != null)
    Container(
        height: 350,
        width: 350,
        margin: const EdgeInsets.only(bottom: 25),
        child: Image.file(
            image!,
            fit: BoxFit.cover,
        ),
    )
else
    Container(
        height: 350,
        width: 350,
        padding: const EdgeInsets.all(100),
        margin: const EdgeInsets.only(bottom: 25),
        child: const CircularProgressIndicator(),
    ),
ElevatedButton(
    style: ElevatedButton.styleFrom(),
    onPressed: () async {
        await getImage();
    },
    child: const Text(
        'Pick Image',
        style: TextStyle(fontSize: 20),
    ),
),
],
),
),
);
}
}
```



<pre>if (image != null)... //widget) else ... (//widget ,</pre>	List<Widget>	Percabangan pada List<Widget> atau children
getImage()	Future<dynamic>	Function yang dibuat untuk mengambil image dari smartphone dan memperbarui state dengan gambar yang dipilih.
image	File?	Menyimpan file image setelah diambil dari smartphone
picker	ImagePicker	Library flutter.dev untuk mengambil image atau aset dari smartphone

CircularProgressIndicator

Widget

Untuk menampilkan loading proses

required nullable

Text

[Text](#) merupakan widget yang digunakan untuk menampilkan teks string dengan menerapkan style. Untuk mengubah `fontFamily`, harus terdapat data font yang dapat diakses. Salah satu caranya adalah dengan menambahkan data font ke folder assets pada project.

1. Tambahkan font yang akan digunakan ke dalam folder `assets/fonts`
 - nama_project
 - assets
 - fonts
 - A font.ttf
2. Buka file `pubspec.yaml` pada project
 - nama_project
 - pubspec.yaml
3. Tambahkan path direktori font pada `pubspec.yaml` lalu save

```
fonts:
  -family:
    fonts: flutter
      -assets: assets/fonts/Thesignature.ttf
```

4. Setelah itu font yang ditambahkan dapat digunakan sebagai fontFamily

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

  @override
  _StatefulWidgetState createState() => _StatefulWidgetState();
}

class _StatefulWidgetState extends State< 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: "TheSignature",  
    ),  
    ),  
    );  
}  
}
```

maxLines	int?	Menentukan jumlah maksimal baris teks yang akan ditampilkan
textAlign	TextAlign?	Mengatur perataan teks di dalam widget `Text`: <code>TextAlign.left</code> , <code>TextAlign.right</code> , <code>TextAlign.center</code> , <code>TextAlign.justify</code> , dll.
fontSize	double?	Mengatur ukuran font dari teks
color	Color?	Warna yang digunakan untuk teks
decoration	TextDecoration?	Mengatur dekorasi teks seperti garis bawah, garis di atas teks, atau garis yang melewati teks.
backgroundColor	Color?	Warna latar belakang teks
fontWeight	FontWeight	Ketebalan teks: <code>FontWeight.w100</code> , <code>FontWeight.w200</code> , <code>FontWeight.w300</code> , <code>FontWeight.w400</code> , <code>FontWeight.w500</code> , <code>FontWeight.w600</code> , <code>FontWeight.w700</code> , <code>FontWeight.w800</code> , <code>FontWeight.w900</code> , <code>FontWeight.bold</code>

required **nullable**

Grid

Grid adalah widget yang digunakan untuk menata children atau daftar widget agar tersusun dalam bentuk grid (tabel) yang teratur, baik secara horizontal maupun vertikal.

GridView

[GridView](#) adalah widget yang digunakan untuk menampilkan daftar item dalam tata letak grid yang dapat digulir (scrollable) secara rapi dalam bentuk grid dengan baris dan kolom yang teratur.

```
class MyHomePage extends StatelessWidget {  
  const MyHomePage({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: const Text('GridView Example'),  
      ),  
      body: GridView.builder(  
        padding: const EdgeInsets.all(10),  
        gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(  
          crossAxisCount: 2,  
          mainAxisExtent: 150,  
        ),  
        itemBuilder: (context, index) {  
          return Container(  
            color: Colors.amber[100 * (index % 10)],  
            child: Center(  
              child: Text('Item $index'),  
            ),  
          );  
        },  
      ),  
    );  
  }  
}
```

```
crossAxisCount: 2,  
crossAxisSpacing: 10,  
mainAxisSpacing: 10,  
,  
itemCount: 20,  
itemBuilder: (context, index) {  
    return Container(  
        decoration: BoxDecoration(  
            color: Colors.blue[100 * (index % 9 + 1)],  
            borderRadius: BorderRadius.circular(10),  
        ),  
        child: Center(  
            child: Text(  
                'Item $index',  
                style: const TextStyle(fontSize: 18, color: Colors.white),  
            ),  
        ),  
    );  
},  
);  
};  
}  
}
```

gridDelegate SliverGridDelegate Menentukan tata letak grid

required nullable

GridView.Count

Widget grid untuk membuat tata letak grid dengan jumlah kolom tetap yang ditentukan.

```
@override
Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
            title: const Text('GridView Example'),
        ),
        body: GridView.builder(
            padding: const EdgeInsets.all(10),
            gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
                crossAxisCount: 2,
                crossAxisSpacing: 10,
                mainAxisSpacing: 10,
            ),
            itemCount: 20,
            itemBuilder: (context, index) {
                return Container(
                    decoration: BoxDecoration(
                        color: Colors.blue[100 * (index % 9 + 1)],
                        borderRadius: BorderRadius.circular(10),
                    ),
                    child: Center(
                        child: Text(
                            'Item $index',
                            style: const TextStyle(fontSize: 18, color: Colors.white),
                        ),
                    ),
                );
            },
        );
    }
}
```

crossAxisCount int Menentukan jumlah kolom dalam grid

required nullable

GridView.Builder

Widget grid untuk membuat tata letak grid yang dapat digulir (scrollable).

```
@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	Menentukan jumlah item dalam grid
itemBuilder	Widget?Function(BuildContext, int)	Fungsi yang build setiap item dalam grid
gridDelegate	SliverGridDelegate	Menentukan tata letak grid

required nullable

Custom Widget

Custom widget adalah widget gabungan yang dibuat untuk mempermudah, menyeragamkan, dan memperindah user interface. Selain itu, custom widget dapat dikombinasikan dengan teknik refactoring seperti extract method, extract variable, extract widget, dll untuk maintainability code.

- Buat file `widget_custom.dart` di dalam direktori 'lib'

- nama_project
- lib
- widget_custom.dart

```

class BorderBox extends StatelessWidget {
  const 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;
  final BoxBorder? border;
  final BorderRadiusGeometry? borderRadius;
  final Gradient? gradient;
  final List<BoxShadow>? boxShadow;
  final AlignmentGeometry? alignment;
  final Widget? child;
}

```

```

@Override
Widget build(BuildContext context) {
    return Container(
        height: height,
        width: width,
        margin: margin,
        padding: padding,
        decoration: BoxDecoration(
            color: color,
            border: border ?? Border.all(color: Colors.black, width: 0.5),
            borderRadius: borderRadius,
            gradient: gradient,
            boxShadow: boxShadow,
        ),
        alignment: alignment,
        child: child,
    );
}
}

```

2. Di dalam file `widget_custom.dart`, definisikan custom widget sesuai kebutuhan. Bisa menggunakan ‘**StatelessWidget**’ atau ‘ **StatefulWidget**’ tergantung apakah widget memerlukan perubahan state atau tidak.

```

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

    @override
    Widget build(BuildContext context) {
        return Scaffold(
            appBar: AppBar(
                title: const Text('Custom Widget Example'),
            ),
            body: Center(
                child: BorderBox(
                    height: 150,
                    width: 150,
                    margin: const EdgeInsets.all(20),
                    padding: const EdgeInsets.all(10),
                    color: Colors.blue[100],
                    border: Border.all(color: Colors.blue, width: 2),
                    borderRadius: BorderRadius.circular(15),
                    boxShadow: [
                        BoxShadow(
                            color: Colors.black.withOpacity(0.1),
                            spreadRadius: 5,
                            blurRadius: 7,
                            offset: const Offset(0, 3),
                        ),
                    ],
                    alignment: Alignment.center,
                    child: const Text(
                        'Custom BorderBox',
                        style: TextStyle(fontSize: 18, color: Colors.blue),
                        textAlign: TextAlign.center,
                    ),
                ),
            ),
        );
    }
}

```

```

        ),
        ),
        );
    }
}

```

itemCount	int	Menentukan jumlah item dalam grid
itemBuilder	Widget?Function(BuildContext, int)	Fungsi yang build setiap item dalam grid
gridDelegate	SliverGridDelegate	Menentukan tata letak grid

required nullable

Navigation

Navigator digunakan untuk berpindah antar screen. Berikut contoh implementasi navigator:

Contoh code 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();
            }));
          },
        ),
      );
    }
}

```

Untuk berpindah ke screen kedua, kita akan menggunakan method ‘Navigator.push’. ‘Navigator.push’ memiliki dua parameter yaitu ‘context’ dan ‘route’.

- **context**: Variabel **BuildContext** yang ada pada method **build**.
- **route**: Menentukan tujuan screen yang akan kita tuju. Diisi dengan **MaterialPageRoute** yang didalamnya terdapat **builder** yang nantinya akan diisi dengan tujuan screen.

Contoh code 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);
          },
        ),
      ),
    );
  }
}
```

Pada SecondScreen, kita dapat menggunakan `Navigator.pop` untuk kembali ke screen sebelumnya. `Navigator.pop` hanya membutuhkan satu parameter yaitu **context**.

- **Navigator.pop**: Digunakan untuk kembali ke screen sebelumnya. Hanya membutuhkan satu parameter yaitu **context**.

Object

Class object pada Dart Flutter tidak jauh berbeda dengan Java. Salah satu cara inisialisasi dapat dilakukan dengan menggunakan konstruktor.

1. Buat file model.dart

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

2. Tuliskan code class pada model.dart dengan menggunakan konstruktor

```
class User {
  String name;
  int age;
  String email;

  // Konstruktor
  User({required this.name, required this.age, required this.email});
}
```

3. model user sudah dapat digunakan

```

@Override
Widget build(BuildContext context) {
    // Membuat instance dari kelas User
    User user = User(name: "Dut", age: 30, email: "dut@gmail.com");

    return Scaffold(
        appBar: AppBar(
            title: const Text('User Info'),
        ),
        body: Center(
            child: Padding(
                padding: const EdgeInsets.all(16.0),
                child: Column(
                    mainAxisAlignment: MainAxisAlignment.center,
                    children: [
                        Text(
                            'Name: ${user.name}',
                            style: const TextStyle(fontSize: 24),
                        ),
                        const SizedBox(height: 10),
                        Text(
                            'Age: ${user.age}',
                            style: const TextStyle(fontSize: 24),
                        ),
                        const SizedBox(height: 10),
                        Text(
                            'Email: ${user.email}',
                            style: const TextStyle(fontSize: 24),
                        ),
                    ],
                ),
            ),
        );
    }
}

```

user **User** Object yang digunakan perlu diinisialisasi terlebih dahulu sebelum digunakan

required **nullable**

CODE LAB

TUGAS 1

1. Lakukan instalasi Flutter dan Software yang diperlukan seperti IDE di perangkat kalian masing-masing.
2. Buatlah 1 project Flutter dan jalankan project tersebut
3. Setelah menjalankan project Flutter dan muncul tampilannya, ubahlah tampilan tersebut seperti warna, teks, icon, dan lain-lain. Selain itu bisa juga menambahkan image atau menambahkan teks.

KRITERIA & DETAIL PENILAIAN

Modul 0 ini tidak ada masuk ke dalam penilaian, **tapi sangat di anjurkan untuk praktikan ikuti dan pahami.**