Kursus Flutter

Fahri Firdausillah, Syaifur Rohman, dkk

2023-03-07

Daftar Isi

Selamat Datang

Ini adalah buku yang membahas tentang Flutter. Susunan dari buku ini adalah sebagai berikut.

Flutter Dasar

1. **Pengenalan Flutter**
    1. Apa itu Flutter
    2. Instalasi Flutter
    3. Membuat proyek Flutter pertama
2. **Dasar Pemrograman Dart**
    1. Variabel dan Tipe Data
    2. Struktur Kontrol
    3. Fungsi dan Kelas
    4. Penggunaan Pustaka
3. **User Interface**
    1. Widget
    2. Layout
    3. Styling
    4. Input Handling (Stateless & Statefull)
4. **Proyek Pertama: Aplikasi Kalkulator**
5. **Routing dan Navigasi**
    1. Navigasi antar halaman
    2. Widget navigator
    3. Mengirim data antar halaman
    4. State Management
6. **Pengujian dan Debugging**
    1. Debugging di Flutter
    2. Menggunakan widget test untuk uji widget
    3. Menggunakan unit test untuk uji logika
7. **Akses Data Lokal**
    1. Konfigurasi dan pengaturan awal

2. CRUD data dengan SQLite
3. Query pencarian database lokal

8. **Proyek Kedua: Aplikasi Catatan**

Flutter Client Server

9. **Akses Data Melalui API**

    1. Konsep dasar REST API
    2. Menggunakan package HTTP
    3. Mengambil data dari API
    4. Menampilkan data pada widget

10. **Integrasi dengan Firebase/Supabase**

    1. Pengenalan Firebase/Supabase
    2. Integrasi Firebase/Supabase dengan Flutter
    3. Penggunaan Firestore untuk penyimpanan data

11. **Layanan Berbasis Lokasi**

    1. Menggunakan package Location
    2. Menggunakan widget Geolocator
    3. Menampilkan openstreetmap API pada aplikasi

12. **Pengamanan Aplikasi**

    1. Penggunaan protokol jaringan aman dengan HTTPS
    2. Autentikasi pengguna
    3. Obfuscating Dart code

13. **Proyek Ketiga: Lapor Book**

1 Pengenalan Flutter

Pengenalan Flutter

Flutter adalah framework yang sangat keren yang akan kita pelajari di sini.

1.1 Apa itu Flutter?

Flutter adalah sebuah SDK (Software Development Kit) open source yang dikembangkan oleh Google. Tujuan utamanya adalah untuk memudahkan pembuatan aplikasi yang dapat berjalan di berbagai platform, atau yang biasa disebut sebagai multi-platform. Dengan Flutter, pengembang dapat membangun aplikasi untuk sistem operasi Android, iOS, Web, Windows, Linux, dan MacOS dengan menggunakan kode yang sama, tanpa perlu menulis ulang kode secara terpisah. Selain itu, Flutter juga dapat digunakan baik sebagai bagian dari aplikasi native yang sudah ada maupun sebagai dasar pembangunan aplikasi baru.

1.2 Kelebihan dan kekurangan Flutter

1.2.1 kelebihan dari Flutter, antara lain:

1. Flutter memungkinkan pembuatan aplikasi dengan desain yang indah dan kreatif. Framework ini memberikan kebebasan dalam mengatur desain aplikasi tanpa banyak batasan. Dengan kemampuannya untuk mengontrol setiap piksel di layar, Flutter memudahkan pembuatan animasi yang menarik. Selain itu, Flutter menyediakan beragam komponen material design yang dapat digunakan dengan baik pada platform Android, iOS, dan web.
2. Flutter memiliki kinerja yang sangat cepat. Flutter menggunakan mesin grafis bernama Skia-2D, yang juga digunakan oleh Chrome dan Android. Selain itu, kode Flutter ditulis dalam bahasa Dart, yang memungkinkan kompilasi ke kode native 32-bit dan 64-bit ARM untuk iOS dan Android. Dengan demikian, aplikasi Flutter dapat berjalan dengan kinerja tinggi dan responsif.
3. Flutter sangat produktif dalam pengembangan aplikasi. Fitur hot-reload yang dimilikinya memungkinkan pengembang untuk melihat perubahan kode secara langsung dan secara real-time. Dengan hot-reload, perubahan yang dilakukan pada kode dapat segera diterapkan pada perangkat tanpa perlu menunggu proses restart dan tanpa kehilangan state aplikasi.

4. Flutter adalah proyek open source yang bersifat terbuka. Dengan lisensi BSD, Flutter mendorong partisipasi dari komunitas pengembang di seluruh dunia. Banyak kontributor telah berkontribusi dalam pengembangan Flutter, dan ada banyak plugin yang telah dibuat oleh para pengembang.

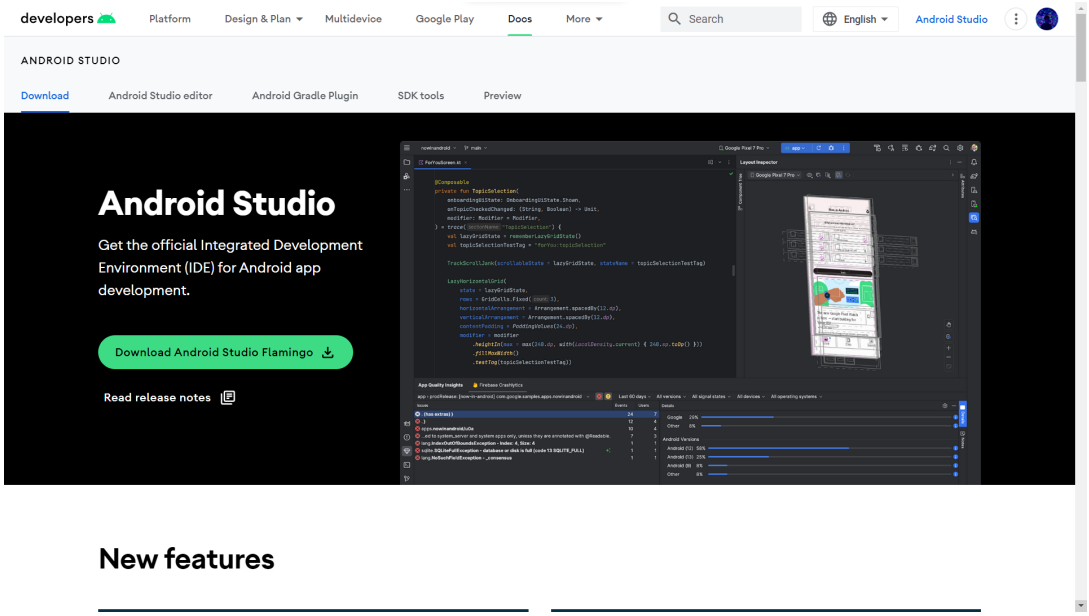
1.2.2 Kekurangan dari Flutter, antara lain:

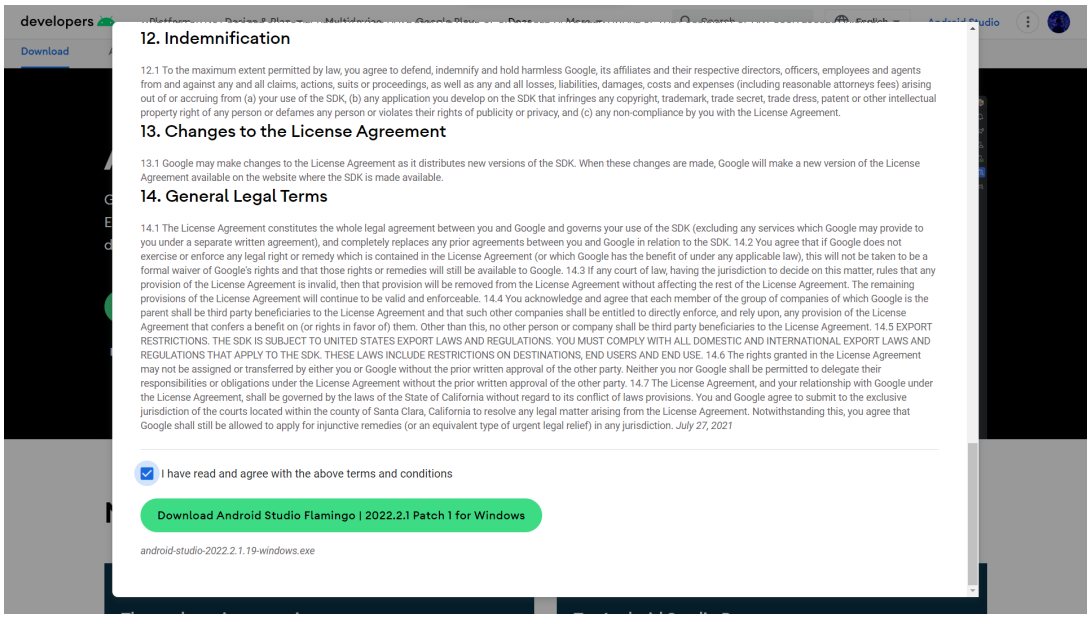
1. Flutter masih baru dibandingkan dengan beberapa framework lainnya. Ini berarti bahwa dokumentasi dan sumber daya yang tersedia mungkin belum sebanyak yang tersedia untuk framework yang lebih mapan.
2. Ukuran file aplikasi Flutter lebih besar dibandingkan dengan aplikasi yang dibangun menggunakan teknologi lain. Ini dapat menjadi masalah dalam situasi di mana ukuran file menjadi perhatian utama, seperti dalam pengembangan aplikasi dengan batasan ruang penyimpanan atau kecepatan internet yang terbatas.
3. Dukungan untuk beberapa fitur platform mungkin masih terbatas atau belum sepenuhnya matang dalam Flutter. Meskipun Flutter berusaha untuk menyediakan pengalaman yang seragam di berbagai platform, ada kemungkinan bahwa beberapa fitur khusus platform tertentu mungkin tidak sepenuhnya didukung atau membutuhkan upaya lebih dalam implementasinya.

1.3 Instalasi Flutter dan Android Studio

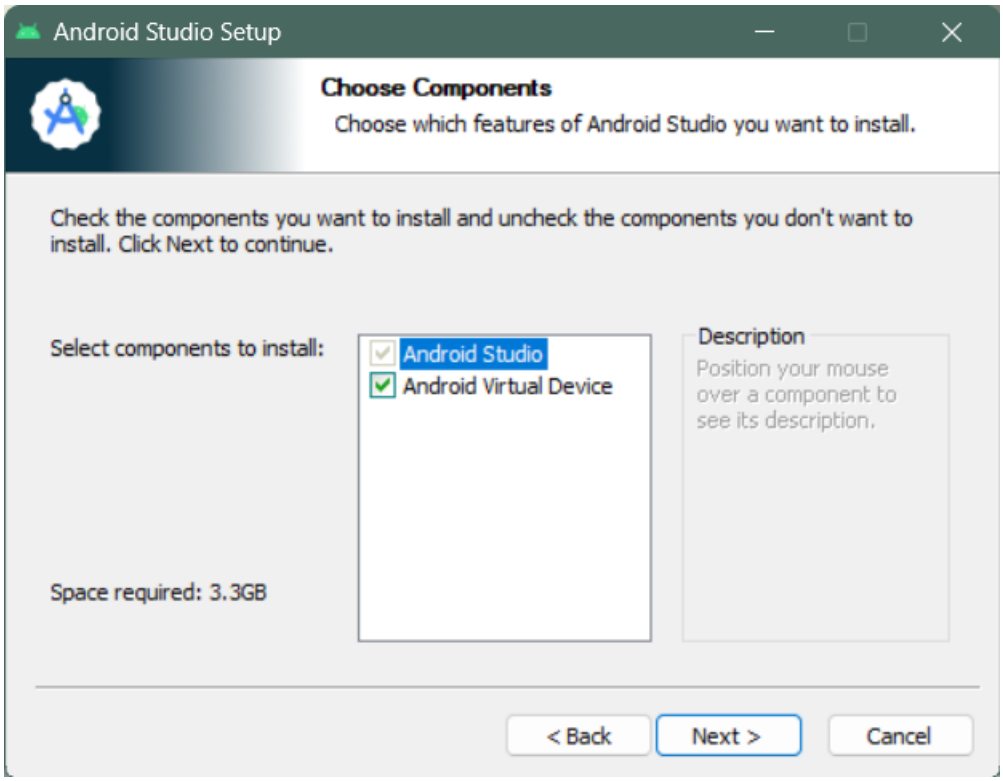
1.3.1 Instalasi Android Studio

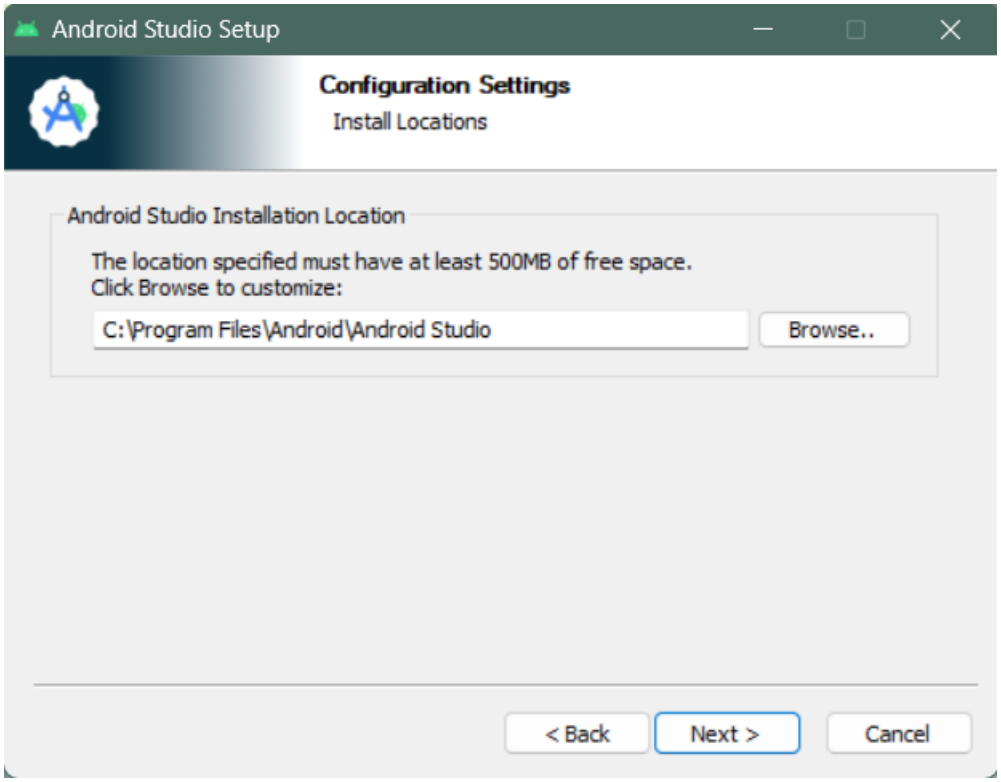
1. Download android studio versi terbaru di link berikut

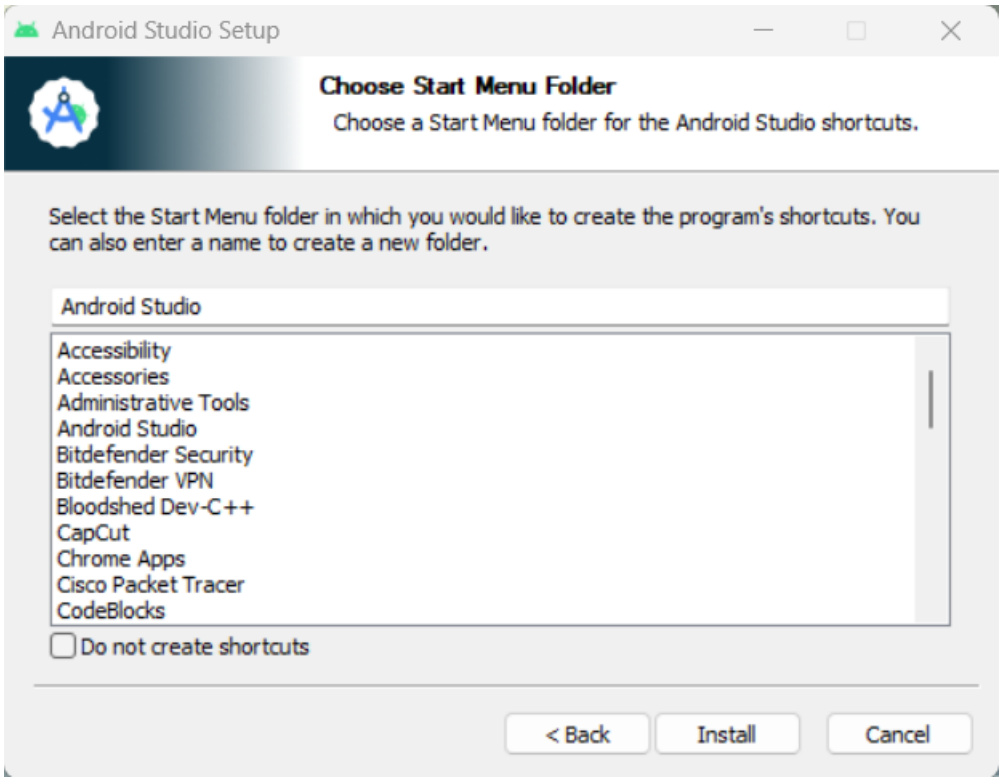


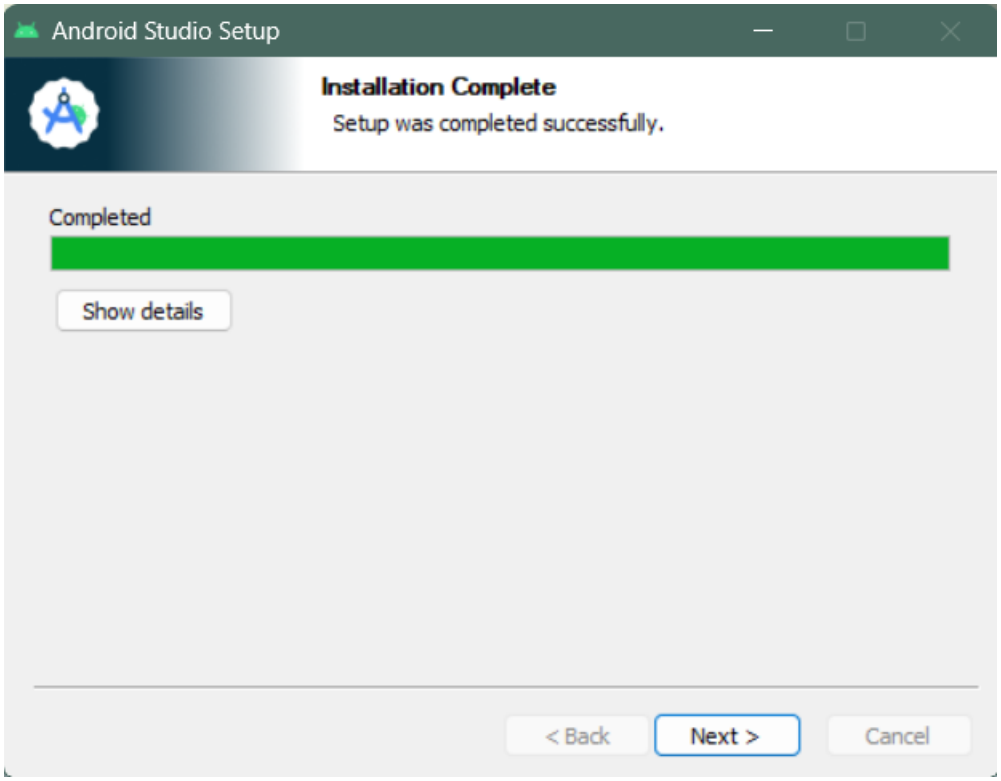


2. Instal android studio

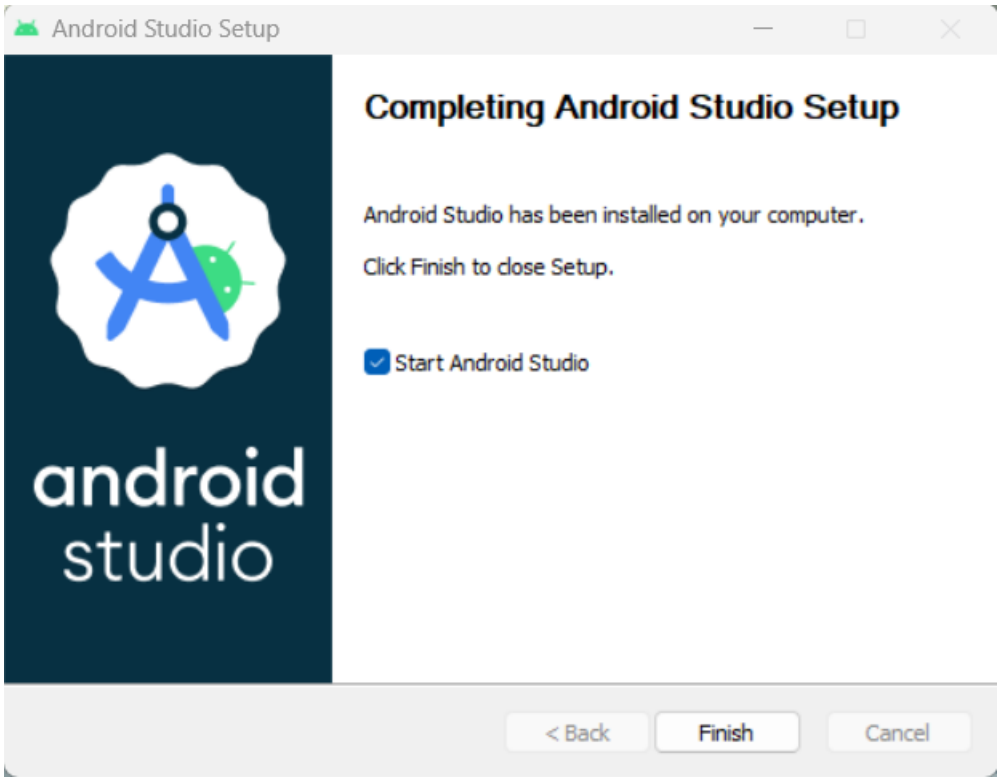

Android Studio Setup
Choose Components
Choose which features of Android Studio you want to install.
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.
Select components to install:
Android Studio
Android Virtual Device
Description
Position your mouse over a component to see its description.
Space required: 3.3GB
< Back
Next >
Cancel

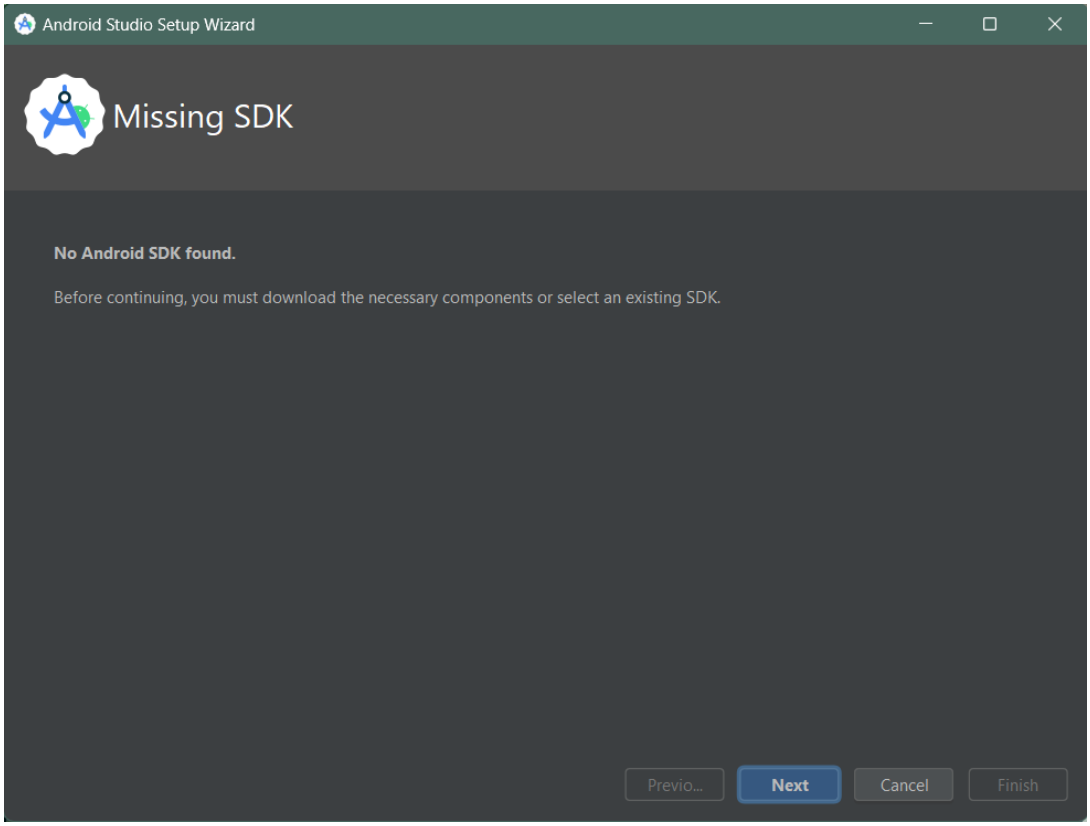

Android Studio Setup
Configuration Settings
Install Locations
Android Studio Installation Location
The location specified must have at least 500MB of free space.
Click Browse to customize:
C:\Program Files\Android\Android Studio
Browse..
< Back
Next >
Cancel

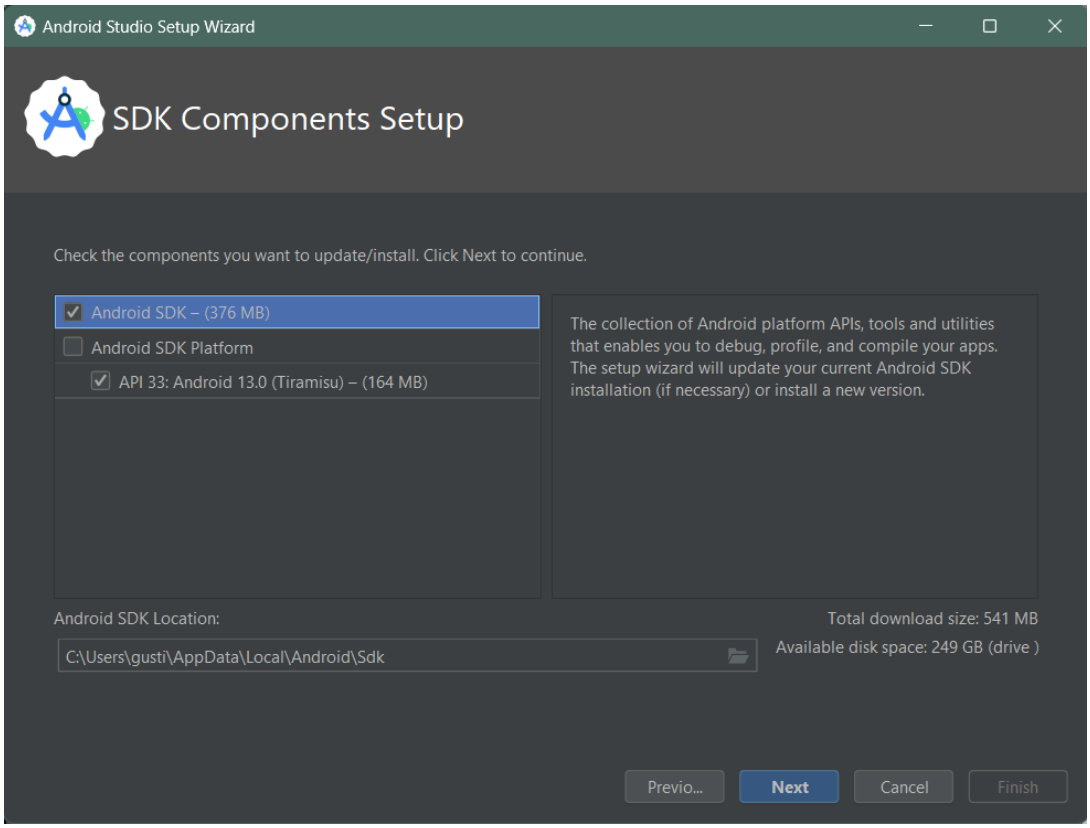


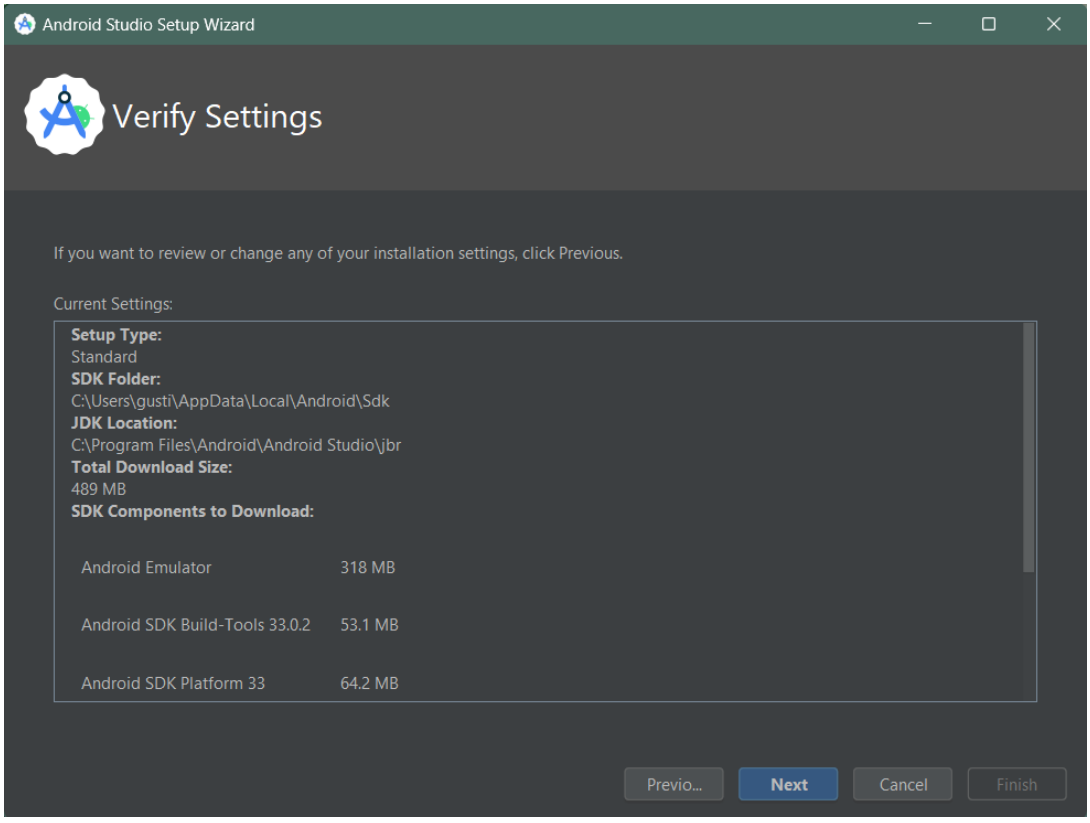


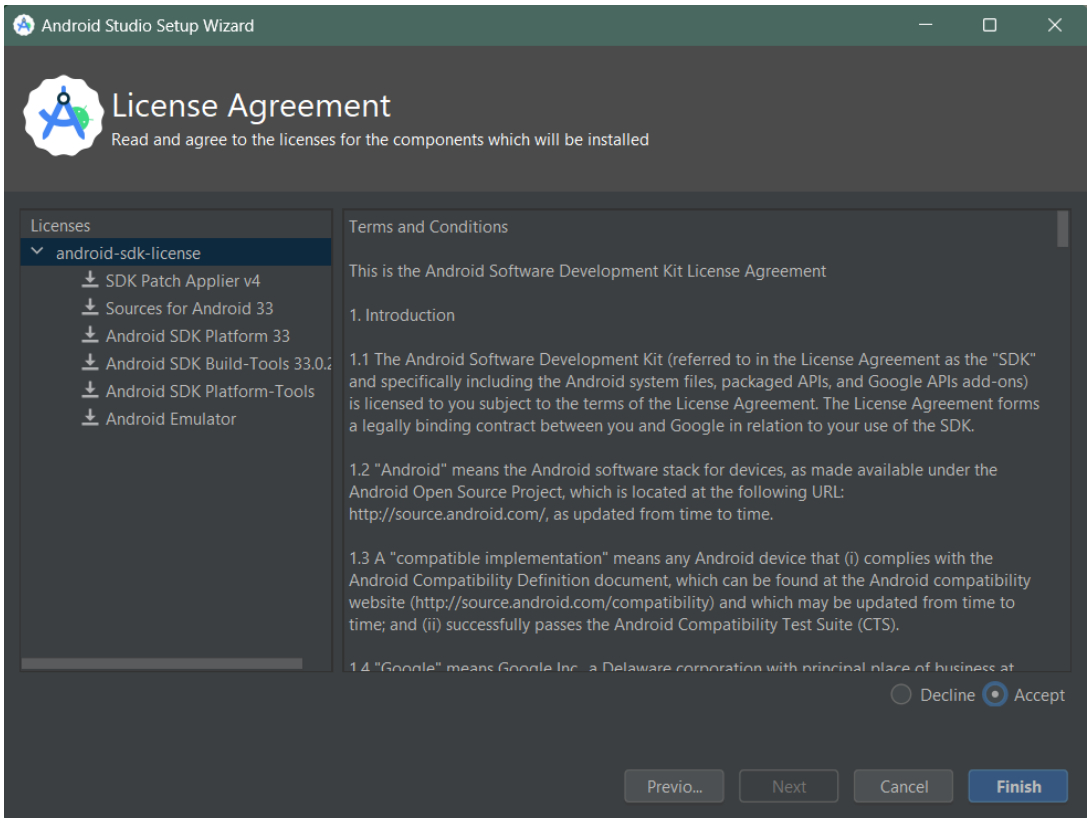
3. Setup android studio

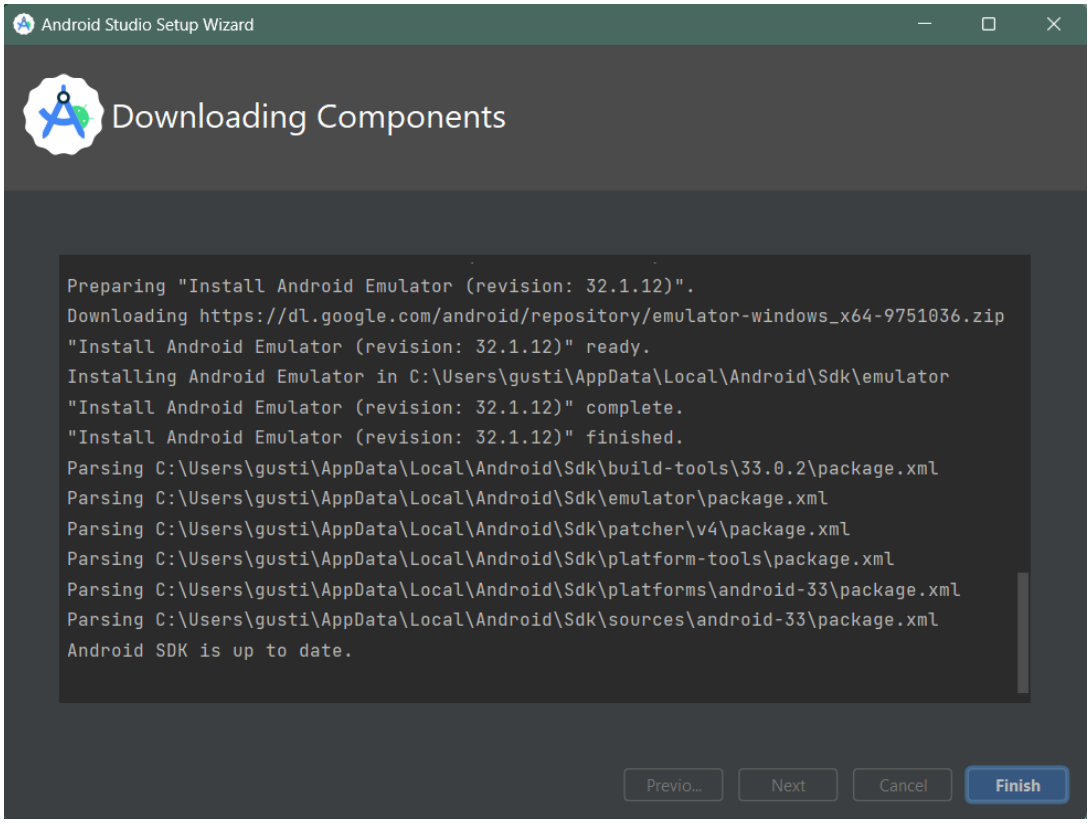

Android Studio Setup
Completing Android Studio Setup
Android Studio has been installed on your computer.
Click Finish to close Setup.
Start Android Studio
android
studio
< Back
Finish
Cancel

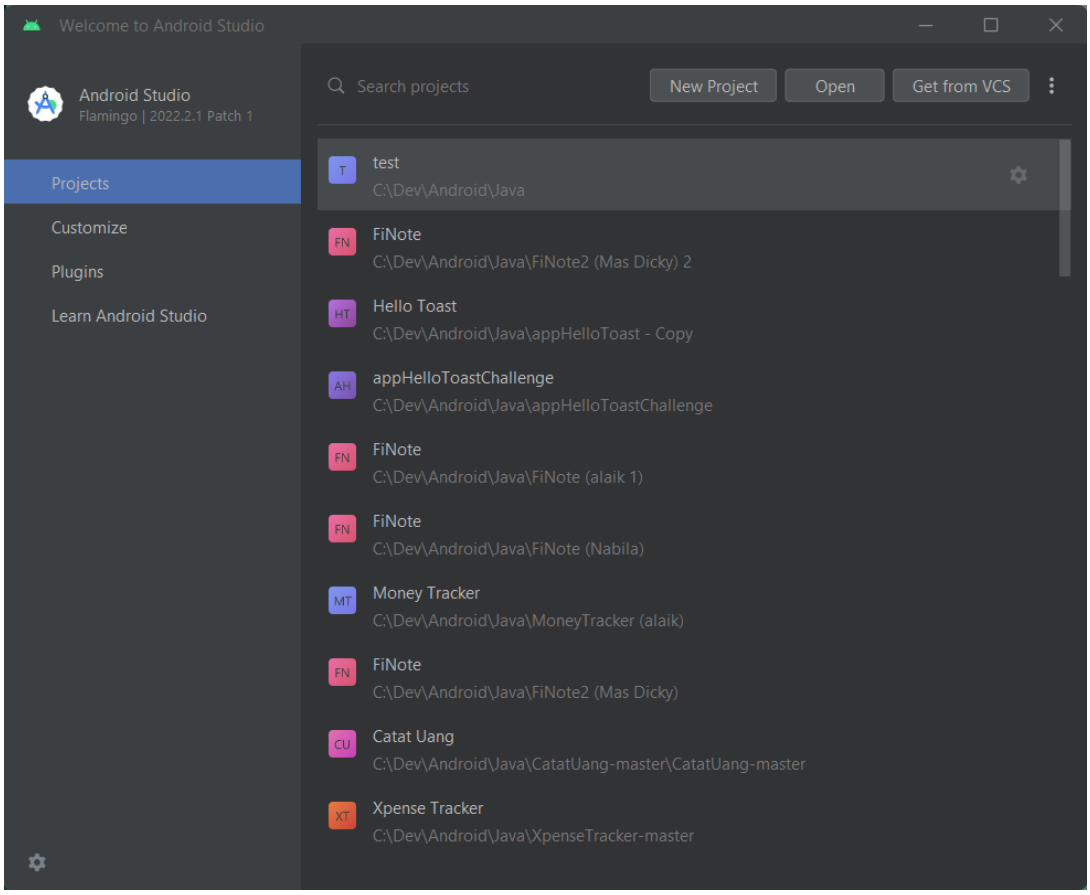

Android Studio Setup Wizard
Missing SDK
No Android SDK found.
Before continuing, you must download the necessary components or select an existing SDK.
Previo...
Next
Cancel
Finish


Android Studio Setup Wizard
SDK Components Setup
Check the components you want to update/install. Click Next to continue.
Android SDK – (376 MB)
Android SDK Platform
API 33: Android 13.0 (Tiramisu) – (164 MB)
The collection of Android platform APIs, tools and utilities that enables you to debug, profile, and compile your apps. The setup wizard will update your current Android SDK installation (if necessary) or install a new version.
Android SDK Location:
C:\Users\gusti\AppData\Local\Android\Sdk
Total download size: 541 MB
Available disk space: 249 GB (drive)
Previo...
Next
Cancel
Finish


Android Studio Setup Wizard
Verify Settings
If you want to review or change any of your installation settings, click Previous.
Current Settings:
Setup Type:
Standard
SDK Folder:
C:\Users\gusti\AppData\Local\Android\Sdk
JDK Location:
C:\Program Files\Android\Android Studio\jbr
Total Download Size:
489 MB
SDK Components to Download:
Android Emulator 318 MB
Android SDK Build-Tools 33.0.2 53.1 MB
Android SDK Platform 33 64.2 MB
Previo...
Next
Cancel
Finish


Android Studio Setup Wizard
License Agreement
Read and agree to the licenses for the components which will be installed
Licenses
android-sdk-license
SDK Patch Applier v4
Sources for Android 33
Android SDK Platform 33
Android SDK Build-Tools 33.0.2
Android SDK Platform-Tools
Android Emulator
Terms and Conditions
This is the Android Software Development Kit License Agreement
1. Introduction
1.1 The Android Software Development Kit (referred to in the License Agreement as the "SDK" and specifically including the Android system files, packaged APIs, and Google APIs add-ons) is licensed to you subject to the terms of the License Agreement. The License Agreement forms a legally binding contract between you and Google in relation to your use of the SDK.
1.2 "Android" means the Android software stack for devices, as made available under the Android Open Source Project, which is located at the following URL: http://source.android.com/, as updated from time to time.
1.3 A "compatible implementation" means any Android device that (i) complies with the Android Compatibility Definition document, which can be found at the Android compatibility website (http://source.android.com/compatibility) and which may be updated from time to time; and (ii) successfully passes the Android Compatibility Test Suite (CTS).
Decline
Accept
Previo...
Next
Cancel
Finish

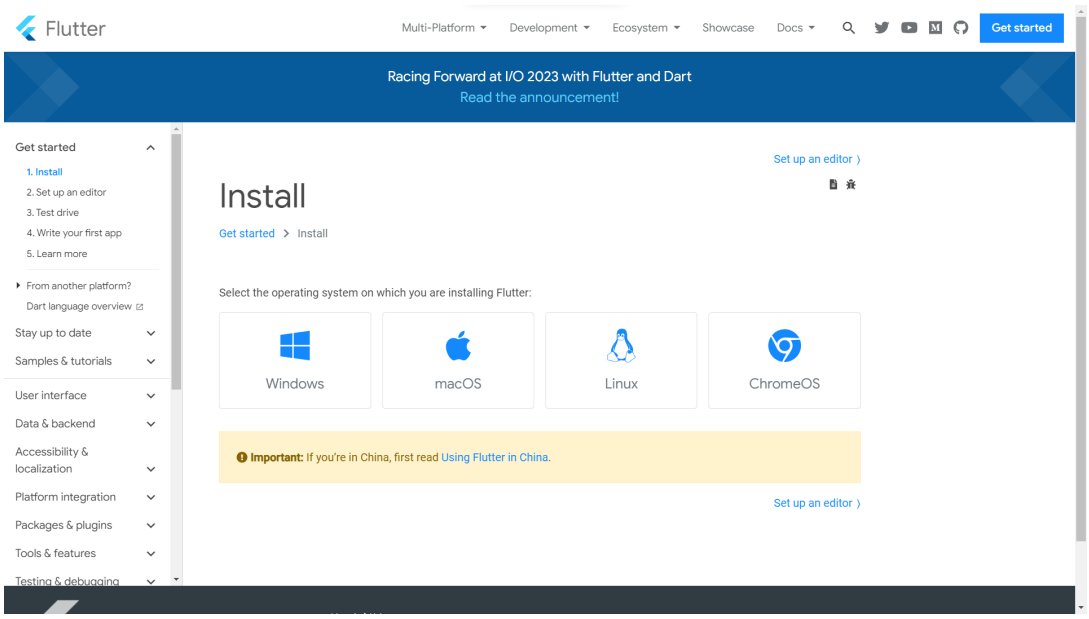

Android Studio Setup Wizard
Downloading Components
Preparing "Install Android Emulator (revision: 32.1.12)".
Downloading https://dl.google.com/android/repository/emulator-windows_x64-9751036.zip
"Install Android Emulator (revision: 32.1.12)" ready.
Installing Android Emulator in C:\Users\gusti\AppData\Local\Android\Sdk\emulator
"Install Android Emulator (revision: 32.1.12)" complete.
"Install Android Emulator (revision: 32.1.12)" finished.
Parsing C:\Users\gusti\AppData\Local\Android\Sdk\build-tools\33.0.2\package.xml
Parsing C:\Users\gusti\AppData\Local\Android\Sdk\emulator\package.xml
Parsing C:\Users\gusti\AppData\Local\Android\Sdk\patcher\v4\package.xml
Parsing C:\Users\gusti\AppData\Local\Android\Sdk\platform-tools\package.xml
Parsing C:\Users\gusti\AppData\Local\Android\Sdk\platforms\android-33\package.xml
Parsing C:\Users\gusti\AppData\Local\Android\Sdk\sources\android-33\package.xml
Android SDK is up to date.
Previo...
Next
Cancel
Finish

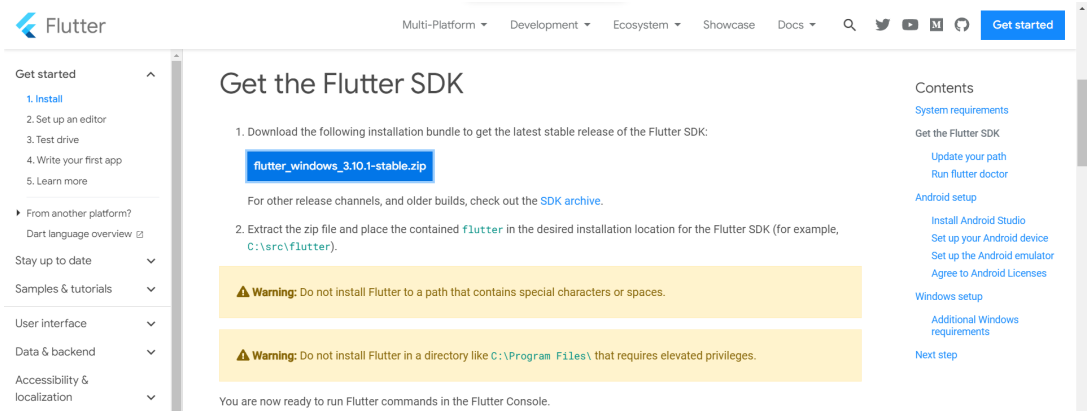


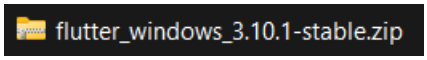
4. Selesai

1.3.2 Instalasi Flutter

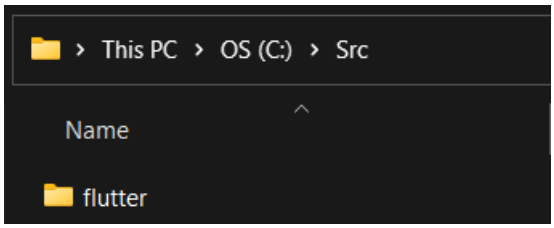
1. Instal android studio jika belum ada
2. Unduh Flutter versi terbaru di link berikut sesuai OS perangkat yang digunakan [**Windows**].





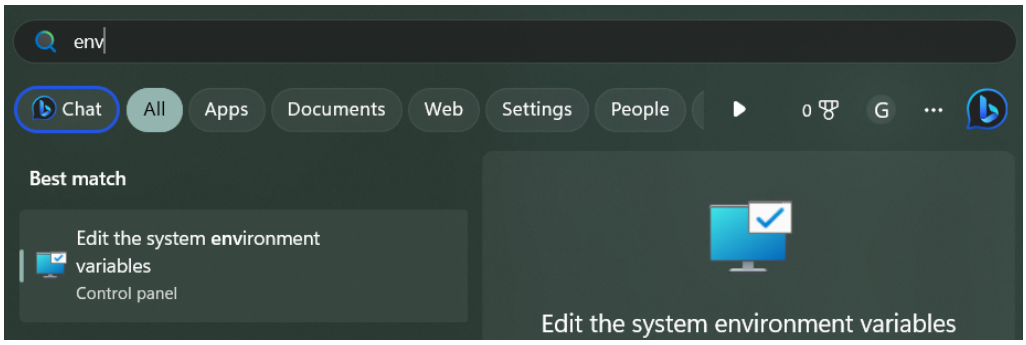


3. Ekstrak file flutter yang telah diunduh dan letakkan di lokasi instalasi yang diinginkan, misal di folder (**"C:/Src/flutter"**)

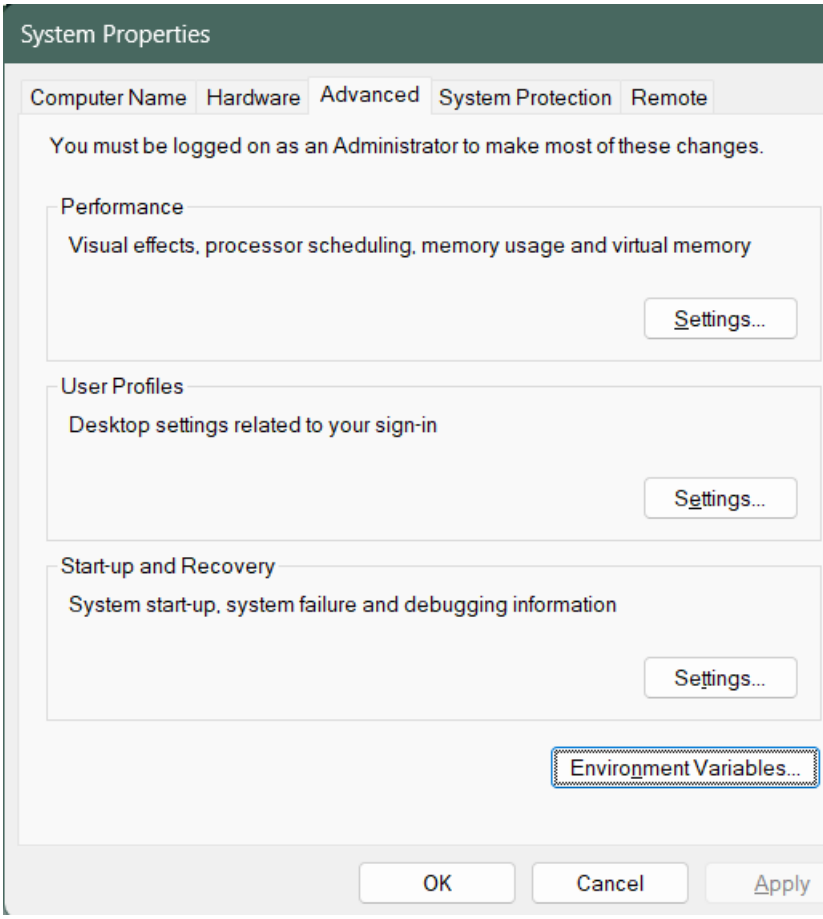


4. Update path
    1. Cari **"env"** di search bar windows, pilih "**Edit environment variables for**

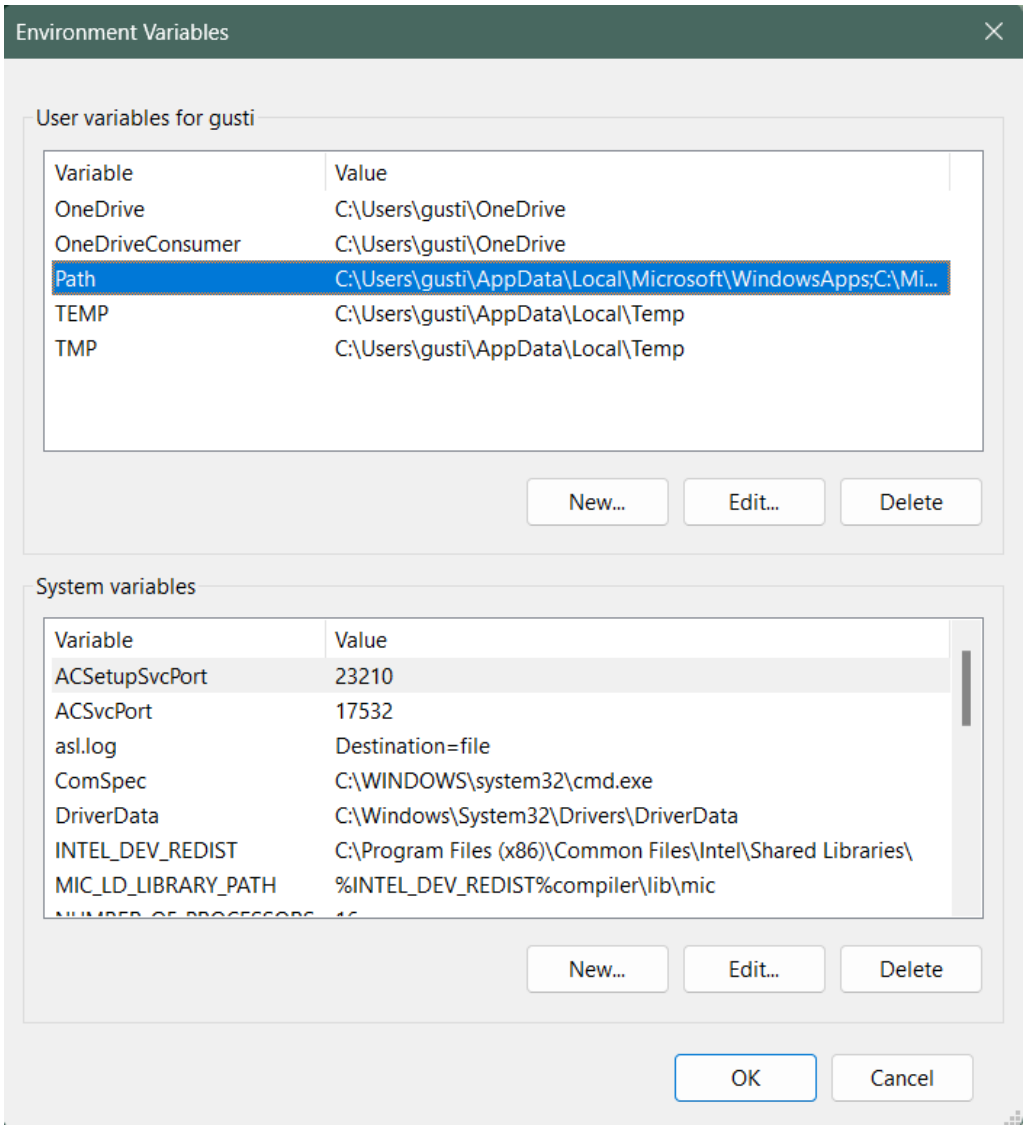
your account”



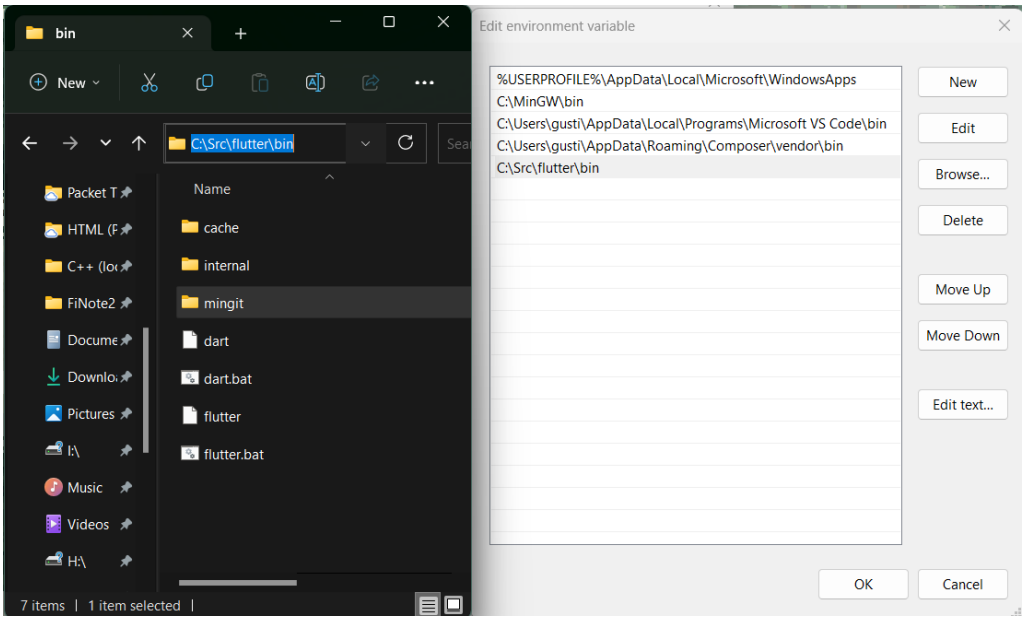
2. pilih “**environment variables**”



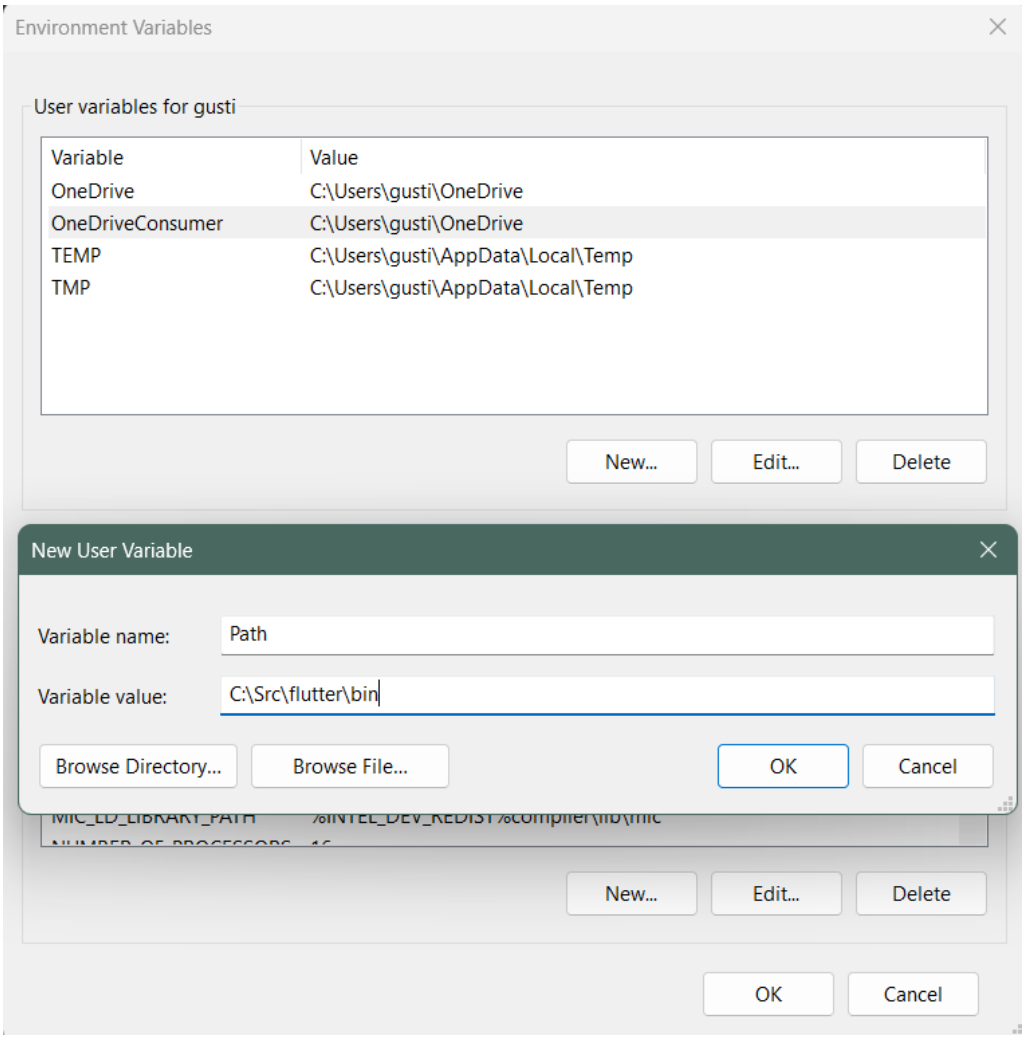
3. pilih “**path**” pada “**user variables ...**”



4. Tambahkan alamat folder bin flutter (**C:/Src/flutter/bin**)

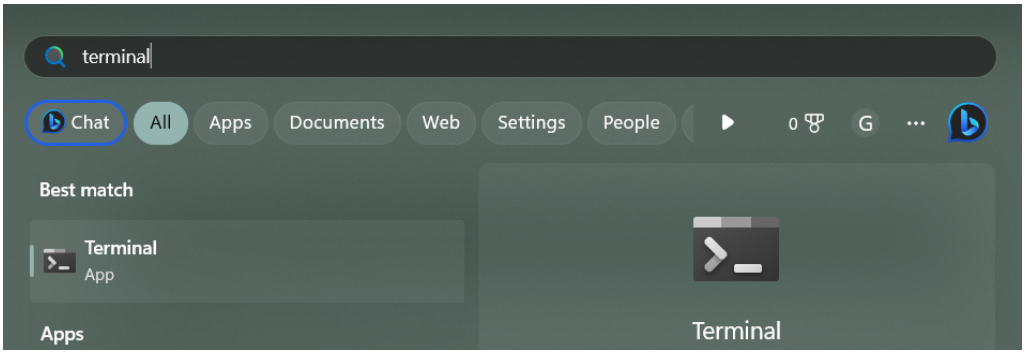


5. Jika belum ada "**Path**" pada "**user variables**", bisa menambahkan sendiri disertai alamat folder bin flutter (**C:/Src/flutter/bin**).

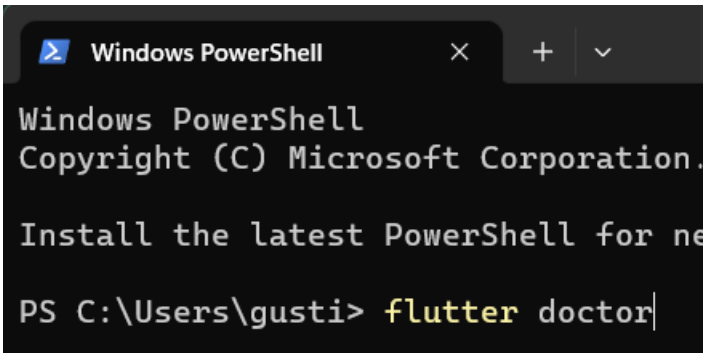


5. Cek instalasi flutter

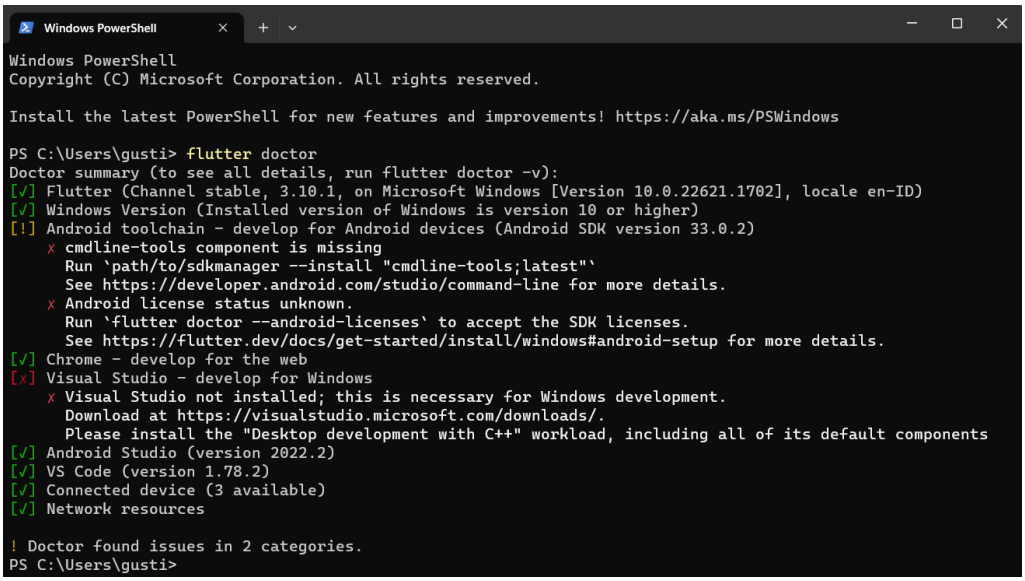
    1. Buka terminal pada windows.



    2. Ketik "**flutter doctor**" dan jalankan.



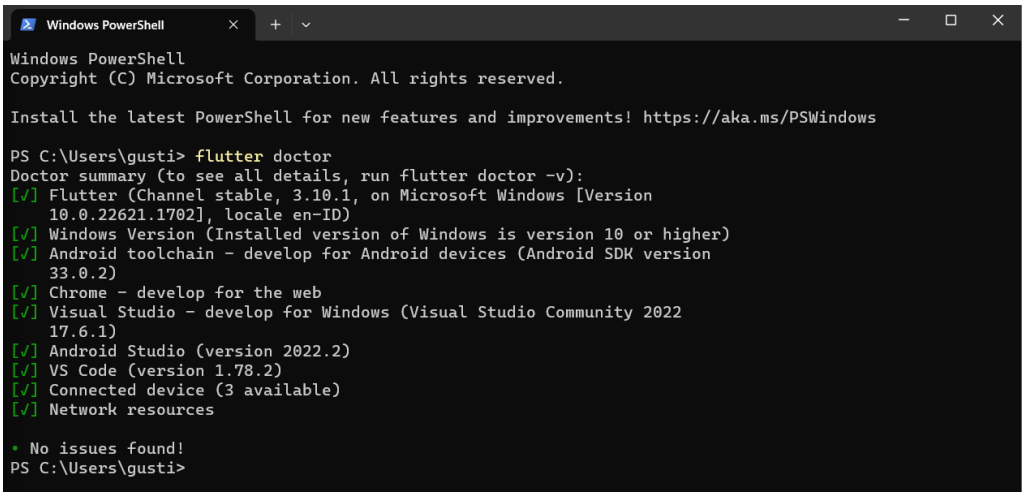
3. Pada hasil ditemukan 2 issue



4. Atasi issue sesuai dengan panduan masing-masing issue (issue "**visual studio ...**" tidak wajib diatasi).

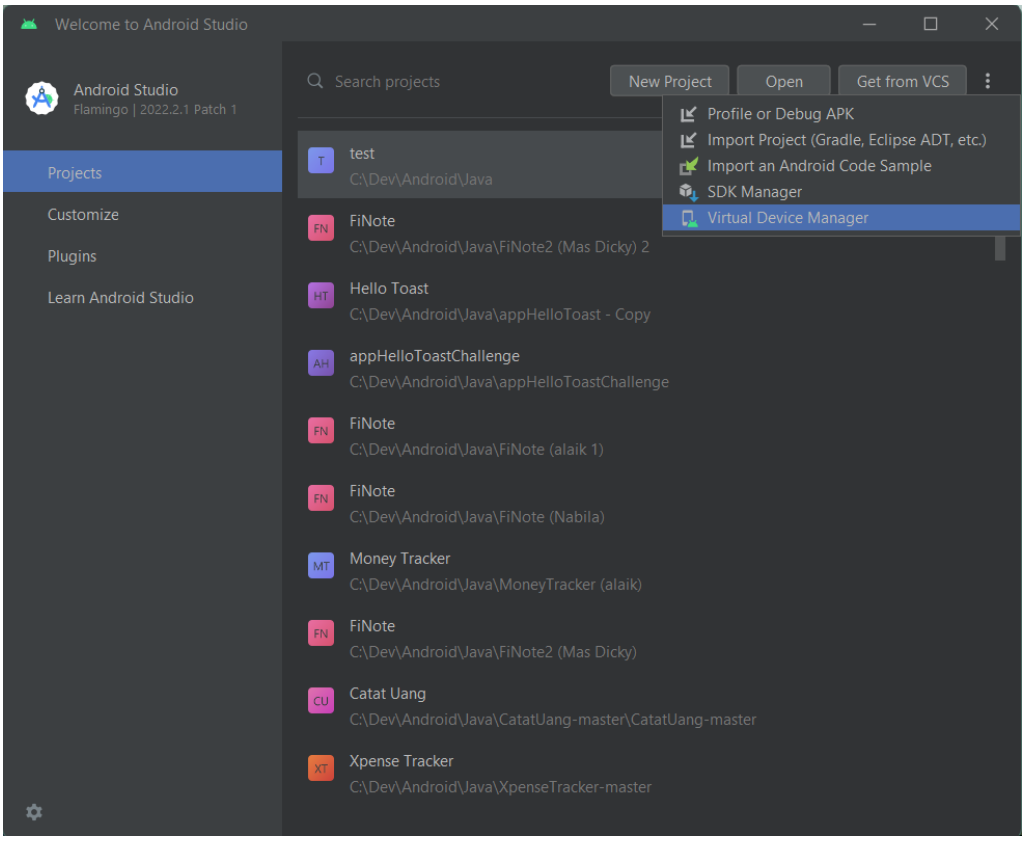
   Mengatasi "cmdline-tools component is missing"

5. Setelah issue diatasi, jalankan "**flutter doctor**" untuk cek apakah issue sudah teratasi.

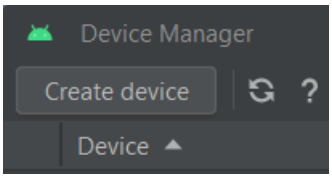


6. Set up Android emulator (jika belum set up)

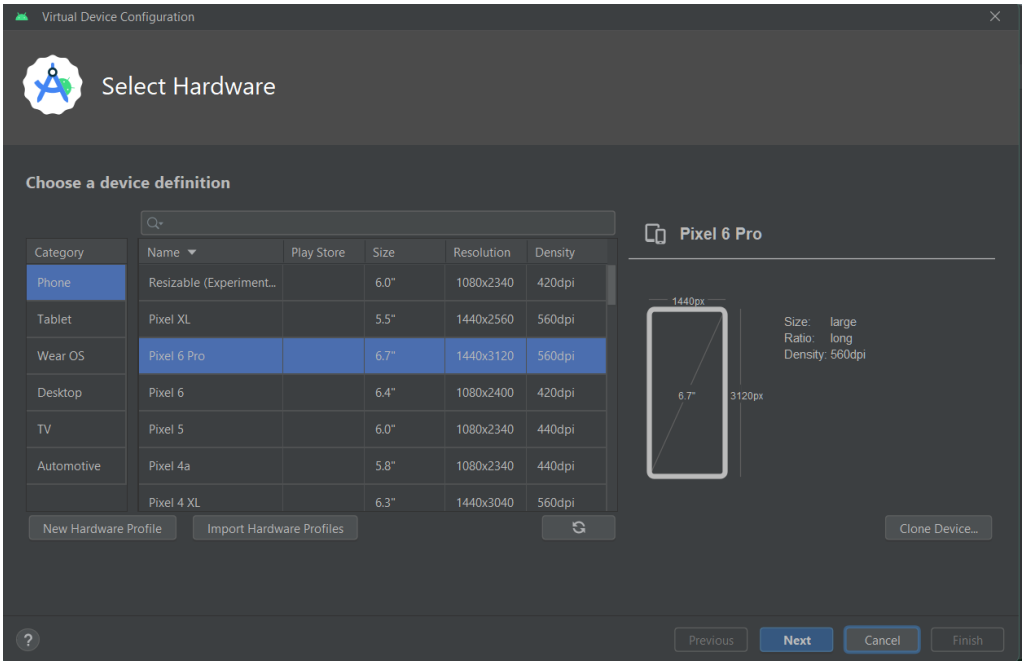
    1. Buka android studio → “**titik tiga (:)**” → “**Virtual Device Manager**”



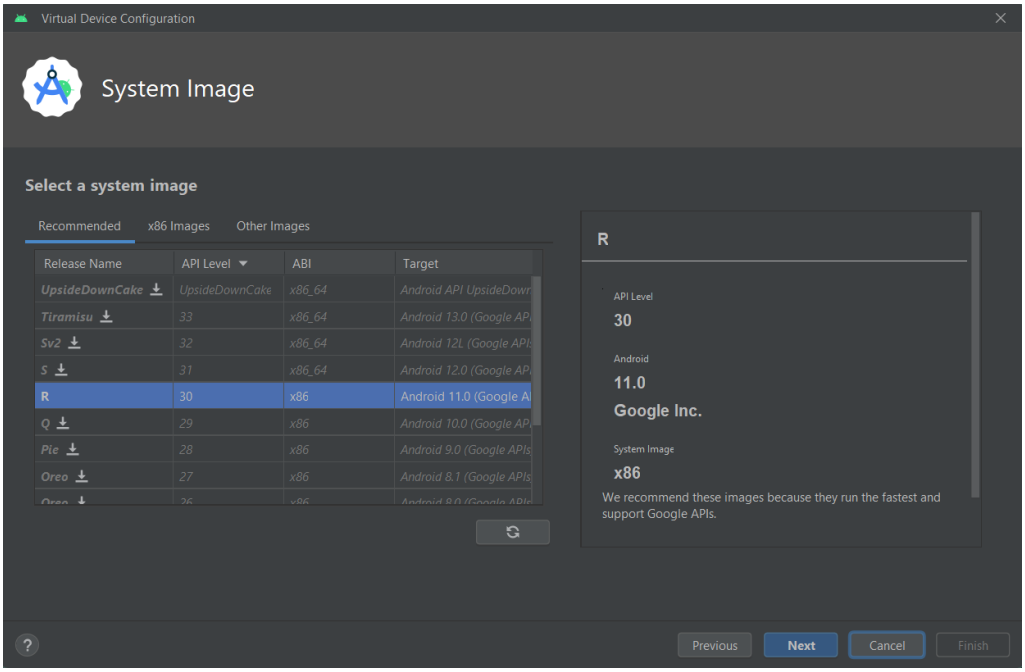
    2. Pilih “**Create device**”



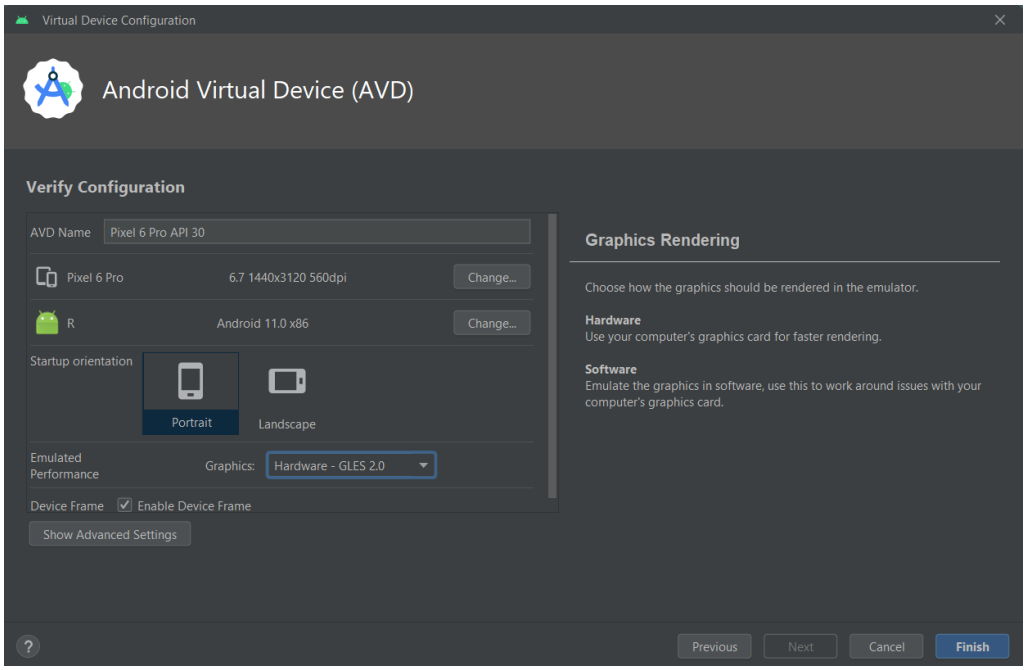
3. Pilih perangkat yang diinginkan



4. Pilih system image yang diinginkan (unduh dahulu jika belum terunduh)



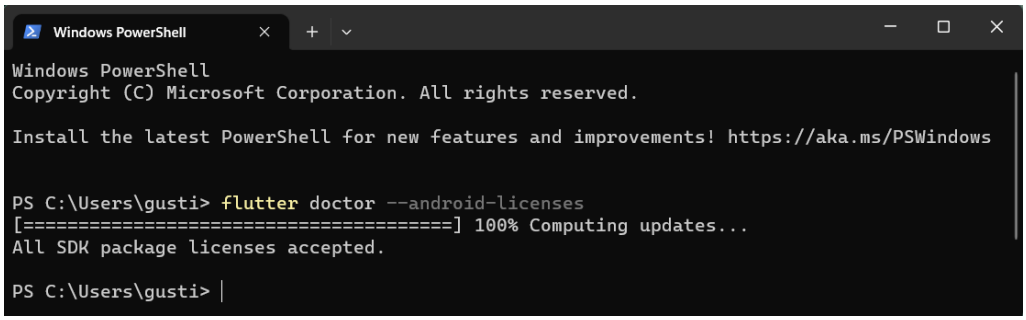
5. Pada “**emulated performance**”, pilih “**Hardware-GLES 2.0**” untuk mengizinkan hardware acceleration, Finish



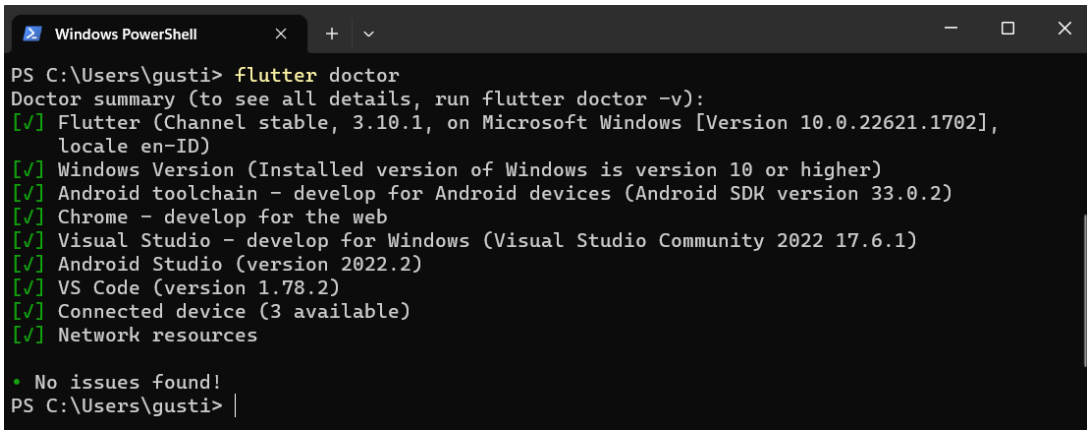
6. Android emulator siap dijalankan

7. Menyetujui lisensi android

    1. Buka terminal pada windows
    2. ketikkan “**flutter doctor —android-licenses**”, enter



8. Jalankan “**flutter doctor**” pada terminal



9. flutter sudah terinstal, siap dipakai.

2 Dasar Pemrograman Dart

Dasar Pemrograman Dart

Dart adalah bahasa pemrograman yang agak aneh. Aneh tapi nyata, emang bener ada bahasanya dan dipake.

2.1 Apa itu Dart?

Dart merupakan bahasa pemrograman yang open source yang dikembangkan oleh Google dengam tujuan untuk membuat aplikasi multiplatform seperti mobile, desktop, dan web. Tujuan awal pembuatan Dart adalah untuk menggantikan JavaScript yang dinilai memiliki banyak kelemahan.

2.2 Tipe data dan Variabel

Dalam dart variabel dapat dideklarasikan secara eksplisit maupun tidak. Deklarasi secara ekspilisit dilakukan dengan cara menambahkan tipe data di depan nama variabel. Sedangkan untuk deklarasi secara tidak eksplisit dilakukan dengan cara menambahkan kata kunci `var` di depan nama variabel. Sebagai bahasa yang memiliki fitur type inference dart dapat menentukan tipe data otomatis dengan kata kunci `var`. Berikut adalah contoh pendeklarasian variabel dalam dart:

```
var greetings = 'Hello World!'; // String
var myAge = 20;                 // integers
var isHuman = true;             // boolean
```

```
String greetings = 'Hello World!'; // String
int myAge = 20;                 // integers
bool isHuman = true;             // boolean
```

Sebagai tambah berikut adalah tabel berbagai jenis tipe data yang dapat digunakan dalam bahasa pemrograman Dart beserta deskripsi dan contoh singkatnya:

Tipe	Dekripsi	Contoh
int	Bilangan bulat	5,7,8
double	Bilangan desimal	3.14,1.23
num	Bilangan bulat dan bilangan desimal	5, 3.14, -99.00

Tipe	Dekripsi	Contoh
bool	Boolean	true,false
String	Teks yang terdiri dari 0 atau beberapa karakter	“udinus”,““,”G”
List	Daftar nilai	[1, 2, 3], [‘a’, ‘b’, ‘c’]
Map	Pasangan key-value	{“x”: 4, “y”: 10}
dynamic	Tipe data apa pun	

2.3 Control Flow (Kondisi dan Perulangan)

2.3.1 Kondisi

Saat membuat atau mengembangkan sebuah program, terdapat situasi di mana aliran eksekusi bercabang berdasarkan kondisi tertentu. Dalam bahasa Dart, terdapat dua kata kunci yang dapat digunakan untuk mengakomodasi dan menguji kondisi tersebut, yaitu if dan switch-if. Berikut merupakan contoh penggunaan if dan switch-case dalam dart :

```
// if

if (kondisi) {
    // Tindakan jika kondisi benar
} else {
    // Tindakan jika kondisi salah
}
```

```
// switch

switch (nilai) {
    case 1:
        // Tindakan jika nilai sama dengan 1
        break;
    case 2:
        // Tindakan jika nilai sama dengan 2
        break;
    default:
        // Tindakan jika nilai tidak sama dengan kasus di atas
}
```

2.3.2 Perulangan

Saat membuat atau mengembangkan sebuah program, terdapat situasi di mana program melakukan hal yang sama berkali-kali. Dart memiliki banyak opsi untuk melakukan peru-

langan kode, seperti for, while, dan fo-while. Berikut merupakan contoh penggunaan for, while, dan do-while dalam dart dengan study case menampilkan angka 1 sampai 100 :

```
// for

for (var i = 1; i <= 100; i++) {
    print(i);
}
```

```
// while

var i = 1;

while (i <= 100) {
    print(i);
    i++;
}
```

```
// do-while

var i = 1;

do {
    print(i);
    i++;
} while (i <= 100);
```

2.4 Pembuatan Fungsi

Functions dalam Dart digunakan untuk mengelompokkan dan mengorganisir blok kode yang dapat digunakan secara berulang. Fungsi tersebut dapat menerima parameter input, melakukan tindakan tertentu, dan mengembalikan nilai balik (return value) jika diperlukan. Berikut ini adalah contoh pendeklarasian dan pemanggilan sebuah fungsi dalam Dart:

```
// fungsi bentuk umum
TipeNilaiBalik namaFungsi(TipeParameter parameter1, TipeParameter parameter2) {
    return nilaiBalik;
}

// fungsi tanpa parameter
TipeNilaiBalik namaFungsi() {
    return nilaiBalik;
```

```
}

namaFungsi();

// fungsi tanpa pengembalian nilai
void namaFungsi(TipeParameter parameter1, TipeParameter parameter2) {
    print(parameter1, parameter2);
}

namaFungsi(parameter1, parameter2);

// fungsi dengan parameter opsional
void namaFungsi([TipeParameter parameter1, TipeParameter parameter2]) {}

namaFungsi(parameter1);
namaFungsi(parameter2);

// fungsi dengan lambda syntax
TipeNilaiBalik namaFungsi(TipeParameter parameter1) => parameter1 * parameter1;
```

2.5 Class dan Objek

Sebagai sebuah bahasa pemrograman yang mendukung konsep OOP (Object-Oriented Programming), Dart menyediakan fitur untuk membuat class. Class dalam Dart berfungsi sebagai sebuah cetak biru (blueprint) untuk membuat objek. Di dalam class, kita dapat mendefinisikan sifat-sifat (attribute) dan perilaku-perilaku (behavior) yang dimiliki oleh objek yang akan kita buat.

Dalam Dart, sifat-sifat objek didefinisikan menggunakan variabel, sedangkan perilaku objek seringkali direpresentasikan sebagai fungsi (function). Untuk mendeklarasikan sebuah class dalam Dart, kita menggunakan kata kunci `class`. Berikut ini adalah contoh pendeklarasian class, attribute, dan method dalam Dart:

```
class namaClass {
    TipeAtribut namaAtribut;

    TipeNilaiBalik namaMethod() {}
}
```

Selanjutnya berikut adalah cara membuat sebuah objek dari suatu class, mengakses atribut, dan menggunakan method:

```
NamaClass namaObjek = NamaClass();
namaObjek.namaAtribut;
namaObjek.namaMethod();
```

2.6 Pewarisan dan Polimorfisme

Inheritance atau pewarisan adalah kemampuan suatu program untuk membuat kelas baru berdasarkan kelas yang sudah ada. Konsep inheritance dapat dibayangkan layaknya seorang anak mewarisi sifat dari orang tuanya. Di dalam OOP kelas yang menurunkan sifat disebut sebagai kelas induk (parent class/superclass) sementara kelas yang mewarisi kelas induknya disebut sebagai kelas anak (child class/subclass). Dalam dart pewarisan dapat dideklarasikan sebagai berikut:

```
class ParentClass {
    void parentMethod() {}
}

class ChildClass extends ParentClass {
    void childMethod() {}
}

var childObj = ChildClass();
childObj.parentMethod(); // Memanggil metode dari kelas induk
childObj.childMethod(); // Memanggil metode dari kelas anak
```

Polimorfisme adalah konsep di mana suatu objek dapat muncul dalam berbagai bentuk atau tipe. Dalam konteks Dart, polimorfisme memungkinkan penggunaan objek dari kelas turunan sebagai objek dari kelas induknya, sehingga objek-objek tersebut dapat diperlakukan secara umum tanpa perlu mengetahui tipe spesifik dari objek.

Berikut adalah contoh sederhana untuk pendeklarasian polimorfisme dalam Dart:

```
class Shape {
    void draw() {}
}

class Circle extends Shape {
    @override
    void draw() {}
}

class Rectangle extends Shape {
    @override
```

```
    void draw() {}
}

List<Shape> shapes = [Circle(), Rectangle()];

for (var shape in shapes) {
    shape.draw(); // Polimorfisme terjadi di sini
}
```

3 Antarmuka Pengguna

Antarmuka Pengguna

Aplikasi yang baik memiliki antarmuka yang dapat digunakan oleh penggunanya.

3.1 Widget

Widget pada Flutter merupakan elemen dasar dalam membangun antarmuka pengguna (UI). Dalam Flutter, hampir semua komponen yang berada di layar adalah widget, termasuk tombol, teks, gambar, kotak, daftar, dan sebagainya. Berikut merupakan widget yang umum digunakan dalam flutter :

3.1.1 Scaffold

Widget yang digunakan untuk membuat tampilan dasar aplikasi Flutter. Memiliki 3 bagian yaitu AppBar, Body, dan FloatingActionButton.

```
Scaffold(
    appBar: AppBar(
        title: const Text('First Screen'),
        actions: [
            IconButton(
                icon: const Icon(
                Icons.search,
                color: Colors.white,
                ),
                onPressed: () {},
            ),
        ],
        leading: IconButton(
            icon: const Icon(
                Icons.menu,
                color: Colors.white,
        ),
            onPressed: () {},
        ),
    ),
    body: const Center(
```

```
        child: Text('Hello world!'),
    ),
    floatingActionButton: FloatingActionButton(
        child: const Icon(Icons.add),
        onPressed: () {},
    ),
);
```

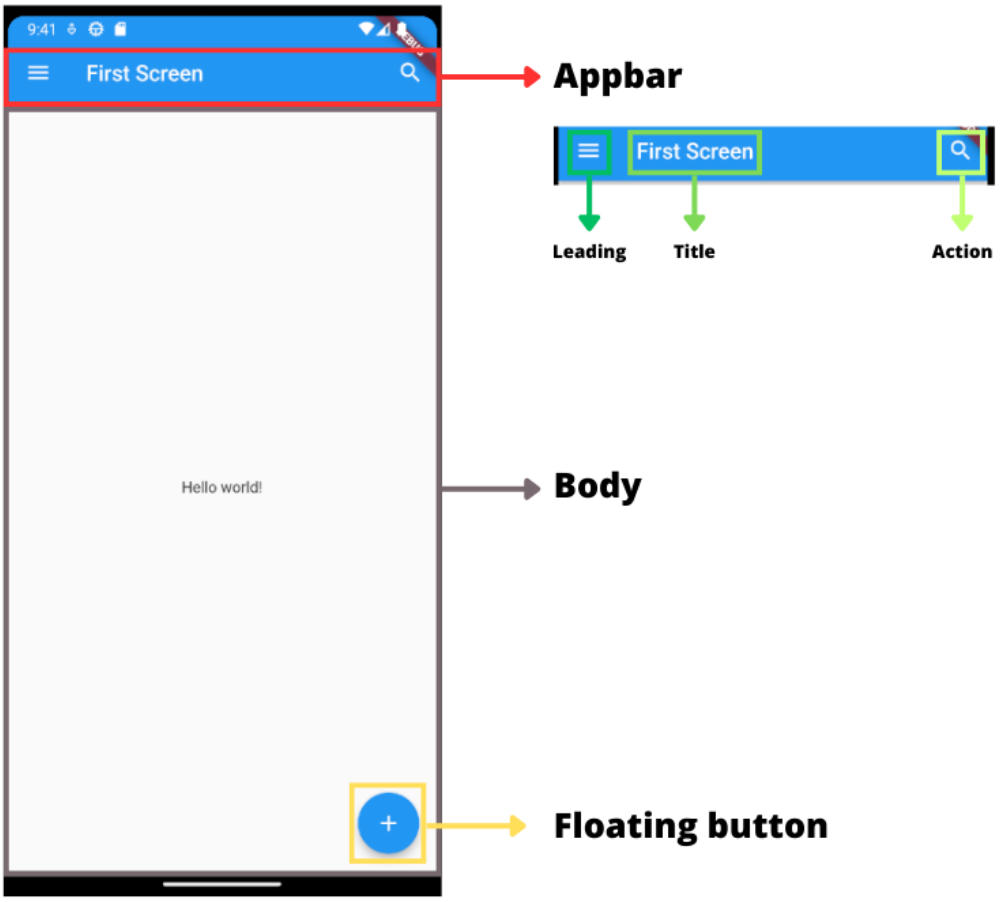


Figure 3.1: Appbar Details

Penjelasan

1. `appBar: AppBar(...)`: Ini adalah bagian yang mendefinisikan AppBar (bilah aplikasi) di atas tampilan utama. Di dalam AppBar, kita menentukan judul dengan menggunakan widget `Text`. Selain itu, ada juga dua tombol ikon yang

ditempatkan di sebelah kanan dan kiri AppBar. Tombol ikon kanan menggunakan `IconButton` dengan ikon `Icons.search`, sedangkan tombol ikon kiri menggunakan `IconButton` dengan ikon `Icons.menu`.

2. `body:...` : Ini adalah bagian yang mendefinisikan konten utama dari tampilan aplikasi.

3. `floatingActionButton: FloatingActionButton(...)` : Ini adalah bagian yang mendefinisikan tombol aksi mengambang (floating action button) di sudut kanan bawah tampilan. Menggunakan widget `FloatingActionButton` dan menentukan ikon dengan `Icon(Icons.add)`.

3.1.2 Container

Widget yang digunakan untuk melakukan styling, membuat sebuah shape (bentuk), dan layout pada widget child-nya.

```
Container(
    alignment: Alignment.center,
    width: 300,
    height: 300,
    color: Colors.blue,
    padding: const EdgeInsets.all(10),
    margin: const EdgeInsets.all(10),
    child: const Text(
    'Hello',
    ),
),
```

Penjelasan

1. `alignment:...`: Properti ini mengatur posisi atau penempatan konten di dalam container.

2. `width:..., height:...`: Properti ini mengatur lebar (width) dan tinggi (height) dari container.

3. `color:...`: Properti ini mengatur warna latar belakang dari container.

4. `padding:...`: Properti ini mengatur ruang padding (jarak) di sekeliling konten di dalam container.

5. `margin:...`: Properti ini mengatur ruang margin (jarak) di sekeliling container itu sendiri.

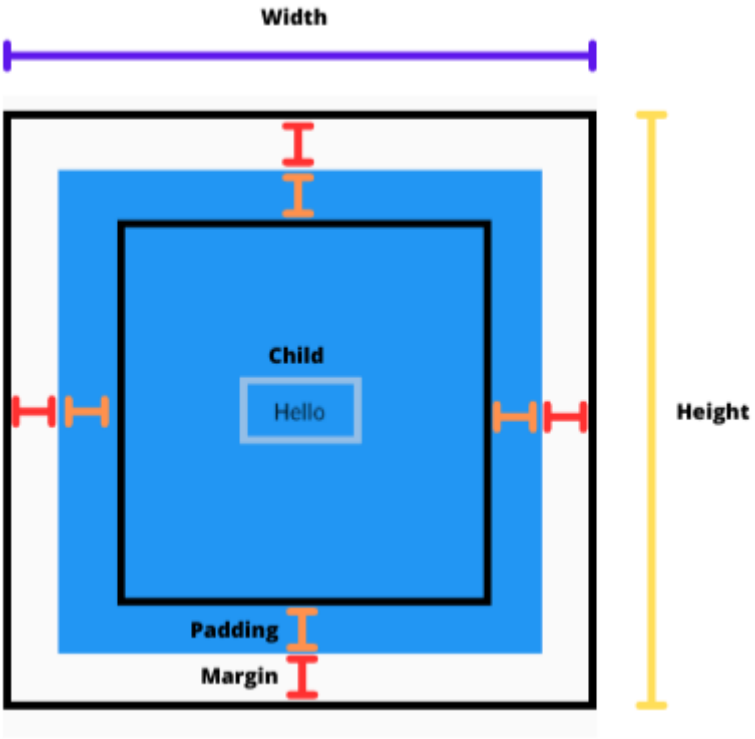


Figure 3.2: Container Details

6. `child:...`: Properti ini menyediakan konten yang akan ditampilkan di dalam container.

3.1.3 Center

Widget Center merupakan sebuah widget yang digunakan untuk membuat suatu widget berada pada posisi tengah.

```
Center(
    // konten yang ingin diposisikan ketengah
    child: const Text('Text berada di tengah'),
),
```

3.1.4 Safe Area

Widget dalam Flutter yang digunakan untuk mengatur area konten yang "aman" atau bebas dari gangguan seperti bilah status (status bar) atau bilah navigasi (navigation bar) pada perangkat.

```
SafeArea(
    // konten yang ingin diposisi dibawah bilah navigasi
    child: Scaffold(
        body: Center(
        child: Text('Konten utama aplikasi'),
        ),
    ),
)
```

3.1.5 SizedBox

Widget dalam Flutter yang digunakan untuk mengatur ukuran ruang kosong yang tetap dalam tata letak.

```
SizedBox(
    width: 200.0,
    height: 100.0,
)
```

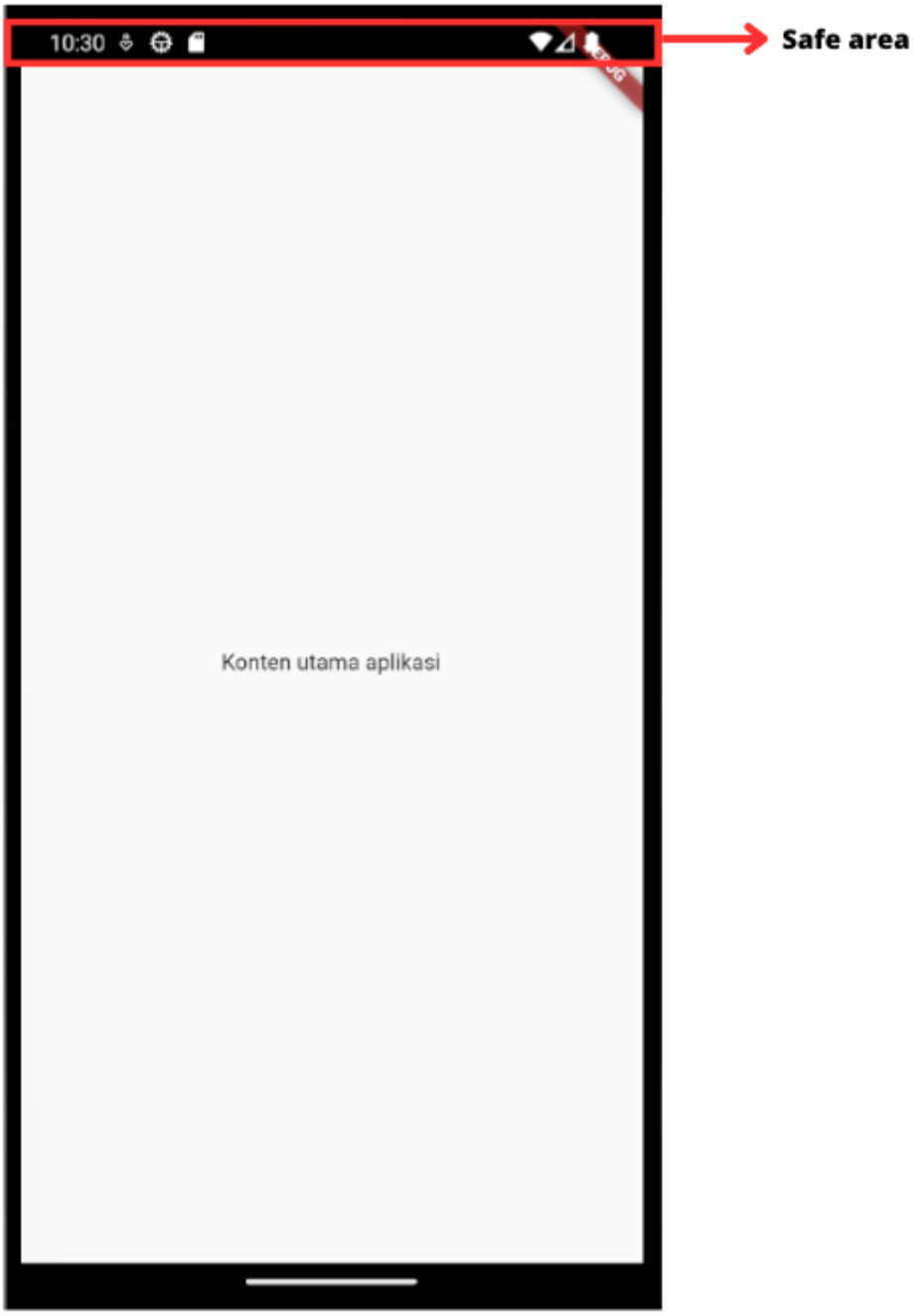


Figure 3.3: Safe Area Details

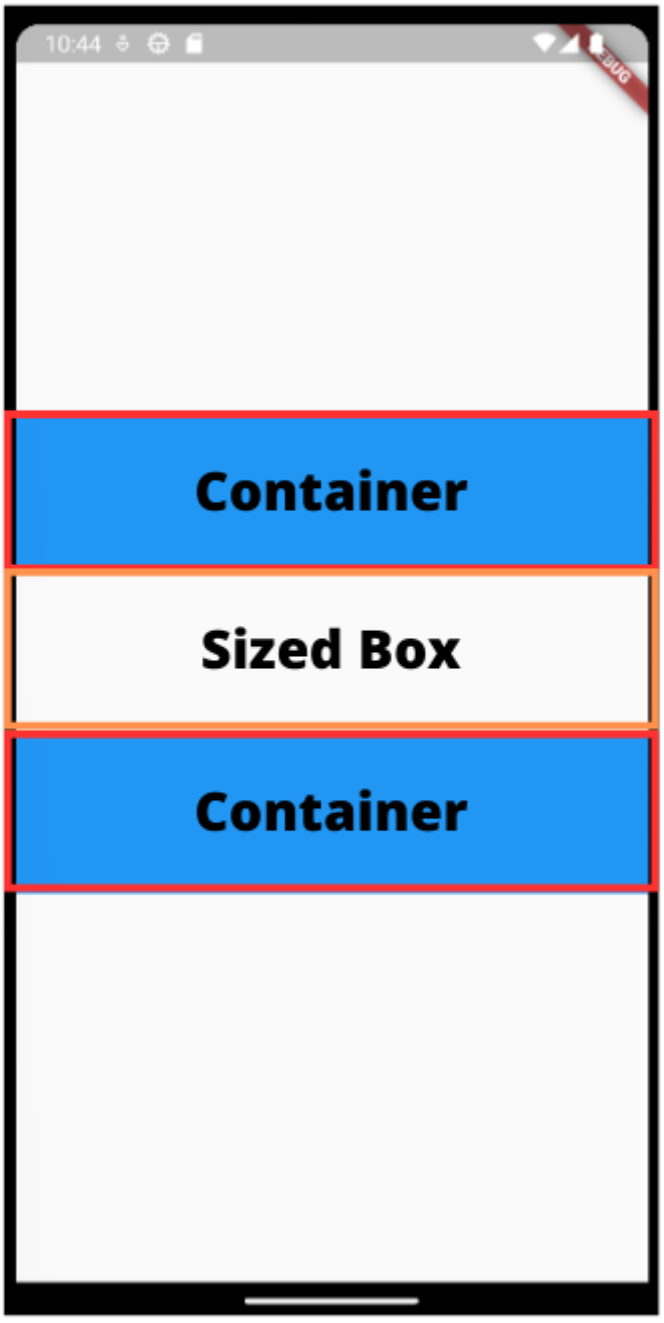


Figure 3.4: SizedBox Details

Penjelasan

1. `width:...` - Properti ini mengatur lebar kotak yang akan dibuat.
2. `height:...` - Properti ini mengatur tinggi kotak yang akan dibuat.

3.1.6 Text

Widget dalam Flutter yang digunakan untuk menampilkan teks.

```
Text('Hello world!')
```

3.1.7 Button

Widget dalam Flutter yang digunakan untuk membuat elemen yang dapat diinteraksi oleh pengguna. Flutter menyediakan beberapa jenis button yang dapat digunakan, seperti ElevatedButton, TextButton, IconButton, FloatingActionButton, dan lainnya.

```
ElevatedButton(
    onPressed: () {
    print('Tombol ditekan!');
    },
    child: Text('Tombol'),
),
IconButton(
    onPressed: () {
    print('Tombol ditekan!');
    },
    icon: Icon(Icons.radio_button_checked),
),
TextButton(
    onPressed: () {
    print('Tombol ditekan!');
    },
    // text di dalam button
    child: Text('Tombol'),
),
floatingActionButton: FloatingActionButton(
    onPressed: () {},
    child: Text('Tombol'),
),
```

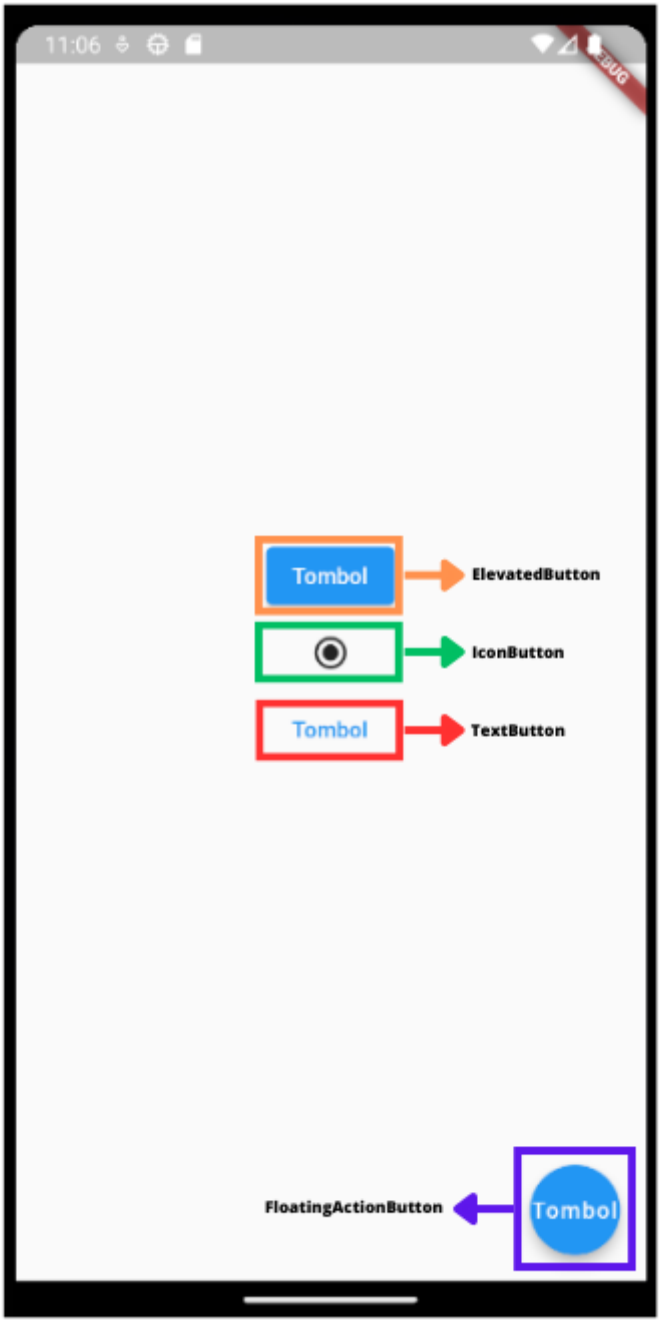


Figure 3.5: Button Details

Penjelasan

1. `ElevatedButton`: Ini adalah tombol dengan tampilan yang sedikit lebih "mengangkat" atau menonjol daripada tombol standar. Saat tombol ini ditekan, logika yang diberikan di dalam fungsi `onPressed` akan dijalankan. `child` merupakan properti yang digunakan untuk menambahkan konten teks ke dalam tombol.

2. `IconButton`: Ini adalah tombol yang berisi sebuah ikon. Ikon tersebut diberikan melalui properti `icon`. Ketika tombol ini ditekan, logika di dalam fungsi `onPressed` akan dijalankan.

3. `TextButton`: Ini adalah tombol dengan tampilan teks sederhana. Properti `child` digunakan untuk menentukan teks yang akan ditampilkan di dalam tombol. Saat tombol ini ditekan, logika di dalam fungsi `onPressed` akan dijalankan.

4. `FloatingActionButton`: Ini adalah tombol tindakan yang mengambang (floating action button) yang umumnya digunakan untuk tindakan utama dalam aplikasi. Dalam contoh ini, tombol ini tidak memiliki logika yang ditentukan dalam fungsi `onPressed`. `child` digunakan untuk menambahkan teks "Tombol" ke dalam tombol tersebut.

3.2 Layout

Layout dalam Flutter mengacu pada tata letak atau susunan widget di antarmuka pengguna. Dalam Flutter, tata letak dapat diatur menggunakan berbagai widget dan komponen yang tersedia untuk mengatur posisi, ukuran, dan hubungan antara elemen-elemen dalam antarmuka. Berikut adalah beberapa contoh layout umum dalam Flutter:

3.2.1 Column

Layout ini mengatur widget secara vertikal, mulai dari atas ke bawah.

```
Column(
    mainAxisAlignment: MainAxisAlignment.center,
    crossAxisAlignment: CrossAxisAlignment.end,
    children: const <Widget>[
        Text('Sebuah Judul'),
        Text('Lorem ipsum dolor sit amet'),
    ],
),
```

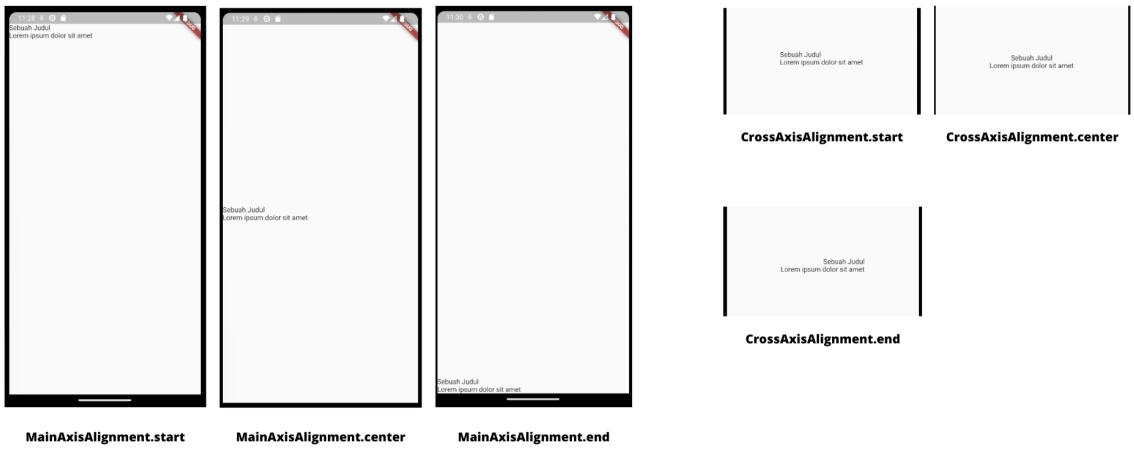


Figure 3.6: Column Details

Penjelasan

1. `mainAxisAlignment:...` : Atribut ini digunakan untuk mengatur perataan utama (main axis alignment) dari widget-column.
2. `crossAxisAlignment:...` : Atribut ini digunakan untuk mengatur perataan lintang (cross axis alignment) dari widget-column.
3. `children: const <Widget>[...]`: Ini adalah daftar widget yang akan ditampilkan di dalam column secara vertikal. Penggunaan `const` menandakan bahwa daftar widget ini merupakan konstanta yang tetap, dan tidak akan berubah selama runtime.

3.2.2 Row

Layout ini mengatur widget secara horizontal, mulai dari kiri ke kanan.

```
Row(
    mainAxisAlignment: MainAxisAlignment.start,
    crossAxisAlignment: CrossAxisAlignment.start,
    children: const <Widget>[
        Icon(Icons.share),
        Icon(Icons.thumb_up),
        Icon(Icons.thumb_down),
    ],
)
```

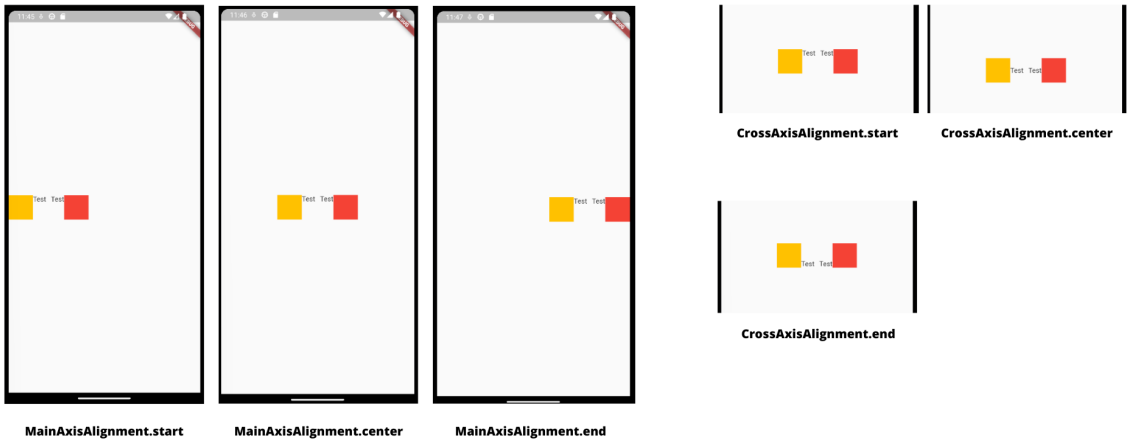


Figure 3.7: Row Details

Penjelasan

1. `mainAxisAlignment:...` : menentukan posisi utama (main axis) dari elemen-elemen di dalam Row.
2. `crossAxisAlignment:...` : menentukan posisi silang (cross axis) dari elemen-elemen di dalam Row.
3. `children` : merupakan properti yang berisi daftar elemen-elemen yang ingin ditampilkan dalam Row.
4. `<Widget>[...]` menandakan bahwa tipe elemen yang ditampilkan dalam `children` adalah Widget.
5. `const` digunakan untuk mengindikasikan bahwa daftar elemen dalam `children` adalah konstan, artinya tidak akan berubah.

3. ListView

Layout ini mengatur widget-widget didalamnya dalam tampilan daftar item yang dapat di-scroll secara vertikal. Widget ini berguna untuk kumpulan data yang ingin ditampilkan dalam bentuk daftar, seperti daftar kontak, daftar pesan, atau daftar produk.

```
ListView(
    children: [
        // ....

        Container(
        color: Colors.amber,
```

```
        width: double.infinity,
        height: 100,
        ),
        Container(
        color: Colors.blueAccent,
        width: double.infinity,
        height: 100,
        ),
        Container(
        color: Colors.brown,
        width: double.infinity,
        height: 100,
        ),

        // ...
    ],
)
```

Penjelasan

1. Properti `children` adalah properti wajib pada `ListView` yang berisi daftar widget yang akan ditampilkan sebagai elemen dalam daftar.

4. GridView

Layout ini digunakan untuk menampilkan daftar item dalam bentuk grid. Grid ini dapat berisi item-item yang ditampilkan secara berbaris dan berkolom. Layout ini berguna ketika menampilkan data dalam tata letak grid yang teratur.

```
GridView(
    gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
        crossAxisCount: 2,
        mainAxisSpacing: 10.0,
        crossAxisSpacing: 10.0,
    ),
    children: <Widget>[
        Container(color: Colors.red),
        Container(color: Colors.green),
        Container(color: Colors.blue),
        Container(color: Colors.yellow),
    ],
)
```

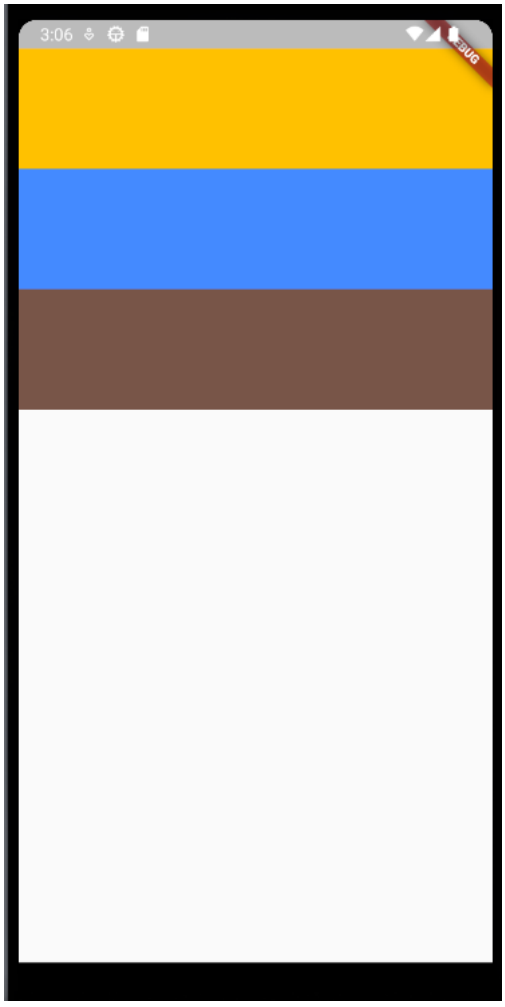


Figure 3.8: Listview Details

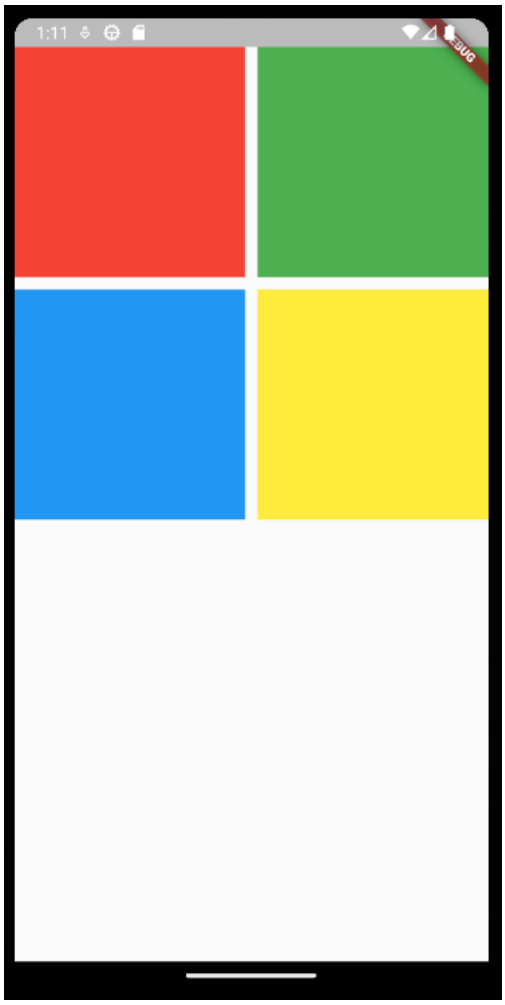


Figure 3.9: Gridview Details

Penjelasan

1. `gridDelegate`: Properti ini mengatur tata letak dan konfigurasi grid. Pada kode di atas, digunakan `SliverGridDelegateWithFixedCrossAxisCount` yang merupakan salah satu implementasi dari `SliverGridDelegate`. `crossAxisCount` menentukan jumlah kolom dalam grid. `mainAxisSpacing` mengatur ruang antara item secara vertikal, sedangkan `crossAxisSpacing` mengatur ruang antara item secara horizontal.

2. `children`: Properti ini berisi daftar widget yang akan ditampilkan dalam grid. Widget ini akan ditampilkan dalam bentuk grid sesuai dengan konfigurasi yang diberikan oleh `gridDelegate`.

5. Expanded

Layout ini digunakan untuk mengalokasikan ruang yang tersisa dalam tata letak yang fleksibel. Expanded memungkinkan widget untuk memperluas dan mengisi ruang yang tersedia sebanyak mungkin, sesuai dengan aturan penempatan dan proporsi yang ditentukan.

```
Row(
    children: [
        Container(
        color: Colors.red,
        height: 100,
        width: 100,
        ),
        Expanded(
        child: Container(
            color: Colors.blue,
        ),
        ),
        Container(
        color: Colors.green,
        height: 100,
        width: 100,
        ),
    ],
)
```

3.3 Styling

Styling dalam Flutter mengacu pada pengaturan penampilan visual dari widget. Dalam Flutter, berbagai properti gaya seperti warna, ukuran, jenis huruf, latar belakang, dan sebagainya dapat diatur untuk memodifikasi penampilan widget.

Figure 3.10: Expanded Details

Berikut adalah beberapa contoh penggunaan styling dalam Flutter:

1. Pada Teks:

```
Text(
  'Hello, world!',
  style: TextStyle(
    color: Colors.blue,
    fontSize: 20,
    fontWeight: FontWeight.bold,
    fontStyle: FontStyle.italic,
    letterSpacing: 2,
    wordSpacing: 5,
    background: Paint()..color = Colors.yellow,
    decoration: TextDecoration.underline,
    decorationColor: Colors.red,
    decorationStyle: TextDecorationStyle.dashed,
  ),
)
```

Penjelasan

1. `color:...` : digunakan untuk mengatur warna teks.
2. `fontSize:...` : digunakan untuk mengatur ukuran font teks.
3. `fontWeight:...` : digunakan untuk mengatur ketebalan font teks.
4. `fontStyle:...` : digunakan untuk mengatur gaya font teks.
5. `letterSpacing:...` : digunakan untuk mengatur jarak antar huruf pada teks.
6. `wordSpacing:...` : digunakan untuk mengatur jarak antar kata pada teks.
7. `background: Paint()..color = ...` : digunakan untuk memberikan latar belakang pada teks.
8. `decoration: TextDecoration.underline` : digunakan untuk menambahkan garis bawah pada teks.
9. `decorationColor:...` : digunakan untuk mengatur warna garis bawah pada teks.
10. `decorationStyle:...` : digunakan untuk mengatur garis bawah pada teks.

2. Pada Container:

```
Container(
  width: 200,
  height: 200,
  decoration: BoxDecoration(
    color: Colors.blue,
    borderRadius: BorderRadius.circular(10),
    boxShadow: [
```

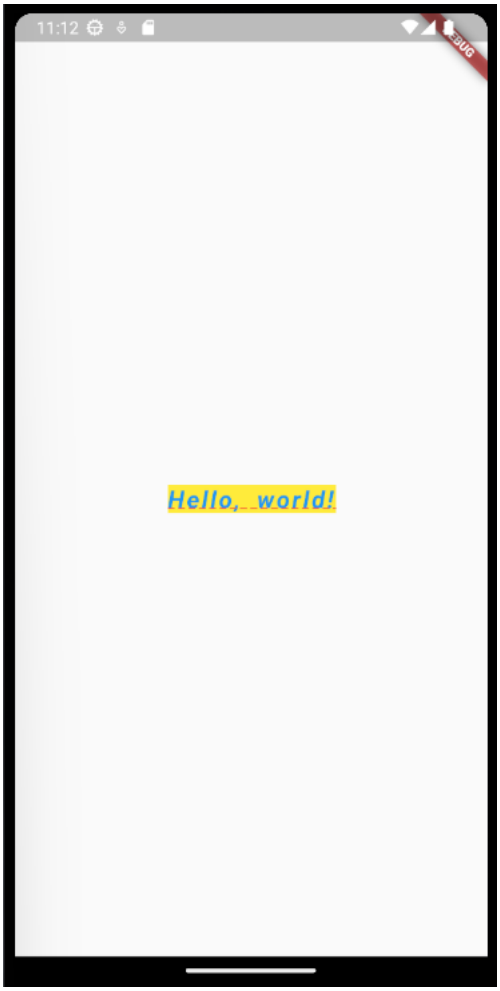


Figure 3.11: Font Styling Details

```
      BoxShadow(
        color: Colors.grey,
        blurRadius: 10,
        offset: Offset(5, 5),
      ),
    ],
    gradient: LinearGradient(
      begin: Alignment.topLeft,
      end: Alignment.bottomRight,
      colors: [Colors.blue, Colors.green],
    ),
  ),
  padding: EdgeInsets.all(16),
  margin: EdgeInsets.all(16),
  alignment: Alignment.center,
  child: Text(
    'Hello, world!',
    style: TextStyle(
      color: Colors.white,
      fontSize: 20,
      fontWeight: FontWeight.bold,
    ),
  ),
)
```

Penjelasan

Dalam `BoxDecoration`, terdapat beberapa properti yang digunakan untuk mengatur dekorasi pada `Container`, yaitu:

- `color:...`: digunakan untuk mengatur warna latar belakang `Container`.
- `borderRadius:...`: digunakan untuk mengatur radius border `Container`.
- `boxShadow:...`: digunakan untuk memberikan bayangan pada `Container`.
- `gradient: LinearGradient(...)`: digunakan untuk mengatur warna gradient pada `Container`.

3. Pada Elevated Button

```
ElevatedButton(
  style: ButtonStyle(
    backgroundColor: MaterialStateProperty.all<Color>(Colors.blue),
    shape: MaterialStateProperty.all<RoundedRectangleBorder>(
```

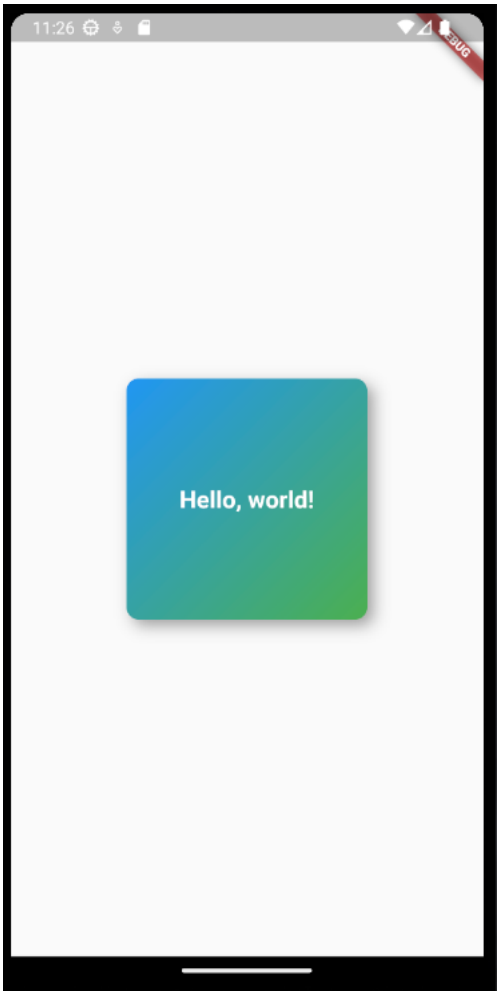


Figure 3.12: Container Styling Details

```
        RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(18.0),
        ),
      ),
    ),
    onPressed: () {
      // lakukan sesuatu saat tombol ditekan
    },
    child: Text(
      'Click me!',
      style: TextStyle(
        color: Colors.white,
        fontSize: 20,
        fontWeight: FontWeight.bold,
      ),
    ),
  )
```

Penjelasan

1. `style: ButtonStyle()`: Properti ini digunakan untuk mengatur gaya (style) dari tombol. Di dalamnya, terdapat beberapa properti yang bisa dikonfigurasi.
2. `backgroundColor: MaterialStateProperty.all<Color>(Colors.blue)`: Properti ini digunakan untuk mengatur warna latar belakang (background color) tombol.
3. `shape: MaterialStateProperty.all<RoundedRectangleBorder>(RoundedRectangleBorder(borderRadius: BorderRadius.circular(18.0)))`: Properti ini digunakan untuk mengatur bentuk (shape) tombol. Di contoh ini, sebuah `RoundedRectangleBorder` digunakan dengan jari-jari sudut sebesar 18.0.
4. `onPressed: () { ... }`: Properti ini digunakan untuk menentukan aksi yang dilakukan saat tombol ditekan. Di dalam tanda kurung kurawal ({…}), Anda dapat menulis kode untuk menjalankan aksi tertentu.

3.4 Handling Input

1. GestureDetector:

GestureDetector adalah widget yang dapat digunakan untuk mendeteksi dan menangani berbagai jenis gestur pengguna. Widget ini dapat digunakan untuk menangkap ketukan, gesekan, geseran, dan gestur lainnya.

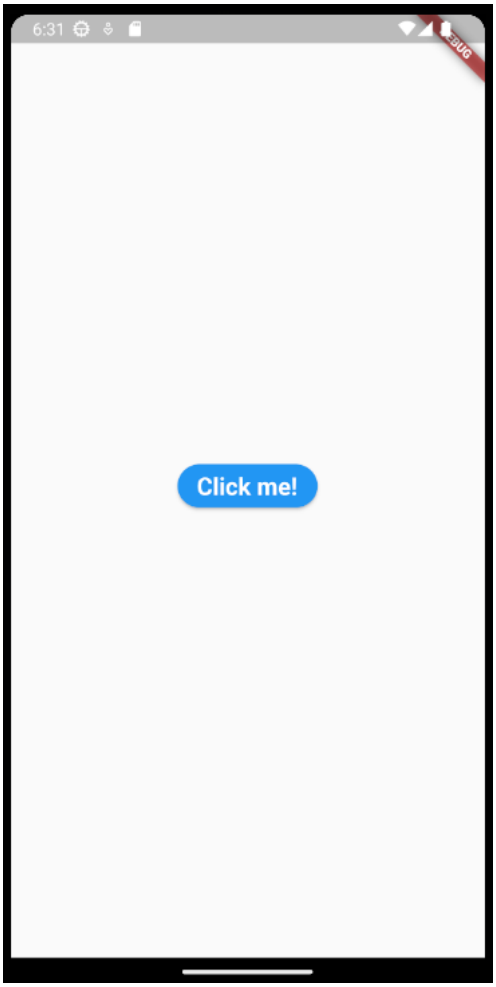


Figure 3.13: Button Styling Details

```
GestureDetector(
    onTap: () {
       print('Tap!');
    },
    onDoubleTap: () {
       print('Double tap!');
    },
    onLongPress: () {
       print('Long tap!');
    },
    onSwipeUp: () {
       print('Swipe Up!');
    },
    child: Container(
    alignment: Alignment.center,
    width: 200,
    height: 200,
    color: Colors.grey[300],
    child: Text(
        _message,
        textAlign: TextAlign.center,
        style: TextStyle(fontSize: 20),
    ),
    ),
)
```

Penjelasan

1. `onTap`: Properti ini menentukan aksi yang akan dilakukan ketika pengguna melakukan satu ketukan (tap) pada area yang diberikan.
2. `onDoubleTap`: Properti ini menentukan aksi yang akan dilakukan ketika pengguna melakukan dua ketukan (double tap) pada area yang diberikan.
3. `onLongPress`: Properti ini menentukan aksi yang akan dilakukan ketika pengguna menahan lama (long press) pada area yang diberikan.
4. `onSwipeUp`: Properti ini menentukan aksi yang akan dilakukan ketika pengguna melakukan geseran ke atas (swipe up) pada area yang diberikan.
5. `child`: Properti ini digunakan untuk menentukan widget yang akan diletakkan di dalam `GestureDetector`.

2. InkWell:

InkWell adalah widget yang memberikan umpan balik visual saat pengguna menyentuhnya. Ketika pengguna menyentuh InkWell, widget ini akan menampilkan animasi “splash” atau

efek “ink”.

```
InkWell(
    onTap: () {
       print('Tap!');
    },
    onLongPress: () {
       print('Long tap!');
    },
    child: Container(
    alignment: Alignment.center,
    width: 200,
    height: 200,
    color: Colors.grey[300],
    child: Text(
        _message,
        textAlign: TextAlign.center,
        style: TextStyle(fontSize: 20),
    ),
    ),
)
```

Penjelasan

1. `onTap`: Properti ini adalah sebuah fungsi yang akan dieksekusi ketika pengguna mengetuk widget yang dibungkus oleh InkWell.
2. `onLongPress`: Properti ini juga merupakan sebuah fungsi yang akan dieksekusi ketika pengguna menahan lama (long press) pada widget yang dibungkus oleh InkWell.
3. `child`: Properti ini adalah widget lain yang akan ditampilkan di dalam InkWell.

3.5 Studi Kasus

Dalam studi kasus ini, kita akan membuat sebuah aplikasi satu halaman yang menampilkan daftar makanan dalam bentuk daftar yang terlihat di layar. Berikut adalah tampilan aplikasi yang dihasilkan dari studi kasus ini.

Tahap-tahap pembuatan studi kasus adalah sebagai berikut:

1. Buatlah projek flutter baru.
2. Buka file main.dart dan hapus kode template yang ada di dalamnya, kemudian gantikan dengan kode berikut ini:

Figure 3.14: Hasil

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

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

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

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: Scaffold(
        body: Center(
          child: Text('Hello World!'),
        ),
      ),
    );
  }
}
```

3. Buatlah sebuah file baru dengan nama "home_page.dart" di dalam direktori "lib". Pada file ini, tampilan utama dari aplikasi akan disimpan, sehingga membuat kode menjadi lebih mudah dibaca dan dikelola.

4. Selanjutnya buat sebuah class baru bernama `HomePage` pada file `home_page.dart` dengan tipe `StatelessWidget`.

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

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

  @override
  Widget build(BuildContext context) {
    return const Placeholder();
  }
}
```

5. Setelah itu kembali ke dalam file "main.dart", ubah kode pada bagian "body" menjadi "HomePage()". Dengan melakukan perubahan tersebut, tampilan di emulator akan mengikuti konten yang ada di halaman utama (HomePage). Selain itu, tambahkan judul pada aplikasi dengan menggunakan komponen AppBar.

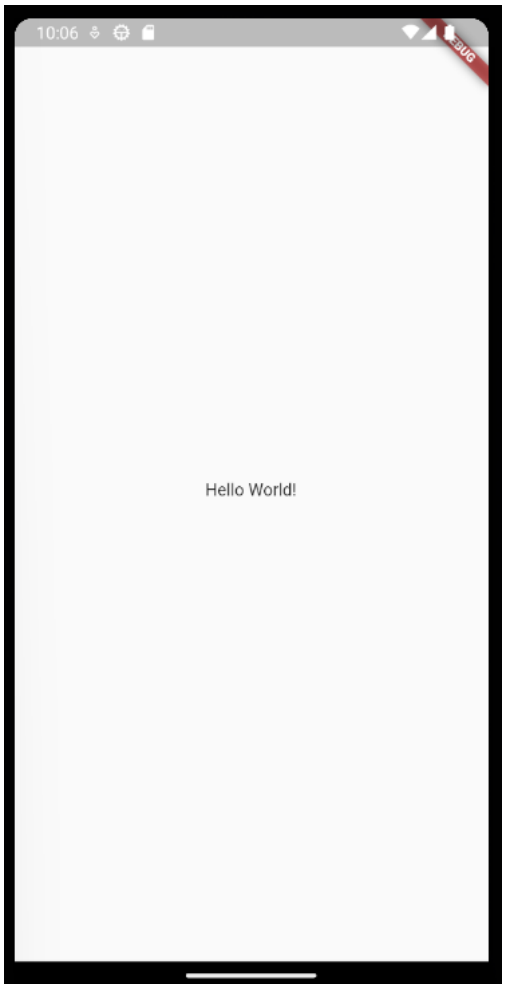


Figure 3.15: Tampilan langkah ke 2

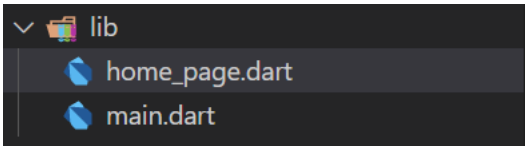


Figure 3.16: Tampilan langkah ke 3

```
Scaffold(
    appBar: AppBar(
        title: Text("List Kuliner"),
    ),
    body: HomePage(),
),
```

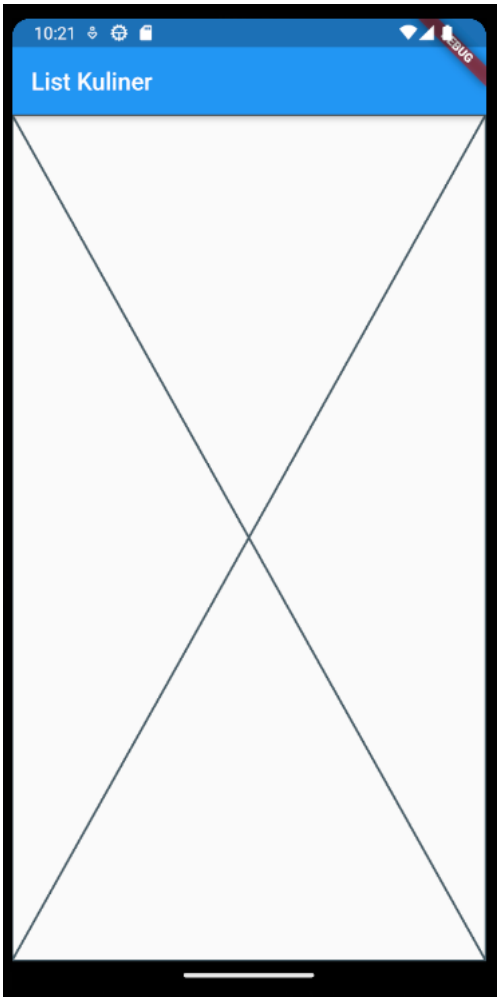


Figure 3.17: Langkah ke 5

6. Buka file "home_page.dart" dan gantilah placeholder dengan widget ListView. Selanjutnya, tambahkan widget Container di dalam ListView tersebut. Sesuaikan gaya tampilan (styling) pada ListView dan Container sesuai dengan preferensi Anda. Tampilan ini akan menjadi dasar dari aplikasi studi kasus yang sedang dibangun.

```
ListView(
    padding: EdgeInsets.all(10),
```

```
    children: [
        Container(
            margin: EdgeInsets.symmetric(vertical: 5),
            decoration: BoxDecoration(
                color: Colors.blueAccent,
                borderRadius:
                BorderRadius.all(Radius.circular(10)),
                boxShadow: [
                BoxShadow(
                    color: Colors.black,
                    offset: Offset(3.0, 5.0),
                    blurRadius: 2.0,
                ),
                ],
            ),
            height: 100,
            padding: EdgeInsets.symmetric(
                horizontal: 10,
                vertical: 15,
            ),
        ),
    ],
),
```

7. Buatlah beberapa widget di dalam sebuah container, menggunakan widget Column atau Row, untuk mengubah tampilan container menjadi sebuah kartu (card) yang dapat digunakan untuk menyimpan data makanan. Sesuaikan juga gaya (styling) seperti margin, padding, dan warna.

```
Container(
    //...

    child: Row(
        crossAxisAlignment: CrossAxisAlignment.center,
        children: [
        Container(
            width: 75,
            height: 75,
            color: Colors.black,
        ),
        SizedBox(
            width: 10,
        ),
        Column(
```

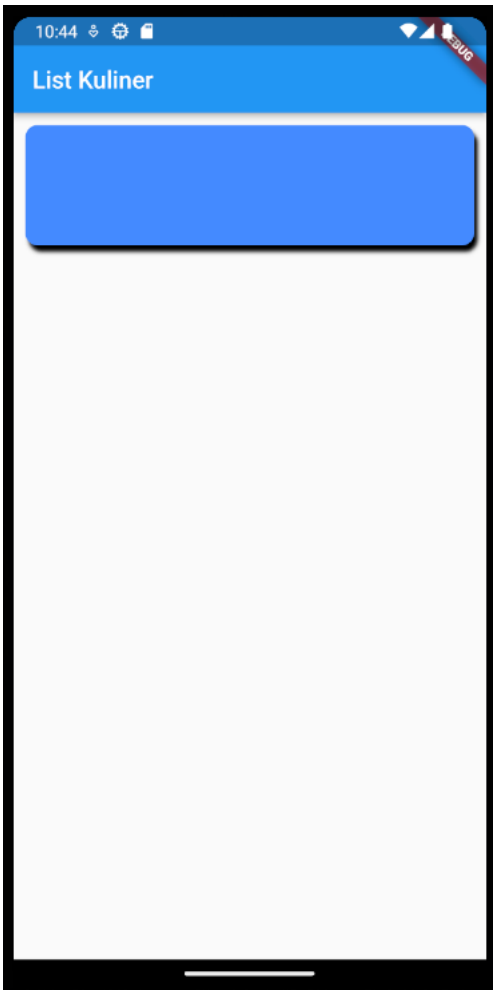


Figure 3.18: Langkah ke 6

```
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
            Text(
                "Title",
                style: TextStyle(
                fontSize: 30,
                fontWeight: FontWeight.bold,
                ),
            ),
            Text(
                "Sub Title",
                style: TextStyle(
                fontWeight: FontWeight.w500,
                ),
            ),
            ],
        )
        ],
    ),
),
```

8. Untuk mempermudah pembacaan kode, buatlah sebuah file baru bernama "list_item.dart". Di dalam file tersebut, buatlah sebuah class baru dengan nama "ListItem" yang merupakan turunan dari StatelessWidget. Kemudian, pindahkan seluruh kode yang berada di dalam container ke dalam class ListItem tersebut.

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

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

  @override
  Widget build(BuildContext context) {
    return Container(
      //...
    );
  }
}
```

Pastikan untuk menyertakan Widget ListItem() di dalam ListView pada file home_page.dart agar kontainer dapat ditampilkan dengan benar.

```
ListView(
    padding: EdgeInsets.all(10),
```

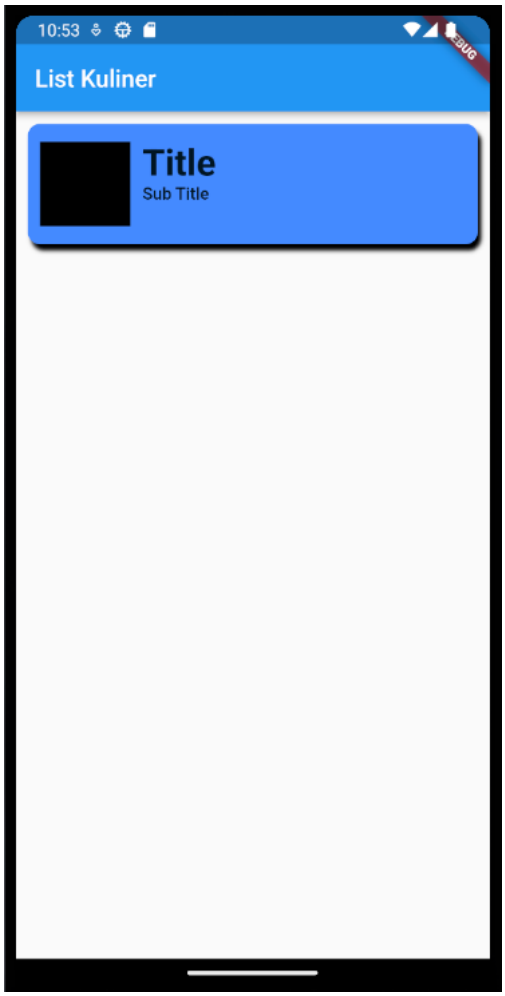


Figure 3.19: Langkah ke 7

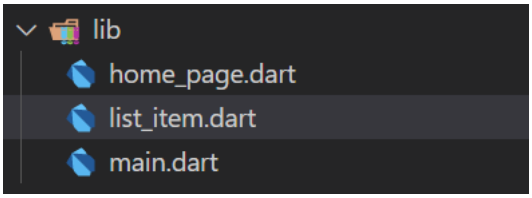


Figure 3.20: Langkah ke 8

```
    children: [
        ListItem(),
    ],
),
```

9. Samapai pada tahap ke 8 tampilan statis dari aplikasi telah selesai. Langkah selanjutnya adalah membuat data dan menampilkannya dalam aplikasi.

Pertama-tama Anda perlu membuat file baru bernama makanan.dart untuk menyimpan kelas makanan beserta atribut-atributnya.

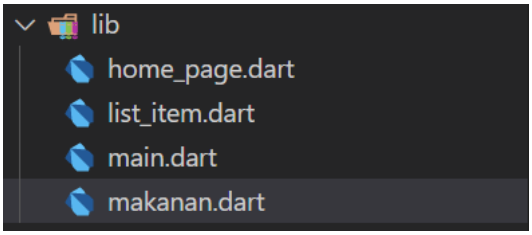


Figure 3.21: Langkah ke 9

```
class Makanan {
  final String nama;
  final String deskripsi;
  final String gambar;

  Makanan({
    required this.nama,
    required this.deskripsi,
    required this.gambar,
  });
}
```

Penjelasan

Kode tersebut adalah definisi dari sebuah kelas (class) yang disebut “Makanan”. Kelas ini memiliki tiga properti (properties) yaitu “nama”, “deskripsi”, dan “gambar”, yang semuanya memiliki tipe data String. Properti-properti tersebut ditandai dengan kata kunci “final” yang menunjukkan bahwa setelah objek Makanan dibuat, nilai-nilai properti tersebut tidak dapat diubah.
Selain itu, kelas Makanan juga memiliki sebuah constructor dengan sintaksis yang sedikit berbeda. Constructor ini menggunakan named parameters (parameter yang diberi nama) dengan menggunakan kurung kurawal {}. Constructor ini memiliki tiga parameter yaitu “nama”, “deskripsi”, dan “gambar”, dan ketiga parameter tersebut ditandai dengan kata kunci “required” yang menunjukkan bahwa nilai-nilai parameter

tersebut harus disediakan saat membuat objek Makanan.

10. Sebelum membuat daftar makanan yang akan ditampilkan, langkah pertama adalah mengunduh gambar-gambar yang akan digunakan. Untuk itu, buatlah folder khusus bernama "assets" di dalam proyek Anda dan simpan gambar-gambar tersebut di dalam folder tersebut.

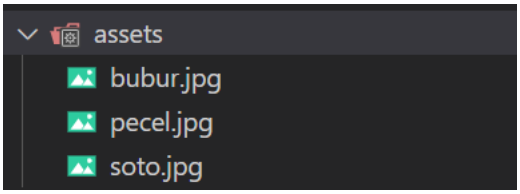


Figure 3.22: Langkah ke 10

Selanjutnya, Anda perlu menambahkan folder "assets" pada file pubspec.yaml agar gambar yang terdapat di dalamnya dapat diakses oleh aplikasi.

```yaml
flutter:
  uses-material-design: true
  assets:
    - assets/
```

Figure 3.23: Langkah ke 10

11. Buatlah sebuah file bernama "makanan.dart" yang berisi sebuah list array untuk menyimpan data makanan yang akan di-load dalam aplikasi.

```dart
List<Makanan> list_makanan = [
  Makanan(
    nama: 'Bubur',
    deskripsi: 'Nasi Lembek',
    gambar: 'assets/bubur.jpg',
  ),
  Makanan(
    nama: 'Soto',
    deskripsi: 'Makanan berkuah',
    gambar: 'assets/soto.jpg',
  ),
  Makanan(
    nama: 'Pecel',
    deskripsi: 'Sayuran dengan bumbu kacang',
    gambar: 'assets/pecel.jpg',
  ),
```

```
];
```

Penjelasan

1. Deklarasi List: `List<Makanan> list_makanan = [...]`
    - `List<Makanan>` mengindikasikan bahwa `list_makanan` adalah sebuah List yang berisi objek-objek bertipe `Makanan`.
    - `[]` menandakan bahwa List tersebut akan diinisialisasi dengan sejumlah elemen objek Makanan.
2. Inisialisasi elemen-elemen List:
    - Setiap elemen dalam List adalah sebuah objek Makanan yang diinisialisasi dengan menggunakan sintaksis `Makanan(...)`.
    - Objek Makanan memiliki tiga atribut: `nama`, `deskripsi`, dan `gambar`. Nilai-nilai atribut tersebut diberikan melalui parameter nama-nama yang sesuai dalam sintaksis inisialisasi objek.

12. Silakan tambahkan atribut yang sesuai dalam ListItem() sesuai dengan atribut yang ada dalam kelas makanan. Gantikan widget container yang berada di dalam row dengan Image.asset(). Kemudian, ubah data dalam teks agar menggunakan atribut yang telah ditambahkan sebelumnya.

```
final String nama;
final String deskripsi;
final String gambar;
const ListItem(
    {super.key,
    required this.nama,
    required this.deskripsi,
    required this.gambar,
    });
```

```
Row(
    crossAxisAlignment: CrossAxisAlignment.center,
    children: [
        // widget untuk menampilkan gambar lokal
        Image.asset(
            gambar,
            height: 75,
            width: 75,
        ),
        SizedBox(
        width: 10,
```

```
        ),
        Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
            Text(
            nama,
            style: TextStyle(
                fontSize: 30,
                fontWeight: FontWeight.bold,
            ),
            ),
            Text(
            deskripsi,
            style: TextStyle(
                fontWeight: FontWeight.w500,
            ),
            ),
        ],
        )
    ],
),
```

13. Silakan buka kembali file home_page.dart dan ubah bagian yang menampilkan ListView menjadi menggunakan ListView.builder.

```
ListView.builder(
    // mengatur panjang / jumlah item dalam list
    itemCount: list_makanan.length,
    padding: EdgeInsets.all(10),
    itemBuilder: (context, index) {
        return ListItem(
        nama: list_makanan[index].nama,
        deskripsi: list_makanan[index].deskripsi,
        gambar: list_makanan[index].gambar,
        );
    },
),
```

Penjelasan

1. `itemCount: list_makanan.length,`: Ini mengatur jumlah item dalam daftar, yang diambil dari panjang (`length`) dari daftar `list_makanan`. Jadi, jumlah item dalam daftar akan sesuai dengan jumlah elemen dalam `list_makanan`.
2. `itemBuilder: (context, index) { ... },`: Ini adalah fungsi yang digunakan

untuk membangun tampilan item dalam daftar. Fungsi ini akan dipanggil secara berulang untuk setiap item dalam daftar.

- `context`: Objek `context` yang memberikan informasi tentang lingkungan tampilan saat ini.
- `index`: Indeks item saat ini dalam daftar.

3. `return ListItem(...);`: Di dalam `itemBuilder`, setiap item dalam daftar dibangun dengan menggunakan widget `ListItem`. Nilai-nilai yang diperlukan untuk membangun `ListItem` diambil dari `list_makanan` dengan menggunakan `index` saat ini. Ini berarti setiap item dalam daftar akan memiliki nama, deskripsi, dan gambar yang sesuai dengan elemen dalam `list_makanan`

Setelah melalui beberapa tahapan, aplikasi studi kasus akhirnya selesai dan siap digunakan. Anda memiliki kebebasan untuk menambahkan data baru atau mengubah tampilan sesuai dengan preferensi Anda.

4 Proyek Kalkulator

Proyek Kalkulator

Aplikasi Kalkulator adalah proyek Flutter yang bertujuan untuk membuat sebuah kalkulator fungsional yang dapat digunakan untuk melakukan operasi matematika dasar. Aplikasi ini akan menyediakan antarmuka pengguna yang intuitif dan responsif untuk memudahkan pengguna dalam melakukan perhitungan.

4.1 Fitur Utama:

1. Tampilan Antarmuka Pengguna:
    - Aplikasi akan memiliki tampilan antarmuka pengguna yang terdiri dari tombol-tombol angka, operator, dan fungsi matematika seperti tambah, kurang, kali, bagi, dan sebagainya.
    - Tampilan akan dirancang agar mudah digunakan dan memberikan pengalaman pengguna yang intuitif.
2. Perhitungan Matematika:
    - Pengguna dapat melakukan perhitungan matematika dasar seperti penjumlahan, pengurangan, perkalian, dan pembagian.
    - Aplikasi akan memiliki logika yang memungkinkan pengguna melakukan perhitungan secara berurutan, menggabungkan beberapa operasi, dan menghasilkan hasil yang akurat.
3. Desain Responsif:
    - Aplikasi akan memiliki desain yang responsif dan bisa diakses dengan baik di berbagai perangkat dengan ukuran layar yang berbeda.

4.2 Teknologi yang Digunakan:

- Flutter sebagai kerangka kerja untuk pengembangan aplikasi mobile lintas platform.
- Dart sebagai bahasa pemrograman untuk mengembangkan logika aplikasi.
- Widget dan State Management dari Flutter untuk mengatur tampilan dan interaksi pengguna.
- Matematika dasar menggunakan operator dan fungsi bawaan dari Dart.

Tujuan dari proyek ini adalah memberikan pengguna sebuah kalkulator yang dapat digunakan dengan mudah, memiliki performa yang baik, dan memberikan hasil perhitungan yang akurat. Aplikasi Kalkulator ini akan menjadi alat yang berguna untuk keperluan perhitungan sehari-hari, baik untuk kebutuhan pribadi maupun profesional.

Selama mengembangkan aplikasi Kalkulator, penting untuk memastikan bahwa logika perhitungan matematika benar, antarmuka pengguna responsif dan intuitif, serta mengikuti pedoman desain terbaik dari Flutter.

4.3 Contoh tampilan aplikasi:

Aplikasi kalkulator hanya memiliki satu halaman saja dengan tampilan kurang lebih seperti di bawah ini.

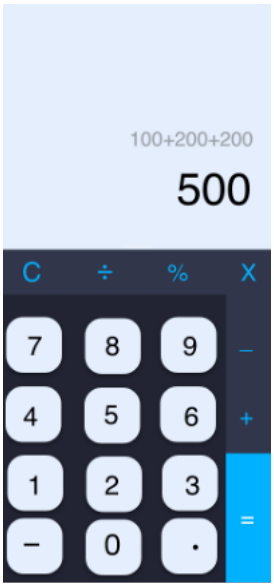


Figure 4.1: Contoh tampilan kalkulator

5 Routing dan Navigasi

Routing dan Navigasi

Membuat aplikasi yang lebih dari satu halaman dan menavigasikannya.

5.1 Navigasi Antar Halaman

5.2 Widget Navigator

5.3 Berbagi Data Antar Halaman

5.4 State Management

6 Pengujian dan Debugging

Pengujian dan Debugging

6.1 Debugging di Flutter

6.2 Menggunakan Unit Test untuk Uji Logika

6.3 Menggunakan Widget Test untuk Uji Widget

7 Akses Data Lokal

Akses Data Lokal

7.1 Konfigurasi dan Pengaturan Awal

7.2 CRUD Data dengan SQLite

7.3 Query Pencarian Database Lokal

8 Proyek Password Vault

Proyek Password Vault

Aplikasi **Password Vault** adalah sebuah proyek Flutter yang bertujuan untuk menyediakan tempat aman untuk menyimpan dan mengelola kata sandi (password) pengguna. Aplikasi ini akan menggunakan penyimpanan lokal untuk menyimpan data pengguna dan kata sandi yang dienkripsi.

8.1 Fitur Utama:

1. Registrasi dan Login Pengguna:
    - Pengguna dapat membuat akun baru dengan menyediakan email dan kata sandi.
    - Pengguna dapat masuk ke akun mereka dengan mengautentikasi email dan kata sandi.
2. Penyimpanan Data Kata Sandi:
    - Pengguna dapat menyimpan data kata sandi mereka dengan memasukkan judul, username, dan kata sandi untuk setiap entri.
    - Data kata sandi akan disimpan secara aman dalam penyimpanan lokal dengan menggunakan enkripsi.
3. Manajemen Data Kata Sandi:
    - Pengguna dapat melihat daftar entri kata sandi yang disimpan.
    - Pengguna dapat menambahkan, mengedit, atau menghapus entri kata sandi.
    - Pengguna dapat mencari entri kata sandi berdasarkan judul atau username.
4. Keamanan dan Enkripsi:
    - Data kata sandi disimpan dalam penyimpanan lokal dengan menggunakan teknik enkripsi yang aman.
    - Pengguna akan diminta memasukkan kata sandi master saat masuk ke aplikasi untuk membuka akses ke data kata sandi.
5. UI yang Responsif:
    - Aplikasi akan memiliki antarmuka pengguna yang responsif dan menarik, memungkinkan pengguna untuk dengan mudah menavigasi dan menggunakan fitur-fitur yang ada.

8.2 Teknologi yang Digunakan:

- Flutter sebagai kerangka kerja untuk pengembangan aplikasi mobile lintas platform.
- Package `sqflite` untuk mengakses dan menyimpan data dalam penyimpanan lokal SQLite.
- Package `encrypt` untuk melakukan enkripsi dan dekripsi data kata sandi.
- Package `flutter_secure_storage` untuk menyimpan kata sandi master dengan aman.

Tujuan dari proyek ini adalah memberikan pengguna aplikasi sebuah solusi yang aman dan mudah digunakan untuk mengelola kata sandi mereka. Aplikasi Password Vault ini akan memberikan perlindungan terhadap pencurian kata sandi dan memberikan akses cepat ke informasi penting yang dibutuhkan oleh pengguna dalam kehidupan digital mereka.

Catatan: Penting untuk selalu mengutamakan keamanan dalam pengembangan aplikasi semacam ini. Pastikan untuk menggunakan praktik terbaik dalam hal pengamanan dan enkripsi data yang sensitif.

8.3 Contoh tampilan aplikasi:

Aplikasi Password Vault memiliki beberapa halaman dengan tampilan kurang lebih seperti di bawah ini.

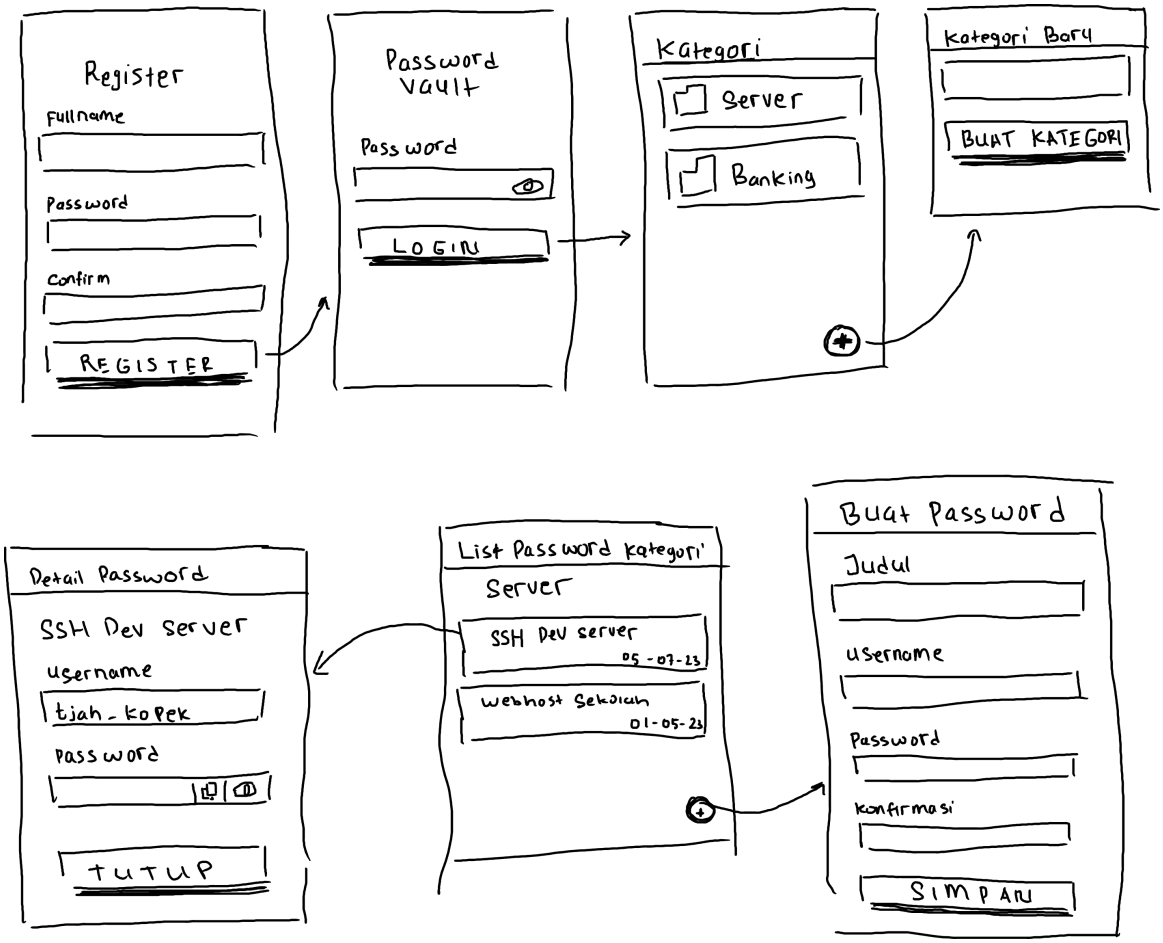


Figure 8.1: Rancangan tampilan password vault

9 Akses Data Melalui API

Akses Data Melalui API

9.1 Konsep Dasar REST API

9.2 Menggunakan Package HTTP

9.3 Mengambil Data dari REST API

9.4 Menampilkan Data pada Widget

10 Integrasi dengan BAaS

Integrasi dengan BAaS

10.1 Konsep BAaS dan Serverless Application

10.2 Pengenalan Supabase

10.3 Integrasi Supabase dengan Flutter

10.4 Otentikasi dengan Supabase

10.5 Pengunaan Supabase untuk Penyimpanan Data

11 Layanan Berbasis Lokasi

Layanan Berbasis Lokasi

11.1 Menggunakan Package Location

11.2 Mengunakan Widget Geolocator

11.3 Menampilkan OpenStreetMap API pada Aplikasi

12 Pengamanan Aplikasi

Pengamanan Aplikasi

12.1 Konsep Protokol Jaringan Aman dengan HTTPS

12.2 Autentikasi Pengguna

12.3 Obfuscating Dart Code

13 Proyek Lapor Book

Proyek Lapor Book

References