

# **BAB 1. INSTALASI DAN PENGENALAN FLUTTER**

## **1.1 Apa itu Flutter**

Flutter adalah sebuah framework open source yang dibuat oleh Google. Google membuat flutter dengan tujuan membangun sebuah framework untuk membuat UI yang modern, native dan reactive yang dapat berjalan di sistem operasi iOS maupun Android. Tidak hanya pada smartphone google juga membuat flutter untuk desktop, web dan embedded device.

Flutter diprogram dengan menggunakan bahasa Dart sebuah bahasa moderen yang dapat dicompile ke arsitektur processor ARM atau javascript. Flutter menggunakan Skia 2D rendering engine yang dapat bekerja pada hardware atau software yang berbeda platform.

Dart menggunakan metode compilasi ahead of time (AOT) untuk mengubah kode Dart menjadi kode native untuk sistem operasi yang digunakan, oleh karena itu aplikasi yang dibangun menggunakan flutter memiliki kecepatan yang hampir sama dengan aplikasi native. Dart juga menggunakan konsep just-in-time (JIT) sehingga memungkinkan programmer dapat membuat perubahan pada kode program dan langsung melihat hasilnya melalui fitur hot reload yang dimiliki Flutter.

Flutter menggunakan Dart untuk membuat User Interface, sehingga memudahkan dalam membuat aplikasi karena menggunakan satu bahasa (Dart) dalam pembuatan UI maupun logika program. Flutter menggunakan pendekatan declarative dimana Flutter membangun UI mengikuti “State” yang dimiliki oleh aplikasi. Ketika state berubah maka UI akan digambar ulang .

Flutter juga memudahkan programmer karena dari satu kode program dapat dikompilasi ke kode native ARM, menggunakan GPU dan mengakses fitur spesifik dari smartphone baik yang menggunakan sistem operasi iOS ataupun yang menggunakan sistem operasi Android. Jadi dengan satu kali membuat program dapat membuat 2 aplikasi yang sama untuk sistem operasi yang berbeda (iOS atau Android).

## **1.2 Widget dan Element pada Flutter**

Gaya pengembangan aplikasi menggunakan flutter sedikit berbeda dengan gaya pengembangan aplikasi pada umumnya, dimana UI pada flutter dibuat menggunakan *Widget*. Widget adalah sebuah konsep dimana UI dapat dianggap sebagai sebuah balok LEGO, sebuah bentuk baru dapat disusun dari beberapa balok dan masing masing kumpulan balok dapat dikombinasikan dengan kumpulan balok lain sehingga membentuk sebuah bentuk baru yang lebih kompleks. Flutter menggunakan widget ini sebagai balok dasar pembangunan aplikasi.

Widget dapat disusun dan dikombinasikan dalam satu layar, sama halnya dengan xml pada pemrograman android native, widget dapat disusun dalam bentuk tree dimana satu widget menjadi parent dan widget lain menjadi child. Masing masing widget dapat diberikan konfigurasi sesuai dengan kebutuhan aplikasi.

Flutter memiliki dua jenis widget yaitu StatelessWidget dan StatefulWidget. StatelessWidget widget digunakan ketika value (state /konfigurasi) dari widget tersebut tidak pernah berubah, dan StatefulWidget digunakan ketika value (state / konfigurasi) dari widget dapat berubah. Baik StatelessWidget maupun StatefulWidget sama sama memiliki sebuah method bernama “build” yang memiliki BuildContext untuk mengatur posisi widget didalam widget tree detail mengenai widget dan bagaimana membuatnya akan dibahas pada bab selanjutnya.

### 1.3 Praktikum Installasi Flutter

#### 1.3.1 Alat dan Bahan

Pada praktikum kali ini anda akan melakukan installasi flutter pada sistem operasi Windows. Berikut ini Alat dan bahan yang dibutuhkan untuk menginstall flutter :

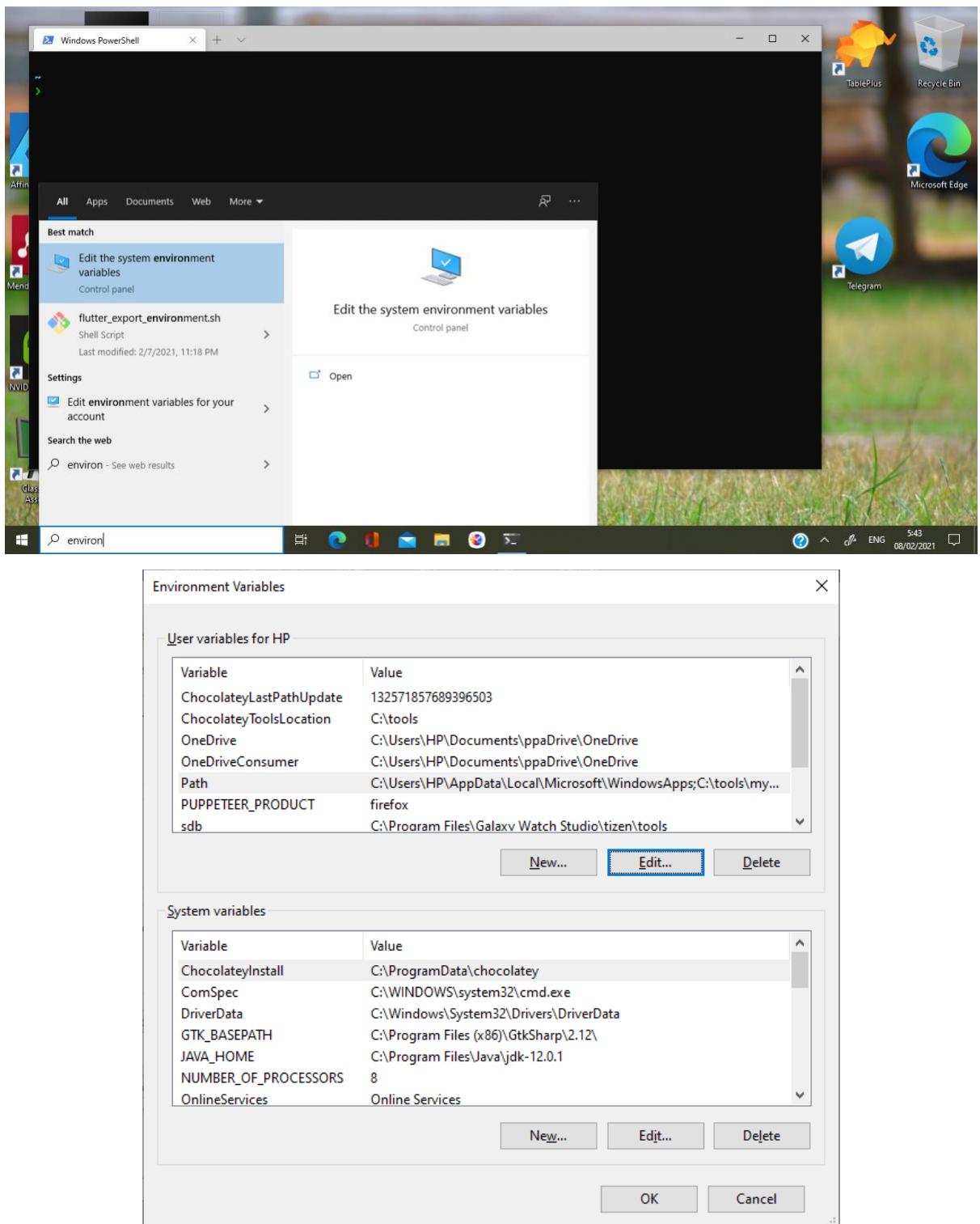
1. Sistem Operasi Windows minimal Windows 7 SP1 atau setelahnya 64 bit dan x86-64.
2. Harddisk 1,32 GB untuk Flutter saja selain IDE atau alat lain
3. Windows PowerShell 5.0 atau lebih baru <https://docs.microsoft.com/en-us/powershell/scripting/install/installing-windows-powershell>
4. Git For Windows 2.x <https://git-scm.com/download/win>

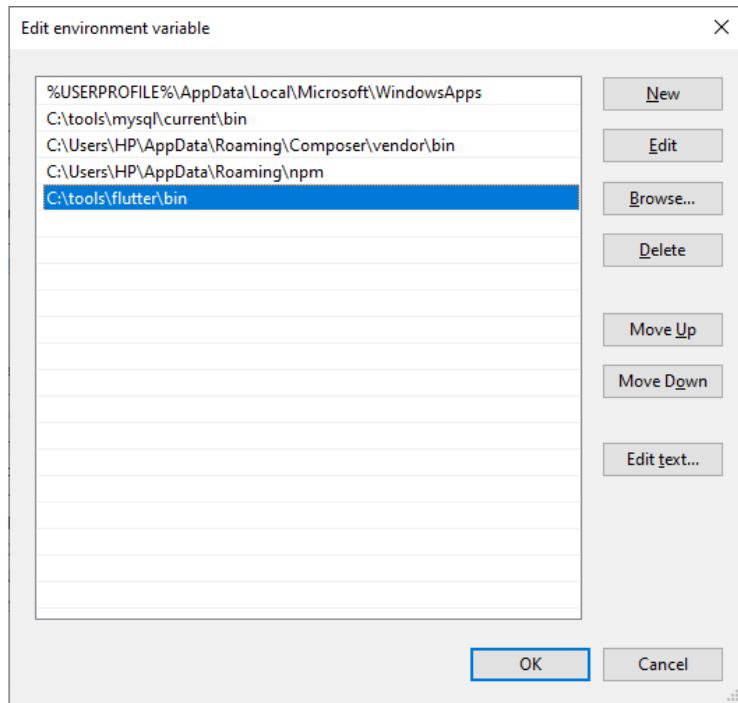
#### 1.3.2 Langkah Langkah Praktikum

1. Download Flutter SDK, pada saat modul praktikum ini dibuat SDK untuk windows ada di tautan berikut  
[https://storage.googleapis.com/flutter\\_infra/releases/stable/windows/flutter\\_windows\\_1.22.6-stable.zip](https://storage.googleapis.com/flutter_infra/releases/stable/windows/flutter_windows_1.22.6-stable.zip)
2. Extract file yang di download ke harddisk anda contoh lokasi ke C:\src\flutter (**JANGAN** di install ke folder C:\Program Files\ karena membutuhkan akses admin )
3. Atau jika anda sudah menginstall git buatlah folder src di dalam drive C kemudian buka terminal di folder tersebut dan ketik

```
git clone https://github.com/flutter/flutter.git -b stable
```

4. Update Windows PATH tambahkan path menuju folder C:\src\flutter\bin





5. Install Android Studio
6. Setup Android Device / Emulator
7. Install Visual Studio Code <https://code.visualstudio.com/>
8. Install Flutter Plugin
9. Validasi Install dengan mengetikkan perintah berikut di terminal

```
flutter doctor
```

```
Documents/2021/pemrogramanMobile
> flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[!] Flutter (Channel dev, 1.26.0-17.2.pre, on Microsoft Windows [Version 10.0.19041.746], locale en-ID)
[!] Android toolchain - develop for Android devices (Android SDK version 30.0.2)
  x Android license status unknown.
    Run 'flutter doctor --android-licenses' to accept the SDK licenses.
    See https://flutter.dev/docs/get-started/install/windows#android-setup for more details.
[!] Chrome - develop for the web (Cannot find Chrome executable at .\Google\Chrome\Application\chrome.exe)
  ! Cannot find Chrome. Try setting CHROME_EXECUTABLE to a Chrome executable.
[!] Android Studio (version 4.1.0)
[!] VS Code, 64-bit edition (version 1.53.0)
[!] Connected device (1 available)

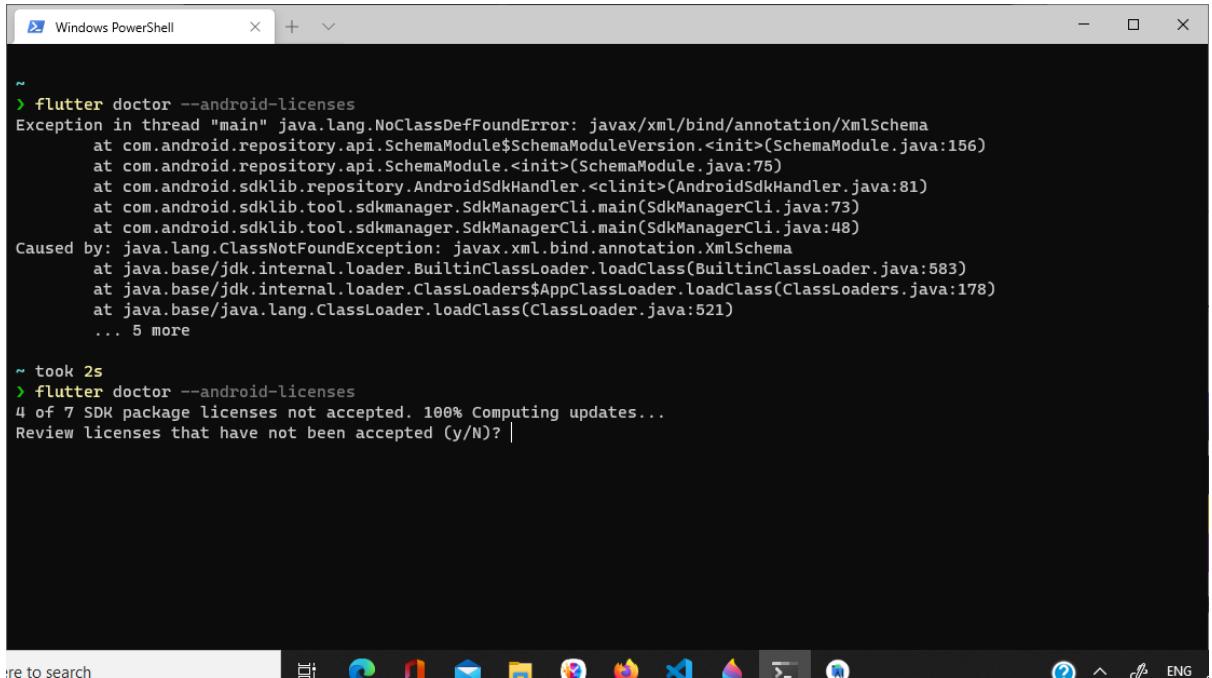
! Doctor found issues in 2 categories.

Documents/2021/pemrogramanMobile took 2s
> |
```

10. Jika perintah flutter doctor mengeluarkan hasil seperti gambar di atas maka anda perlu menerima lisensi android SDK dengan mengetikkan perintah berikut ini di terminal.

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

11. Jika tidak ada error lanjutkan ke langkah praktikum selanjutnya, jika muncul error seperti gambar dibawah ini.

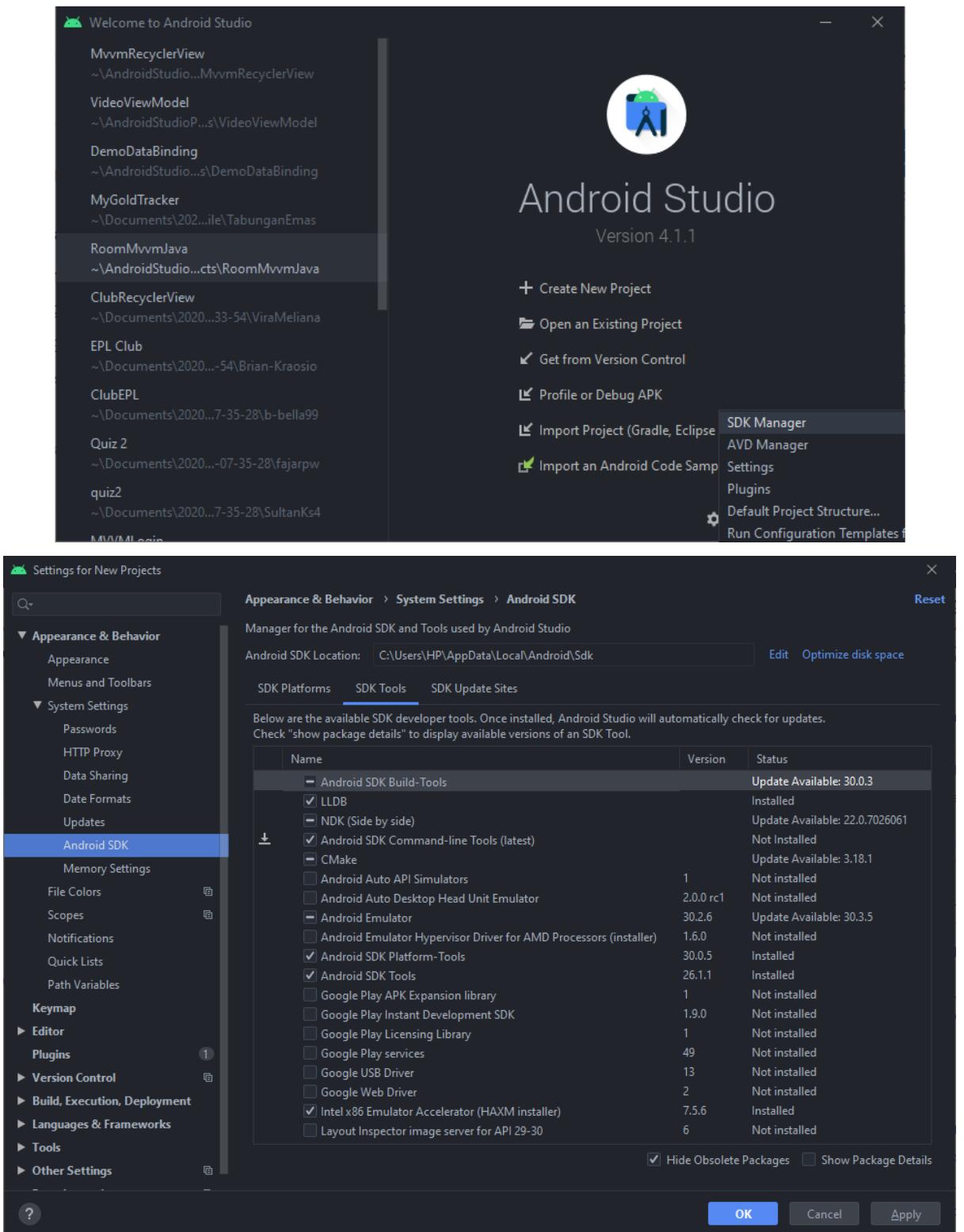


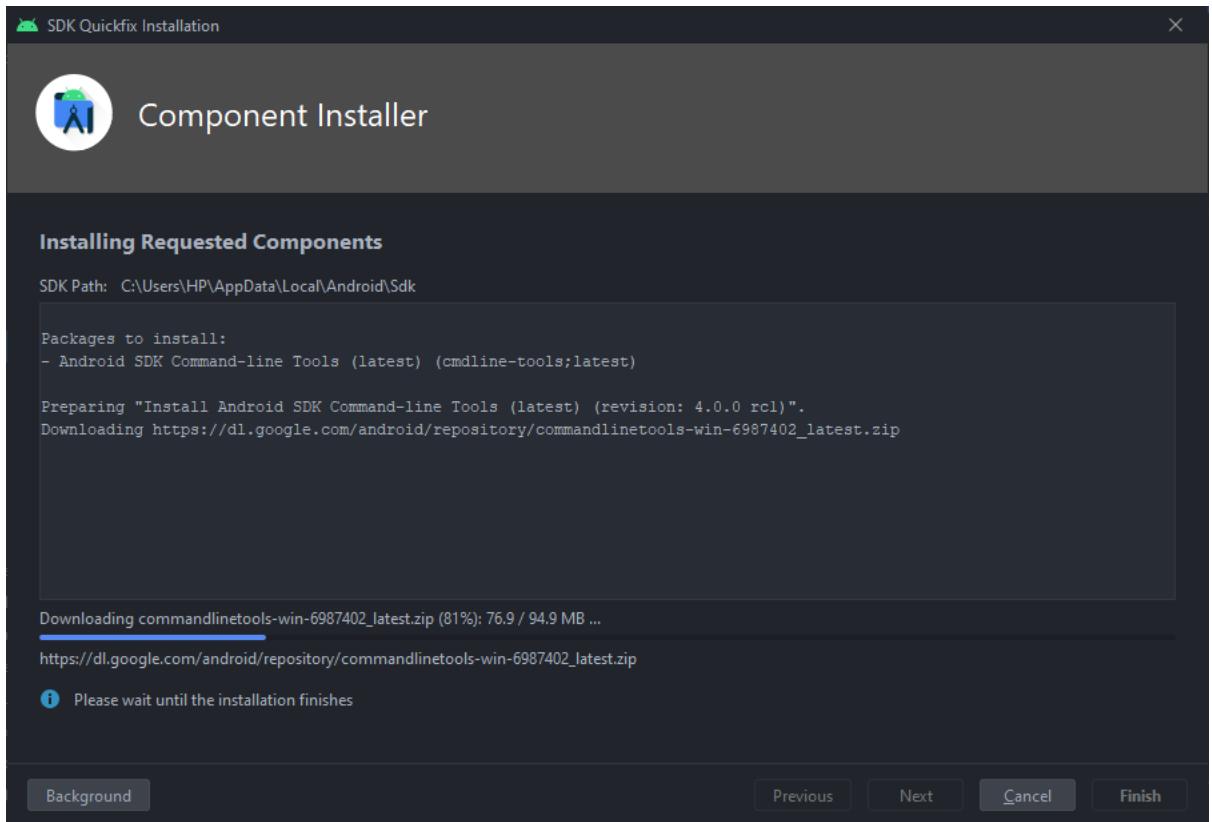
```
Windows PowerShell

~ > flutter doctor --android-licenses
Exception in thread "main" java.lang.NoClassDefFoundError: javax/xml/bind/annotation/XmlSchema
        at com.android.repository.api.SchemaModule$SchemaModuleVersion.<init>(SchemaModule.java:156)
        at com.android.repository.api.SchemaModule.<init>(SchemaModule.java:75)
        at com.android.sdklib.repository.AndroidSdkHandler.<clinit>(AndroidSdkHandler.java:81)
        at com.android.sdklib.tool.sdkmanager.SdkManagerCli.main(SdkManagerCli.java:73)
        at com.android.sdklib.tool.sdkmanager.SdkManagerCli.main(SdkManagerCli.java:48)
Caused by: java.lang.ClassNotFoundException: javax.xml.bind.annotation.XmlSchema
        at java.base/jdk.internal.loader.BuiltinClassLoader.loadClass(BuiltinClassLoader.java:583)
        at java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(ClassLoaders.java:178)
        at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:521)
        ... 5 more

~ took 2s
> flutter doctor --android-licenses
4 of 7 SDK package licenses not accepted. 100% Computing updates...
Review licenses that have not been accepted (y/N)? |
```

12. Lakukan update di Android SDK pada Android Studio dengan menginstall Android Command Line Tools pada android SDK langkah melakukannya lihat gamabar selanjutnya.





## 1.4 Praktikum Hello World

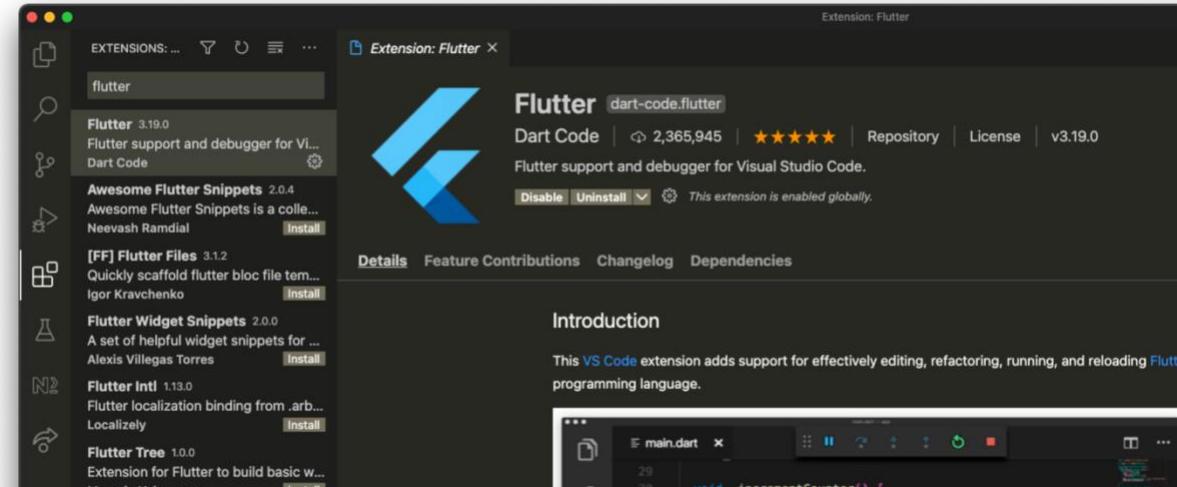
### 1.4.1 Alat dan Bahan

Pada praktikum kali ini anda akan membuat sebuah project hello world flutter

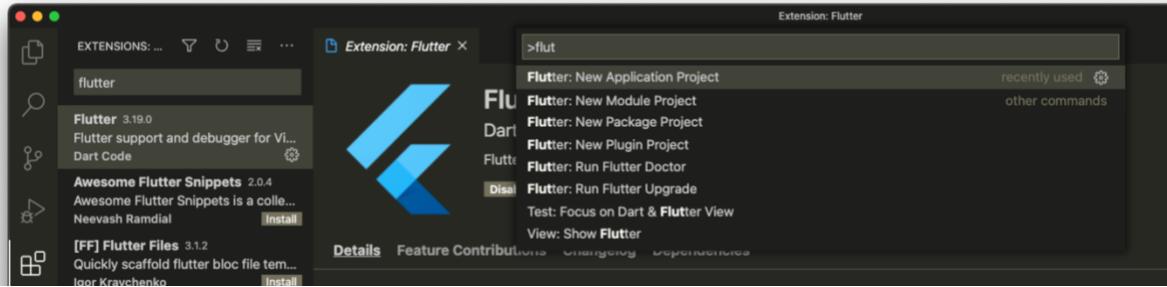
1. Visual Studio Code
2. Flutter SDK
3. Emulator

### 1.4.2 Langkah Langkah Praktikum

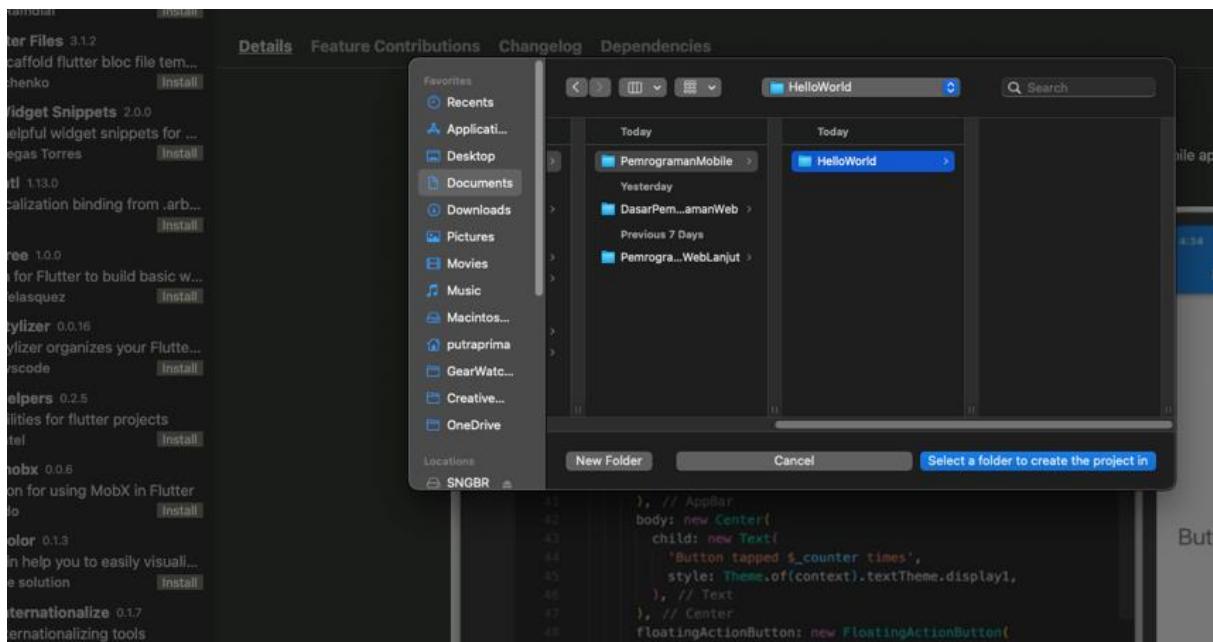
1. Install Flutter Plugin di Visual Studio Code



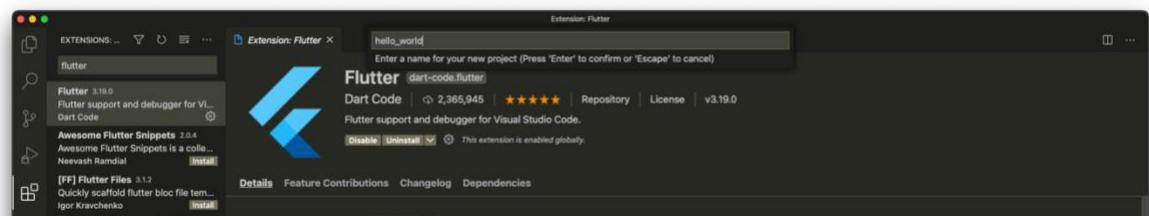
2. Tekan command Ctrl + Shift + P kemudian ketik perintah Flutter : New Application Project



3. Pilih Folder tempat project akan dibuat



4. Berilah nama project yang sesuai Flutter mewajibkan nama project menggunakan huruf kecil dan dipisahkan dengan underscore contoh : hello\_world



5. Berikut ini hasil project hello\_world jika berhasil di generate

```

main.dart - hello_world
main.dart X MyApp > build
1 import 'package:flutter/material.dart';
2
3 void main() {
4   runApp(MyApp());
5 }
6
7 class MyApp extends StatelessWidget {
8   // This widget is the root of your application.
9   @override
10  Widget build(BuildContext context) {
11    return MaterialApp(
12      title: 'Flutter Demo',
13      theme: ThemeData(
14        // This is the theme of your application.
15        //
16        // Try running your application with "flutter run". You'll see the
17        // application has a blue toolbar. Then, without quitting the app, try
18        // changing the primarySwatch below to Colors.green and then invoke
19        // "hot reload" (press "r" in the console where you ran "flutter run",
20        // or simply save your changes to "hot reload" in a Flutter IDE).
21        // Notice that the counter didn't reset back to zero; the application
22        // is not restarted.
23        primarySwatch: Colors.blue,
24        // This makes the visual density adapt to the platform that you run
25        // the app on. For desktop platforms, the controls will be smaller and
26        // closer together (more dense) than on mobile platforms.
27        visualDensity: VisualDensity.adaptivePlatformDensity,
28      ),
29      home: MyHomePage(title: 'Flutter Demo Home Page'),
30    );
31  }
32}
33
34 class MyHomePage extends StatefulWidget {
35   MyHomePage({Key key, this.title}) : super(key: key);
36
37   // This widget is the home page of your application. It is stateful, meaning
38   // that it has a State object (defined below) that contains fields that affect
39   // how it looks.
40
41   // This class is the configuration for the state. It holds the values (in this
42   // case the title) provided by the parent (in this case the App widget) and

```

Ln 29, Col 56 Spaces: 2 UTF-8 LF Dart Flutter: 1.22.4 No Device

6. Untuk menjalankan project pastikan anda sudah melakukan setup emulator / setup device sesuai dengan perintah pada praktikum sebelumnya.
7. Jika emulator / device sudah disetup tekan tombol F5 untuk menjalankan aplikasi ke emulator / device.

```

main.dart — hello_world
Select a device to use
<
dart > MyApp Start iOS Simulator mobile simulator
t 'package:flutter' Start Pixel 2 API 24 mobile emulator
Start Pixel_3a_API_30_x86 mobile emulator
ebug Create Android emulator
main() {
App(MyApp());

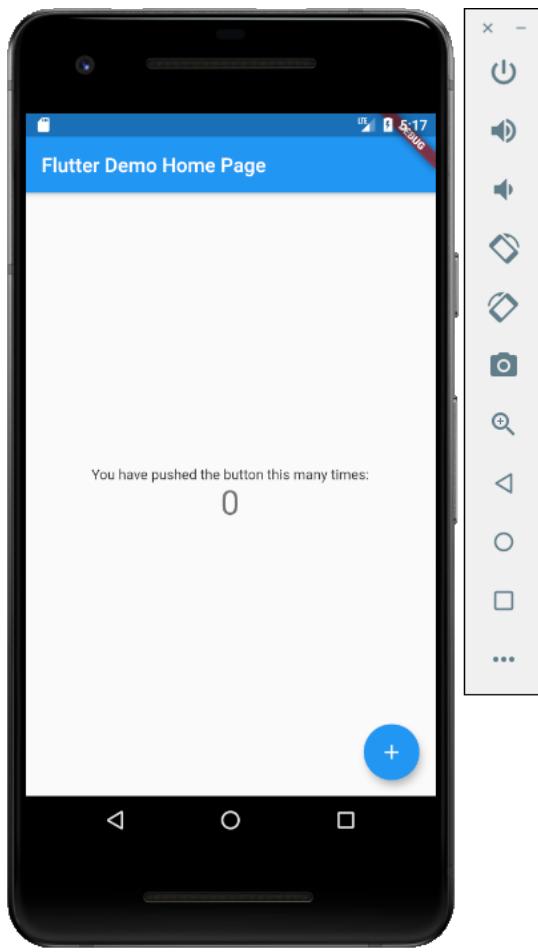
```

```

MyApp extends StatelessWidget {
This widget is the root of your application.
@override
get build(BuildContext context) {
return MaterialApp(

```

8. Pilih emulator / device dan klik enter.



## 1.5 Praktikum 2 Membuat Git Repository dan Publish Ke Github

Pada praktikum kali ini anda akan mempublikasikan project flutter anda ke public repository github.

### 1.5.1 Alat dan Bahan

1. Visual Studio Code
2. Git
3. Akun Github
4. Source Code Flutter hello\_world

### 1.5.2 Langkah Langkah praktikum

1. Mendaftar akun github ke <https://github.com/> , pilihlah username anda dengan bijaksana karena akun github anda akan sangat sering digunakan dan dapat dijadikan portofolio untuk mendaftar pekerjaan.
2. Update konfigurasi git di komputer anda sesuai dengan username dan email yang anda buat di github.com.
- 3.

```
git config --global user.name usernameAnda  
git config --global user.email emailAnda
```

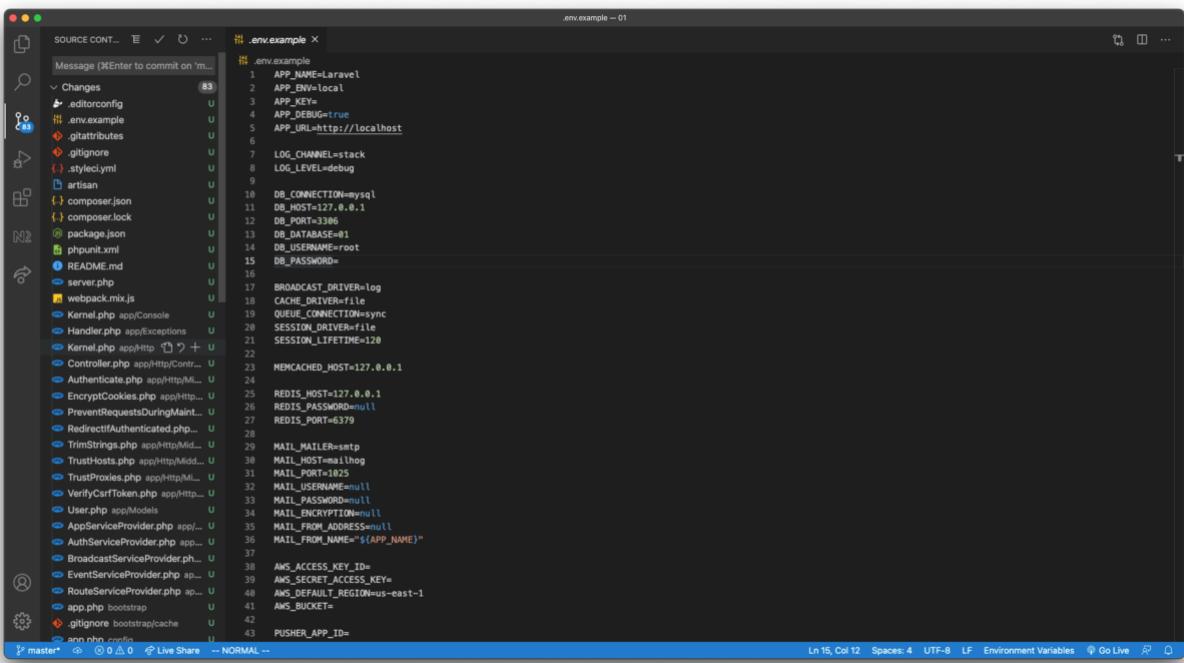
4. Bukalah project laravel yang anda buat sebelumnya pada Praktikum 2 menggunakan Visual Studio Code
5. Jika anda belum menginisialisasi repository git pada project tersebut akan keluar tampilan seperti ini ketika anda mengklik icon source control di sebelah kiri.

The screenshot shows the Visual Studio Code interface with the Source Control sidebar open. The sidebar has a dark theme and includes icons for file operations like Open, Save, and Publish. A message box is displayed, providing instructions on how to use Git and Source Control in VS Code, mentioning the 'Initialize Repository' and 'Publish to GitHub' buttons. The main editor area shows a file named '.env.example' with the following content:

```
APP_NAME=Laravel  
APP_ENV=local  
APP_KEY=  
APP_DEBUG=true  
APP_URL=http://localhost  
  
LOG_CHANNEL=stack  
LOG_LEVEL=debug  
  
DB_CONNECTION=mysql  
DB_HOST=127.0.0.1  
DB_PORT=3306  
DB_DATABASE=bd1  
DB_USERNAME=root  
DB_PASSWORD=  
  
BROADCAST_DRIVER=log  
CACHE_DRIVER=file  
QUEUE_CONNECTION=sync  
SESSION_DRIVER=file  
SESSION_LIFETIME=120  
  
MEMCACHED_HOST=127.0.0.1  
  
REDIS_HOST=127.0.0.1  
REDIS_PASSWORD=null  
REDIS_PORT=6379  
  
MAIL_MAILER=smtp  
MAIL_HOST=mailhog  
MAIL_PORT=1025  
MAIL_USERNAME=null  
MAIL_PASSWORD=null  
MAIL_ENCRYPTION=null  
MAIL_FROM_ADDRESS=null  
MAIL_FROM_NAME="${APP_NAME}"  
  
AWS_ACCESS_KEY_ID=  
AWS_SECRET_ACCESS_KEY=  
AWS_DEFAULT_REGION=us-east-1  
AWS_BUCKET=  
  
PUSHER_APP_ID=
```

At the bottom of the interface, there are status indicators for Live Share, Normal mode, and various file and environment settings.

6. Perhatikan ada dua tombol yaitu Initialize Repository dan Publish to Github, Klik lah tombol Initialize Repository. Sidebar akan berubah menjadi seperti berikut ini.



```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=
APP_DEBUG=true
APP_URL=http://localhost
LOG_CHANNEL=stack
LOG_LEVEL=debug
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=01
DB_USERNAME=root
DB_PASSWORD=
BROADCAST_DRIVER=log
CACHE_DRIVER=file
QUEUE_CONNECTION=sync
SESSION_DRIVER=file
SESSION_LIFETIME=120
MEMCACHED_HOST=127.0.0.1
REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379
MAIL_MAILER=smtp
MAIL_HOST=mailhog
MAIL_PORT=1025
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null
MAIL_FROM_ADDRESS=null
MAIL_FROM_NAME="${APP_NAME}"
AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
AWS_DEFAULT_REGION=us-east-1
AWS_BUCKET=
PUSHER_APP_ID=
```

7. Perhatikan pada langkah ini kita sudah membuat repository git dan git mulai melacak perubahan yang ada pada root folder project kita. Perhatikan dengan seksama di sidebar terdapat daftar file yang berubah dibawah dropdown menu Changes dan status file nya memiliki status U dengan warna hijau.
8. Klik dropdown changes ini sehingga daftar file yang berubah diminimzie, kemudian klik kanan dan klik menu “Stage All Changes”

```
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8 LOG_LEVEL=debug
9
10 DB_CONNECTION=mysql
11 DB_HOST=127.0.0.1
12 DB_PORT=3306
13 DB_DATABASE=01
14 DB_USERNAME=root
15 DB_PASSWORD=
16
17 BROADCAST_DRIVER=log
18 CACHE_DRIVER=file
19 QUEUE_CONNECTION=sync
20 SESSION_DRIVER=file
21 SESSION_LIFETIME=120
```

9. Setelah di klik git mencatat semua perubahan yang kita buat pada kode program dan sekarang semua file yang ada di Changes berpindah ke Dropdown Staged Changes.

The screenshot shows a Mac OS X desktop environment. In the foreground, a terminal window is open with the command `git commit` entered, and a message field is empty. In the background, a code editor (Visual Studio Code) is open, showing the `.env.example` configuration file for a Laravel application. The file contains the following environment variables:

```
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8 LOG_LEVEL=debug
9
10 DB_CONNECTION=mysql
11 DB_HOST=127.0.0.1
12 DB_PORT=3306
13 DB_DATABASE=01
14 DB_USERNAME=root
15 DB_PASSWORD=
16
17 BROADCAST_DRIVER=log
18 CACHE_DRIVER=file
19 QUEUE_CONNECTION=sync
20 SESSION_DRIVER=file
21 SESSION_LIFETIME=120
22
```

10. Isilah pesan tentang perubahan kode yang kita buat, pada kolom isian Message, kemudian klik tombol centang di atas.

The screenshot shows the Visual Studio Code interface. The top menu bar includes Code, File, Edit, Selection, View, Go, Run, Terminal, Window, and Help. The left sidebar features icons for Source Control, Search, Git (with 83 changes), Split Editor, and Activity. The main editor area displays a file named '.env.example' with the following content:

```
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8 LOG_LEVEL=debug
9
10 DB_CONNECTION=mysql
11 DB_HOST=127.0.0.1
12 DB_PORT=3306
13 DB_DATABASE=01
14 DB_USERNAME=root
15 DB_PASSWORD=
16
17 BROADCAST_DRIVER=log
18 CACHE_DRIVER=file
19 QUEUE_CONNECTION=sync
20 SESSION_DRIVER=file
21 SESSION_LIFETIME=120
22
```

11. Jika di klik menu “Staged Change” akan hilang dan semua perubahan sudah di catat oleh git.
12. Selamat anda sudah berhasil menggunakan git pada repository lokal.
13. Selanjutnya untuk melakukan publikasi repository lokal ke github lakukan perintah berikut ini tekan Ctrl + Shift + P kemudian ketik publish to github

welcome.blade.php — 01

```
<ip> >
```

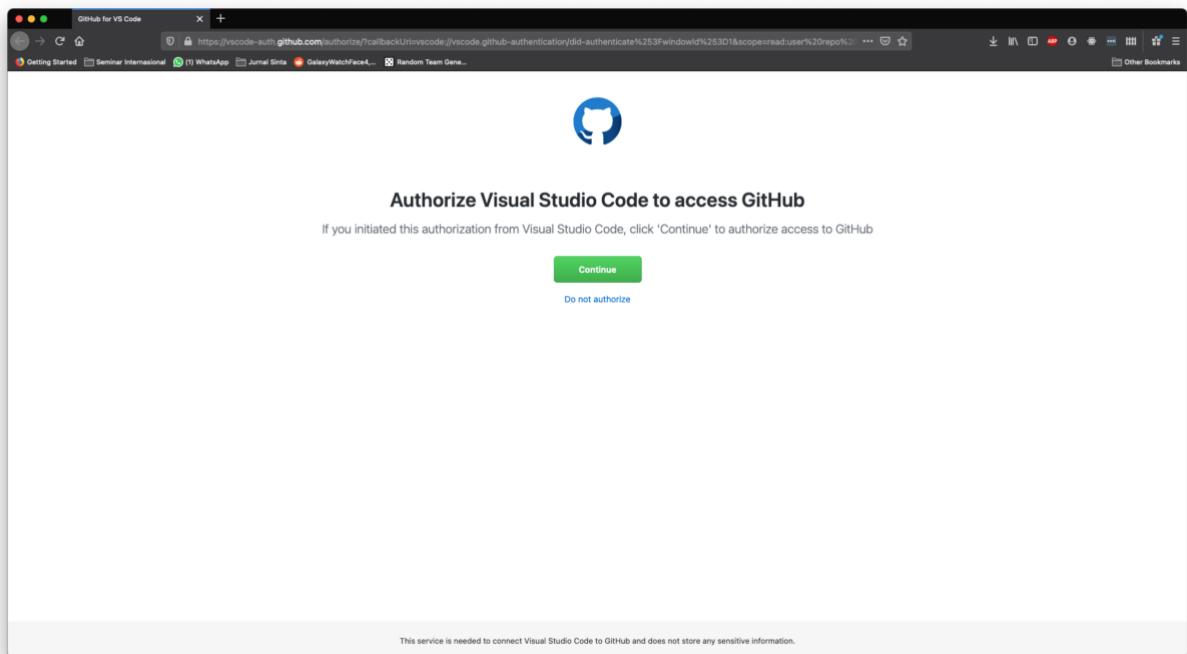
recently used ⚙

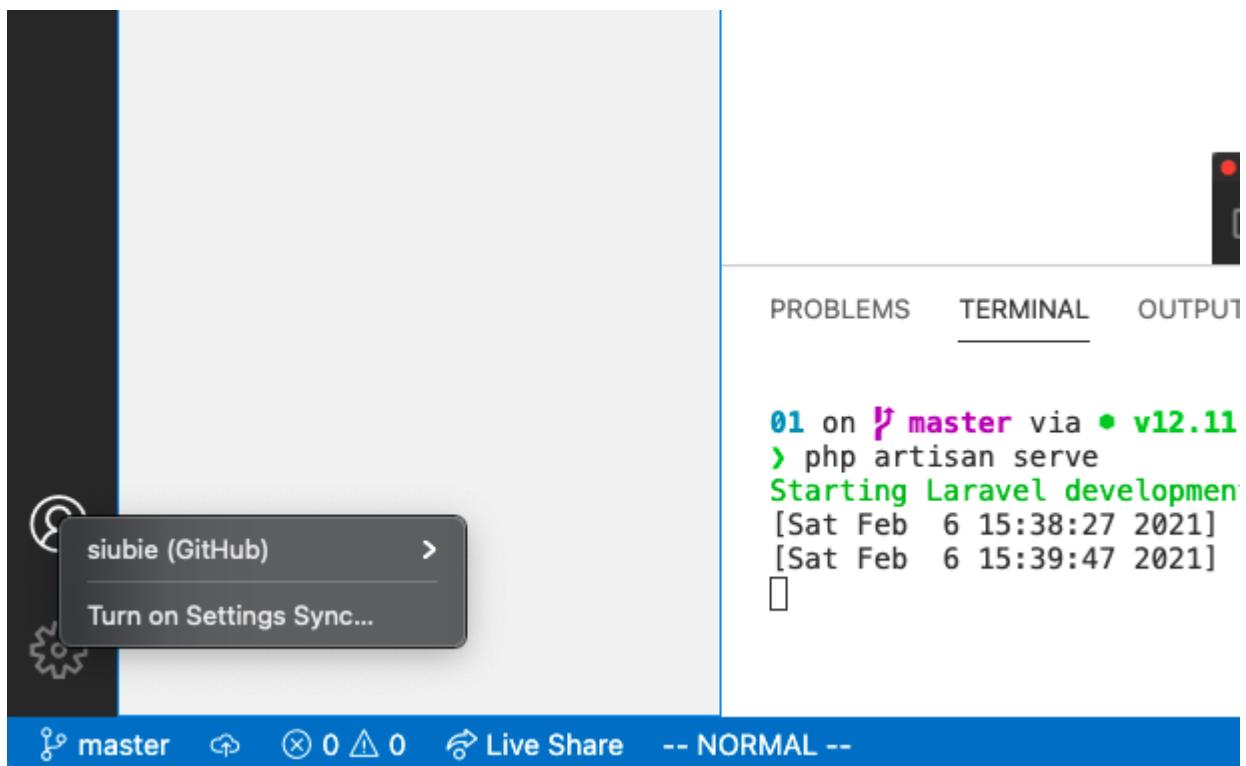
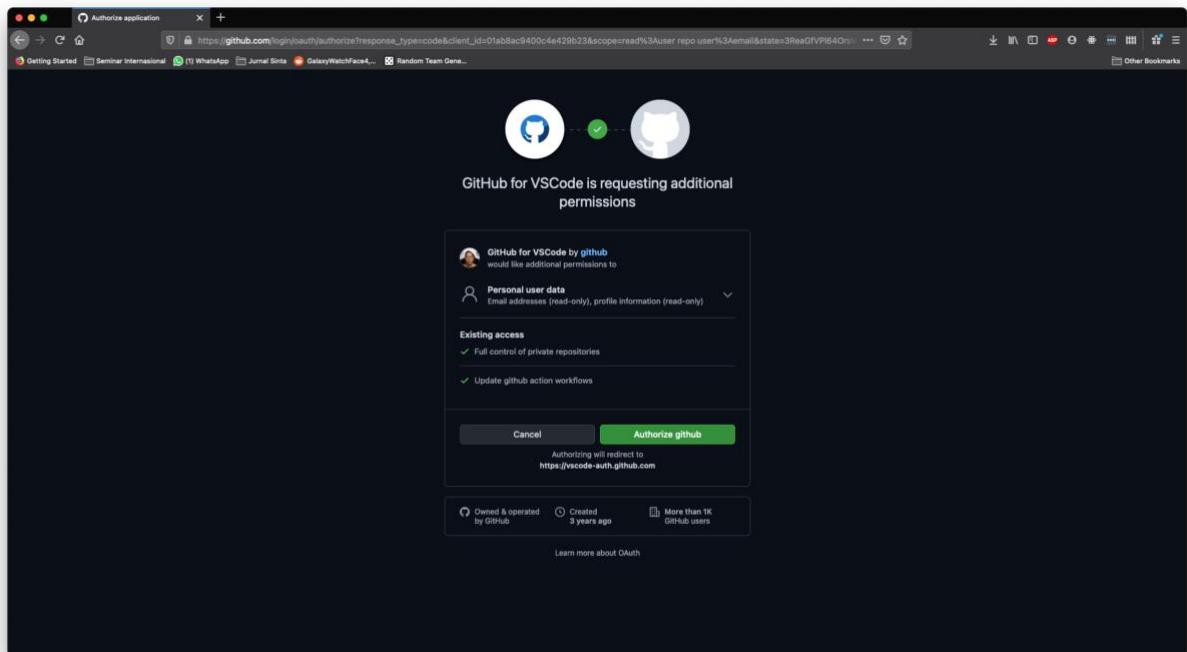
- Publish to GitHub
- Git: Push
- Git: Add Remote...
- Preferences: Color Theme
- Unfold All
- Fold All
- XML Tools: Format as XML
- Format Document
- Add gitignore
- Fold Recursively
- Expand Selection
- Collapse Folders in Explorer
- WakaTime: Api Key
- Copy With Syntax Highlighting

body {  
 font-family: 'Nunito';  
}

ht:1.1

14. Visual studio code akan membuka browser dan meminta otorisasi untuk akun github





15. Setelah otorisasi berhasil anda dapat melanjutkan kembali command Ctrl + Shift + P dan pilih publish to github kemudian berilah nama flutter-hello-world

```
welcome.blade.php - 01

laravel-hello-world

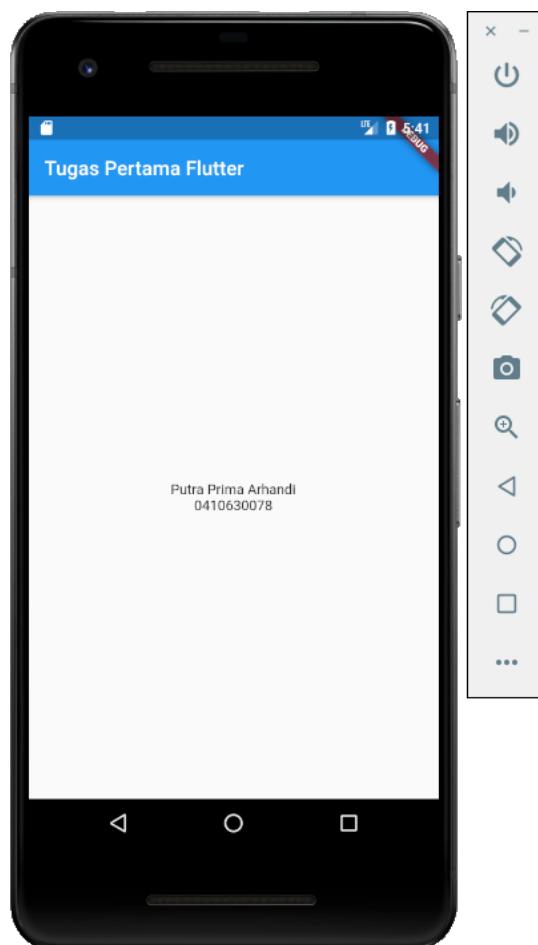
elcor [ ] Publish to GitHub private repository siubie/laravel-hello-world
[ ] Publish to GitHub public repository siubie/laravel-hello-world
str_----- / + + + + ~

charset="utf-8">
ame="viewport" content="width=device-width, initial-scale=1">

+----- / + + + + ~
```

## 1.6 Tugas Praktikum

1. Pastikan anda dapat melakukan installasi flutter dan melakukan kompilasi project hello\_world ke emulator / device android.
2. Push repository project hello\_world anda ke github
3. Buat satu project lagi yang menggunakan flutter new project namun kali ini isi ubahlah tampilan aplikasi menjadi seperti gambar berikut dan publish ke repository github dengan nama repository polinema\_mobile\_flutter\_hello.

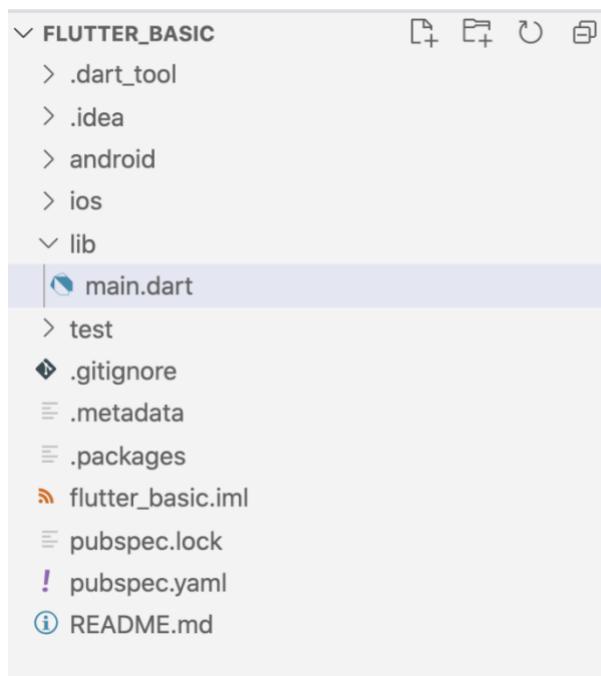


## BAB 2. BASIC APLIKASI FLUTTER

### 2.1 Struktur Project Flutter

#### 2.1.1 Struktur folder

Ketika membuat project flutter secara default berikut adalah struktur folder dan filenya. Dapat kita lihat strukturnya terdiri dari .dart\_tool, .idea, android, ios, lib, test, .gitignore, .metadata, .packages, .flutter\_basic.iml, pubspec.lock, pubspec.yaml, README.md.



Berikut adalah penjelasan yang lebih detail tentang struktur files dari flutter.

1. .dart\_tools : Konfigurasi untuk dart language
2. .idea : Konfigurasi untuk android studio
3. gitignore : File git yang digunakan untuk mengelola source code. Hal ini akan berguna jika developer sudah bekerja dengan git.
4. metadata : File yang berisi metadata dari project
5. packages : File yang berisi alamat path
6. flutter\_basic.iml: File yang berisi detail dari project.
7. pubspec.lock : File yang berisi versi library atau package yang digunakan pada project yang degenerate sesuai dengan file pubspec.yaml.
8. pubspec.yaml : File yang berisi library atau package yang dibutuhkan untuk pengembangan aplikasi.
9. Readme.md : File markdown yang dapat digunakan untuk menjelaskan cara setup

aplikasi atau informasi penting yang perlu untuk diketahui oleh developer lain.

Untuk struktur folder akan dijelaskan pada subbab 2.1.2 sampai 2.1.7.

### **2.1.2 Multi os ios dan android**

Perhatikan pada folder project flutter terdapat dua folder yaitu folder ios dan folder android, dengan menggunakan kedua folder tersebut flutter dapat membuat aplikasi berbasis ios dan berbasis android dalam satu project.

### **2.1.3 Folder android**

Folder android berisi file file pendukung untuk mengenerate project android dan akan dikompilasi menjadi sebuah apk pada folder build. Namun anda jarang atau bahkan tidak perlu mengedit yang ada di folder android. Anda akan banyak bekerja di folder lib.

### **2.1.4 Folder ios**

Berisi project ios, folder ini sama dengan folder android, sangat jarang dan bahkan tidak perlu untuk mengubah apapun pada folder ios. Folder ios dan android dikelola oleh flutter sdk yang akan dimerge (disatukan) dengan code yang ada di folder lib untuk membuat aplikasi ios dan android.

### **2.1.5 Folder lib**

Folder lib berisi kode program dengan bahasa dart yang berupa widget yang dapat dibuat sesuai dengan kebutuhan aplikasi anda.

### **2.1.6 Folder test**

Berisi source code dart yang digunakan untuk melakukan test secara otomatis pada aplikasi flutter.

### **2.1.7 Pubspec.yaml**

Pada file ini berisi konfigurasi konfigurasi project flutter yang dibuat dimana anda dapat mendata asset berupa font, gambar dan lain lain. Pada file ini anda juga dapat mengkonfigurasi flutter sdk dan konfigurasi yang terkait flutter.

## 2.2 Flutter Hot Reload

Pada flutter terdapat fungsi hot reload dan hot restart yang digunakan untuk pengembangan aplikasi dengan flutter. Hot reload mencompile source code yang baru ditambahkan dan dikirimkan ke dart virtual machine diupdate. Setelah selesai update, dart virtual machine akan memperbarui UI sesuai dengan perubahan. Keunggulan hot reload adalah waktu prosesnya cepat dibanding hot restart. Akan tetapi hot reload mempertahankan state yang ada sehingga jika menggunakan state maka nilai dari widget tidak akan berubah.

## 2.3 Flutter Hot Restart

Hot restart akan mencompile ulang aplikasi dan mereset (destroy) state yang ada. Jadi hot restart akan membuild ulang widget tree sesuai dengan code yang telah diperbarui.

## 2.4 Bedah Hello World Project

### 2.4.1 Import Statement

Seperti halnya kode program pada umumnya dart dapat menggunakan statement import untuk mengimport package, library, atau file lain yang digunakan pada file yang dieksekusi.

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

### 2.4.2 Main function

Main function pada flutter dibuat dengan menggunakan kode program berikut ini dimana semua proses aplikasi dimulai dari mengeksekusi fungsi main.

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

Perhatikan pada fungsi main ini yang di eksekusi adalah class MyApp dimana class MyApp harus mengextend salah satu StatelessWidget atau StatefulWidget, ingat bahwa flutter membangun UI menggunakan widget.

### 2.4.3 Stateless Widget

Flutter menggunakan Widget sebagai elemen-elemen pembangun UI, widget ini adalah kode program yang diterjemahkan menjadi tampilan yang dapat dilihat dan diinteraksikan oleh pengguna. Stateless widget bersifat statis / final dimana nilai atau konfigurasi telah diinisiasi sejak awal, nilai variabel pada widget ini tidak dapat diubah oleh widget ini sendiri tetapi dapat diubah oleh parent widgetnya jika parentnya adalah StatefulWidget. Struktur dasar stateless widget adalah sebagai berikut:

```
class exampleStateless extends StatelessWidget{  
    @override  
    Widget build(BuildContext context) {  
  
    }  
}
```

#### 2.4.4 Statefull Widget

Statefull widget bersifat dinamis, widget ini dapat diperbarui ketika dibutuhkan sesuai dengan action pengguna atau jika ada perubahan data. Perubahan data pada statefull widget di trigger oleh perubahan state oleh karena itu sebuah StatefullWidget selalu memiliki State. Struktur dasar statefull widget adalah sebagai berikut:

```
class exampleStateless extends StatefulWidget{  
    @override  
    State<StatefulWidget> createState() {  
    }  
}
```

### 2.5 Build in Widget

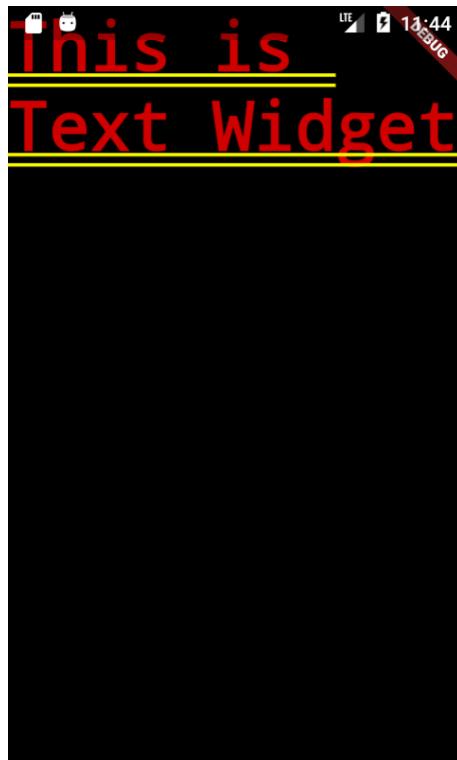
Pada framework flutter terdapat banyak widget. Widget yang telah disediakan dapat digunakan mengembangkan aplikasi yang mobile, desktop dan web yang memiliki tampilan menarik. Secara lebih detail widget yang tersedia di flutter dapat dicek pada <https://flutter.dev/docs/development/ui/widgets>. Berikut adalah beberapa contoh widget yang sering digunakan pada pembuatan aplikasi dengan flutter.

#### 2.5.1 Text Widget

Text widget digunakan untuk menampilkan string yang dapat terdiri satu baris maupun beberapa baris. Contoh penggunaan text widget pada source code dan outputnya adalah sebagai berikut:

```
class MyApp extends StatelessWidget {  
    @override  
    Widget build(BuildContext context) {  
        return MaterialApp(
```

```
        home : Text('This is Text Widget',),
    );
}
}
```



### 2.5.2 Image Widget

Image widget digunakan untuk menampilkan image. Contoh penggunaan image widget pada source code dan outputnya adalah sebagai berikut:

```
class MyApp extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return MaterialApp(
            home: Image(image: NetworkImage('https://flutter.github.io/assets-
for-api-docs/assets/widgets/owl.jpg'),),
        );
    }
}
```



### 2.5.3 Material Design dan iOS Cupertino

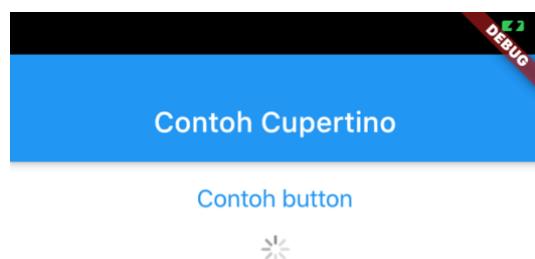
Cupertino widget digunakan untuk mendesain sesuai dengan standar desain pada iOS. Contoh penggunaan cupertino widget pada source code dan ouputnya adalah sebagai berikut:

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Container(
        margin: EdgeInsets.only(top: 30),
        color: Colors.white,
        child: Column(
          children: <Widget>[
            AppBar(title: Text('Contoh Cupertino')),
            CupertinoButton(
              child: Text('Cupertino Button'),
              onPressed: () {
                print('Cupertino button pressed');
              },
            ),
          ],
        ),
      ),
    );
  }
}
```

```
        child: Text("Contoh button"),
        onPressed: () {},
    ),
    CupertinoActivityIndicator(),
],
),
),
);
}
}
```



#### 2.5.4 Button

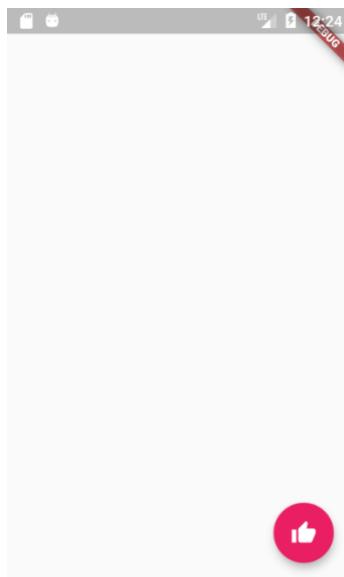
Button widget terdapat beberapa macam pada flutter yaitu antara lain ButtonBar, DropdownButton, FlatButton, FloatingActionButton, IconButton, OutlineButton, PopupMenuItem, dan RaisedButton. Contoh penggunaan Cupertino widget pada source code dan outputnya adalah sebagai berikut:

```
class MyApp extends StatelessWidget {
```

```

@Override
Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            floatingActionButton:FloatingActionButton(
                onPressed: () {
                    // Add your onPressed code here!
                },
                child: Icon(Icons.thumb_up),
                backgroundColor: Colors.pink,
            ),
        ),
    );
}
}

```



### 2.5.5 Scaffold

Scaffold widget digunakan untuk mengatur tata letak sesuai dengan material design.

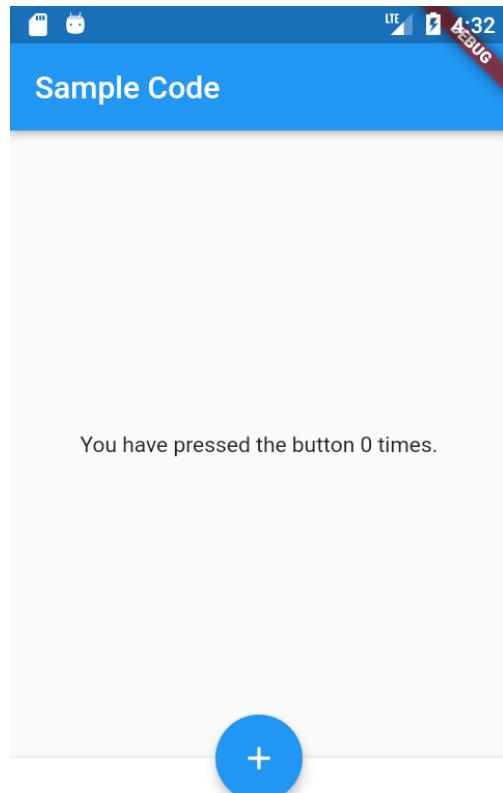
Contoh penggunaan scaffold widget pada source code dan ouputnya adalah sebagai berikut:

```

class MyApp extends StatelessWidget {
    int _count = 0;
    @override
    Widget build(BuildContext context) {
        return MaterialApp(
            home: Scaffold(

```

```
appBar: AppBar(
    title: Text('Sample Code'),
),
body: Center(
    child: Text('You have pressed the button $_count times.'),
),
bottomNavigationBar: BottomAppBar(
    child: Container(
        height: 50.0,
    ),
),
floatingActionButton: FloatingActionButton(
    onPressed: () => 0,
    tooltip: 'Increment Counter',
    child: Icon(Icons.add),
),
floatingActionButtonLocation:
FloatingActionButtonLocation.centerDocked,
),
);
}
}
```



### 2.5.6 Dialog

Dialog widget pada flutter memiliki dua jenis dialog yaitu AlertDialog dan SimpleDialog. Contoh penggunaan AlertDialog widget pada source code dan outputnya adalah sebagai berikut:

```
class MyApp extends StatelessWidget {
    int _count = 0;
    @override
    Widget build(BuildContext context) {
        return MaterialApp(
            home: Scaffold(
                body: MyLayout(),
            ),
        );
    }
}

class MyLayout extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
```

```
        return Padding(
            padding: const EdgeInsets.all(8.0),
            child: RaisedButton(
                child: Text('Show alert'),
                onPressed: () {
                    showAlertDialog(context);
                },
            ),
        );
    }

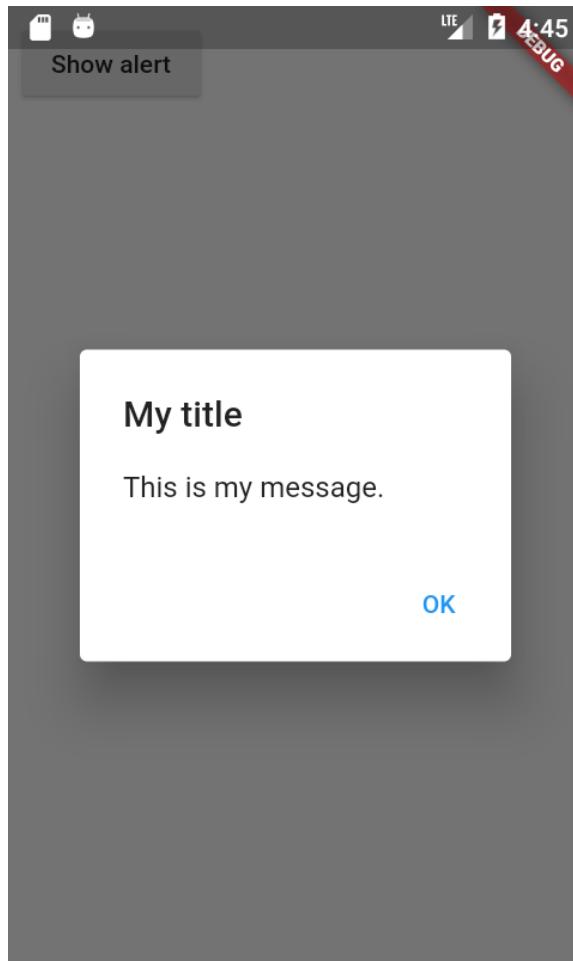
}

showAlertDialog(BuildContext context) {

    // set up the button
    Widget okButton = FlatButton(
        child: Text("OK"),
        onPressed: () { },
    );

    // set up the AlertDialog
    AlertDialog alert = AlertDialog(
        title: Text("My title"),
        content: Text("This is my message."),
        actions: [
            okButton,
        ],
    );

    // show the dialog
    showDialog(
        context: context,
        builder: (BuildContext context) {
            return alert;
        },
    );
}
```



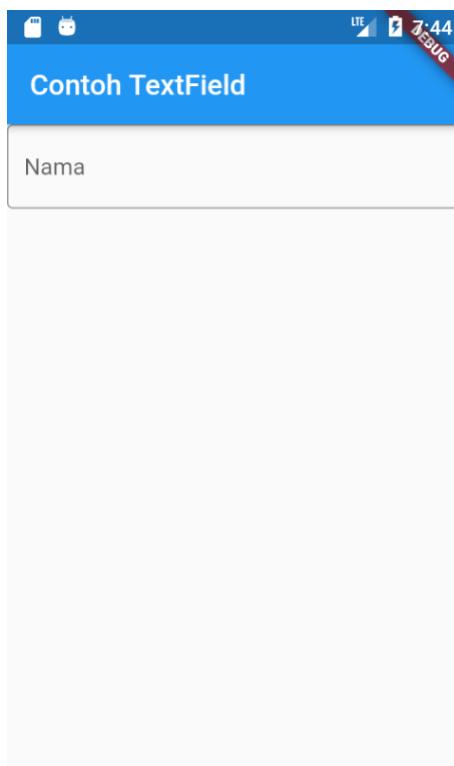
### 2.5.7 Input dan Selection Widget

Flutter menyediakan widget yang dapat menerima input dari pengguna aplikasi yaitu antara lain Checkbox, Date and Time Pickers, Radio Button, Slider, Switch, TextField.

Contoh penggunaan TextField widget pada source code dan outputnya adalah sebagai berikut:

```
class MyApp extends StatelessWidget {  
    @override  
    Widget build(BuildContext context) {  
        return MaterialApp(  
  
            home: Scaffold(  
                appBar: AppBar(  
                    title: Text("Contoh TextField")  
                ),  
                body: TextField(  
                    obscureText: false,  
                    decoration: InputDecoration(  
                        border: OutlineInputBorder(),  
                ),  
            ),  
        );  
    }  
}
```

```
        labelText: 'Nama',
    ),
),
),
);
}
}
```



### 2.5.8 Date and Time Pickers

Date and Time Pickers termasuk pada kategori input dan selection widget, berikut adalah contoh penggunaan Date and Time Pickers.

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

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

class MyApp extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return MaterialApp(

```

```
        title: 'Contoh Date Picker',
        home: MyHomePage(title: 'Contoh Date Picker'),
    );
}

class MyHomePage extends StatefulWidget {
    MyHomePage({Key key, this.title}) : super(key: key);

    final String title;

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

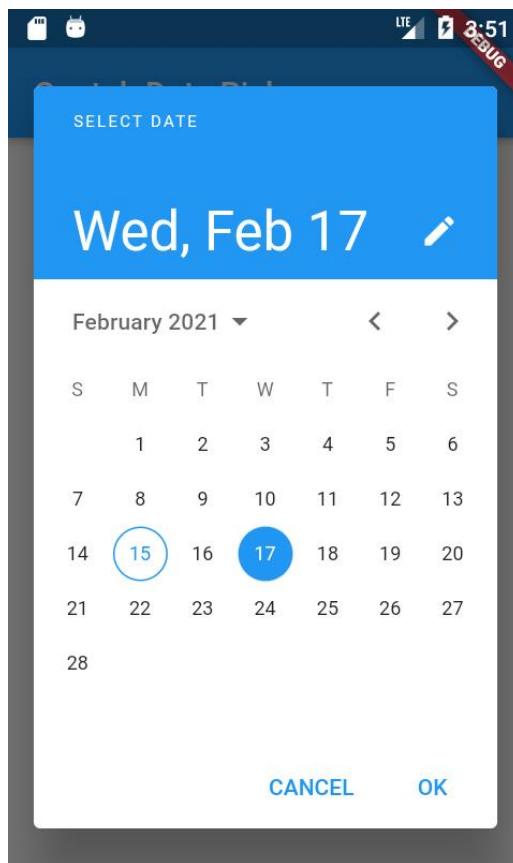
class _MyHomePageState extends State<MyHomePage> {
    // Variable/State untuk mengambil tanggal
    DateTime selectedDate = DateTime.now();

    // Initial SelectDate Flutter
    Future<Null> _selectDate(BuildContext context) async {
        // Initial DateTime Final Picked
        final DateTime picked = await showDatePicker(
            context: context,
            initialDate: selectedDate,
            firstDate: DateTime(2015, 8),
            lastDate: DateTime(2101));
        if (picked != null && picked != selectedDate)
            setState(() {
                selectedDate = picked;
            });
    }

    @override
    Widget build(BuildContext context) {
        return Scaffold(
            appBar: AppBar(
                title: Text(widget.title),
            ),
            body: Center(
                child: Column(

```

```
        mainAxisSize: MainAxisSize.min,
        children: <Widget>[
            Text("${selectedDate.toLocal()}" .split(' ') [0]),
            SizedBox(height: 20.0, ),
            RaisedButton(
                onPressed: () => {
                    _selectDate(context),
                    print(selectedDate.day + selectedDate.month +
selectedDate.year )
                },
            ),
            child: Text('Pilih Tanggal'),
        ],
    ],
),
),
),
),
),
);
}
}
```



## 2.6 Build in Layout Widget

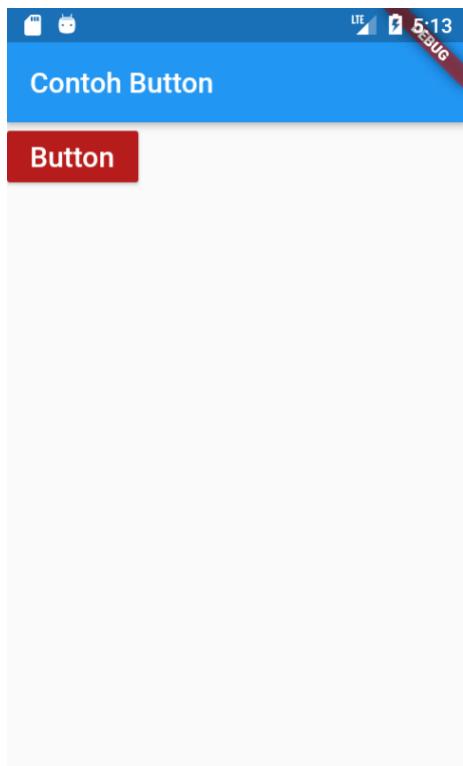
### 2.6.1 Container

Container widget berguna untuk menyimpan berbagai macam attribute dan menampung berbagai macam fungsi objek. Container merupakan **single child objek** yang artinya hanya dapat memiliki satu buah child widget, akan tetapi dalam sebuah container kita dapat menempatkan row, column, text dan container lain. Container memiliki beberapa property yaitu:

- A. property child : digunakan untuk membuat menampung widget didalam container.

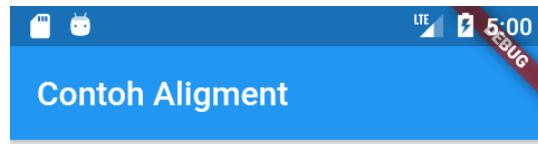
Widget yang ditampung antara lain Text, Column, ListView, Buton dan lain sebagainya.

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text("Contoh Button")),
        body: Container(
          child: RaisedButton(
            textColor: Colors.white,
            onPressed: (){},
            color: Colors.red[900],
            child: Text(
              "Button",
              style: TextStyle(fontSize: 20)
            ),
          )));
  }
}
```



B. property alignment : mengatur posisi child widget menggunakan property Alignment.

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text("Contoh Alignment")),
        body: Container(
          alignment: Alignment.bottomCenter,
          child: Text(
            'Semangat Belajar',
            style: TextStyle(
              fontSize: 20,
            ),
          )),
    );
  }
}
```



## Semangat Belajar

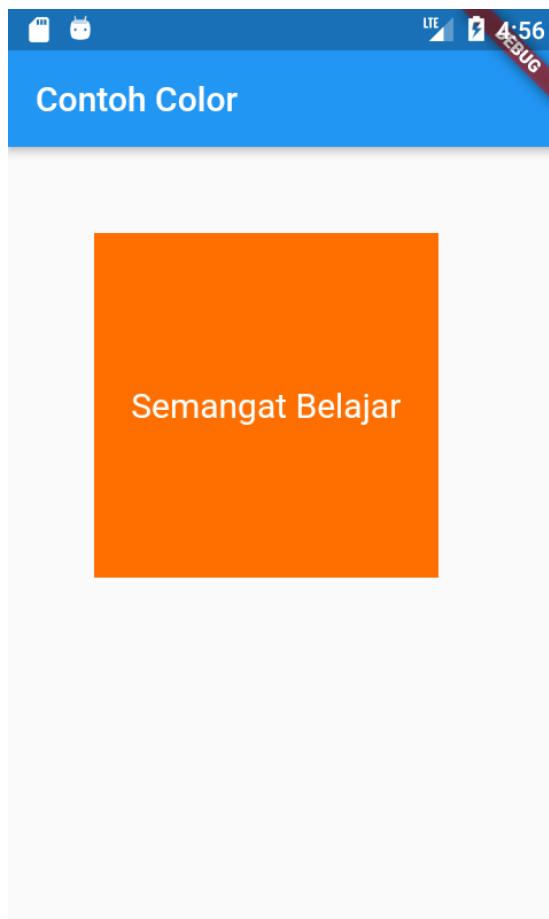
- C. property color : digunakan untuk mengubah warna latar belakang container. Untuk memilih warna dapat menghover warna maka akan muncul pilihan warna yang dapat kita gunakan seperti berikut:

Colors.amber[50]	0xFFFFF8E1
Colors.amber[100]	0xFFFFECB3
Colors.amber[200]	0xFFFFE082
Colors.amber[300]	0xFFFFD54F

Contoh penggunaan color pada container adalah sebagai berikut:

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
```

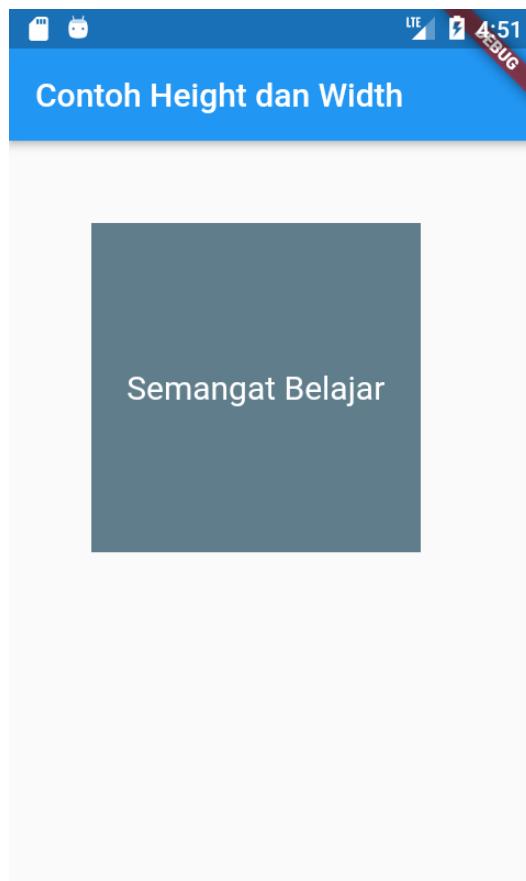
```
        appBar: AppBar(title: Text("Contoh Color")),
        body: Container(
            margin: EdgeInsets.all(50),
            height: 200,
            width: 200,
            alignment: Alignment.center,
            color: Colors.amber[900],
            child: Text(
                'Semangat Belajar',
                style: TextStyle(fontSize: 20, color: Colors.white),
            )));
    }
}
```



- D. Property height dan width : Secara default ukuran container menyesuaikan dengan ukuran body layar, maka untuk mengatur layoutnya dapat menggunakan property height dan width.

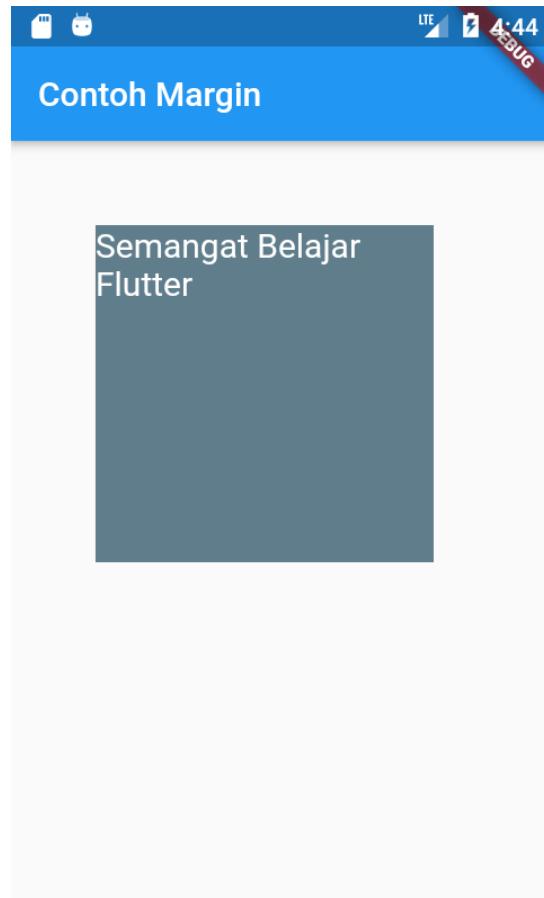
```
class MyApp extends StatelessWidget {
```

```
@override
Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Contoh Height dan Width")),
            body: Container(
                margin: EdgeInsets.all(50),
                height: 200,
                width: 200,
                alignment: Alignment.center,
                color: Colors.blueGrey,
                child: Text(
                    'Semangat Belajar',
                    style: TextStyle(fontSize: 20, color:
Colors.white),
                ))),
    );
}
```



E. property margin : membuat jarak container dengan dengan widget lainnya.

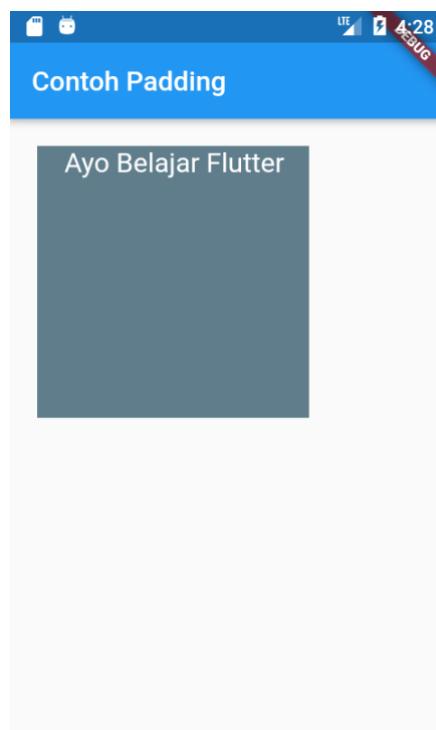
```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text("Contoh Margin")),
        body: Container(
          margin: EdgeInsets.all(50),
          height: 200,
          width: 200,
          alignment: Alignment.topLeft,
          color: Colors.blueGrey,
          child: Text(
            'Semangat Belajar Flutter',
            style: TextStyle(fontSize: 20, color:
            Colors.white),
          )));
  }
}
```



F. property padding : digunakan untuk menambahkan jarak antara container dengan widget yang ada didalam container tersebut.

```
class MyApp extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return MaterialApp(
            home: Scaffold(
                appBar: AppBar(title: Text("Contoh Padding")),
                body: Container(
                    padding: EdgeInsets.only(left: 20),
                    margin: EdgeInsets.all(20),
                    height: 200,
                    width: 200,
                    alignment: Alignment.topLeft,
                    color: Colors.blueGrey,
                    child: Text(
                        'Ayo Belajar Flutter',
                        style: TextStyle(fontSize: 20, color: Colors.white),
                    )));
    }
}
```

```
) ;  
}  
}
```



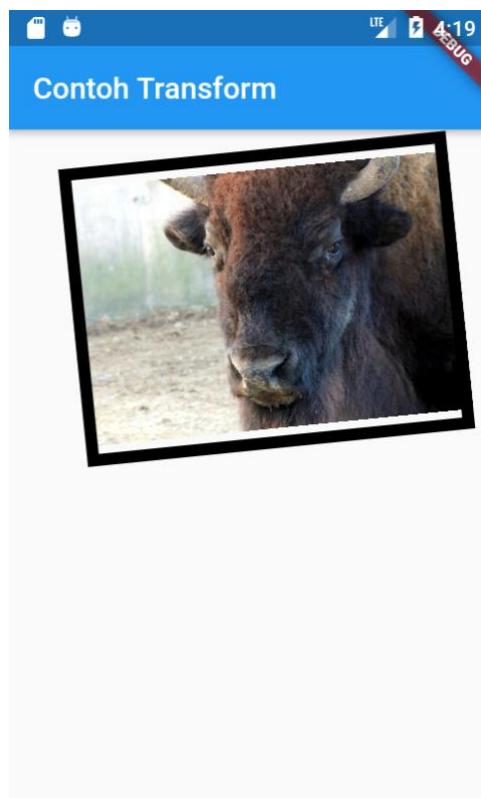
- G. property transform : berfungsi untuk melakukan rotasi pada container dengan melakukan dari berbagai sumbu putar misalnya X,Y, dan Z.

```
class MyApp extends StatelessWidget {  
    @override  
    Widget build(BuildContext context) {  
        return MaterialApp(  
            home: Scaffold(  
                appBar: AppBar(title: Text("Contoh Transform")),  
                body: Container(  
                    decoration: BoxDecoration(  
                        image: const DecorationImage(  
                            image: NetworkImage(  
                                'https://pixnio.com/free-  
images/2017/03/07/2017-03-07-10-59-39-900x600.jpg'),  
                            fit: BoxFit.fitWidth,  
                        ),  
                    border: Border.all(  
                        color: Colors.black,  
                        width: 8,
```

```

        ) ,
    ) ,
    height: 200,
    width: 300,
    margin: const EdgeInsets.only(left: 30.0, right:
30.0, top: 30),
    transform: Matrix4.rotationZ(-0.1),
)
),
);
}
}

```



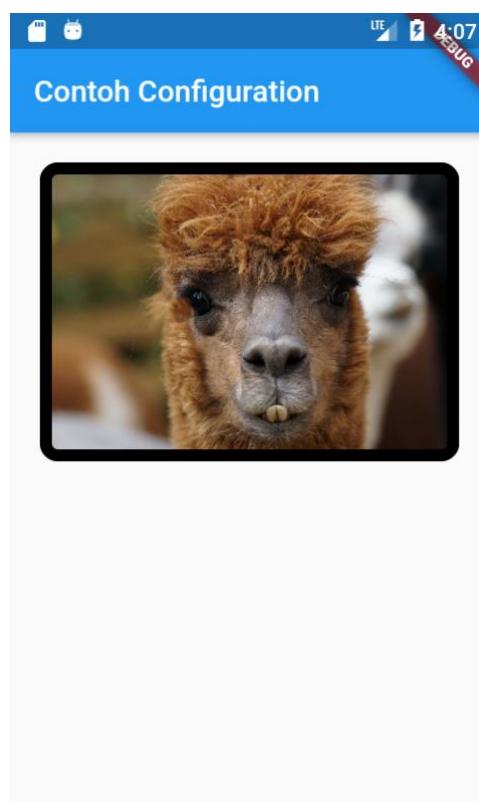
H. property decoration : untuk mencustom container dengan berbagai macam efek misalnya dengan mengubah warna border, memberikan gambar, dan membuat efek bayangan.

```

class MyApp extends StatelessWidget {
@override
Widget build(BuildContext context) {
return MaterialApp(
home: Scaffold(
appBar: AppBar(title: Text("Contoh Configuration")),

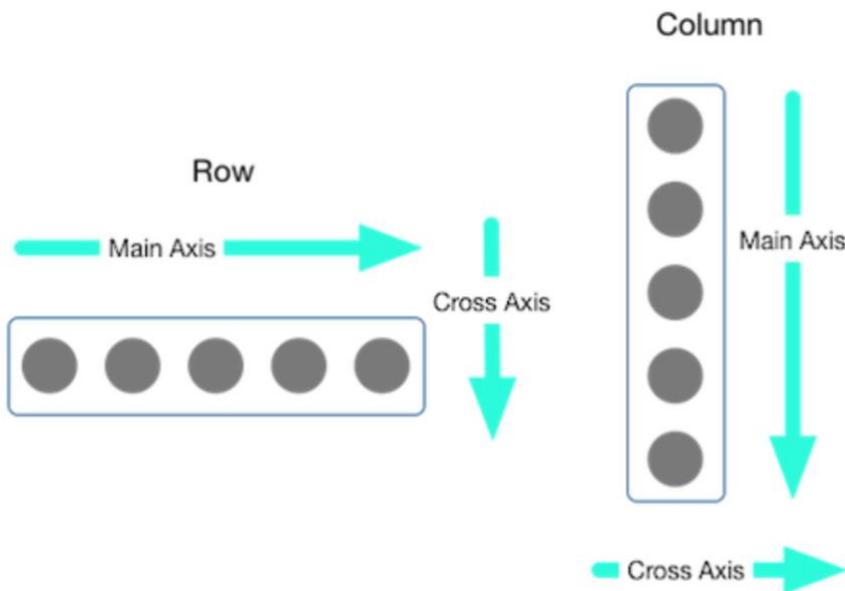
```

```
body: Container(  
    decoration: BoxDecoration(  
        color: const Color(0xff7c94b6),  
        image: const DecorationImage(  
            image: NetworkImage(  
                'https://pixnio.com/free-  
images/2018/12/02/2018-12-02-19-17-12.jpg'),  
            fit: BoxFit.fitWidth,  
        ),  
        border: Border.all(  
            color: Colors.black,  
            width: 8,  
        ),  
        borderRadius: BorderRadius.circular(12),  
    ),  
    height: 200,  
    width: 300,  
    margin: EdgeInsets.all(20))),  
);  
}  
}
```



## 2.6.2 Row dan Column

Column widget digunakan untuk mengatur tata letak widget secara vertikal. Sedangkan row digunakan untuk mengatur tata letak widget secara horizontal. Berikut adalah gambaran perbedaan row dan column widget adalah sebagai berikut:



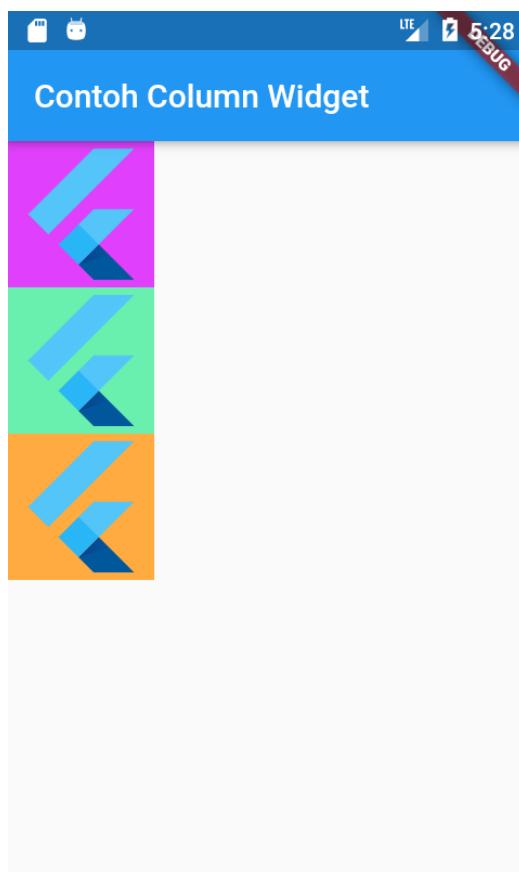
(sumber : <https://flutter.dev/docs>)

10. Berikut adalah contoh penggunaan Column Widget

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text("Contoh Column Widget")),
        body: Column(
          children: [
            Container(
              color: Colors.purpleAccent,
              child: FlutterLogo(
                size: 90.0,
              ),
            ),
            Container(
              color: Colors.greenAccent,
              child: FlutterLogo(

```

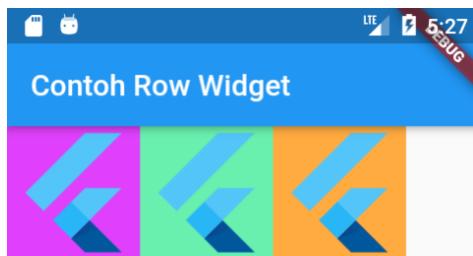
```
        size: 90.0,  
    ),  
),  
Container(  
    color: Colors.orangeAccent,  
    child: FlutterLogo(  
        size: 90.0,  
    ),  
),  
],  
)  
,  
);  
}  
}
```



11. Berikut adalah contoh penggunaan Row Widget

```
class MyApp extends StatelessWidget {  
    @override  
    Widget build(BuildContext context) {  
        return MaterialApp(  
            title: 'Contoh Row Widget',  
            home: Scaffold(  
                appBar: AppBar(  
                    title: Text('Contoh Row Widget'),  
                ),  
                body: Center(  
                    child: Row(  
                        mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
                        children:   
                            [  
                                Container(  
                                    width: 50,  
                                    height: 50,  
                                    color: Colors.pink,  
                                    child: FlutterLogo(  
                                        size: 30,  
                                    ),  
                                ),  
                                Container(  
                                    width: 50,  
                                    height: 50,  
                                    color: Colors.teal,  
                                    child: FlutterLogo(  
                                        size: 30,  
                                    ),  
                                ),  
                                Container(  
                                    width: 50,  
                                    height: 50,  
                                    color: Colors.orange,  
                                    child: FlutterLogo(  
                                        size: 30,  
                                    ),  
                                ),  
                            ],  
                        ),  
                ),  
            ),  
        );  
    }  
}
```

```
home: Scaffold(  
    appBar: AppBar(title: Text("Contoh Row Widget")),  
    body: Row(  
        children: [  
            Container(  
                color: Colors.purpleAccent,  
                child: FlutterLogo(  
                    size: 90.0,  
                ),  
            ),  
            Container(  
                color: Colors.greenAccent,  
                child: FlutterLogo(  
                    size: 90.0,  
                ),  
            ),  
            Container(  
                color: Colors.orangeAccent,  
                child: FlutterLogo(  
                    size: 90.0,  
                ),  
            ),  
        ],  
    ),  
);  
}  
}
```



### 2.6.3 Stack

Stack Widget digunakan untuk menumpuk beberapa widget pada beberapa lapisan.

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Scaffold(
        appBar: AppBar(
          title: Text("Contoh Stack Widget"),
        ),
        body: Stack(
          children: <Widget>[
            Container(
              color: Colors.green,
              alignment: Alignment.bottomCenter,
              child: Text("Satu", style: TextStyle(fontSize: 30, color:
                Colors.white)),
            ),
            Container(
              color: Colors.red,
              alignment: Alignment.bottomCenter,
            ),
          ],
        ),
      ),
    );
  }
}
```

```
        child: Text("Dua", style: TextStyle(fontSize:30,color:  
Colors.white)),  
        height: 400.0,  
        width: 300.0,  
      ),  
      Container(  
        color: Colors.deepPurple,  
        alignment:Alignment.bottomCenter,  
        child: Text("Tiga", style: TextStyle(fontSize:30,color:  
Colors.white)),  
        height: 200.0,  
        width: 200.0,  
      ),  
    ],  
  ),  
),  
);  
}  
}  
}
```

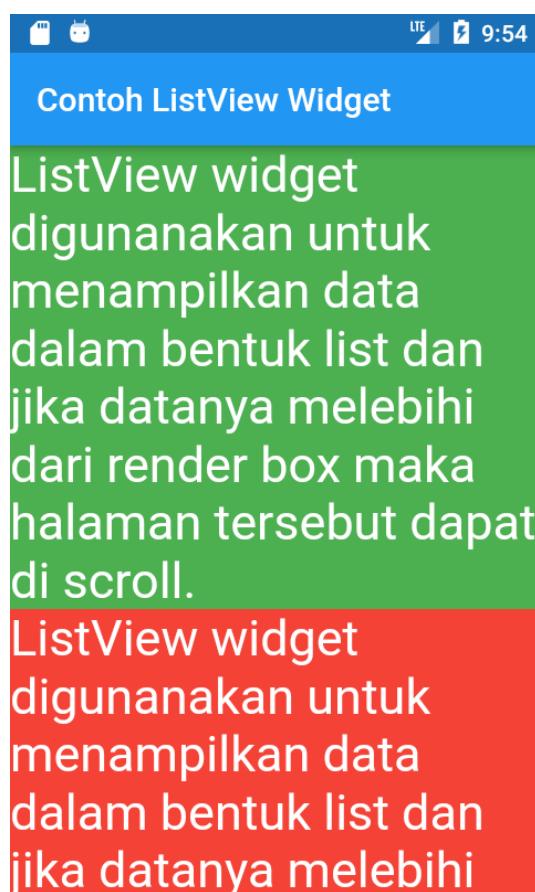


## 2.6.4 ListView

ListView widget digunakan untuk menampilkan data dalam bentuk list dan jika datanya melebihi dari render box maka halaman tersebut dapat di scroll.

```
class MyApp extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return MaterialApp(
            debugShowCheckedModeBanner: false,
            home: Scaffold(
                appBar: AppBar(
                    title: Text("Contoh ListView Widget"),
                ),
                body: ListView(
                    children: <Widget>[
                        Container(
                            color: Colors.green,
                            alignment: Alignment.topLeft,
                            child: Text("ListView widget digunakan untuk menampilkan
data dalam bentuk list dan jika datanya melebihi dari render box maka
halaman tersebut dapat di scroll.", style: TextStyle(fontSize:30,color:
Colors.white)),
                        ),
                        Container(
                            color: Colors.red,
                            alignment: Alignment.topLeft,
                            child: Text("ListView widget digunakan untuk menampilkan
data dalam bentuk list dan jika datanya melebihi dari render box maka
halaman tersebut dapat di scroll.", style: TextStyle(fontSize:30,color:
Colors.white)),
                        height: 400.0,
                        width: 300.0,
                    ),
                    Container(
                        color: Colors.deepPurple,
                        alignment: Alignment.topLeft,
                        child: Text("ListView widget digunakan untuk menampilkan
data dalam bentuk list dan jika datanya melebihi dari render box maka
halaman tersebut dapat di scroll.", style: TextStyle(fontSize:30,color:
Colors.white)),
                ],
            ),
        );
    }
}
```

```
        height: 200.0,
        width: 200.0,
    ),
],
),
),
);
}
}
```



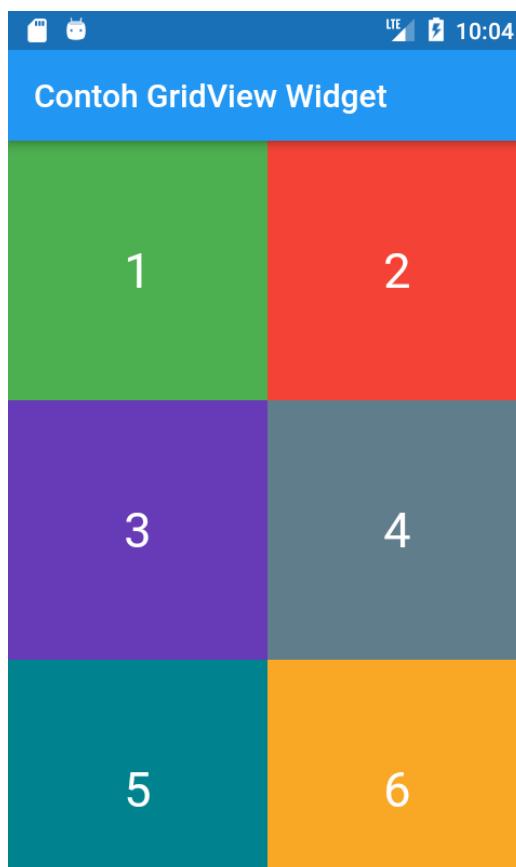
## 2.6.5 GridView

Gridview digunakan untuk menata tata letak widget pada list 2 dimensi. GridView juga secara otomatis menyediakan scrolling ketika konten melebihi render box.

```
class MyApp extends StatelessWidget {
@Override
Widget build(BuildContext context) {
    return MaterialApp(
```

```
home: Scaffold(
    appBar: AppBar(
        title: Text("Contoh GridView Widget"),
    ),
    body: GridView.count(
        crossAxisCount: 2,
        children: <Widget>[
            Container(
                color: Colors.green,
                alignment: Alignment.center,
                child: Text("1", style: TextStyle(fontSize: 30, color:
Colors.white)),
            ),
            Container(
                color: Colors.red,
                alignment: Alignment.center,
                child: Text("2", style: TextStyle(fontSize: 30, color:
Colors.white)),
            ),
            Container(
                color: Colors.deepPurple,
                alignment: Alignment.center,
                child: Text("3", style: TextStyle(fontSize: 30, color:
Colors.white)),
            ),
            Container(
                color: Colors.blueGrey,
                alignment: Alignment.center,
                child: Text("4", style: TextStyle(fontSize: 30, color:
Colors.white)),
            ),
            Container(
                color: Colors.cyan[800],
                alignment: Alignment.center,
```

```
        child: Text("5", style: TextStyle(fontSize:30,color:  
Colors.white)),  
        height: 200.0,  
        width: 200.0,  
      ),  
      Container(  
        color: Colors.yellow[800],  
        alignment:Alignment.center,  
        child: Text("6", style: TextStyle(fontSize:30,color:  
Colors.white)),  
        height: 200.0,  
        width: 200.0,  
      ),  
    ],  
  ),  
),  
);  
}  
}  
}
```



## 2.7 Praktikum 1 Membuat UI Sederhana

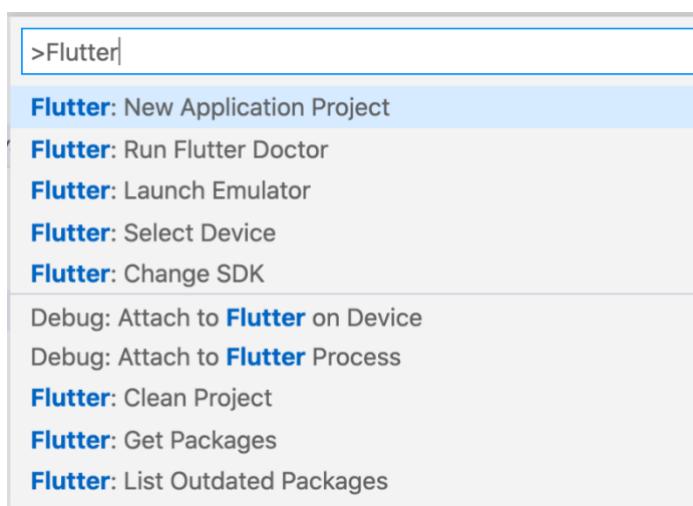
### 2.7.1 Alat dan Bahan

Pada praktikum pertama anda akan membuat sebuah project memanfaatkan widget flutter.

1. Visual Studio Code
2. Flutter SDK
3. Emulator atau device android

### 2.7.2 Langkah Praktikum

1. Buat project baru dengan menggunakan vscode misalnya dengan nama flutter\_basic.  
Pada vscode dapat dilakukan dengan cara memilih menu '**View => Command Pallette**' selanjutnya klik Flutter akan muncul tampilan seperti pada gambar di bawah ini. Selanjutnya pilih **Flutter: New Application Project**, anda akan diminta untuk menentukan path dimana project akan dibuat dan menentukan nama project di folder tersebut.



Untuk membuat project flutter dapat juga dilakukan pada terminal atau cmd dengan menggunakan perintah seperti berikut:

```
flutter create nama_project
```

2. Dengan informasi yang anda dapatkan dari teori diatas buatlah UI sederhana seperti tampilan dibawah ini menggunakan flutter.

LTE 8:11

# MyApp

BERITA TERBARU PERTANDINGAN HARI INI



## Costa Mendekat Ke Palmeiras

Transfer



Pique Bilang Wasit Untungkan Madrid, Koeman Tepok Jidat

Barcelona Feb 13, 2021



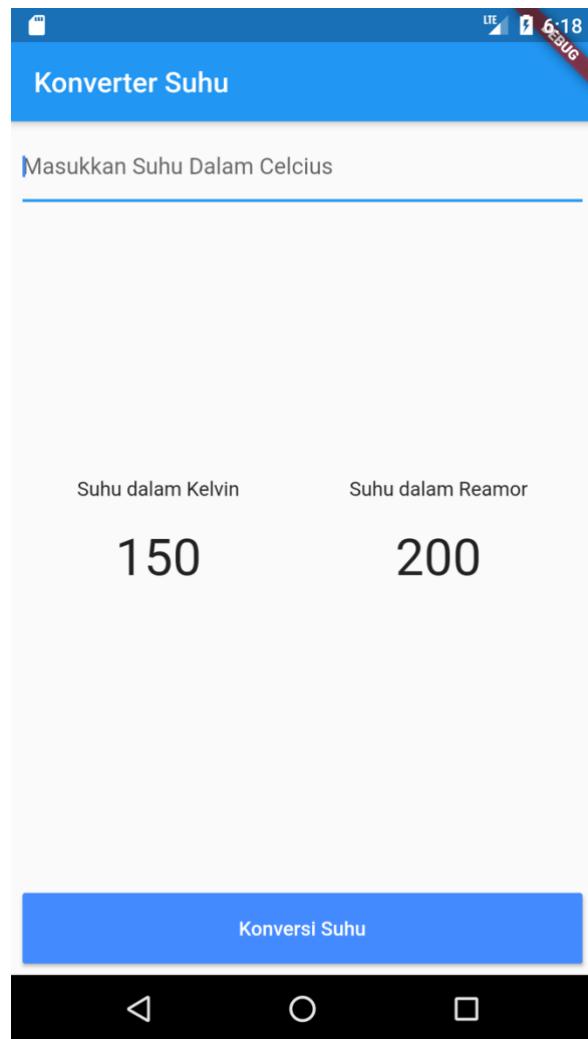
Pique Bilang Wasit Untungkan Madrid, Koeman Tepok Jidat

< O □

## BAB 3. APLIKASI DENGAN STATEFULL WIDGET

### 3.1 Desain Aplikasi

Pada praktikum kali ini anda akan belajar mengenai cara pembuatan aplikasi sederhana yang memiliki statefull widget. Untuk pembuatan aplikasi ini dimulai dengan membuat sebuah desain yang dapat dimulai dengan membuat wireframe atau desain utuh di perangkat lunak seperti adobe xd atau sketch dan yang lain. Untuk menyederhanakan proses desain anda sudah disediakan template sederhana di starter code dengan menggunakan template material design. Berikut ini desain mockup aplikasi yang dibuat.



### 3.2 Konversi Desain Ke StatelessWidget

Untuk mempermudah proses konversi dari desain ke stateless widget perhatikanlah item item yang ada pada desain tersebut, apa saja widget yang terlibat dan apa saja widget layout yang digunakan.

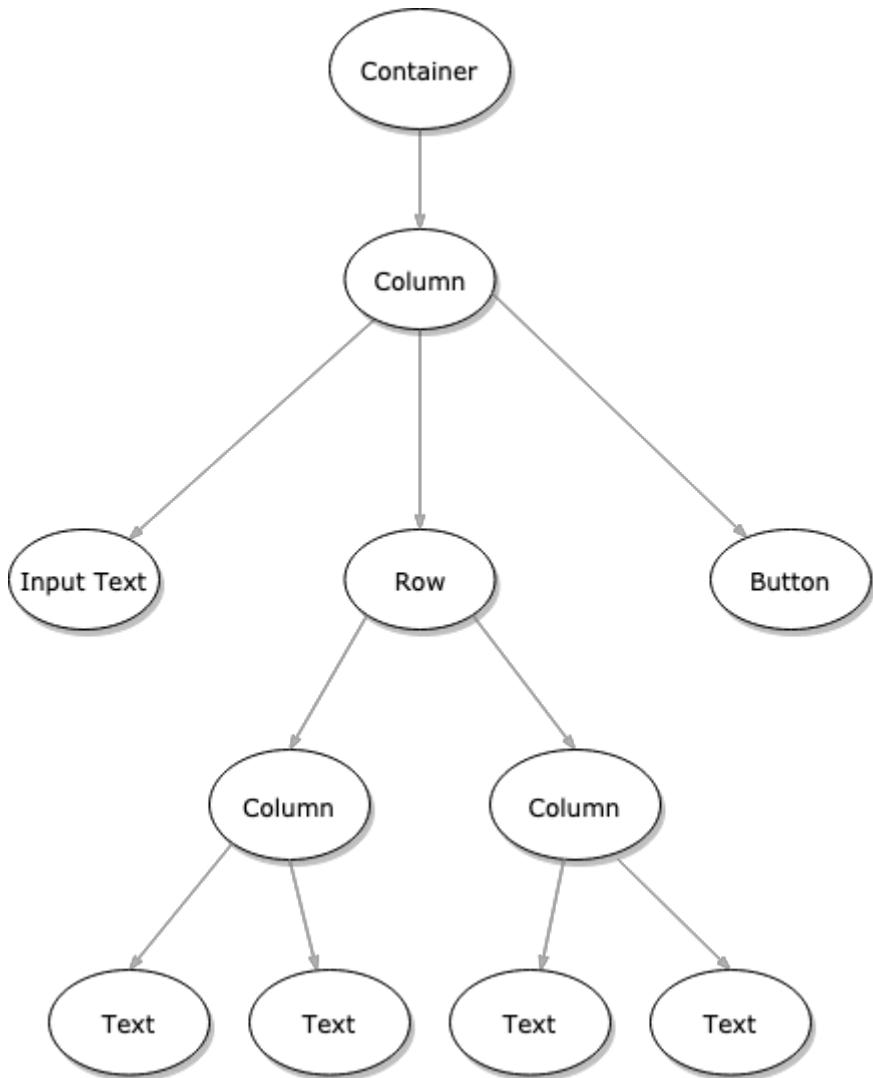
Sekilas dari UI desain yang ada dapat anda temukan bahwa terdapat satu input text, empat text, dan satu buah button. Secara lebih dalam perhatikan karakteristik item-item

tersebut seperti hint pada input, ukuran text yang berbeda, warna text pada tombol dan ukuran tombol yang memenuhi lebar layar.

Perhatikan juga bagaimana UI pada desain tersebut disusun dalam cara pandang row dan column, secara sederhana aplikasi ini terdiri dari satu kolom dan tiga baris dimana baris pertama berisi input, baris kedua berisi info suhu dan nilainya, serta baris ketiga berisi sebuah button. Khusus pada baris kedua terbagi menjadi dua buah kolom yang memiliki dua buah text.



Jika dikonversi menjadi sebuah tree layout diatas dapat direpresentasikan menjadi tree berikut ini.



### 3.2.1 Langkah Mengkonversi Desain Ke Widget

1. Buat Project flutter baru
2. Hapus starter code di main.dart sehingga menjadi seperti kode program dibawah ini

```

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

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

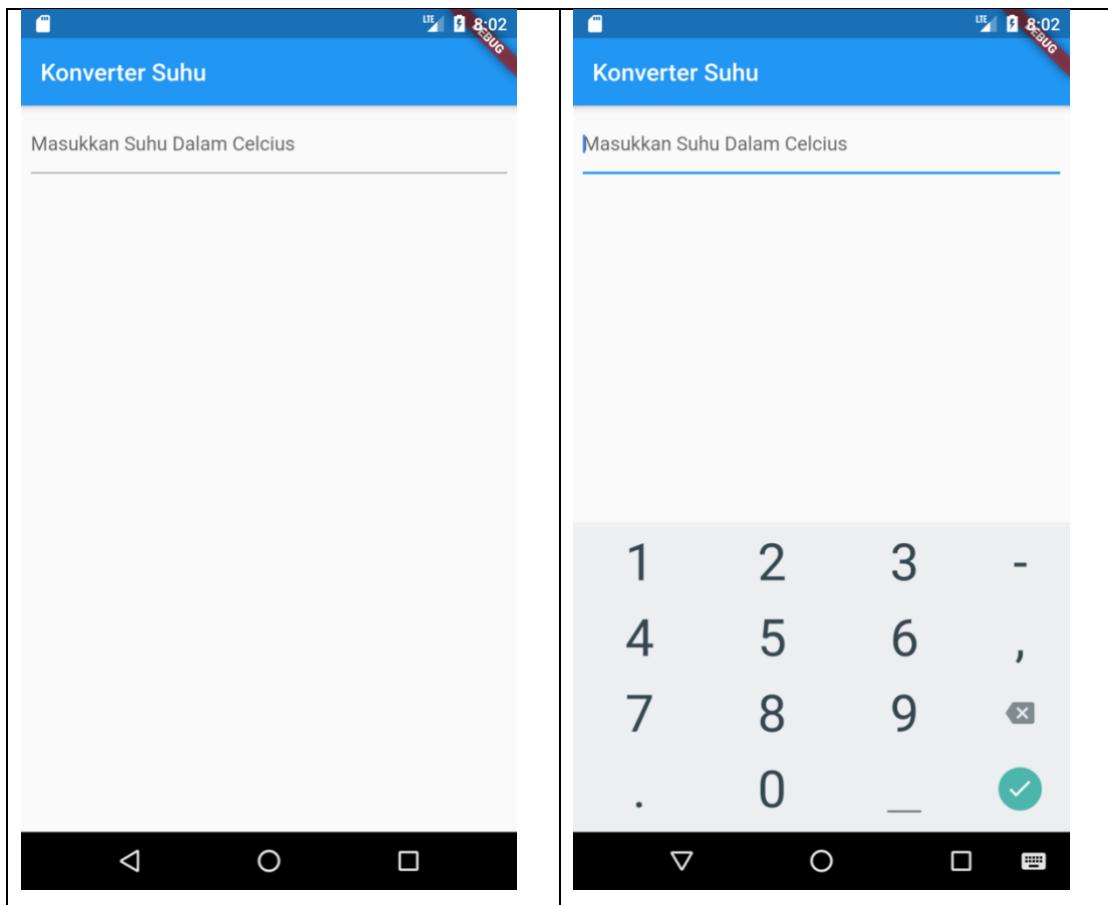
class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
  }
}
  
```

```
return MaterialApp(
    title: 'Flutter Demo',
    theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
    ),
    home: Scaffold(
        appBar: AppBar(
            title: Text("Konverter Suhu"),
        ),
        body: Container(),
    ),
);
}
```

3. Mulai lah dengan menambahkan margin pada container dan memberikan child berupa sebuah InputForm pada container tersebut.

```
body: Container(
    margin: EdgeInsets.all(8),
    child: TextFormField(),
),
```

4. Mulailah dengan cara seperti ini walaupun TextFormField bukan merupakan parent widget pada widget tree yang kita butuhkan namun kita dapat melakukan wrapping dengan widget lain menggunakan shortcut **Ctrl + .**
5. Lanjutkan dengan melakukan modifikasi pada TextFormField carilah properties agar
  - a. Memiliki hint dengan text “Masukkan Suhu Dalam Celcius”
  - b. Memiliki validasi hanya angka.
  - c. Memiliki tampilan input keyboard khusus angka.
6. Setelah memasang properties tersebut berikut ini tampilan yang diharapkan



7. Wrap TextFormField kedalam sebuah kolom dengan menekan Ctrl + .

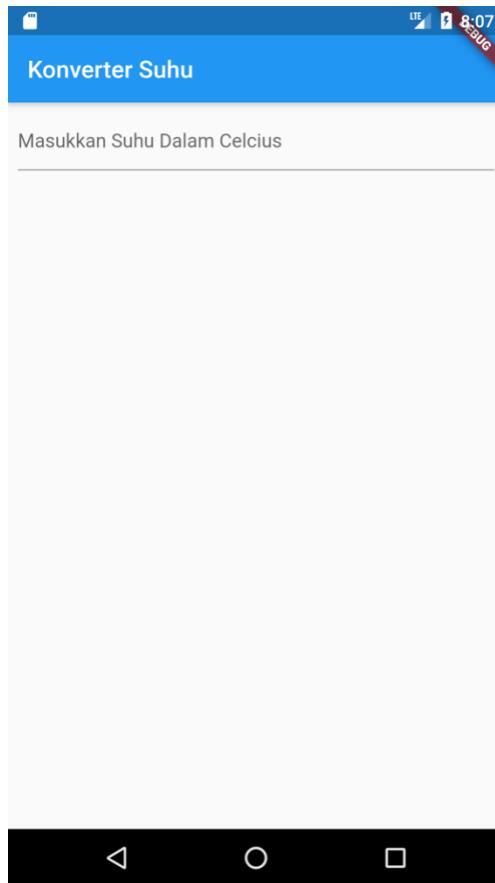
```
body: Container( margin: EdgeInsets.all(8), child: TextFormField( // TextFormField )], // Container ), // Scaffold ); // MaterialApp } }
```

The code editor shows a context menu for the TextFormField widget. The menu items are:

- Wrap with widget...
- Wrap with Center
- Wrap with Column** (highlighted)
- Wrap with Container
- Wrap with Padding
- Wrap with Row
- Wrap with SizedBox
- Wrap with StreamBuilder
- Extract Method
- Extract Widget

At the bottom of the screen, there are tabs for EMS, TERMINAL, and OUTPUT. The TERMINAL tab shows the IP address 192.168.1.95:5555.

8. Berikut ini tampilan program setelah di wrap ke column



9. Langkah selanjutnya adalah mengisi Row untuk Text Info Suhu dalam Kelvin dan Reamur.
10. Langkah selanjutnya adalah mengisi sebuah button yang memiliki warna biru text putih dan even on click null.
11. Lakukanlah langkah 9 dan 10 secara mandiri sehingga layout yang dibentuk menjadi seperti pada gambar desain. Berikut ini resource yang dapat anda gunakan untuk menyelesaikan langkah 9 dan 10.
  - a. <https://flutter.dev/docs/cookbook/forms/text-input>
  - b. <https://stackoverflow.com/questions/50014342/button-width-match-parent>
  - c. <https://medium.com/flutter-community/beginners-guide-to-text-styling-in-flutter-3939085d6607>
  - d. <https://medium.com/@yuvrajpandey24/working-with-raised-button-in-flutter-6f5c0f71aab3>
  - e. <https://daengweb.id/mengenal-widget-flutter-2-container-stack-positioned>
  - f. <https://medium.com/jlouage/container-de5b0d3ad184>
  - g. <https://stackoverflow.com/questions/53850149/flutter-crossaxisalignment-vs-mainaxisalignment>

### 3.3 Membuat Aplikasi Interaktif

Setelah berhasil membuat layout selanjutnya adalah membuat aplikasi yang mampu melakukan konversi suhu dari input berupa derajat celcius ke tiga derajat lain yaitu kelvin, reamur dan fahrenheit.

#### 3.3.1 Convert Ke Statefull Widget

Langkah pertama adalah dengan mengkonversi widget ke StatefullWidget. Hal ini dapat dilakukan dengan menekan tombol `Ctrl + .` pada widget yang ingin dikonversi. Berikut ini kode program sebelum dikonversi.

```
class MyApp extends StatelessWidget {  
    // This widget is the root of your application.  
  
    @override  
  
    Widget build(BuildContext context) {  
  
        return MaterialApp(  
            title: 'Flutter Demo',  
            theme: ThemeData(  
                primarySwatch: Colors.blue,  
                visualDensity: VisualDensity.adaptivePlatformDensity,  
            ),  
    }  
}
```

Setelah dilakukan konversi kode program berubah memiliki state seperti dibawah ini :

```
class MyApp extends StatefulWidget {  
  
    @override  
  
    _MyAppState createState() => _MyAppState();  
}  
  
class _MyAppState extends State<MyApp> {  
    // text controller
```

#### 3.3.2 Membuat State

Untuk membuat state perhatikan beberapa best practice berikut ini mengenai elemen mana dari UI yang akan dimasukkan kedalam state.

1. Apakah ada elemen pada UI yang berubah ?
2. Apakah yang menyebabkan elemen tersebut berubah ?

Jika diperhatikan dari UI yang disediakan maka elemen yang dapat berubah adalah input dari user, dan output dari proses konversi suhu. Sedangkan yang menjadi trigger perubahan adalah ketika tombol konversi ditekan. Pada pengembangan aplikasi menggunakan flutter

elemen yang berubah ini akan dimasukkan kedalam state dan elemen yang mentrigger perubahan akan disebut event.

Untuk membuat state tambahkanlah kode program yang menurut anda perlu ditambahkan berikut ini contoh kode program pada state dari widget MyApp yang sekarang berubah menjadi statefull widget.

```
class _MyAppState extends State<MyApp> {  
    //variabel berubah  
    double _inputUser = 0;  
    double _kelvin = 0;  
    //tambahkan variabel lain yang dibutuhkan
```

Berikut ini beberapa url yang dapat membantu proses pembuatan variabel state

1. <https://www.w3adda.com/dart-tutorial/dart-variables>

### 3.3.3 Mengambil Data Input Suhu

Selain memiliki variabel input tentu saja input ini harus diambil dari TextFormField yang menjadi cara user menginputkan angka untuk dikonversi, carilah cara menambahkan controller terhadap TextFormField ini. Berikut ini beberapa url yang dapat membantu proses tersebut.

1. <https://flutter.dev/docs/cookbook/forms/retrieve-input>
2. <https://stackoverflow.com/questions/61538657/how-to-get-value-from-textformfield-on-flutter>
3. <https://medium.com/flutter-developer-indonesia/retrieve-value-from-text-field-to-widget-text-e12a9f162307>

### 3.3.4 Membuat Event

Langkah selanjutnya adalah membuat event dimana event perhitungan konversi dan update hasil konversi dilakukan saat tombol Konversi ditekan. Untuk menyambungkan proses klik button ke perubahan state ada beberapa langkah yang harus anda lakukan.

1. Buat Fungsi di dalam state AppState
2. Panggil Fungsi ini ketika tombol konversi di tekan
3. Tambahkan Set State di fungsi tersebut dan ubahlah state dari variabel yang anda gunakan sebagai dasar perhitungan konversi.

4. Tambahkan kode program untuk melakukan konversi suhu dan set state variabel yang anda gunakan untuk kelvin, reamur dan fahrenheit.

Berikut ini url yang dapat anda gunakan untuk membantu proses ini.

1. <https://rumusrumus.com/rumus-konversi-suhu/>
2. <https://stackoverflow.com/questions/50758704/how-to-convert-to-double>
3. Project Hello World Flutter cek bagaimana function increment di panggil dan bagaimana state di set.

### 3.3.5 Menampilkan Output Result

Untuk menampilkan output anda hanya perlu menyambungkan variabel yang berubah pada state ke widget Text yang menampilkan hasil. Berikut ini beberapa url yang dapat membantu anda menyelesaikan proses ini.

1. <https://stackoverflow.com/questions/28419255/how-do-you-round-a-double-in-dart-to-a-given-degree-of-precision-after-the-decim>
2. Project Hello World Flutter cek bagaimana variabel counter dihubungkan ke Text Widget.

## 3.4 Membagi Ke Widget Yang Lebih Kecil

Selamat jika anda sampai pada langkah ini berarti aplikasi yang anda buat sudah dapat melakukan konversi dan selesai secara fitur, namun untuk proses development yang baik widget yang ada di main.dart harus dipotong potong menjadi widget yang lebih sederhana sehingga mudah untuk di maintain dan di perbaiki.

Lakukan lah extraksi widget ke file yang berbeda dengan cara Ctrl + . pada widget yang ingin anda potong dan pilih menu Extract Widget, berikanlah nama class yang sesuai dan pindahkan ke file baru.

Berikut ini nama nama widget dan ekstraksi yang dapat dicontoh :

1. Input : Berisi Widget TextFormField dan controller nya
2. Result : Berisi widget hasil konversi yang memiliki variabel nama dan hasil.
3. Convert : Berisi button dan reference ke function yang digunakan untuk melakukan setState().

Berikut ini resource yang dapat anda pelajari untuk melakukan konversi widget :

1. <https://medium.com/flutter-community/flutter-visual-studio-code-shortcuts-for-fast-and-efficient-development-7235bc6c3b7d>

### 3.4.1 Send Parameter ke Child

Sebuah widget dapat mengirim parameter ke child widgetnya melalui constructor yang dimiliki oleh widget tersebut. Parameter ini dapat menyesuaikan dengan kebutuhan yang dimiliki oleh widget tersebut. Berikut ini contoh pembuatan widget dan constructor dari widget yang dibuat.

```
Input(etInput: etInput),
```

```
class Input extends StatelessWidget {
    const Input({
        Key key,
        @required this.etInput,
    }) : super(key: key);

    final TextEditingController etInput;

    @override
    Widget build(BuildContext context) {
        return TextFormField(
            decoration: InputDecoration(hintText: "Masukkan Suhu Dalam Celcius"),
            inputFormatters: [FilteringTextInputFormatter.digitsOnly],
            controller: etInput,
            keyboardType: TextInputType.number,
        );
    }
}
```

### 3.4.2 Send Event Handler Ke Child

Selain mengirimkan parameter ke child widget parent juga dapat mengirim fungsi turun ke child widget dimana fungsi ini akan dipanggil lagi oleh parent widget yang memiliki state. Berikut ini contoh kode program untuk menurunkan fungsi ke child widget.

```
Convert(konvertHandler: _konversiSuhu),
```

```
class Convert extends StatelessWidget {
    final Function konvertHandler;

    Convert({Key key, @required this.konvertHandler});
```

```
    @override
    Widget build(BuildContext context) {
        return Container(
            width: double.infinity,
            height: 50,
            child: RaisedButton(
                onPressed: konvertHandler,
                color: Colors.blueAccent,
                textColor: Colors.white,
                child: Text("Konversi Suhu"),
            ),
        );
    }
}
```

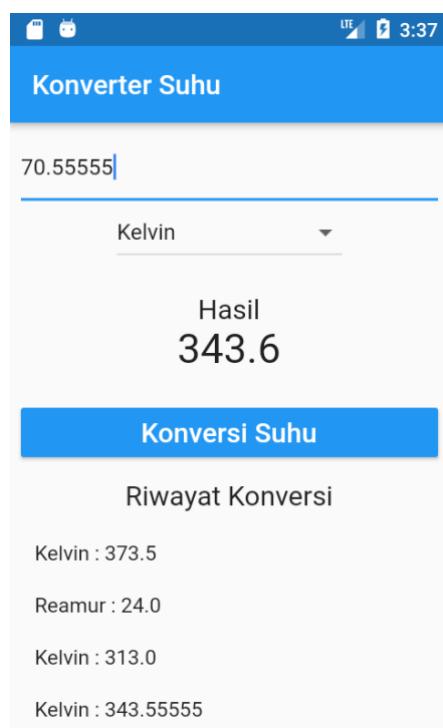
### 3.5 Praktikum 1

1. Selesaikan Langkah Praktikum Di Atas dan Submit ke Repository Github

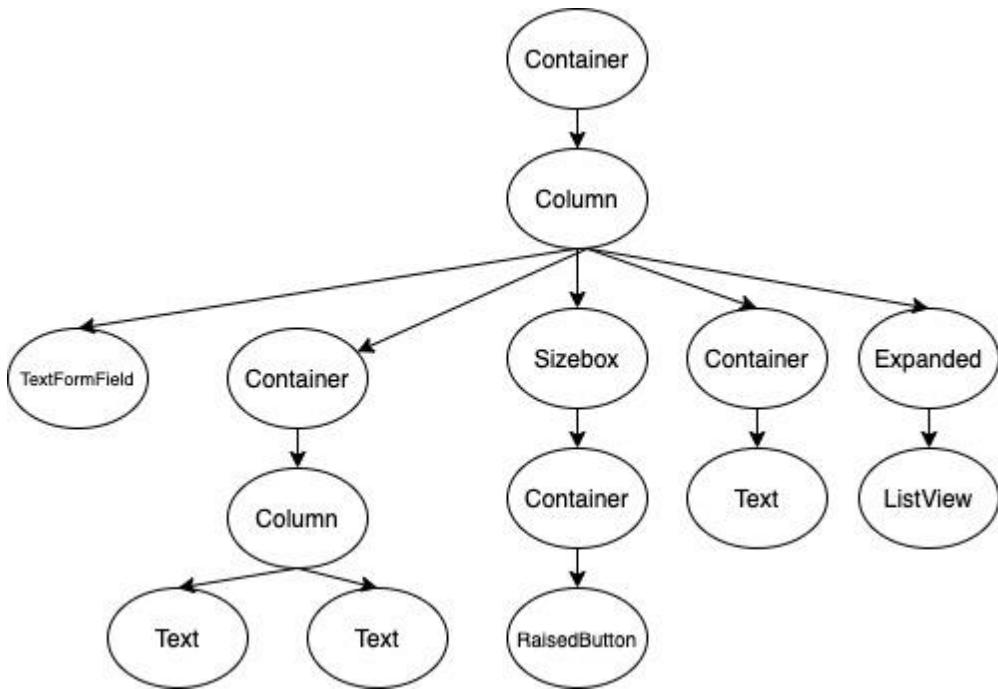
## BAB 4. STATEFULL WIDGET DAN MAP

### 4.1 Desain Aplikasi

Pada praktikum bab 4 anda akan belajar mengenai cara menggunakan array atau list untuk membuat aplikasi dengan menggunakan flutter. Aplikasi yang dibuat akan mengembangkan aplikasi yang telah dibuat pada bab 3 yaitu untuk mengkonversi suhu. Berikut adalah desain mockup aplikasi yang akan dibuat. Pada aplikasi bab 4 akan ditambahkan DropdownButton dan ListView widget.



Widget tree yang digunakan penulis dapat diilustrasikan pada gambar di bawah ini. Widget tree yang dibuat tidak terbatas atau harus seperti pada gambar di bawah ini untuk membuat tampilan seperti mockup di atas.



## 4.2 Teori

### 4.2.1 List

Sebelum masuk pada praktikum akan dijelaskan pengantar tentang List pada Dart. List adalah collection yang seperti pada bahasa pemrograman yang lain. Pada bahasa pemrograman Dart, array adalah list of object sehingga disebut juga dengan list. Berikut ini adalah contoh list pada Dart

```
//contoh list of integer
var list = [1, 2, 3];

//contoh list of string
var list = [ 'Car', 'Boat', 'Plane', ];
```

List pada Dart menggunakan index 0 sebagai nilai awal dan **list.length – 1** adalah index nilai terakhir dari sebuah list.

List pada Dart menggunakan index 0 sebagai nilai awal dan **list.length – 1** adalah index nilai terakhir dari sebuah list.

```
var list = [1, 2, 3];
print(list.length == 3);
```

```
print (list[1] == 2);
```

Untuk membuat list pada saat compile time dapat menggunakan const

```
var constantList = const [1, 2, 3];
```

Mulai Dart 2.3 dikenalkan **spread operator** (...) dan **null-aware spread operator** (...?) yang digunakan insert banyak nilai ke collection. Contohnya pada source code di bawah ini, digunakan untuk memasukkan nilai dari list ke list2.

```
var list = [1, 2, 3];
var list2 = [0, ...list];
print(list2.length == 4);
```

Anda juga dapat menggunakan **null-aware spread operator** untuk menghindari exception ketika list bernilai null.

```
var list;
var list2 = [0, ...?list];
print(list2.length == 1);
```

Dart juga menyediakan **collection if** dan **collection for** yang digunakan untuk membuat collection dengan kondisi dan repetisi / perulangan. Berikut adalah contoh **collection if** untuk membuat list yang terdiri dari 3 atau 4 item sesuai dengan kondisi.

```
var nav = [
  'Home',
  'Furniture',
  'Plants',
  if (promoActive) 'Outlet'
];
```

Berikut adalah contoh **collection for** untuk memanipulasi list item sebelum ditambahkan pada list yang lain.

```
var listOfInts = [1, 2, 3];
var listOfStrings = [
  '#0', for (var i in listOfInts) '#$i'
];
print(listOfStrings[1] == '#1');
```

## 4.3 Praktikum 1 Menambahkan Dropdown Widget

### 4.3.1 Tambahkan code berikut dibawah Input

Pada bab 3, anda telah membuat source code untuk mengkonvert dari kelvin dan reamur. Cuplikan source code anda adalah sebagai berikut :

Berdasarkan ilustrasi mockup, pada praktikum ini kita akan menambahkan dropdown dan listview. Pada praktikum pertama akan ditambahkan dropdown dibawah Input. Tambahkan code berikut.

```
DropdownButton(
  items: [
    DropdownMenuItem(
      value: "Kelvin", child: Container(child: Text("Kelvin"))),
    DropdownMenuItem(
      value: "Reamur", child: Container(child: Text("Reamur"))),
  ],
  value: null,
  onChanged: (String changeValue) {},
),
```

### 4.3.2 Menggunakan Map pada DropDownButton

Dapat diamati pada source code dropdown items merupakan list yang terdiri dari DropdownMenuItem Widget. Hal ini akan tidak efektif karena code kita akan panjang jika

terdapat banyak item yang harus dimasukkan ke items. Hal ini dapat ditangani dengan menggunakan fungsi map pada list. Pertama buat list seperti berikut:

```
var listItem = [
    "Kelvin",
    "Reamur"
];
```

Selanjutnya ubah DropdownButton menjadi seperti berikut. Maksut dari code dibawah ini adalah :

`DropdownButton<String>` = String digunakan untuk memberi tipe data value dari Dropdown adalah bertipe String.

`listItem.map((String value)` = Digunakan untuk melakukan iterasi untuk setiap item dari listItem sesuai dengan parameter bertipe String.

```
DropdownButton<String>(
    items:
        listItem.map((String value) {
            return DropdownMenuItem<String>(
                value: value,
                child: Text(value),
            );
        }).toList(),
        value: null,
        onChanged: (String changeValue) {},
    ),
```

#### 4.3.3 Mengeset Parameter value pada DropdownButton

Parameter value pada DropdownButton digunakan untuk mengeset nilai pada DropdownButton. Agar nilai defaultnya pada saat pertama muncul Text “Pilih Konversi ke Suhu” maka langkah yang dapat dilakukan adalah membuat variabel bertipe String seperti berikut pada class yang mengextend State.

```
class MyAppState extends State<MyApp> {

    double _inputUser = 0;
    double _kelvin = 0;
    double _reamur = 0;
    final TextEditingController inputController = TextEditingController();
    String _newValue = "Kelvin";
    double _result = 0;

    .
    .
    .
    .
}
```

Selanjutnya isi value dengan variabel `_newValue`.

```
DropdownButton<String>(
    items:
        listItem.map((String value) {
            return DropdownMenuItem<String>(
                value: value,
                child: Text(value),
            );
        }).toList(),
    value: _newValue,
    onChanged: (String changeValue) {},
),
```

#### 4.3.4 Mengisi Fungsi pada onChanged

Pada saat ini nilai dari `_newValue` tidak diupdate ketika user mengubah pilihan pada dropdown. Hal tersebut dapat ditangani pada fungsi `onChanged` seperti berikut.

```
onChanged: (String changeValue) {
    setState(() {
        _newValue = changeValue;
    });
},
```

Atau anda juga dapat menggunakan fungsi terpisah kemudian gunakan fungsi `dropdownOnChanged` pada parameter `value`

```
void dropdownOnChanged(String changeValue) {
    setState(() {
        _newValue = changeValue;
    });
}
```

#### 4.3.5 Mengubah Result.dart

Langkah selanjutnya adalah mengubah Result.dart sehingga memiliki tampilan dan hanya menerima 1 variabel.

```
class Result extends StatelessWidget {
    const Result({
        Key key,
        @required this.result,
    }) : super(key: key);

    final double result;

    @override
    Widget build(BuildContext context) {
        return
            Container(
                margin: EdgeInsets.only(top: 20,bottom: 20),
                child: Column(
                    mainAxisAlignment: MainAxisAlignment.center,
                    children: [
                        Text("Hasil",style: TextStyle(fontSize: 20),),
                        Text(
                            result.toStringAsFixed(1),
                            style: TextStyle(fontSize: 30),
                        )
                    ],
                ),
            );
    }
}
```

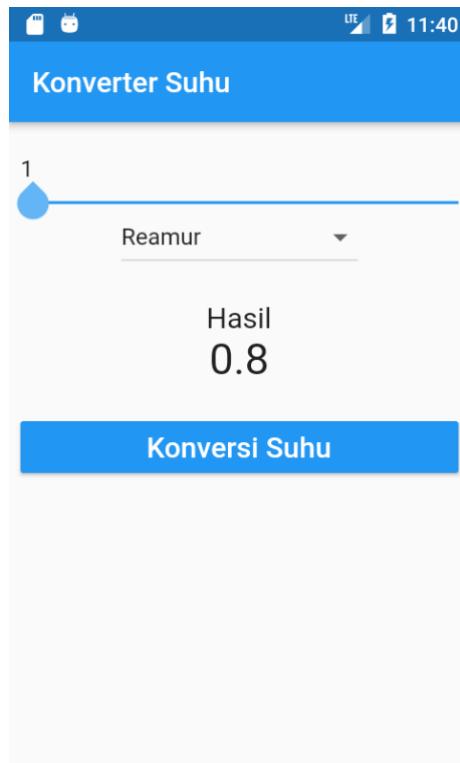
#### 4.3.6 Mengubah Fungsi Perhitungan Suhu

Fungsi perhitungan suhu perlu untuk diubah sehingga hanya memproses konversi sesuai dengan pilihan pengguna.

```
void perhitunganSuhu() {
    setState(() {
        _inputUser = double.parse(inputController.text);

        if (_newValue == "Kelvin")
            _result = _inputUser + 273;
        else
            _result = (4 / 5) * _inputUser;
    });
}
```

Sampai Pada tahap ini anda sudah belajar cara untuk menggunakan list dan fungsi map yang ada di Dart. Dan hasil aplikasi yang ada buat adalah sebagai berikut.



#### 4.3.7 Tugas Praktikum 1

Kalian ubah proses pada aplikasi sehingga dapat memproses ketika ada perubahan dropdown tanpa di klik konversi suhu.

#### 4.4 Tugas Praktikum 2 Menambahkan ListView Widget

Pada praktikum kedua sama dengan praktikum pertama yaitu dengan memanfaatkan map pada list dapat digunakan untuk mengisi history dari ListView. Langkah-langkah untuk menambah history adalah sebagai berikut.

1. Membuat variable bertipe List<String> dengan menggunakan code berikut

```
List<String> listViewItem = List<String>();
```

2. Mengisi listViewItem setiap kali terjadi proses konversi
3. Menampilkan hasil listViewItem menggunakan map, menggunakan code berikut.

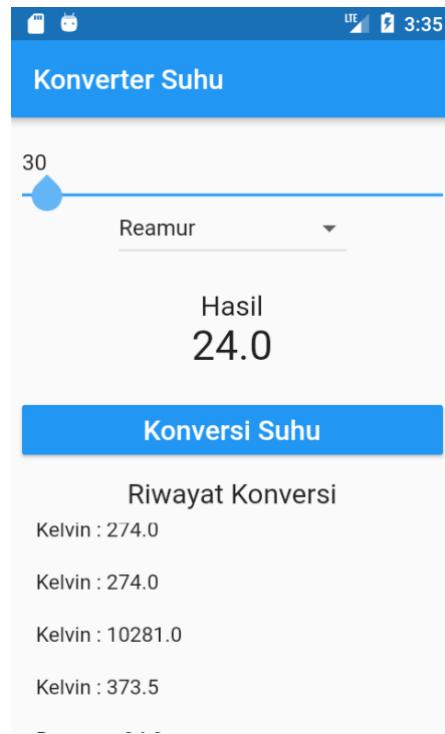
```
listViewItem.map((String value) {  
    return Container(  
        margin: EdgeInsets.all(10),  
        child: Text(  
            value,  
            style: TextStyle(fontSize: 15),  
        ),  
    ).toList()
```

4. Menjadikan widget yang lebih kecil dengan menggunakan extract widget. Hasil akhir setelah diextract widget adalah seperti berikut.

```
body: Container(  
    margin: EdgeInsets.all(8),  
    child: Column(  
        children: [  
            Input(inputController: inputController),  
            DropdownKonversi(listItem: listItem, newValue: _newValue,  
dropdownOnChanged : dropdownOnChanged),  
            Result(result: _result),  
            Convert(konvertHandler: perhitunganSuhu),  
            Container(  
                margin: EdgeInsets.all(10),  
                child: Text(  
                    result,  
                    style: TextStyle(fontSize: 15),  
                ),  
            ),  
        ],  
    ),  
);
```

```
margin: EdgeInsets.only(top: 10, bottom: 10),  
child: Text(  
    "Riwayat Konversi",  
    style: TextStyle(fontSize: 20),  
) ,  
) ,  
Expanded(  
    child: RiwayatKonversi(listViewItem: listViewItem),  
) ,  
] ,  
) ,  
) ,
```

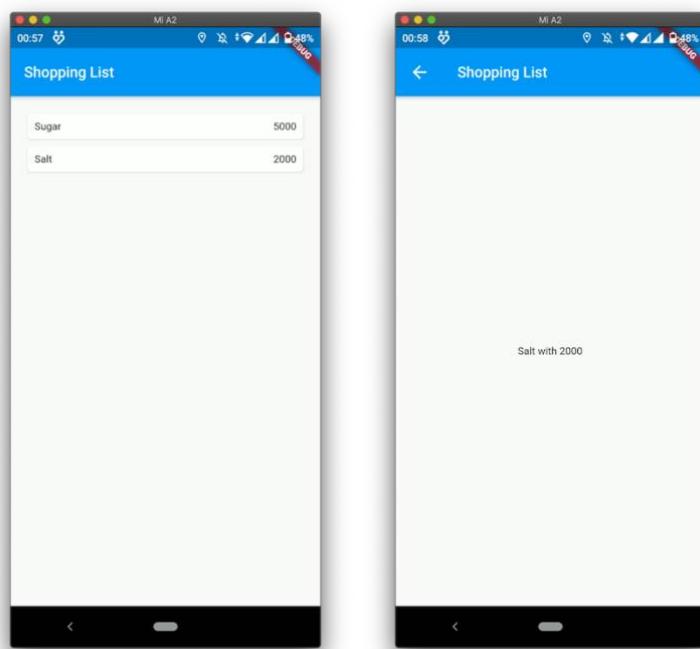
Lengkapilah source code yang kurang sehingga mendapatkan hasil akhir praktikum seperti berikut.



## BAB 5. NAVIGASI DAN RUTE

### 5.1 Desain Aplikasi

Pada praktikum bab 5 ini anda akan belajar mengenai pembangunan aplikasi bergerak multi halaman. Aplikasi yang dikembangkan berupa kasus daftar barang belanja. Pada aplikasi ini anda akan belajar untuk berpindah halaman dan mengirimkan data ke halaman lainnya. Gambaran mockup hasil akhir aplikasi dapat anda lihat pada gambar berikut (mockup dibuat sesederhana mungkin, sehingga anda mempunyai banyak ruang untuk berkreasi). Desain aplikasi menampilkan sebuah ListView widget yang datanya bersumber dari List. Ketika item ditekan, data akan dikirimkan ke halaman berikutnya.



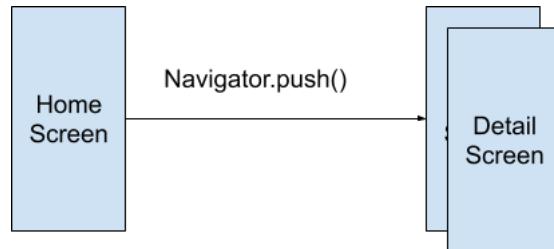
### 5.2 Teori

Perpindahan halaman di Flutter, ditangani oleh Navigator dengan melibatkan konsep sebagai berikut:

- Navigator: sebuah widget yang mengatur tumpukan (struktur data stack) dari obyek rute
- Route: sebuah obyek yang merepresentasikan tampilan, umumnya diimplementasikan oleh class seperti MaterialPageRoute

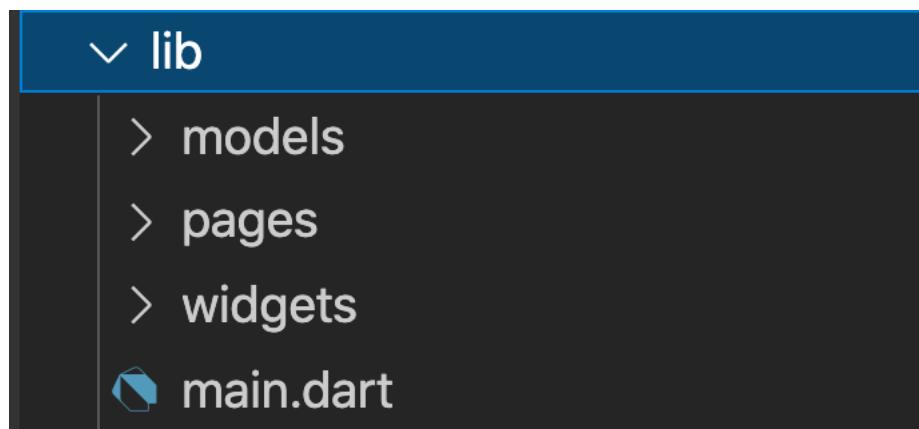
Sebuah Route umumnya dimasukkan (*push*) atau diambil (*pop*) dari dan ke tumpukan Navigator. Ketika sebuah halaman dilakukan operasi push, maka halaman tersebut akan

diletakkan di atas halaman yang memanggilnya. Ilustrasi tersebut dapat anda lihat pada gambar berikut. Dan jika pop dipanggil (tombol back ditekan) maka aplikasi akan menampilkan halaman sebelumnya. Selain itu Flutter juga mendukung adanya penamaan Route yang didefinisikan di awal.



### 5.3 Praktikum

Sebelum melanjutkan praktikum, buatlah sebuah project baru Flutter dengan nama belanja dan susunan folder seperti pada gambar berikut. Penyusunan ini dimaksudkan untuk mengorganisasi kode yang lebih mudah.



#### 5.3.1 Praktikum 1 - Mendefinisikan Route

Buatlah dua buah file dart dengan nama home\_page.dart dan item\_page.dart pada folder pages. Untuk masing-masing file, deklarasikan class HomePage pada file home\_page.dart dan ItemPage pada item\_page.dart. Turunkan class dari StatelessWidget. Gambaran potongan kode, dapat anda lihat pada potongan berikut.

```
class HomePage extends StatelessWidget {  
    @override  
    Widget build(BuildContext context) {  
        // TODO: implement build  
        throw UnimplementedError();  
    }  
}
```

Setelah kedua halaman telah dibuat dan didefinisikan, bukalah file main.dart. Pada langkah ini anda akan mendefinisikan Route untuk kedua halaman tersebut. Definisi penamaan route harus bersifat *unique*. Halaman HomePage didefinisikan sebagai /. Dan halaman ItemPage didefinisikan sebagai /item. Untuk mendefinisikan halaman awal, anda dapat menggunakan named argument initialRoute. Gambaran tahapan ini, dapat anda lihat pada potongan kode berikut.

```
void main() {  
    runApp(MaterialApp(  
        initialRoute: '/',  
        routes: {  
            '/': (context) => HomePage(),  
            '/item': (context) => ItemPage(),  
        },  
    )); // MaterialApp  
}
```

### 5.3.2 Praktikum 2 - Berpindah Halaman

Sebelum melakukan perpindahan halaman dari HomePage ke ItemPage, dibutuhkan proses pemodelan data. Pada desain mockup, dibutuhkan dua informasi yaitu nama dan harga. Untuk menangani hal ini, buatlah sebuah file dengan nama item.dart dan letakkan pada folder models. Pada file ini didefinisikan pemodelan data yang dibutuhkan. Ilustrasi kode yang dibutuhkan, dapat anda lihat pada potongan kode berikut.

```
class Item {  
    String name;  
    int price;  
  
    Item({this.name, this.price});  
}
```

Pada halaman HomePage terdapat ListView widget. Sumber data ListView diambil dari model List dari object Item. Gambaran kode yang dibutuhkan untuk melakukan definisi model dapat anda lihat sebagai berikut.

```
class HomePage extends StatelessWidget {  
    final List<Item> items = [  
        Item(name: 'Sugar', price: 5000),  
        Item(name: 'Salt', price: 2000)  
   ];
```

Untuk menampilkan ListView pada praktikum ini digunakan itemBuilder. Data diambil dari definisi model yang telah dibuat sebelumnya. Untuk menunjukkan batas data satu dan berikutnya digunakan juga widget Card. Kode yang telah umum, pada bagian ini tidak ditampilkan. Gambaran kode yang dibutuhkan dapat anda lihat sebagai berikut.

```
body: Container(
    margin: EdgeInsets.all(8),
    child: ListView.builder(
        padding: EdgeInsets.all(8),
        itemCount: items.length,
        itemBuilder: (context, index) {
            final item = items[index];
            return Card(
                child: Container(
                    margin: EdgeInsets.all(8),
                    child: Row(
                        children: [
                            Expanded(child: Text(item.name)),
                            Expanded(
                                child: Text(
                                    item.price.toString(),
                                    textAlign: TextAlign.end,
                                ), // Text
                            ) // Expanded
                        ],
                    ), // Row
                ), // Container
            ); // Card
        },
    ), // ListView.builder
), // Container
```

Jalankan aplikasi pada emulator atau pada device anda. Pastikan pada halaman awal telah berhasil menampilkan ListView. Jika ada kesalahan, segera perbaiki sebelum melanjutkan ke langkah berikutnya.

Item pada ListView saat ini ketika ditekan masih belum memberikan aksi tertentu. Untuk menambahkan aksi pada ListView dapat digunakan widget InkWell atau GestureDetector. Perbedaan utamanya InkWell merupakan material widget yang memberikan efek ketika ditekan. Sedangkan GestureDetector bersifat umum dan bisa juga digunakan untuk gesture lain selain sentuhan. Pada praktikum kali ini akan digunakan widget InkWell.

Untuk menambahkan sentuhan, letakkan cursor pada widget pembuka Card. Kemudian gunakan shortcut quick fix dari VSCode (`Ctrl + .` atau `Cmd + .` pada

MacOS). Sorot menu wrap with widget... Ubah nilai widget menjadi InkWell serta tambahkan named argument onTap yang berisi fungsi untuk berpindah ke halaman ItemPage. Ilustrasi potongan kode dapat anda lihat pada potongan berikut.

```
return InkWell(  
    onTap: () {  
        Navigator.pushNamed(context, '/item');  
    },
```

Jalankan aplikasi kembali dan pastikan ListView dapat disentuh dan berpindah ke halaman berikutnya. Periksa kembali jika terdapat kesalahan.

### 5.3.3 Praktikum 3 - Pengiriman Data

Untuk melakukan pengiriman data ke halaman berikutnya, cukup menambahkan informasi arguments pada penggunaan Navigator. Perbaharui kode pada bagian Navigator menjadi berikut.

```
Navigator.pushNamed(context, '/item', arguments: item);
```

### 5.3.4 Praktikum 4 - Pembacaan Data

Pembacaan nilai yang dikirimkan pada halaman sebelumnya dapat dilakukan menggunakan ModalRoute. Tambahkan kode berikut pada blok fungsi build dalam halaman ItemPage. Setelah nilai didapatkan, anda dapat menggunakannya seperti penggunaan variabel pada umumnya.

```
final Item itemArgs = ModalRoute.of(context).settings.arguments;
```

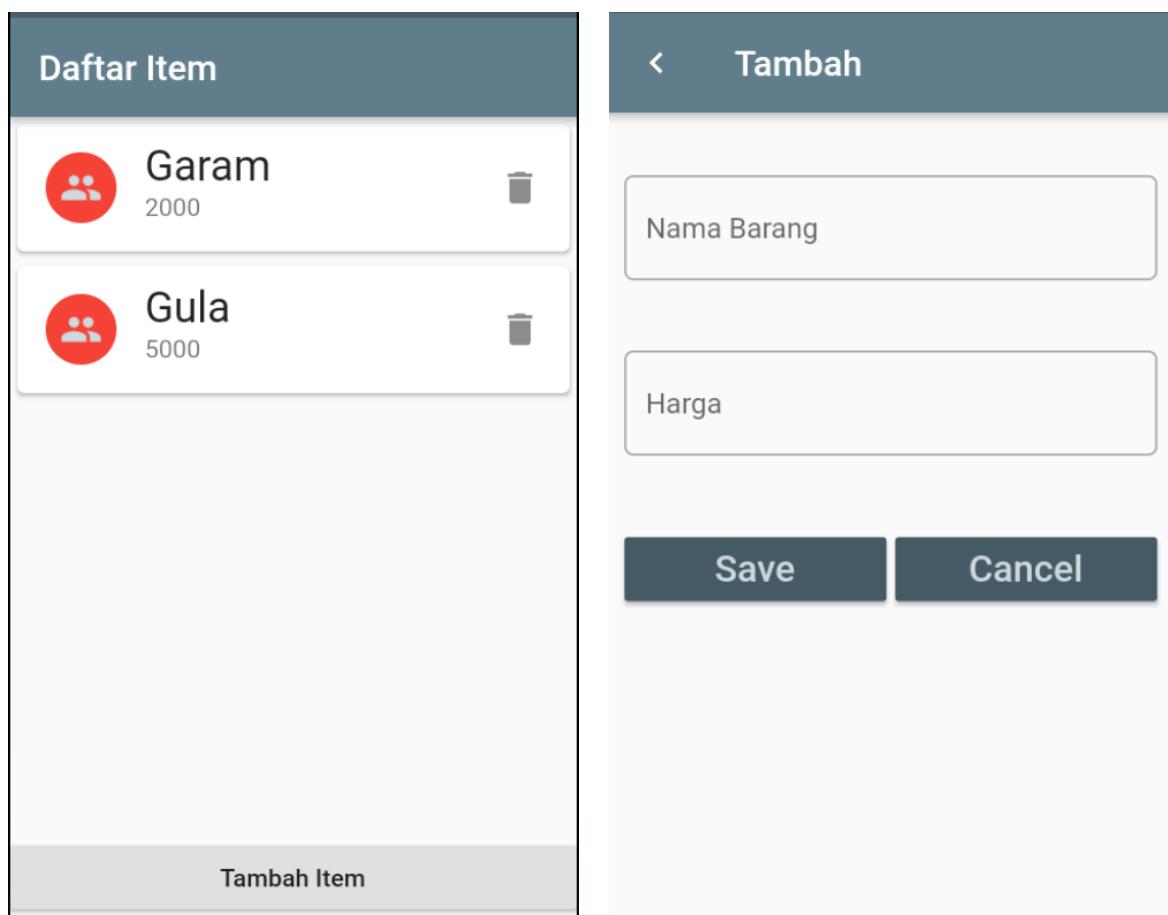
### 5.3.5 Tugas Praktikum

Pada hasil akhir dari aplikasi multi halaman yang telah anda selesaikan, tambahkan setidaknya satu atribut tambahan. Sesuaikan dan modifikasi tampilan sehingga menjadi aplikasi yang menarik. Selain itu, pecah widget menjadi kode yang lebih kecil.

## BAB 6. DATABASE SQLITE

### 6.1 Desain Aplikasi

Pada praktikum bab 6 anda akan belajar cara untuk melakukan CRUD (create, read, update dan delete) pada SQLite. Untuk pembuatan aplikasi ini dimulai dengan membuat sebuah desain yang dapat dimulai dengan membuat wireframe atau desain utuh di perangkat lunak seperti adobe xd atau sketch dan yang lain. Untuk menyederhanakan proses desain anda sudah disediakan template sederhana di starter code dengan menggunakan template material design. Berikut ini desain mockup aplikasi yang dibuat.



### 6.2 Teori

#### 6.2.1 Operasi pada SQLite

Berikut adalah daftar SQL statement yang dapat anda lakukan pada SQLite berdasarkan sumber <https://www.sqlitetutorial.net/> :

1. Simple Query, contohnya Select

2. Mengurutkan (sorting), contohnya Order By
3. Filtering data, contohnya Select Distinct, Where, Between, In, Like, Glob, IS NULL
4. Join tables, contohnya SQLite Join, Inner Join, Left Join, Cross Join, Self Join, Full Outer Join
5. Group Data, contohnya Group By, Having
6. Set Operator, contohnya Union, Except dan Intersect
7. Subquery
8. Case statement
9. Mengubah data, contohnya Update, Delete, Insert dan Replace
10. Transaction Statement
11. Constraints
12. View
13. Indexed
14. Triggered
15. Full text search
16. Sqlite Tools, contohnya SQLite Commands, SQLite Show Table, SQLite Dump, SQLite Import CSV dan SQLite Export CSV

Pada praktikum ini kita fokus untuk melakukan simple query dan mengubah data pada SQLite.

### 6.3 Praktikum

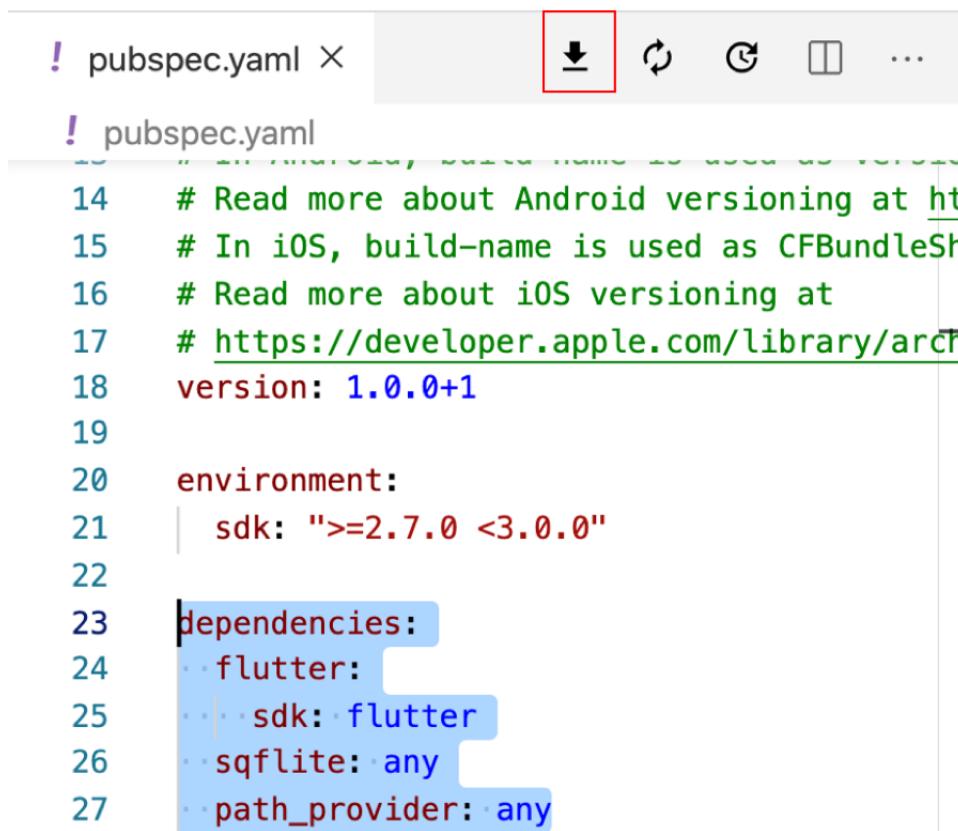
Untuk memanfaatkan SQLite pada Flutter berikut adalah langkah-langkah yang perlu untuk dilakukan:

#### 6.3.1 Mengatur dependencies pada pubspec.yaml

Tambahkan dependency sqlite dan path\_provider seperti pada code di bawah ini selanjutnya tekan yang diberi kotal merah untuk download packages yang baru. Untuk nilai **Any** dapat diisi versi terbaru atau sesuai kebutuhan. Versi packages sqflite dapat dicek di

<https://pub.dev/packages/sqlite> dan path\_provider dapat dicek di

[https://pub.dev/packages/path\\_provider](https://pub.dev/packages/path_provider).



```
! pubspec.yaml X
! pubspec.yaml
14 # Read more about Android versioning at ht...
15 # In iOS, build-name is used as CFBundleSh...
16 # Read more about iOS versioning at
17 # https://developer.apple.com/library/arch...
18 version: 1.0.0+1
19
20 environment:
21   sdk: ">=2.7.0 <3.0.0"
22
23 dependencies:
24   flutter:
25     sdk: flutter
26     sqflite: any
27     path_provider: any
```

### 6.3.2 Membuat Data Model

Buat class Model Item dengan nama **item.dart** yang dibuat hamper sama dengan praktikum sebelumnya hanya ditambahkan atribut `_id`. Jadi total ada 3 atribut yaitu `_id`, `_name`, dan `_price`.

```
class Item{
  int _id;
  String _name;
  int _price;
}
```

Selanjutnya buat getter dan setter untuk masing-masing variabel. Untuk getter dan setter secara otomatis dapat degenerate menggunakan extension pada vscode dengan nama Dart Getter and Setters. Jika menggunakan extension tersebut restart VSCode dan klik kanan pada variabel yang akan degenerate getter dan setternya.

getter akan mengambil nilai yang dimasukkan ke constructor dan setter ini akan dipakai untuk mengembalikan nilai yang dimasukkan dari constructor, untuk setiap variable.

```
class Item{  
  
    int _id;  
  
    String _name;  
  
    int _price;  
  
  
  
    int get id => _id;  
  
  
    String get name => this._name;  
  
    set name(String value) => this._name = value;  
  
  
    get price => this._price;  
  
    set price( value) => this._price = value;  
  
  
    // konstruktor versi 1  
  
    Item(this._name, this._price);  
}
```

Kita akan membuat beberapa constructor pada class Item. Pertama buat constructor untuk mengeset nilai name dan price secara bersama-sama.

```
// konstruktor versi 1  
  
Item(this._name, this._price);
```

Constructor kedua adalah berbentuk map digunakan untuk mengambil data dari sql yang tersimpan berbentuk Map setelah itu akan disimpan kembali dalam bentuk variabel

```
// konstruktor versi 2: konversi dari Map ke Item

Item.fromMap(Map<String, dynamic> map) {

    this._id = map['id'];

    this._name = map['name'];

    this._price = map['price'];

}
```

Terakhir membuat method Map untuk melakukan update dan insert.

```
// konversi dari Item ke Map

Map<String, dynamic> toMap() {

    Map<String, dynamic> map = Map<String, dynamic>();

    map['id'] = this._id;

    map['name'] = name;

    map['price'] = price;

    return map;

}

}
```

Code lengkap dari Item.Dart adalah sebagai berikut:

```
class Item{

    int _id;

    String _name;

    int _price;
```

```
int get id => _id;

String get name => this._name;

set name(String value) => this._name = value;

get price => this._price;

set price( value) => this._price = value;

// konstruktor versi 1

Item(this._name, this._price);

// konstruktor versi 2: konversi dari Map ke Item

Item.fromMap(Map<String, dynamic> map) {

    this._id = map['id'];

    this._name = map['name'];

    this._price = map['price'];

}

// konversi dari Item ke Map

Map<String, dynamic> toMap() {

    Map<String, dynamic> map = Map<String, dynamic>();

    map['id'] = this._id;

    map['name'] = name;

    map['price'] = price;

    return map;

}
```

```
}
```

### 6.3.3 Membuat Db Helper

Langkah praktikum kedua adalah membuat dbhelper.dart. Pertama adalah membuat fungsi untuk menginisialisasi database.

Future adalah “tipe data” yang terpanggil dengan adanya delay atau “keterlambatan”. Tidak seperti method lainnya, sistem akan terus menjalankan method tersebut sampai method itu selesai berjalan. Contohnya ketika kita akan mengambil data yang ada di dalam database/API , kita membutuhkan method Future untuk mengambil data di dalam database/API tersebut. Untuk lebih jelasnya kalian bisa membuka sumber dibawah:

1. <https://medium.com/flutter-community/a-guide-to-using-futures-in-flutter-for-beginners-ebeddfbf967>
2. <https://www.youtube.com/watch?v=g9Uk1Xou0m4>

Didalam flutter ada async dan await.

- async : menggunakan future pada sebuah method, sehingga membuat sistem menunggu sampai terjadi Blocking. Makanya, method tersebut harus ditandai dengan async.
- await : Jika ada method yang ditandai await, maka artinya sistem harus menunggu sampai syntax tersebut selesai berjalan.

Pada source code, variable directory akan menunggu sampai method getApplicationDocumentsDirectory mengerjakan tugasnya. Method getApplicationDocumentsDirectory() berfungsi untuk mengambil direktori folder aplikasi untuk menempatkan data yang dibuat pengguna sehingga tidak dapat dibuat ulang oleh aplikasi tersebut. Setelah itu kita gunakan variable String path, untuk membuat nama database kita dengan mengambil lokasi directory nya dan menambahkannya dengan nama database item.db.

Setelah itu kita baru membuat database dan akses untuk membuka database-nya dengan openDatabase. Dalam method ini akan membutuhkan nama database-nya, dengan variable path yang kita buat sebelumnya. Kemudian ada parameter version dan onCreate.

- version: bisa dikatakan version adalah level untuk penggunaan databasenya. Karena kita bahkan belum memiliki tablenya, kita cukup menulisnya ‘1’.
- onCreate: bukan cuma onCreate, kita bisa menambahkanya dengan onConfigure, onUpgrade, dll. Tapi yang kita bahas hanya onCreate saja. Seperti namanya, fungsinya untuk membuat table supaya kita dapat mengaksesnya.

```
Future<Database> initDb() async {  
  
    //untuk menentukan nama database dan lokasi yg dibuat  
  
    Directory directory = await getApplicationDocumentsDirectory();  
  
    String path = directory.path + 'item.db';  
  
    //create, read databases  
  
    var itemDatabase = openDatabase(path, version: 1, onCreate: _createDb);  
  
    //mengembalikan nilai object sebagai hasil dari fungsinya  
  
    return itemDatabase;  
}
```

Selanjutnya membuat method untuk mengcreate database seperti berikut.

```
//buat tabel baru dengan nama item  
  
void _createDb(Database db, int version) async {  
  
    await db.execute(  
        "CREATE TABLE item ("  
            "id INTEGER PRIMARY KEY AUTOINCREMENT,"  
            "name TEXT,"
```

```
    price INTEGER  
  
)  
  
""");  
  
}
```

Selanjutnya membuat fungsi untuk melakukan CRUD (create, read, update dan delete).

Method pada insert, update, dan delete struktur dan logikanya sama. Hanya syntax SQL nya yang berbeda.

Pada source code variable Database db akan membuka akses ke database. Variable count digunakan untuk menampung hasil SQL — nya. Bertipe Integer karena ketika sistem berhasil dieksekusi, nilai yang dikeluarkan adalah 1.

```
//create databases  
  
Future<int> insert(Item object) async {  
  
    Database db = await this.database;  
  
    int count = await db.insert('item', object.toMap());  
  
    return count;  
  
}  
  
//update databases  
  
Future<int> update(Item object) async {  
  
    Database db = await this.database;  
  
    int count = await db.update('item', object.toMap(),  
  
        where: 'id=?',  
  
        whereArgs: [object.id]);  
  
    return count;  
  
}
```

```
//delete databases

Future<int> delete(int id) async {

    Database db = await this.database;

    int count = await db.delete('item',
        where: 'id=?',
        whereArgs: [id]);

    return count;
}

Future<List<Item>> getItemList() async {

    var itemMapList = await select();

    int count = itemMapList.length;

    List<Item> itemList = List<Item>();

    for (int i=0; i<count; i++) {

        itemList.add(Item.fromMap(itemMapList[i]));

    }

    return itemList;
}
```

Untuk source code lengkapnya adalah sebagai berikut.

```
import 'package:sqflite/sqflite.dart';

import 'dart:async';

import 'dart:io';

import 'package:path_provider/path_provider.dart';

import 'item.dart';
```

```
class DBHelper {  
  
    static DBHelper _dbHelper;  
  
    static Database _database;  
  
    DBHelper._createObject();  
  
  
    Future<Database> initDb() async {  
  
        //untuk menentukan nama database dan lokasi yg dibuat  
  
        Directory directory = await getApplicationDocumentsDirectory();  
  
        String path = directory.path + 'item.db';  
  
  
        //create, read databases  
  
        var itemDatabase = openDatabase(path, version: 4, onCreate: _createDb);  
  
  
        //mengembalikan nilai object sebagai hasil dari fungsinya  
  
        return itemDatabase;  
  
    }  
  
    //buat tabel baru dengan nama item  
  
    void _createDb(Database db, int version) async {  
  
        await db.execute(  
            "CREATE TABLE item (  
                id INTEGER PRIMARY KEY AUTOINCREMENT,  
                name TEXT,  
                price INTEGER  
            )  
        ");  
    }  
}
```

```
}

//select databases

Future<List<Map<String, dynamic>>> select() async {

    Database db = await this.initDb();

    var mapList = await db.query('item', orderBy: 'name');

    return mapList;

}

//create databases

Future<int> insert(Item object) async {

    Database db = await this.initDb();

    int count = await db.insert('item', object.toMap());

    return count;

}

//update databases

Future<int> update(Item object) async {

    Database db = await this.initDb();

    int count = await db.update('item', object.toMap(),

        where: 'id=?',

        whereArgs: [object.id]);

    return count;

}

//delete databases

Future<int> delete(int id) async {

    Database db = await this.initDb();
```

```
int count = await db.delete('item',
    where: 'id=?',
    whereArgs: [id]);
}

Future<List<Item>> getItemList() async {
    var itemMapList = await select();
    int count = itemMapList.length;
    List<Item> itemList = List<Item>();
    for (int i=0; i<count; i++) {
        itemList.add(Item.fromMap(itemMapList[i]));
    }
    return itemList;
}

factory DbHelper() {
    if (_dbHelper == null) {
        _dbHelper = DbHelper._createObject();
    }
    return _dbHelper;
}

Future<Database> get database async {
    if (_database == null) {
        _database = await initDb();
    }
    return _database;
```

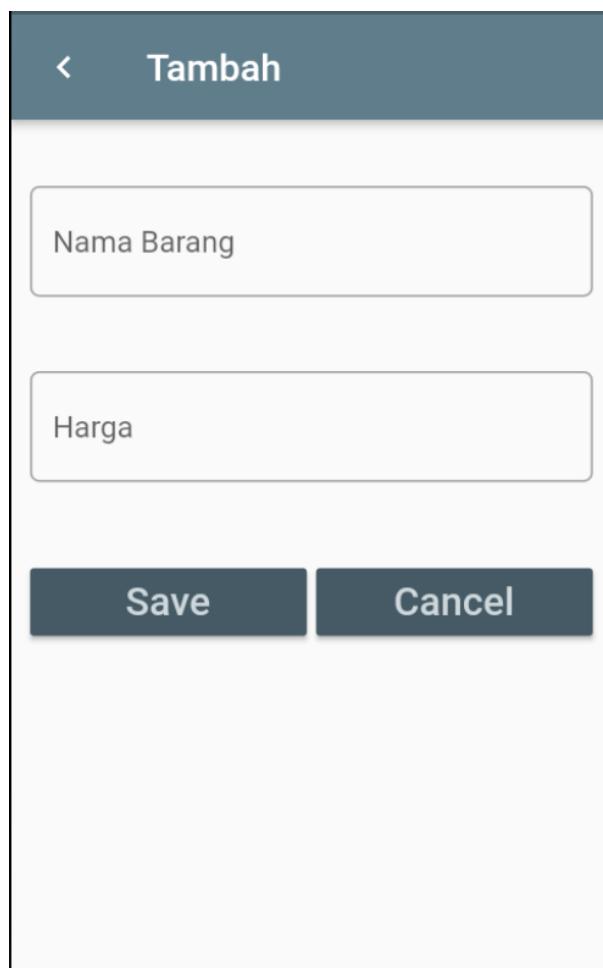
```
}
```

```
}
```

Pada tahap ini sebenarnya kita sudah selesai untuk membuat database SQLite pada flutter yang kurang hanya tampilannya saja. Untuk memanfaatkan fungsi diatas dapat dipanggil pada fungsi test di flutter.

#### 6.3.4 Membuat Entry Form

Buatlah UI untuk menginput dan mengedit isi dari table yang kita buat seperti berikut. Tampilan tidak harus sama persis dengan screenshoot di bawah ini yang paling penting adalah variabel TextEditingControllernya gunakan nama nameController dan priceControlle sehingga mudah dalam mengikuti jobsheet ini.



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

import 'item.dart';

class EntryForm extends StatefulWidget {

    final Item item;

    EntryForm(this.item);

    @override
    EntryFormState createState() => EntryFormState(this.item);
}

//class controller

class EntryFormState extends State<EntryForm> {

    Item item;

    EntryFormState(this.item);

    TextEditingController nameController = TextEditingController();
    TextEditingController priceController = TextEditingController();

    @override
    Widget build(BuildContext context) {
        //kondisi
        if (item != null) {
            nameController.text = item.name;
            priceController.text = item.price.toString();
        }
    }
}
```

```
//rubah

return Scaffold(
  appBar: AppBar(
    title: item == null ? Text('Tambah') : Text('Ubah'),
    leading: Icon(Icons.keyboard_arrow_left),
  ),
  body: Padding(
    padding: EdgeInsets.only(top: 15.0, left:10.0, right:10.0),
    child: ListView(
      children: <Widget> [
        // nama
        Padding (
          padding: EdgeInsets.only(top:20.0, bottom:20.0),
          child: TextField(
            controller: nameController,
            keyboardType: TextInputType.text,
            decoration: InputDecoration(
              labelText: 'Nama Barang',
              border: OutlineInputBorder(
                borderRadius: BorderRadius.circular(5.0),
              ),
            ),
            onChanged: (value) {
              //
            },
          ),
        ),
      ],
    ),
  ),
)
```

```
),

// harga

Padding (
    padding: EdgeInsets.only(top:20.0, bottom:20.0),
    child: TextField(
        controller: priceController,
        keyboardType: TextInputType.number,
        decoration: InputDecoration(
            labelText: 'Harga',
            border: OutlineInputBorder(
                borderRadius: BorderRadius.circular(5.0),
            ),
        ),
        onChanged: (value) {
            //
        },
    ),
),

// tombol button

Padding (
    padding: EdgeInsets.only(top:20.0, bottom:20.0),
    child: Row(
        children: <Widget> [
            // tombol simpan
            Expanded(
```

```
        child: RaisedButton(
```

```
            color: Theme.of(context).primaryColorDark,
```

```
            textColor: Theme.of(context).primaryColorLight,
```

```
            child: Text(
```

```
                'Save',
```

```
                textScaleFactor: 1.5,
```

```
            ),
```

```
            onPressed: () {
```

```
                if (item == null) {
```

```
                    // tambah data
```

```
                    item = Item(nameController.text, int.parse(priceController.text));
```

```
                } else {
```

```
                    // ubah data
```

```
                    item.name = nameController.text;
```

```
                    item.price = int.parse(priceController.text);
```

```
                }
```

```
                // kembali ke layar sebelumnya dengan membawa objek item
```

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

```
            },
```

```
        ),
```

```
        ),
```

```
        Container(width: 5.0,),
```

```
        // tombol batal
```

```
        Expanded(
```

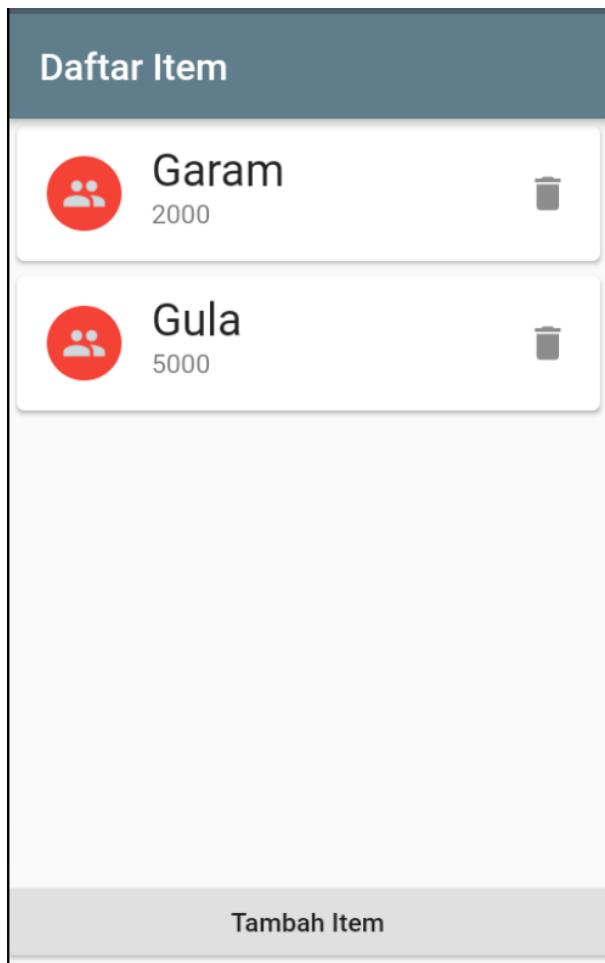
```
            child: RaisedButton(
```

```
                color: Theme.of(context).primaryColorDark,
```

```
        textColor: Theme.of(context).primaryColorLight,  
  
        child: Text(  
            'Cancel',  
            textScaleFactor: 1.5,  
        ),  
  
        onPressed: () {  
            Navigator.pop(context);  
        },  
    ),  
),  
],  
),  
),  
],  
),  
)  
);  
}  
}
```

### 6.3.5 Membuat Home

Buatlah UI untuk menampilkan record yang ada di SQLite pada class home.dart untuk melakukan aksi delete, edit dan menambah data seperti berikut.



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

import 'package:sqflite/sqflite.dart';

import 'dart:async';

import 'package:sqlite/dbhelper.dart';

import 'package:sqlite/entryform.dart';

import 'item.dart';

//pendukung program asinkron

class Home extends StatefulWidget {

  @override

  HomeState createState() => HomeState();
}
```

```
}
```

```
class HomeState extends State<Home> {
```

```
    DBHelper dbHelper = DBHelper();
```

```
    int count = 0;
```

```
    List<Item> itemList;
```

```
    @override
```

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

```
        if (itemList == null) {
```

```
            itemList = List<Item>();
```

```
        }
```

```
        return Scaffold(
```

```
            appBar: AppBar(
```

```
                title: Text('Daftar Item'),
```

```
            ),
```

```
            body: Column(children : [
```

```
                Expanded(child: createListView(),),
```

```
                Container(
```

```
                    alignment: Alignment.bottomCenter,
```

```
                    child: SizedBox(
```

```
                        width: double.infinity,
```

```
                        child: RaisedButton(
```

```
                            child: Text("Tambah Item"),
```

```
                            onPressed: () async {
```

```
var item = await navigateToEntryForm(context, null);

if (item != null) {

//TODO 2 Panggil Fungsi untuk Insert ke DB

int result = await dbHelper.insert(item);

if (result > 0) {

updateListView();

}

}

),

),

),

[]);

);

}

Future<Item> navigateToEntryForm(BuildContext context, Item item) async {

var result = await Navigator.push(

context,

MaterialPageRoute(


builder: (BuildContext context) {

return EntryForm(item);

}

)

);

return result;
}
```

```
}

ListView createListView() {

    TextStyle textStyle = Theme.of(context).textTheme.headline5;

    return ListView.builder(
        itemCount: count,
        itemBuilder: (BuildContext context, int index) {
            return Card(
                color: Colors.white,
                elevation: 2.0,
                child: ListTile(
                    leading: CircleAvatar(
                        backgroundColor: Colors.red,
                        child: Icon(Icons.ad_units),
                    ),
                    title: Text(this.itemList[index].name, style: textStyle,),
                    subtitle: Text(this.itemList[index].price.toString(),),
                    trailing: GestureDetector(
                        child: Icon(Icons.delete),
                        onTap: ()async {
                            //TODO 3 Panggil Fungsi untuk Delete dari DB berdasarkan Item
                        },
                    ),
                    onTap: () async {
                        var item = await navigateToEntryForm(context, this.itemList[index]);
                        //TODO 4 Panggil Fungsi untuk Edit data
                    },
                ),
            );
        }
    );
}
```

```
    },  
    ),  
    );  
},  
);  
}  
  
//update List item  
  
void updateListView() {  
    final Future<Database> dbFuture = dbHelper.initDb();  
  
    dbFuture.then((database) {  
  
        //TODO 1 Select data dari DB  
  
        Future<List<Item>> itemListFuture = dbHelper.getItemList();  
  
        itemListFuture.then((itemList) {  
  
            setState(() {  
  
                this.itemList = itemList;  
  
                this.count = itemList.length;  
  
            });  
        });  
    });  
}  
}
```

### 6.3.6 Membuat Main

```
//kode utama Aplikasi tampilan awal  
  
import 'package:flutter/material.dart';
```

```
import 'package:sqlite/home.dart';

//package letak folder Anda

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

class MyApp extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Tambahkan Item',
      theme: ThemeData(
        primarySwatch: Colors.blueGrey,
      ),
      home: Home(),
    );
  }
}
```

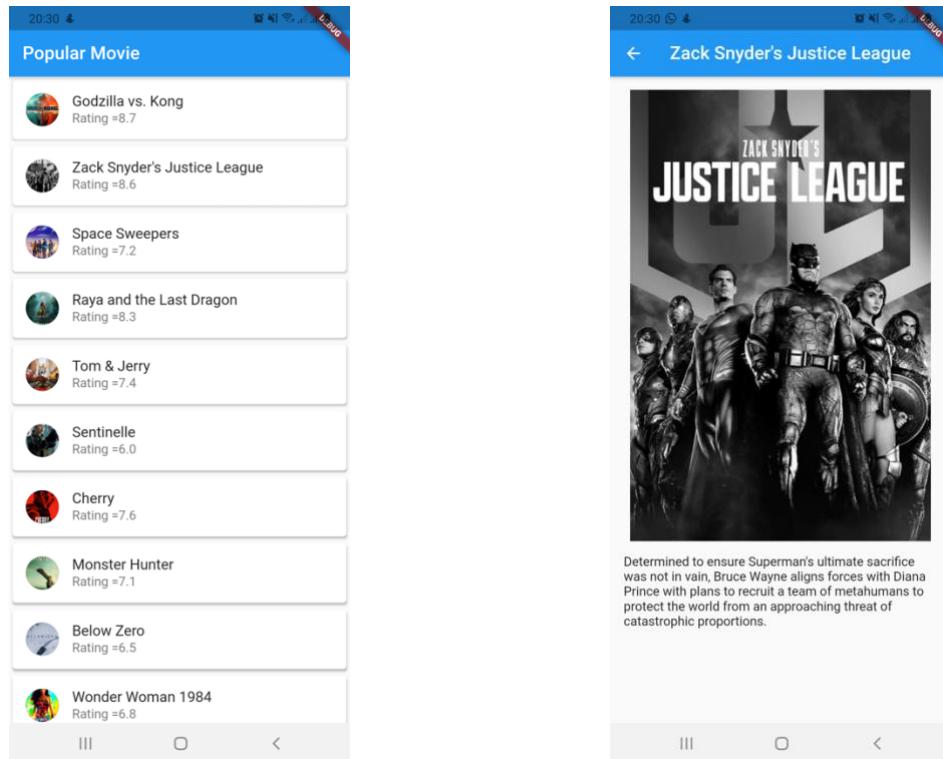
### 6.3.7 Tugas

1. Lengkapi TODO 3 dan 4.
2. Tambahkan variabel stok dan kode barang untuk operasi CRUD

## BAB 7. HTTP REQUEST

### 7.1 Desain Aplikasi

Pada project kali ini anda akan membuat sebuah aplikasi yang mengakses REST API dari themoviedb.org. Aplikasi yang dibuat akan menampilkan filem populer dan menampilkan detail dari filem populer yang di pilih oleh user. Hasil akhir dari aplikasi yang dibuat dapat dilihat pada gambar berikut ini.



Desain yang dibuat mengikuti pola master detail pada praktikum 5 tetapi sekarang berisi data dari REST API.

### 7.2 Persiapan Project

Untuk membuat project ini anda membutuhkan :

1. Api Key dari themoviedb.org untuk mendapatkan api key dapat mengikuti tutorial berikut ini :
  - a. <https://www.dicoding.com/blog/registrasi-testing-themoviedb-api/>
  - b. <https://stackoverflow.com/questions/31047815/api-key-for-themoviedb-org>
2. Library Flutter http
  - a. Setelah project flutter berhasil dibuat tambahkan library http ke project tersebut. Untuk menambahkan library bisa menggunakan plugin vscode pubspec assit atau mengikuti tutorial disini

- b. <https://stackoverflow.com/questions/59915827/how-to-add-a-library-to-flutter-dependencies>

3. Jika ada permintaan update flutter lakukan flutter upgrade.

## 7.3 Koneksi ke REST API

### 7.3.1 Menguji koneksi di dashboard api themoviedb

Untuk menguji koneksi ke REST API themoviedb dapat dilakukan di link berikut ini <https://developers.themoviedb.org/3/movies/get-popular-movies> pada link berikut anda dapat menguji dan membuat request ke server themoviedb dan melihat response nya dalam berbagai format json.

The screenshot shows the Themoviedb API documentation for the 'Get Popular' endpoint. On the left, there's a sidebar with a navigation menu for 'MOVIES' containing several GET requests. The main content area has a title 'Get Popular' and a method 'GET /movie/popular'. Below the title, it says 'Get a list of the current popular movies on TMDb. This list updates daily.' There are two buttons: 'Definition' and 'Try it out'. Under 'Variables', there's a table with one row for 'api\_key'. In the 'Query String' section, there are four rows: 'api\_key' (required), 'language' (optional, en-US), 'page' (optional, 1), and 'region' (optional). At the bottom, there's a 'SEND REQUEST' button and a URL: 'https://api.themoviedb.org/3/movie/popular?api\_key=<>&language=en-US&page=1'.

### 7.3.2 Mengkoneksikan Aplikasi Flutter ke themoviedb dengan package http

Untuk mulai mengkoneksikan aplikasi flutter yang dibuat ke rest api themoviedb.org lakukanlah langkah berikut:

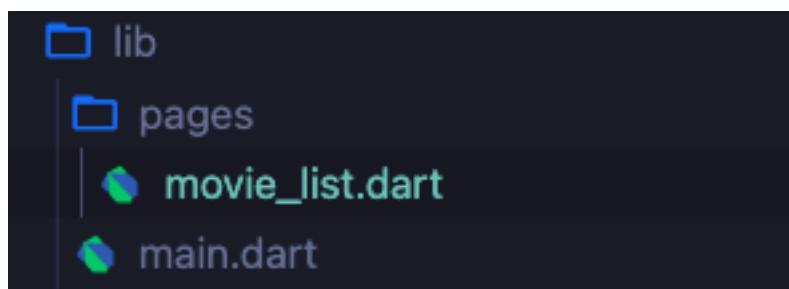
1. Untuk mengakses internet kita harus menambahkan permission internet pada android manifest cari lah file android manifest.xml pada folder android/app/src/main/AndroidManifest.xml kemudian tambahkan permission internet.

```
1 <manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.example.moviedb2">
2   <uses-permission android:name="android.permission.INTERNET" />
3   <application android:label="moviedb2" android:icon="@mipmap/ic_launcher">
4     <activity android:name=".MainActivity" android:launchMode="singleTop" android:theme="@style/LaunchTheme" android:
5       <!-- Specifies an Android theme to apply to this Activity as soon as
          the Android process has started. This theme is visible to the user
          while the Flutter UI initializes. After that, this theme continues
          to determine the Window background behind the Flutter UI. -->
6       <meta-data android:name="io.flutter.embedding.android.NormalTheme" android:resource="@style/NormalTheme" />
7       <!-- Displays an Android View that continues showing the launch screen
          Drawable until Flutter paints its first frame, then this splash
11      | Drawables until Flutter paints its first frame, then this splash
```

2. Hapus komentar yang ada pada file main.dart dari kode program awal.
3. Ubah MyHomePage menjadi stateless widget.

```
1 import 'package:flutter/material.dart';
2
3 Run | Debug
4 void main() {
5   runApp(MyApp());
6 }
7
8 class MyApp extends StatelessWidget {
9   // This widget is the root of your application.
10  @override
11  Widget build(BuildContext context) {
12    return MaterialApp(
13      title: 'Flutter Demo',
14      theme: ThemeData(
15        primarySwatch: Colors.blue,
16      ), // ThemeData
17      home: MyHomePage(),
18    ); // MaterialApp
19  }
20 }
21
22 class MyHomePage extends StatelessWidget {
23   @override
24   Widget build(BuildContext context) {
25     return Container();
26   }
27 }
```

4. Buat file baru baru “pages” pada folder lib dan buat sebuah file dengan nama movie\_list.dart.



5. Buat sebuah statefull widget pada file movie\_list.dart

```
1 import 'package:flutter/material.dart';
2
3 class MovieList extends StatefulWidget {
4   @override
5   _MovieListState createState() => _MovieListState();
6 }
7
8 class _MovieListState extends State<MovieList> {
9   @override
10  Widget build(BuildContext context) {
11    return Container();
12  }
13 }
14
```

6. Import File movie\_list.dart ke main.dart dant gunakan movie\_list.dart sebagai return dari class MyHomePage

```
22 class MyHomePage extends StatelessWidget {
23   @override
24   Widget build(BuildContext context) {
25     return MovieList();
26   }
27 }
```

7. Sebelum melanjutkan ke pembuatan aplikasi kita buat terlebih dahulu sebuah helper class untuk koneksi ke rest api themoviedb. Buatlah sebuah folder dengan nama service dan isi dengan file http\_service.dart.

```
4 import 'package:http/http.dart' as http;
5
6 class HttpService {
7   final String apiKey = 'isi dengan api key anda';
8   final String baseUrl = 'https://api.themoviedb.org/3/movie/popular?api_key=';
9 }
```

8. Selanjutnya buat sebuah function untuk mengambil response dari server themoviedb.org

```

1 ~ import 'dart:io';
2   import 'package:http/http.dart' as http;
3
4 ~ class HttpService {
5   final String apiKey = 'isi dengan api key anda';
6   final String baseUrl = 'https://api.themoviedb.org/3/movie/popular?api_key=';
7
8 ~ Future<String> getPopularMovies() async {
9   final String uri = baseUrl + apiKey;
10
11   http.Response result = await http.get(Uri.parse(uri));
12   if (result.statusCode == HttpStatus.ok) {
13     print("Sukses");
14     String response = result.body;
15     return response;
16   } else {
17     print("Fail");
18     return null;
19   }
20 }
21 }
```

9. Update file movie\_list.dart agar dapat menggunakan file httpService dan mereturn kan widget scaffold.
10. Pada function \_MovieListState tambahkan variabel berikut ini untuk variabel service jangan lupa import dulu file httpservice nya.

```

9  class _MovieListState extends State<MovieList> {
10  String result = "";
11  HttpService service;
12 }
```

11. Kemudian tambahkan method override init state agar permintaan ke rest api dapat dilakukan ketika state di inisialisasi.

```

9  class _MovieListState extends State<MovieList> {
10  String result = "";
11  HttpService service;
12
13  @override
14  void initState() {
15    service = HttpService();
16    super.initState();
17  }
18 }
```

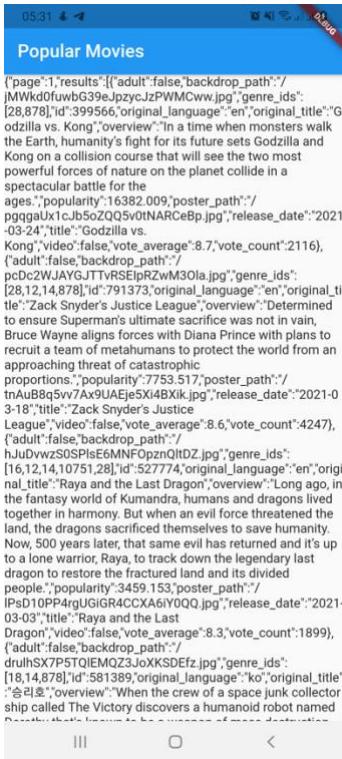
12. Selanjutnya update function build pada movieListState dengan menggunakan widget scaffold.

```

19  |   @override
20  |   Widget build(BuildContext context) {
21  |       service.getPopularMovies().then((value) => {
22  |           setState(() {
23  |               result = value;
24  |           })
25  |       });
26  |       return Scaffold(
27  |           appBar: AppBar(
28  |               title: Text("Popular Movies"),
29  |           ), // AppBar
30  |           body: Container(
31  |               child: Text(result),
32  |           ), // Container
33  |       ); // Scaffold
34  |   }
35  |

```

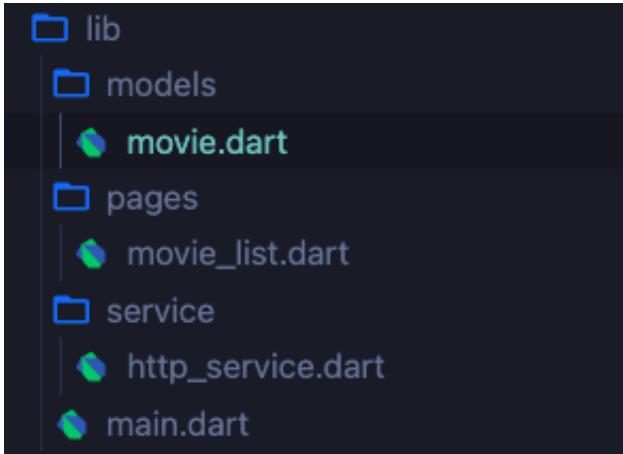
13. Jalankan aplikasi, jika di awal terjadi error reload menggunakan hot restart.



14. Pada langkah ini kita sudah berhasil mendapatkan response dari REST Api ke widget text namun belum tampil dengan baik karena belum menggunakan listview dan models. Selanjutnya kita persiapkan sebuah file models dan listview.

### 7.3.3 Membuat model untuk response http

1. Buat folder models didalam folder lib dan isikan dengan file movie.dart



2. Di dalam movie.dart buatlah sebuah class movie lengkap dengan variabel dan konstruktor seperti dibawah ini.

```
lib > models > movie.dart > ...
1 class Movie {
2     int id;
3     String title;
4     double voteAverage;
5     String overview;
6     String posterPath;
7
8     Movie(this.id, this.title, this.voteAverage, this.overview, this.posterPath);
9 }
```

3. Selanjutnya buatlah sebuah function untuk mengkonversi json menjadi response yang sesuai dengan class movie.

```
lib > models > movie.dart > ...
1 ~ class Movie {
2     int id;
3     String title;
4     double voteAverage;
5     String overview;
6     String posterPath;
7
8     Movie(this.id, this.title, this.voteAverage, this.overview, this.posterPath);
9
10 ~ Movie.fromJson(Map<String, dynamic> parsedJson) {
11     this.id = parsedJson['id'];
12     this.title = parsedJson['title'];
13     this.voteAverage = parsedJson['vote_average'] * 1.0;
14     this.overview = parsedJson['overview'];
15     this.posterPath = parsedJson['poster_path'];
16 }
17 }
```

4. Selanjutnya update function http\_service.dart menjadi seperti dibawah ini.

```

1 √ import 'dart:convert';
2 import 'dart:io';
3 import 'package:http/http.dart' as http;
4 import 'package:moviedb2/models/movie.dart';
5
6 √ class HttpService {
7   final String apiKey = 'api key';
8   final String baseUrl = 'https://api.themoviedb.org/3/movie/popular?api_key=';
9
10 √ Future<List> getPopularMovies() async {
11   final String uri = baseUrl + apiKey;
12
13   http.Response result = await http.get(Uri.parse(uri));
14   if (result.statusCode == HttpStatus.ok) {
15     print("Sukses");
16     final jsonResponse = json.decode(result.body);
17     final moviesMap = jsonResponse['results'];
18     List movies = moviesMap.map((i) => Movie.fromJson(i)).toList();
19     return movies;
20   } else {
21     print("Fail");
22     return null;
23   }
24 }
25 }
```

- Sampai pada tahap ini aplikasi anda tidak dapat berjalan dengan baik, biarkan dan lanjutkan ke langkah selanjutnya untuk membuat list populer movie.

#### 7.4 Membuat halaman list Populer Movie

Untuk membuat list view pada widget movie list anda memerlukan beberapa widget baru antara lain list view dan card. Selain itu juga dibutuhkan data dari http service. Langkah langkah yang perlu dilakukan adalah sebagai berikut :

- Update inisialisasi variabel pada class movieliststate

```

9  class _MovieListState extends State<MovieList> {
10  int moviesCount;
11  List movies;
12  HttpService service;
```

- Tambahkan method initialize() pada class movieliststate

```
9  class _MovieListState extends State<MovieList> {
10 |   int moviesCount;
11 |   List movies;
12 |   HttpService service;
13 |
14 |   Future initialize() async {
15 |     movies = [];
16 |     movies = await service.getPopularMovies();
17 |     setState(() {
18 |       moviesCount = movies.length;
19 |       movies = movies;
20 |     });
21 |   }
22 }
```

3. Tambahkan function initialize pada initState

```
9  class _MovieListState extends State<MovieList> {
10 |   int moviesCount;
11 |   List movies;
12 |   HttpService service;
13 |
14 |   Future initialize() async {
15 |     movies = [];
16 |     movies = await service.getPopularMovies();
17 |     setState(() {
18 |       moviesCount = movies.length;
19 |       movies = movies;
20 |     });
21 |   }
22 |
23 |   @override
24 |   void initState() {
25 |     service = HttpService();
26 |     initialize();
27 |     super.initState();
28 |   }
29 }
```

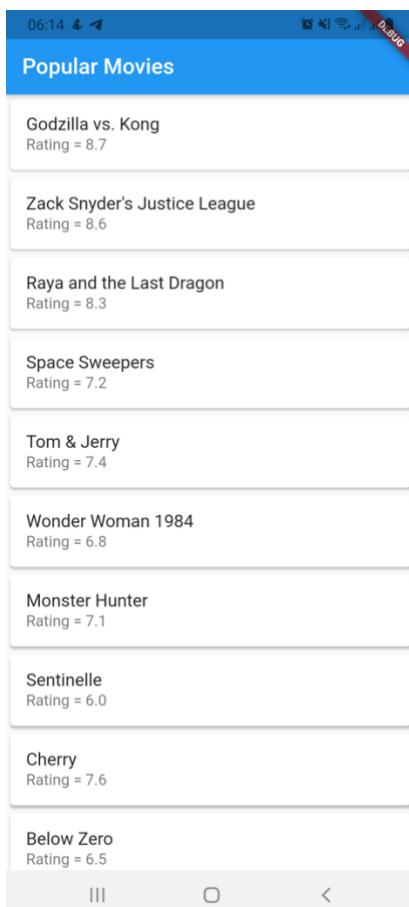
4. Selanjutnya buat listview dan card berdasarkan data dari initialize

```

30  | @override
31  | Widget build(BuildContext context) {
32  |   return Scaffold(
33  |     appBar: AppBar(
34  |       title: Text("Popular Movies"),
35  |     ), // AppBar
36  |     body: ListView.builder(
37  |       itemCount: (this.moviesCount == null) ? 0 : this.moviesCount,
38  |       itemBuilder: (context, int position) {
39  |         return Card(
40  |           color: Colors.white,
41  |           elevation: 2.0,
42  |           child: ListTile(
43  |             title: Text(movies[position].title),
44  |             subtitle: Text(
45  |               'Rating = ' + movies[position].voteAverage.toString(),
46  |             ), // Text
47  |             ), // ListTile
48  |           ); // Card
49  |         },
50  |       ); // ListView.builder // Scaffold
51  |

```

5. Lakukan hot restart dan aplikasi akan berubah menjadi seperti dibawah ini.



6. Selamat anda berhasil membuat listview sekarang saatnya mempercantik listview yang dibuat.

- Challenge : carilah cara menambahkan gambar dari response api ke listview, tambahkan gambar tersebut ke listview.

## 7.5 Membuat halaman detail Populer Movie

- Untuk membuat perpindahan dari movie list ke movie detail buatlah onTap event di listview pada movie list.

```
39 |         return Card(
40 |           color: Colors.white,
41 |           elevation: 2.0,
42 |           child: ListTile(
43 |             title: Text(movies[position].title),
44 |             subtitle: Text(
45 |               'Rating = ' + movies[position].voteAverage.toString(),
46 |             ), // Text
47 |             onTap: () {
48 |               MaterialPageRoute route = MaterialPageRoute(
49 |                 builder: (_) => MovieDetail(movies[position])); // MaterialPageRoute
50 |               Navigator.push(context, route);
51 |             },
52 |           ), // ListTile
53 |         ); // Card
```

- Pada event on tap Widget MovieDetail belum dibuat, buatlah widget ini pada folder pages/movie\_detail.dart.

```
lib > pages > movie_detail.dart > ...
1 import 'package:flutter/material.dart';
2
3 class MovieDetail extends StatelessWidget {
4   @override
5   Widget build(BuildContext context) {
6     return Container();
7   }
8 }
```

- Lengkapi Movie Detail untuk menerima parameter Movies

```
lib > pages > movie_detail.dart > ...
1 import 'package:flutter/material.dart';
2 import 'package:moviedb2/models/movie.dart';
3
4 class MovieDetail extends StatelessWidget {
5   final Movie movie;
6   final String imgPath = 'https://image.tmdb.org/t/p/w500/';
7
8   MovieDetail(this.movie);
9
10  @override
11  Widget build(BuildContext context) {
12    return Container();
13  }
14}
15
```

#### 4. Lengkapi detail widget.

```
10  @override
11  Widget build(BuildContext context) {
12    String path;
13    if (movie.posterPath != null) {
14      path = imgPath + movie.posterPath;
15    } else {
16      path =
17        'https://images.freeimages.com/images/large-previews/5eb/movie-clapboard-1184339.jpg';
18    }
19    double height = MediaQuery.of(context).size.height;
20    return Scaffold(
21      appBar: AppBar(
22        title: Text(movie.title),
23      ), // AppBar
24      body: SingleChildScrollView(
25        child: Center(
26          child: Column(
27            children: [
28              Container(
29                padding: EdgeInsets.all(16),
30                height: height / 1.5,
31                child: Image.network(path)), // Container
32              Container(
33                child: Text(movie.overview),
34                padding: EdgeInsets.only(left: 16, right: 16),
35              ), // Container
36            ],
37          ), // Column
38        ), // Center
39      ), // SingleChildScrollView
40    ); // Scaffold
41 }
```

#### 5. Challenge Modifikasilah Tampilan agar terlihat lebih menarik

## **BAB 8. FIREBASE**

### **8.1 Teori**

Firebase adalah suatu layanan dari Google untuk memberikan kemudahan bahkan mempermudah para developer aplikasi dalam mengembangkan aplikasinya. Firebase sebagai BaaS (Backend as a Service) merupakan solusi yang ditawarkan oleh Google untuk mempercepat pekerjaan developer. Dengan menggunakan Firebase, apps developer bisa fokus dalam mengembangkan aplikasi tanpa memberikan effort yang besar untuk urusan backend.

Fitur yang ditawarkan firebase yaitu

1. Firebase Analytics
2. Firebase Cloud Messaging and Notifications
3. Firebase Authentication
4. Firebase Cloud Firestore
5. Firebase Realtime Database
6. Firebase Hosting
7. Dan lain sebagainya

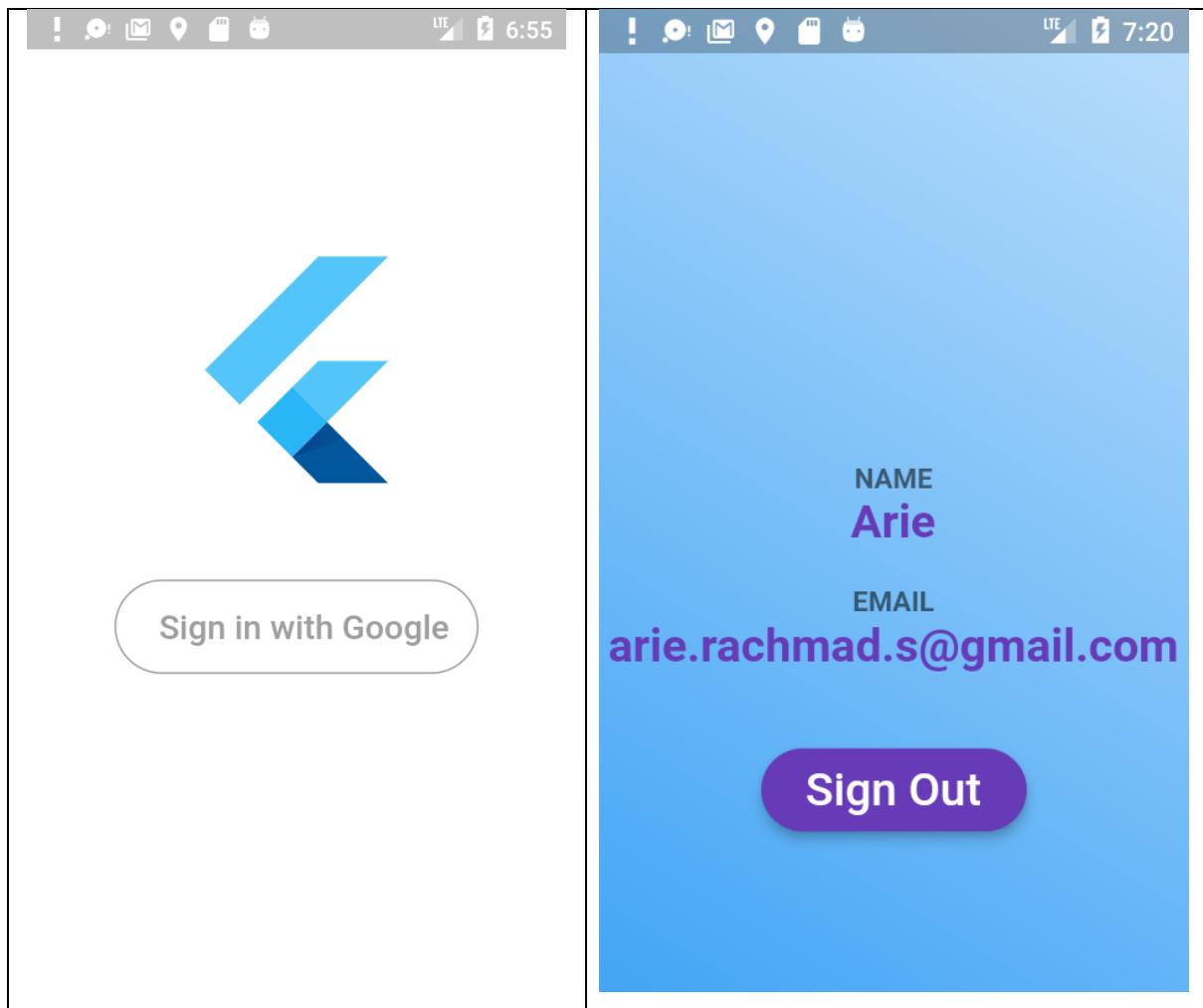
Untuk lebih detail tentang fitur yang dimiliki firebase dapat dilihat pada tautan berikut:

1. <https://firebase.google.com/products-build?hl=en>
2. <https://firebase.google.com/products-release?hl=en>
3. <https://firebase.google.com/products-engage?hl=en>

### **8.2 Desain Aplikasi**

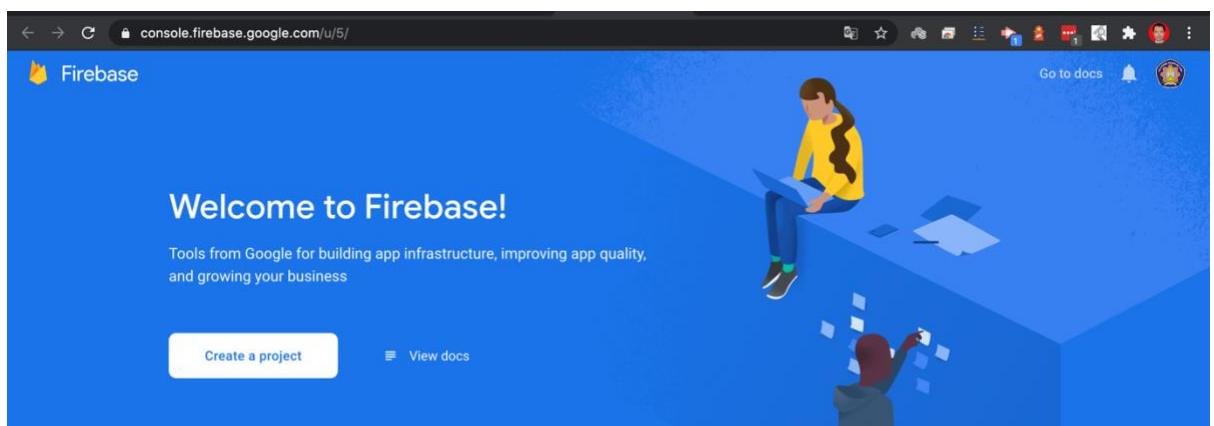
Aplikasi flutter yang akan dibuat pada modul ini adalah memanfaatkan salah satu fitur firebase yaitu firebase authentication yang dapat digunakan untuk melakukan login.

Tampilan aplikasi akhir yang diharapkan adalah:

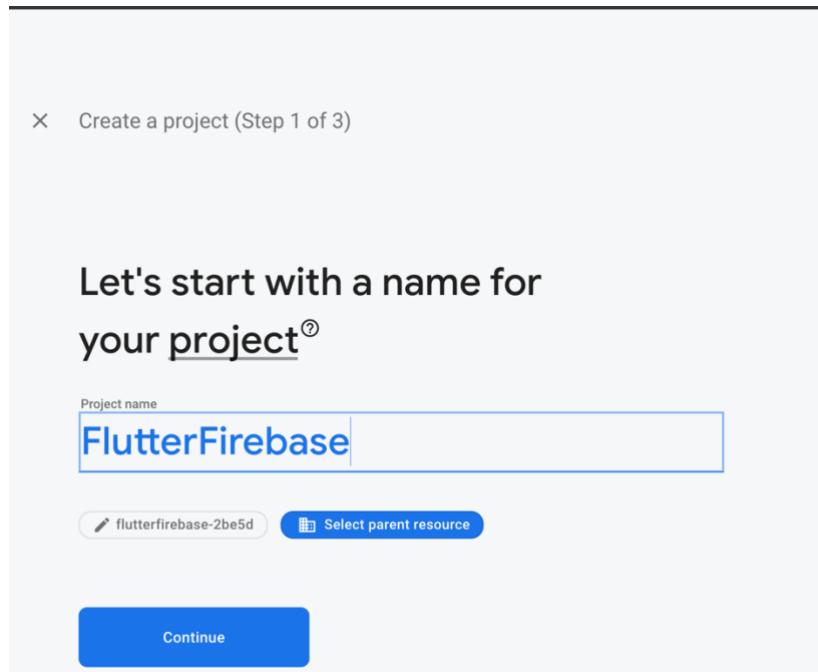


### 8.3 Persiapan Project

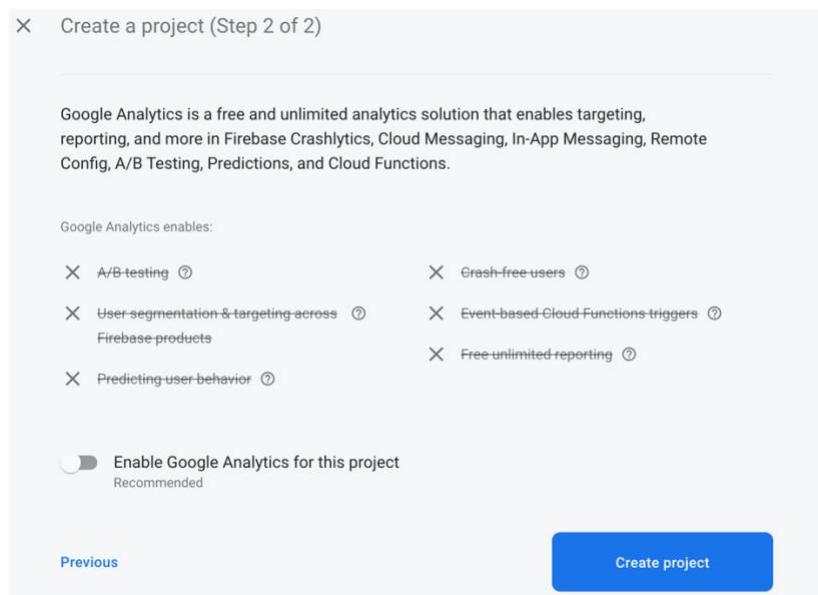
1. Membuat project di firebase console <https://console.firebaseio.google.com/>



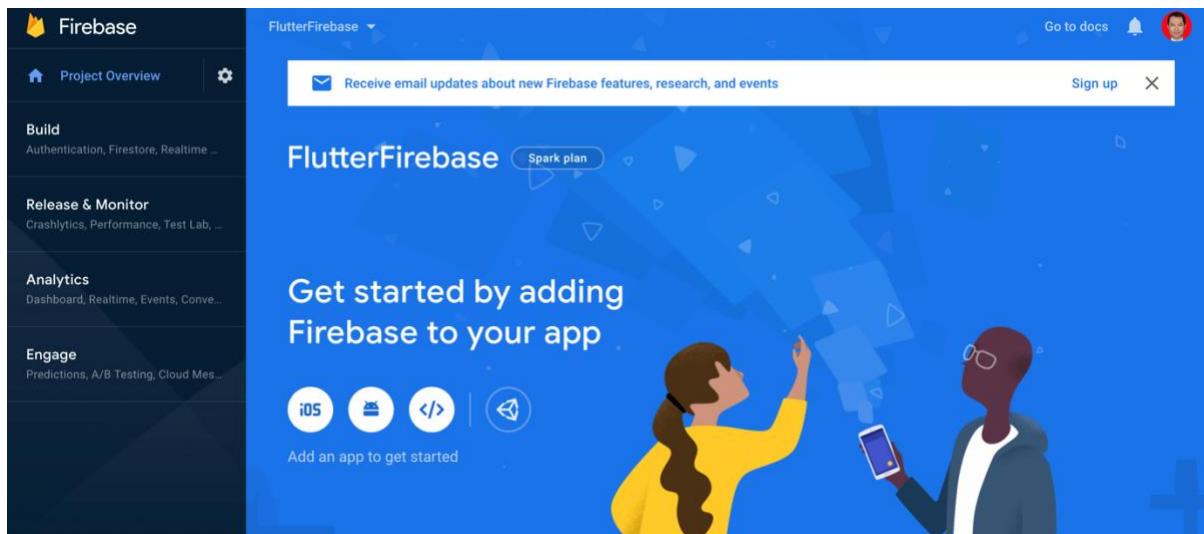
2. Tekan create project dan diisi nama project sesuai dengan project yang akan dibuat, misalnya seperti ini.



3. Tekan continue maka selanjutnya akan ada tampilan seperti berikut:



4. Pada Firebase console, expand Build menu pada panel sebelah kiri.



- Klik Authentication, selanjutnya klik button Get Started, kemudian pilih tab Sign-in method dan terakhir Save pengaturan yang sudah dilakukan, seperti pada ilustrasi di bawah ini

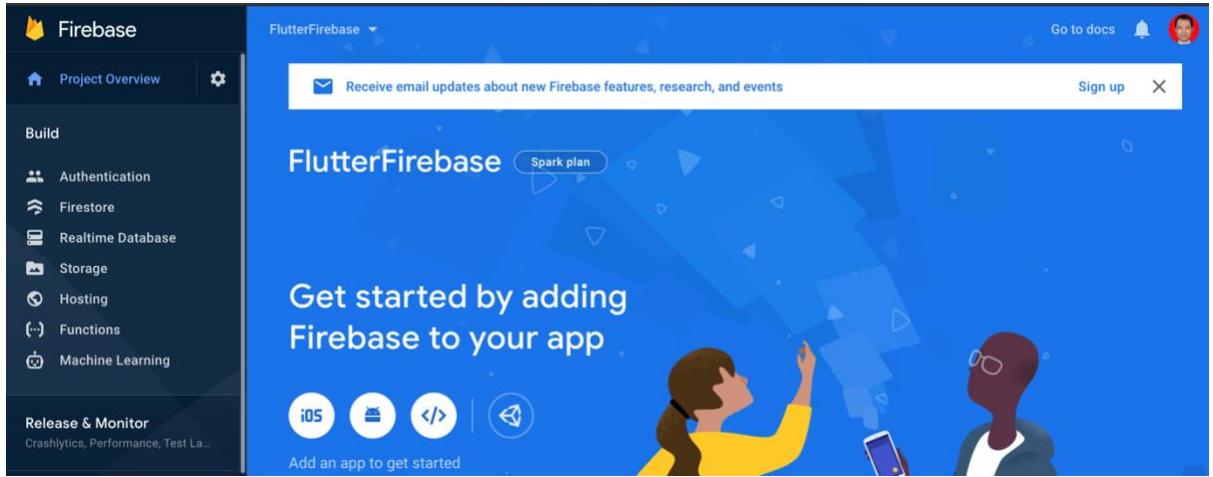
The screenshot shows the Authentication settings page in the Firebase console. The sidebar on the left includes options for Build, Authentication (which is selected), Firestore, Realtime Database, Storage, Hosting, Functions, and Machine Learning. The main content area is titled "Authentication" and has tabs for Users, Sign-in method, Templates, and Usage. The "Sign-in method" tab is active, showing "Sign-in providers". Under "Provider", there is an "Email/Password" entry with an "Enable" toggle switch turned on. Below it, there is another entry for "Email link (passwordless sign-in)" with its own "Enable" toggle switch. At the bottom right of the modal are "Cancel" and "Save" buttons.

- Setup menjadi enable pengaturan authentikasi menggunakan login yang terintegrasi dengan aplikasi lain misalnya google. Tampilannya adalah sebagai berikut:

Provider	Status
✉ Email/Password	Enabled
📞 Phone	Disabled
.Google	Enabled

## 8.4 Konfigurasi Firebase

1. Klik Project Overview maka terdapat tampilan seperti dibawah ini. Yang dapat digunakan untuk menambahkan firebase pada aplikasi yang akan dibuat.



2. Klik logo android karena kita akan fokus untuk membuat aplikasi android. Setelah itu akan muncul tampilan seperti berikut ini.

Tambahkan android packages name yang ada di AndroidManifest.xml pada tampilan di atas. Packages name juga dapat diubah dengan mengkuti langkah pada tautan berikut.

- <https://qastack.id/programming/51534616/how-to-change-package-name-in-flutter>

note : Untuk nicknames optional diisi atau tidak.

- Untuk Debug Signing certificate dapat menggunakan perintah berikut ini diterminal visual code (<https://developers.google.com/android/guides/client-auth>).

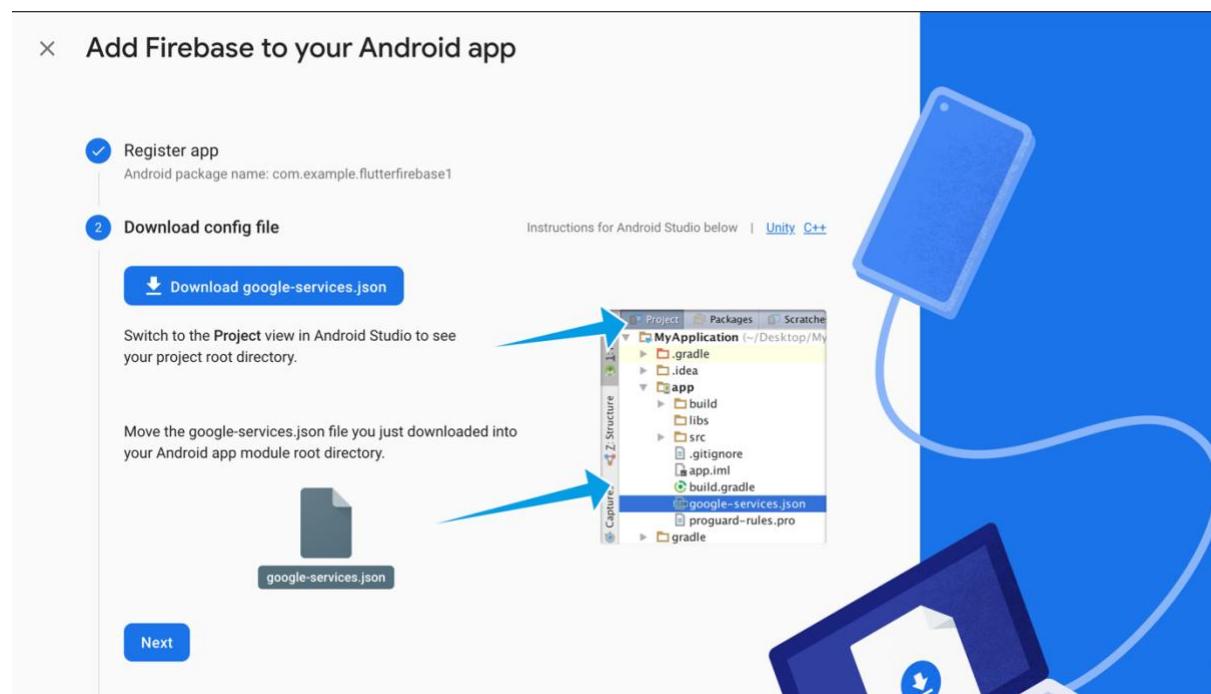
#### Windows

```
keytool -list -v \
-alias androiddebugkey -keystore
%USERPROFILE%\.android\debug.keystore
```

#### Unix/Linux

```
keytool -list -v \
-alias androiddebugkey -keystore ~/.android/debug.keystore
```

- Menambah google-services.json pada folder android seperti berikut



- Edit android/build.gradle untuk menambah google-services plugin dependency seperti berikut.

```
dependencies {
    classpath 'com.android.tools.build:gradle:4.1.0'
```

```
classpath "org.jetbrains.kotlin:kotlin-gradle-
plugin:$kotlin_version"
classpath 'com.google.gms:google-services:4.3.5' // new
}
```

6. Edit android/app/build.gradle untuk mengenable google-services plugin.

```
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
apply from:
"$flutterRoot/packages/flutter_tools/gradle/flutter.gradle"
apply plugin: 'com.google.gms.google-services' // new
```

7. Firebase memerlukan Multidex untuk di enabled, dan satu-satunya cara untuk melakukan hal itu adalah dengan mengupdate minimum supported SDK yaitu 21 atau versi yang lebih baru.

```
defaultConfig {
    applicationId "com.example.gtk_flutter"
    minSdkVersion 21 // Updated
    targetSdkVersion 30
    versionCode flutterVersionCode.toInt()
    versionName flutterVersionName
}
```

8. Terakhir mengubah pubspec.yaml dengan menambah packages firebase yang diperlukan pada project flutter, seperti berikut.

```
dependencies:
  flutter:
    sdk: flutter
  firebase_auth: ^1.0.0 # new
  google_sign_in: ^4.5.3 # new
```

## 8.5 Membuat login\_page .dart

Buatlah halaman login page yang digunakan untuk login menggunakan akun google.

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

class LoginPage extends StatefulWidget {
    @override
    _LoginPageState createState() => _LoginPageState();
}

class _LoginPageState extends State<LoginPage> {
    @override
    Widget build(BuildContext context) {
        return Scaffold(
            body: Container(
                color: Colors.white,
                child: Center(
                    child: Column(
                        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
                        children: <Widget>[
                            FlutterLogo(size: 150),
                            SizedBox(height: 50),
                            _signInButton(),
                        ],
                    ),
                ),
            ),
        );
    }

    Widget _signInButton() {
        return OutlineButton(
            splashColor: Colors.grey,
            onPressed: () {
                signInWithGoogle().then((result) {
                    if (result != null) {
                        Navigator.of(context).push(
                            MaterialPageRoute(
                                builder: (context) {

```

```

        return FirstScreen();
    },
),
);
}
);
},
shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(40)),
highlightElevation: 0,
borderSide: BorderSide(color: Colors.grey),
child: Padding(
padding: const EdgeInsets.fromLTRB(0, 10, 0, 10),
child: Row(
mainAxisSize: MainAxisSize.min,
mainAxisAlignment: MainAxisAlignment.center,
children: <Widget>[
Image(image: AssetImage("assets/google_logo.png"), height: 35.0),
Padding(
padding: const EdgeInsets.only(left: 10),
child: Text(
'Sign in with Google',
style: TextStyle(
fontSize: 20,
color: Colors.grey,
),
),
),
),
],
),
),
);
}
}
}

```

## 8.6 Membuat sign\_in .dart

Pada file sign\_in inilah yang memegang peranan penting pada modul firebase authentikasi.

```

import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:google_sign_in/google_sign_in.dart';

final FirebaseAuth _auth = FirebaseAuth.instance;

```

```
final GoogleSignIn googleSignIn = GoogleSignIn();

String name;
String email;
String imageUrl;

Future<String> signInWithGoogle() async {
  await Firebase.initializeApp();

  final GoogleSignInAccount googleSignInAccount = await googleSignIn.signIn();
  final GoogleSignInAuthentication googleSignInAuthentication =
    await googleSignInAccount.authentication;

  final AuthCredential credential = GoogleAuthProvider.credential(
    accessToken: googleSignInAuthentication.accessToken,
    idToken: googleSignInAuthentication.idToken,
  );

  final UserCredential authResult =
    await _auth.signInWithCredential(credential);
  final User user = authResult.user;

  if (user != null) {
    // Checking if email and name is null
    assert(user.email != null);
    assert(user.displayName != null);
    assert(user.photoURL != null);

    name = user.displayName;
    email = user.email;
    imageUrl = user.photoURL;

    // Only taking the first part of the name, i.e., First Name
    if (name.contains(" ")) {
      name = name.substring(0, name.indexOf(" "));
    }

    assert(!user.isAnonymous);
    assert(await user.getIdToken() != null);

    final User currentUser = _auth.currentUser;
    assert(user.uid == currentUser.uid);

    print('signInWithGoogle succeeded: $user');
  }
}
```

```

        return '$user';
    }

    return null;
}

Future<void> signOutGoogle() async {
    await googleSignIn.signOut();

    print("User Signed Out");
}

```

## 8.7 Membuat first\_screen.dart

```

// Copyright (c) 2019 Souvik Biswas

import 'package:flutter/material.dart';
import 'package:flutterfirebase/login_page.dart';
import 'package:flutterfirebase/sign_in.dart';

class FirstScreen extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return Scaffold(
            body: Container(
                decoration: BoxDecoration(
                    gradient: LinearGradient(
                        begin: Alignment.topRight,
                        end: Alignment.bottomLeft,
                        colors: [Colors.blue[100], Colors.blue[400]],
                    ),
                ),
                child: Center(
                    child: Column(
                        mainAxisAlignment: MainAxisAlignment.center,
                        mainAxisSize: MainAxisSize.max,
                        children: <Widget>[
                            CircleAvatar(
                                backgroundImage: NetworkImage(
                                    imageUrl,
                                ),
                                radius: 60,
                                backgroundColor: Colors.transparent,
                            )
                        ],
                    ),
                )
            )
        );
    }
}

```

```
        ) ,
        SizedBox(height: 40),
        Text(
            'NAME',
            style: TextStyle(
                fontSize: 15,
                fontWeight: FontWeight.bold,
                color: Colors.black54),
        ),
        Text(
            name,
            style: TextStyle(
                fontSize: 25,
                color: Colors.deepPurple,
                fontWeight: FontWeight.bold),
        ),
        SizedBox(height: 20),
        Text(
            'EMAIL',
            style: TextStyle(
                fontSize: 15,
                fontWeight: FontWeight.bold,
                color: Colors.black54),
        ),
        Text(
            email,
            style: TextStyle(
                fontSize: 25,
                color: Colors.deepPurple,
                fontWeight: FontWeight.bold),
        ),
        SizedBox(height: 40),
        RaisedButton(
            onPressed: () {
                signOutGoogle();

Navigator.of(context).pushAndRemoveUntil(MaterialPageRoute(builder: (context)
{return LoginPage();}), ModalRoute.withName('/'));

},
color: Colors.deepPurple,
child: Padding(
padding: const EdgeInsets.all(8.0),
child: Text(
'Sign Out',
style: TextStyle(fontSize: 25, color: Colors.white),
```

```
        ) ,
    ) ,
    elevation: 5,
    shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(40)),
    )
],
),
),
),
),
);
}
}
```

## 8.8 Mengedit main.dart

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

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

class MyApp extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return MaterialApp(
            debugShowCheckedModeBanner: false,
            title: 'Flutter Login',
            theme: ThemeData(
                primarySwatch: Colors.blue,
            ),
            home: LoginPage(),
        );
    }
}
```

## 8.9 Tugas

Edit aplikasi sehingga dapat login menggunakan email dan password. Sehingga terdapat 2 cara untuk login menggunakan email dan akun google.