#06 | Widget Dasar Flutter

| **Summary** | - Mahasiswa mampu menjelaskan bagian-bagian dari project file flutter  - Mahasiswa mampu menggunakan fitur hot reload dan restart  - Mahasiswa mampu menjelaskan perbedaan stateful dan stateless widget  - Mahasiswa mampu menerapkan widget-widget dasar flutter dan tema |
| --- | --- |
| **URL** | 06-widget-dasar-flutter |
| **Category** | Mobile Flutter |
| **Environment** | web |
| **Status** | Published |
| **Feedback Link** | https://github.com/jti-polinema/flutter-codelab/issues |
| **Author** | Habibie Ed Dien |
| **Author LDAP** | hbb-polinema |
| **Analytics Account** | G-H7507PFEJB |

[Flutter Fundamental - Bagian 1](#_ok7k5uux6)

[Tujuan Praktikum](#_21yzqg98x7h6)

[Sumber Daya yang Dibutuhkan](#_dusjnkkj26ob)

[Pengetahuan yang Anda harus Miliki](#_319h1e1sjzdz)

[Struktur Project Flutter](#_t0tqoph2ohlh)

[Flutter Hot Reload dan Restart](#_g7nutupwvu6v)

[Flutter Hot Reload](#_89gt0c2q7m8e)

[Flutter Hot Restart](#_edqdpm43bl6g)

[Bedah Hello World Project](#_sl1nr883oe3v)

[Import Statement](#_ndqn33mzkwzq)

[Main function](#_y4w796pa4qp3)

[Stateless Widget](#_mpg6x3enher)

[Stateful Widget](#_s4fuwn64dog5)

[Stateful dan Stateless Widget](#_el89ede9berg)

[Stateful widgets](#_od6s7wl3f9jr)

[Apa itu business logic?](#_bmcleik48cu0)

[Inherited widgets](#_ylmcjrdx0x6b)

[The widget key property](#_916qwqjprxvz)

[Praktikum 1: Membuat Project Flutter Baru](#_vmpz7ap0erhd)

[Langkah 1:](#_spr93e7x3yo2)

[Langkah 2:](#_g8csgggwssw)

[Langkah 3:](#_2q85gi9dhaiq)

[Langkah 4:](#_b5grkrhzelll)

[Praktikum 2: Membuat Repository GitHub dan Laporan Praktikum](#_183ymolzmz72)

[Langkah 1:](#_fo4sec87xe1y)

[Langkah 2:](#_w1t2iv38s7c4)

[Langkah 3:](#_si1yuc34p8n5)

[Langkah 4:](#_24atpxdvii9o)

[Langkah 5:](#_ux1nm0s2j2m6)

[Langkah 6:](#_y8c7p06bc925)

[Langkah 7:](#_4uwxy9ceorhu)

[Langkah 8:](#_4oqjens3e5h)

[Langkah 9:](#_5qxx0m7o9mwc)

[Langkah 10:](#_mmfuqpf9j8hr)

[Langkah 11:](#_prfanbwcjj2)

[Langkah 12:](#_u89do7qerjky)

[Praktikum 3: Menerapkan Widget Dasar](#_86j27ymyw6sc)

[Langkah 1: Text Widget](#_z4xb79s85ro)

[Langkah 2: Image Widget](#_x89yqkiqmitd)

[Praktikum 4: Menerapkan Widget Material Design dan iOS Cupertino](#_gz7tbatl2uh8)

[Langkah 1: Cupertino Button dan Loading Bar](#_h9m2druzjfj8)

[Langkah 2: Floating Action Button (FAB)](#_2nzf86f8qvki)

[Langkah 3: Scaffold Widget](#_gxsz8i7mkdo3)

[Langkah 4: Dialog Widget](#_qp3pp1metiux)

[Langkah 5: Input dan Selection Widget](#_ybah5qq959kt)

[Langkah 6: Date and Time Pickers](#_e4m3gfagvjh)

[Tugas Praktikum](#_dz35nqtk0wjb)

[Selamat!](#_s22qmcj4zu61)

[Apa selanjutnya?](#_9mqifffit2ew)

[Referensi](#_an95hoqxrlhg)

# Widget Dasar Flutter



**Terakhir diperbarui:** 24 September 2023

**Penulis:** Habibie Ed Dien

Pada codelab ini, Anda akan mempelajari konsep dan praktik untuk dasar-dasar framework Flutter termasuk fitur penggunaan hot reload dan restart serta widget dasar dan temanya.

## Tujuan Praktikum

Setelah menyelesaikan codelab ini Anda akan mampu untuk:

* menjelaskan bagian-bagian dari project file flutter
* menggunakan fitur hot reload dan restart
* menjelaskan perbedaan stateful dan stateless widget
* menerapkan widget-widget dasar flutter dan tema

| **Catatan:** Materi ini diadaptasi dari Buku Flutter for Beginners - second edition dan Buku Ajar Pemrograman Mobile Flutter JTI Polinema serta sumber referensi lainnya. |
| --- |

## Sumber Daya yang Dibutuhkan

Berikut merupakan sumber daya yang diperlukan untuk menyelesaikan praktikum ini:

* PC atau Laptop dengan spesifikasi minimum RAM 4GB Processor Core i3
* Koneksi internet
* Chrome Browser
* Perangkat mobile (Android/iOS)

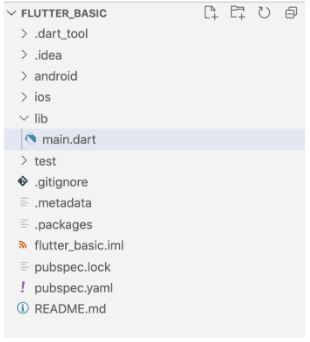
## Pengetahuan yang Anda harus Miliki

* Algoritma dan Pemrograman Dasar
* Pemrograman Berorientasi Objek
* Bahasa Pemrograman Dart

# Struktur Project Flutter

Duration: 15:00

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.

1. **pubspec.yaml** : File yang berisi library atau package yang dibutuhkan untuk

pengembangan aplikasi.

1. **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 bagian berikut.

1. **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.

1. **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.

1. **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.

1. **Folder lib**

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

1. **Folder test**

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

1. **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.

# Flutter Hot Reload dan Restart

Duration: 10:00

## 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 lalu diupdate. Setelah selesai update, dart virtual machine akan memperbarui UI sesuai dengan perubahan. Keunggulan hot reload adalah waktu prosenya yang cepat dibanding hot restart. Akan tetapi hot reload mempertahankan state yang ada sehingga jika menggunakan state maka nilai dari widget tidak akan berubah.

## Flutter Hot Restart

Hot restart akan meng-*compile* ulang aplikasi dan mereset (*destroy*) state yang ada. Jadi hot restart akan mem-*build* ulang *widget tree* sesuai dengan code yang telah diperbarui.

# Bedah Hello World Project

Duration: 20:00

Berikut ini penjelasan terkait project flutter yang nanti akan Anda praktikkan pada Praktikum 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'; |
| --- |

## 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 StatlessWidget atau StatefullWidget, ingat bahwa flutter membangun UI menggunakan widget.

## 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 widget nya jika parent nya adalah StatefullWidget. Struktur dasar stateless widget adalah sebagai berikut:

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

## Stateful Widget

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

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

# Stateful dan Stateless Widget

Duration: 10:00

Widget memiliki peranan sangat penting dalam Pengembangan aplikasi Flutter. Widget adalah bagian yang membentuk UI; kodenya merupakan representasi dari apa yang terlihat oleh pengguna.

UI hampir tidak pernah bersifat statis; mereka sering berubah, seperti yang akan Anda alami ketika Anda menggunakan halaman web atau aplikasi. Meskipun tidak dapat diubah menurut definisi, widget tidak dimaksudkan untuk menjadi final – lagi pula, kita berurusan dengan UI, dan UI pasti akan berubah selama siklus hidup aplikasi apa pun. Itu sebabnya Flutter menyediakan dua jenis widget: stateless dan stateful.

**Immutability**

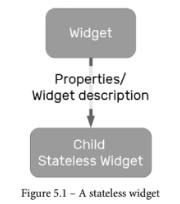
Sebagian besar bahasa pemrograman merujuk pada istilah "immutable". Object Immutable adalah objek yang tidak pernah berubah. Artinya, itu tidak dapat mengubah dirinya sendiri, dan itu tidak dapat diubah secara eksternal. Sebaliknya, jika perubahan diperlukan, maka objeknya adalah diganti begitu saja. Stateless Widget immutable karena tidak dapat mengubah properties atau state, sesuatu yang eksternal juga tidak dapat mengubah properties atau statenya. Jika widget perlu diubah, maka secara efektif diganti dengan widget baru yang memiliki properties atau state yang berbeda.

Seperti yang Anda duga, widget stateless tidak memiliki state, sedangkan widget stateful memiliki state dan beradaptasi berdasarkan state itu. Perbedaan ini berdampak pada life cycle widget, bagaimana kode dibangun, dan disusun. Kewajiban developer untuk memilih jenis widget apa yang digunakan dalam setiap situasi. Secara umum, seorang developer harus menggunakan stateless sebagai opsi default kecuali widget yang perlu menggunakan state. Stateful widget yang dapat digunakan untuk setiap skenario, tetapi ini akan memengaruhi kinerja dan pemeliharaan kode.

Selain itu, Flutter menggunakan konsep widget yang diwarisi (tipe **InheritedWidget**), yang juga merupakan sejenis widget, tetapi sedikit berbeda dari dua jenis lainnya yang telah dijelaskan.

**Stateless widgets**

Sebuah typical UI akan terdiri dari banyak widget, dan beberapa di antaranya tidak akan pernah mengubah propertinya setelah dipakai. Mereka tidak memiliki state; yaitu, mereka tidak berubah dengan sendirinya melalui tindakan atau perilaku internal. Sebaliknya, mereka diubah oleh eksternal event pada parent widget di widget tree. Jadi, aman untuk mengatakan bahwa widget stateless memberikan kontrol tentang bagaimana mereka dibangun menuju parent widget di tree. Diagram berikut menunjukkan representasi dari widget stateless:



Dalam diagram diatas, parent widget membuat instance child stateless widget dan meneruskan sekumpulan properti selama pembuatan instance. Child widget hanya dapat menerima properti ini dari parent widget dan tidak akan mengubah dengan sendirinya. Dalam hal kode, ini berarti bahwa widget stateless hanya memiliki properti akhir yang ditentukan selama construction, dan properti ini hanya dapat diubah melalui pembaruan parent widget dengan perubahan kemudian meneruskan menuju child widget.

Mari kita lihat contoh stateless widget dari Hello World! aplikasi yang kita jelajahi di bab-bab sebelumnya.

**Contoh Kode Program**

Stateless widget pertama di dalam aplikasi adalah kelas aplikasi itu sendiri:

| class MyApp extends StatelessWidget {  @override  Widget build(BuildContext context) {  return MaterialApp(  title: 'Flutter Demo',  theme: ThemeData(  primarySwatch: Colors.blue,  ),  home: MyHomePage(title: 'Flutter Demo Home Page'),  );  }  } |
| --- |

Seperti yang Anda lihat, class MyApp extends StatelessWidget dan overrides metode build(BuildContext). Metode build sangat penting untuk semua widget dan menjelaskan bagaimana widget akan muncul di layar. Ini dilakukan dengan membangun subtree widget di bawahnya. MyApp merupakan root dari widget tree; itu adalah widget top-level yang dipakai dalam metode runApp dalam metode main Dart. Oleh karena itu, ia membangun semua widget di bawah tree. Dalam contoh ini, turunan langsungnya adalah MaterialApp. Menurut dokumentasi, MaterialApp dapat didefinisikan sebagai berikut:

*"A convenience widget that wraps a number of widgets that are commonly required for material design applications."*

**Material Design**

Material Design adalah kumpulan desain dan pengalaman digital standar yang dibuat oleh Google untuk membantu tim membangun UI berkualitas tinggi. Apple memiliki padanan bernama Cupertino, dan kita akan melihat contoh keduanya pada materi ini.

Dalam contoh ini, stateless widget tidak menerima properti apa pun dari parent karena tidak memiliki parent. Kami akan melihat contoh properti yang diteruskan nanti.

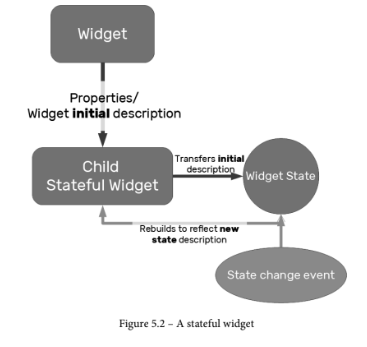
BuildContext adalah argumen yang disediakan untuk metode build sebagai cara yang berguna untuk berinteraksi dengan widget tree. Ini memungkinkan Anda untuk mengakses informasi ancestral penting yang membantu menggambarkan widget yang sedang dibangun. Misalnya, data tema yang ditentukan dalam widget ini dapat diakses oleh semua child widget untuk memastikan ada tampilan dan nuansa yang konsisten pada aplikasi Anda.

Selain properti lainnya, MaterialApp berisi properti home. Ini adalah widget pertama yang ditampilkan dalam aplikasi Anda. Di sini, home mengacu pada widget MyHomePage, yang bukan merupakan widget bawaan, melainkan widget stateful yang didefinisikan dalam Aplikasi Hello World!.

Anda sekarang seharusnya dapat memahami bagaimana Flutter menyusun widget untuk membuat tampilan. Mari kita lihat partner widget stateless, yaitu widget stateful.

## Stateful widgets

Tidak seperti widget stateless, yang menerima properti dari induknya (parent) yang konstan sepanjang masa pakai widget, widget stateful dimaksudkan untuk mengubah propertinya secara dinamis selama masa pakainya. Menurut definisi, widget stateful juga tidak dapat diubah, tetapi mereka memiliki kelas **State** pendamping yang mewakili state widget saat ini. Ini ditunjukkan dalam diagram berikut:



Dalam diagram diatas, widget membuat instance child widget dan, mirip dengan contoh widget stateless, meneruskan properti ke child. Properti ini, sekali lagi, final dan tidak dapat diubah di dalam widget. Namun, tidak seperti contoh widget stateless, objek **State** pendamping juga memiliki akses ke properti widget dan, selain itu, mampu menampung properti lain yang tidak final.

Dengan menahan state widget di objek separate **State**, framework dapat merebuild widget kapan pun diperlukan tanpa kehilangan state terkait saat ini. Elemen pada tree elemen menyimpan referensi ke widget yang sesuai dan juga objek **State** yang terkait dengannya. Objek **State** akan memberi tahu Anda ketika widget perlu dibangun kembali dan kemudian melakukan pembaruan di elemen tree juga.

**Contoh Kode Program**

MyHomePage adalah widget stateful, dan karenanya, itu didefinisikan dengan objek **State** bernama \_MyHomePageState, yang berisi properti yang memengaruhi tampilan widget. Pertama, mari kita lihat widgetnya:

| class MyHomePage extends StatefulWidget {  MyHomePage({Key key, required this.title}) : super(key: key);  final String title;  @override  \_MyHomePageState createState() => \_MyHomePageState();  } |
| --- |

Hal pertama yang harus Anda perhatikan adalah bahwa ini extends StatefulWidget, mengidentifikasi, oleh karena itu, bahwa ini akan memiliki objek pendamping State. Widget stateful harus mengganti metode createState() dan mengembalikan instance objek pendamping. Dalam contoh ini, ia mengembalikan sebuah instance dari \_MyHomePageState. State widget yang valid adalah kelas yang memperluas kelas **State** framework, yang didefinisikan dalam dokumentasi sebagai berikut:

*"The logic and internal state for a StatefulWidget."*

Selain itu, dalam contoh ini, properti telah diteruskan dari parent widget dan ini muncul di konstruktor widget. Perhatikan bahwa properti title telah diteruskan dan disimpan di widget di bidang terakhir, yang diberi nama title. Seperti yang dibahas dalam Bab 4, Kelas dan Konstruksi Dart, Dart memiliki beberapa pintasan cerdas dalam cara menyusun konstruktor, dan dengan menggunakan this.title di badan konstruktor ini, kita dapat secara otomatis menetapkan nilai properti title ke bidang title.

Biasanya, widget stateful mendefinisikan kelas State yang sesuai dalam file yang sama. Selain itu, status biasanya bersifat pribadi untuk pustaka widget, karena klien eksternal tidak perlu berinteraksi dengannya secara langsung.

Kelas \_MyHomePageState berikut mewakili objek **State** dari widget MyHomePage:

| class \_MyHomePageState extends State<MyHomePage> {  int \_counter = 0;  void \_incrementCounter() {  setState(() {  \_counter++;  });  }  @override  Widget build(BuildContext context) {  return Scaffold(  appBar: AppBar(  title: Text(widget.title),  ),  body: Center(  child: Column(  mainAxisAlignment: MainAxisAlignment.center,  children: <Widget>[  Text(  'You have pushed the button this many times:',  ),  Text(  '$\_counter',  style: Theme.of(context).textTheme.headline4,  ),  ],  ),  ),  floatingActionButton: FloatingActionButton(  onPressed: \_incrementCounter,  tooltip: 'Increment',  child: Icon(Icons.add),  ), // This trailing comma makes auto-formatting nicer for  build methods.  );  }  } |
| --- |

Mari kita lihat kode diatas bagian demi bagian.

Pertama, hanya ada satu kelas field, yang bernama \_counter, sehingga Anda dapat menyimpulkan bahwa status widget MyHomePage ditentukan oleh properti tunggal tersebut. Properti \_counter mencatat jumlah penekanan tombol di sudut kanan bawah layar. Bagaimana perubahan properti \_counter ini akan ditentukan dalam business logic Anda.

## Apa itu business logic?

Business logic adalah bagian dari kode tempat Anda menentukan aturan bisnis untuk aplikasi Anda. Tidak seperti kebanyakan kode aplikasi Anda, yang berkaitan dengan detail tingkat yang lebih rendah, seperti cara menampilkan widget atau menghubungkan ke database, business logic menentukan bagaimana pengguna berinteraksi dengan aplikasi Anda, dan bagaimana interaksi tersebut memengaruhi data status.

Misalnya, di mana widget ditempatkan di layar, apa warnanya, dan bagaimana reaksinya saat ditekan bukanlah business logic. Namun, jika dengan menekannya, pengguna dapat mengubah informasi tentang diri mereka sendiri, seperti jenis kelamin, alamat rumah, atau berapa banyak kue yang ingin mereka beli, maka itulah logika bisnis (business logic).

Metode pertama yang kita temui adalah metode \_incrementCounter. Ini tidak membutuhkan argumen dan memiliki kode **void** return yang menentukan bahwa ia tidak mengembalikan nilai. Namun, itu memang melakukan sesuatu yang penting yang harus kita jelajahi lebih detail:

| setState(() {  \_counter++;  }); |
| --- |

Widget stateful dimaksudkan untuk mengubah penampilannya selama masa pakainya – yaitu, mendefinisikan apa yang akan berubah – dan karenanya perlu dibangun kembali untuk mencerminkan perubahan tersebut. Dalam diagram **StatefulWidget** (Figure 5.2), kita melihat bahwa kerangka kerja membangun kembali StatefulWidget untuk mencerminkan keadaan baru. Namun, bagaimana kerangka kerja mengetahui kapan sesuatu di widget berubah dan bahwa pembangunan kembali diperlukan?

Jawabannya adalah metode setState. Metode ini menerima fungsi sebagai parameter yang memperbarui **State** terkait widget. Dalam hal ini, kami telah membuat fungsi anonim, dan di badan fungsi, kami telah menetapkan bahwa variabel \_counter harus bertambah. Dengan memanggil setState, kerangka kerja diberi tahu bahwa ia perlu membangun kembali widget. Setelah dipanggil, widget akan digambar ulang dengan nilai \_counter baru yang sudah ditetapkan.

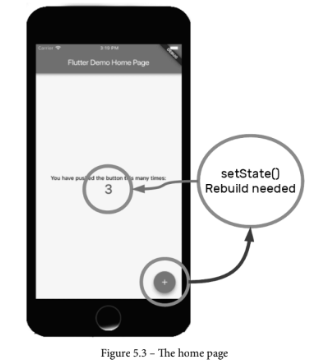
Akhirnya, kita mencapai metode pembuatan widget. Tanda tangan metode dan fungsi yang dimaksud identik dengan metode pembuatan StatelessWidget yang telah kita bahas sebelumnya. Namun, tidak seperti StatelessWidget, kami sekarang memiliki status yang akan memengaruhi cara kami menggambar widget, yang dapat mengarah ke kode yang jauh lebih kompleks yang melibatkan lebih banyak pernyataan bersyarat.

Metode build dapat terlihat menakutkan, dan kita akan melihat masing-masing widget secara lebih rinci nanti di bab ini. Namun, pada titik ini, cobalah untuk merasakan struktur komposisi yang ditampilkan. Metode ini mengembalikan widget Scaffold di tingkat atas dan terdiri dari tiga widget anak melalui tiga argumen konstruktor:

* appBar: Ini menyimpan widget tipe AppBar, yang memiliki satu argumen konstruktor bernama title. Seperti yang dapat Anda tebak, ini menjelaskan widget yang akan muncul di bagian atas layar sebagai bilah aplikasi.
* body: Ini dapat menampung widget apa pun dan muncul di badan utama aplikasi (yaitu, di antara bilah aplikasi atas dan bilah menu bawah). Dalam hal ini, ia memegang widget Center (yang memusatkan anak konten). Ini, pada gilirannya, memegang widget Column(yang membuat kolom widget vertikal). Terakhir, ini menampung dua widget Text(yang menampilkan string teks).
* floatingActionButton: Ini menampung widget jenis FloatingActionButton, yang merupakan tombol yang mengapung di atas badan aplikasi di sudut kanan bawah (sebagai konfigurasi default) dan berfungsi sebagai tombol

Perhatikan bahwa salah satu argumen untuk konstruktor FloatingActionButton adalah onPressed, dan nilainya adalah metode \_incrementCounter. Ini mengikat seluruh flow bersama-sama:

1. Widget MyHomePage memanggil metode build dari status pendamping untuk menampilkan bilah aplikasi, isi, dan tombol tindakan.
2. Pengguna menekan tombol action.
3. Nilai argumen **onPressed** dipicu, yang memanggil metode \_incrementCounter.
4. Metode \_incrementCounter memanggil metode setState dengan fungsi anonim, menetapkan bahwa variabel \_counter harus bertambah.
5. Framework memanggil fungsi anonim, sehingga menambah variabel \_counter.
6. Framework menggambar ulang widget dengan memanggil metode build dari state pendamping lagi untuk menampilkan update app bar, body, dan action button yang diperbarui:

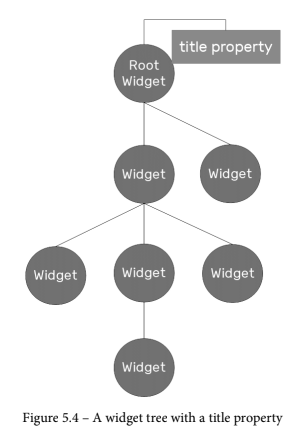


Seperti yang ditunjukkan diagram sebelumnya, metode setState sangat penting untuk keseluruhan flow ini, dan memahami aliran yang digambar ulang ini adalah inti dari cara kerja framework Flutter.

Mari selesaikan bagian ini dengan melihat jenis widget yang kurang terkenal, yaitu jenis InheritedWidget.

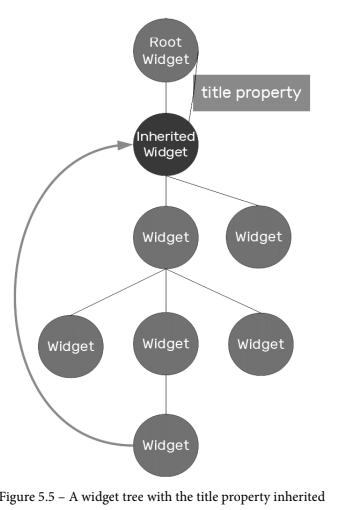
## Inherited widgets

Selain StatelessWidget dan StatefulWidget, ada satu lagi jenis widget di framework Flutter, InheritedWidget. Terkadang, satu widget mungkin memerlukan akses ke data lebih jauh di atas widget tree. Dalam skenario seperti itu, salah satu solusinya adalah mereplikasi informasi ke widget yang tertarik dengan meneruskannya melalui semua widget perantara. Mari kita lihat contoh struktur komposisi widget sehingga kita dapat memeriksanya lebih detail:



Dalam skenario ini, anggaplah salah satu widget di bawah pohon memerlukan akses ke title properti dari widget root. Agar hal ini terjadi di dunia di mana hanya ada widget stateful dan stateless, kita perlu meneruskan properti title ke setiap child widget melalui konstruktor sehingga pada gilirannya widget turunan dapat meneruskan properti title ke widget turunannya. Ini dapat menyebabkan banyak kode boilerplate, bisa rawan kesalahan jika salah satu widget tidak dikodekan dengan benar, dan bisa sangat menyakitkan jika diputuskan bahwa child widget memerlukan properti lain, yang berarti bahwa semua perantara widget perlu diperbarui.

Untuk mengatasi masalah ini, Flutter menyediakan kelas InheritedWidget. Ini adalah jenis widget tambahan yang membantu menyebarkan informasi ke bawah pohon, seperti yang ditunjukkan pada diagram berikut:



Dengan menambahkan InheritedWidget ke pohon, widget apa pun di bawahnya dapat mengakses data yang dieksposnya dengan menggunakan metode of(InheritedWidget) dari kelas BuildContext yang menerima tipe InheritedWidget sebagai parameter dan menggunakan pohon untuk menemukan widget ancestral pertama dari jenis yang diminta.

Ada beberapa penggunaan InheritedWidget yang sangat umum di Flutter. Salah satu kegunaan paling umum adalah dari class Theme, yang membantu mendeskripsikan warna untuk keseluruhan aplikasi. Kita akan membahas ini di Bab 6, Menangani Masukan dan Gestur Pengguna.

Sekarang, mari kita selidiki sesuatu yang mungkin telah Anda lihat di beberapa contoh kode sebelumnya, yaitu, properti konstruktor dengan nama **key**.

## The widget key property

Jika Anda melihat konstruktor kelas StatelessWidget dan StatefulWidget, Anda akan melihat parameter bernama **key**. Ini adalah properti penting untuk widget di Flutter.

Properti **key** memungkinkan Anda untuk mempertahankan status widget di antara pembuatan ulang. Anda mungkin ingat bahwa framework mengambil widget tree dan merendernya ke tree elemen. Tree elemen adalah representasi yang sangat sederhana dari widget tree yang hanya menyimpan jenis widget dan referensi ke widget turunannya. Saat terjadi perubahan atau pembangunan kembali, framework menggunakan tipe widget dan referensi turunan untuk menentukan apakah gambar ulang diperlukan. Dalam situasi di mana ada banyak widget dengan tipe yang sama dengan anak-anak dari widget yang sama (misalnya, baris atau kolom), mungkin ada situasi di mana pengurutan berubah tetapi tidak membatalkan pohon elemen. Dalam situasi ini, perilaku Flutter bisa jadi tidak terduga.

Tanpa key, tree elemen tidak akan tahu state mana yang sesuai dengan widget mana, karena semuanya memiliki tipe yang sama. Ketika sebuah widget memiliki state, ia membutuhkan state yang sesuai untuk dipindahkan dengannya. Sederhananya, itulah yang dilakukan oleh **key** untuk membantu kerangka kerja. Dengan menahan nilai key, elemen yang bersangkutan akan mengetahui state widget terkait yang harus ada dengannya.

Dalam kebanyakan situasi, key tidak diperlukan (dan tidak boleh digunakan); namun, jika Anda melihat beberapa perilaku aneh di mana Anda yakin widget Anda berubah state tetapi ini tidak tercermin dengan benar di UI, maka ada baiknya memeriksa apakah kerangka kerja salah memahami tree elemen dan apakah key mungkin merupakan obat yang cocok untuk situasi. Jika Anda memerlukan informasi lebih lanjut tentang bagaimana key memengaruhi widget dan jenis key yang tersedia, silakan lihat pengantar dokumentasi resmi untuk key:

<https://flutter.dev/docs/development/ui/widgets-intro#keys>.

Kami sekarang telah menjelajahi blok bangunan dasar Flutter: widget. Secara khusus, kami telah melihat tiga jenis widget, stateless, stateful, dan inheritance, bersama dengan situasi di mana Anda akan menggunakannya. Sekarang Anda memiliki pengetahuan dasar tentang dasar-dasar widget, mari kita lihat beberapa widget yang Anda dapatkan gratis saat Anda memulai UI.

# Praktikum 1: Membuat Project Flutter Baru

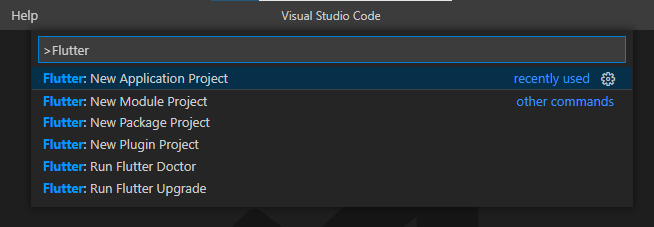
Duration: 10:00

Selesaikan langkah-langkah praktikum berikut ini menggunakan editor Visual Studio Code (VS Code) atau Android Studio atau code editor lain kesukaan Anda.

| **Perhatian:** Diasumsikan Anda telah berhasil melakukan setup environment Flutter SDK, VS Code, Flutter Plugin, dan Android/iOS SDK pada [Codelabs#1](https://jti-polinema.github.io/flutter-codelab/01-pengenalan/). |
| --- |

## Langkah 1:

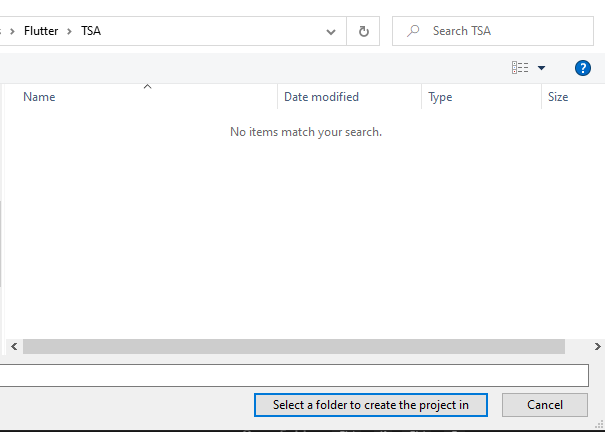
Buka VS Code, lalu tekan tombol **Ctrl + Shift + P** maka akan tampil *Command Palette*, lalu ketik **Flutter**. Pilih **New Application Project**.



| **Catatan:** Anda dapat mengakses **Command Palette** dengan cara lain, yaitu pilih menu **View > Command Palette**. Tombol shortcut **Ctrl + Shift + P** mungkin hanya berlaku di Windows. |
| --- |

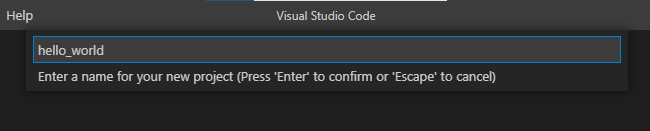
## Langkah 2:

Kemudian buat folder **sesuai style** laporan praktikum yang Anda pilih. Disarankan pada folder dokumen atau desktop atau alamat folder lain yang tidak terlalu dalam atau panjang. Lalu pilih **Select a folder to create the project in**.



## Langkah 3:

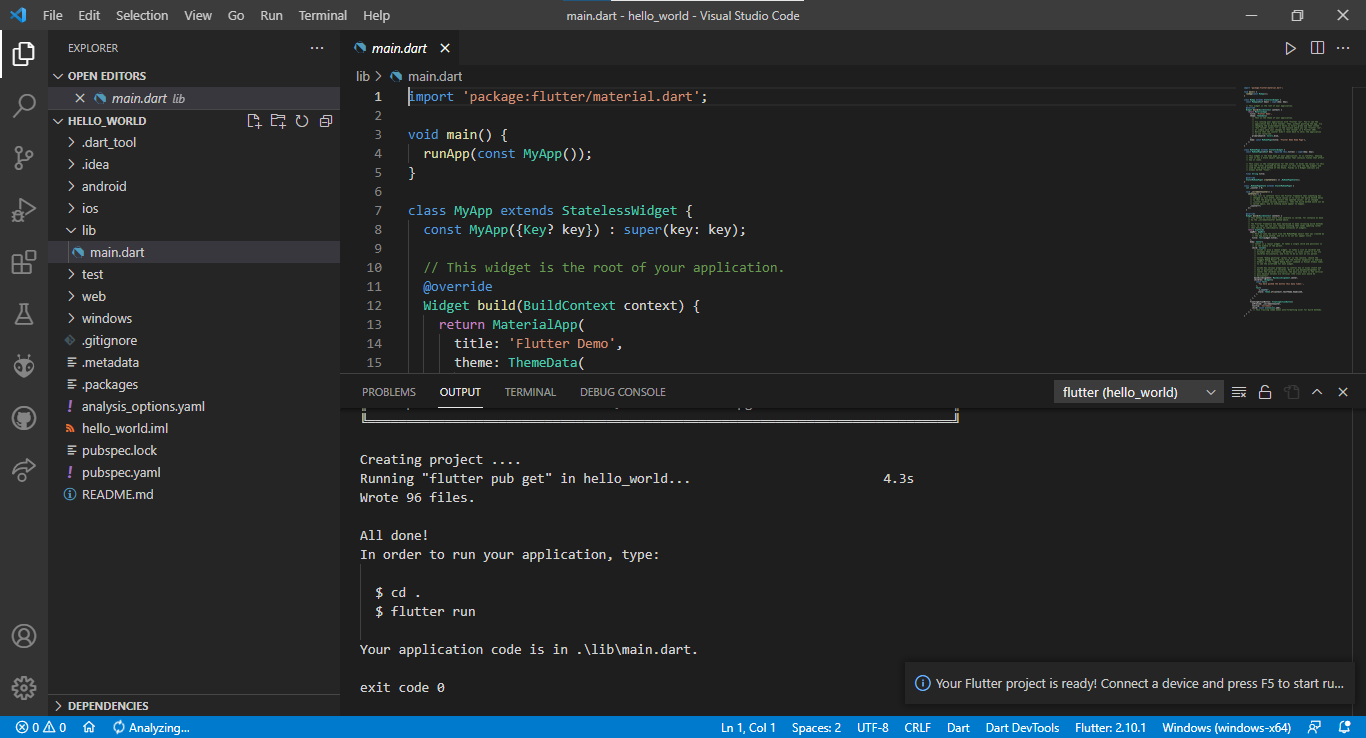
Buat nama project flutter **hello\_world** seperti berikut, lalu tekan **Enter**. Tunggu hingga proses pembuatan project baru selesai.



| **Perhatian:** Nama project ini harus *lowercase* (huruf kecil semua) tanpa menggunakan spasi. Untuk memisahkan kata, bisa menggunakan *underline* (garis bawah). Nama project tidak dapat diawali dengan angka atau karakter khusus lain. Nama project ini bukan nama aplikasi yang akan tampil di Play Store atau App Store. Untuk nama aplikasi, nanti dapat diatur ketika melakukan *deployment*. |
| --- |

## Langkah 4:

Jika sudah selesai proses pembuatan project baru, pastikan tampilan seperti berikut. Pesan akan tampil berupa “**Your Flutter Project is ready!**” artinya Anda telah berhasil membuat project Flutter baru.



# Praktikum 2: Membuat Repository GitHub dan Laporan Praktikum

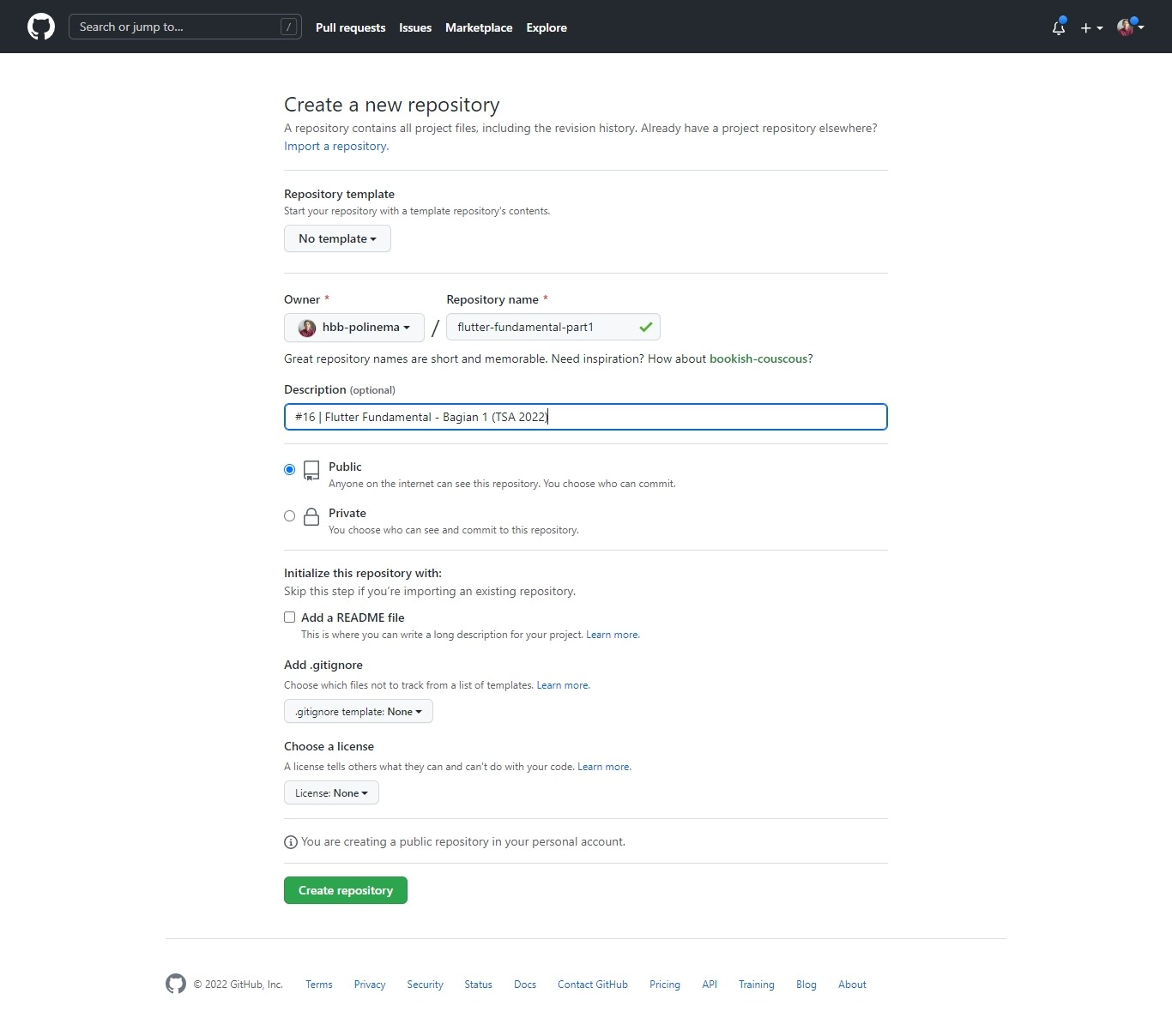
Duration: 30:00

Melanjutkan dari praktikum 1, silakan selesaikan langkah-langkah berikut ini.

| **Perhatian:** Diasumsikan Anda telah mempunyai akun GitHub dan Anda telah memahami konsep dasar dalam bekerja menggunakan Git pada pertemuan pertama.  Praktikum ini dapat Anda **lewati langsung ke langkah 11** jika sudah paham cara membuat laporan praktikum sesuai *style* yang Anda pilih. |
| --- |

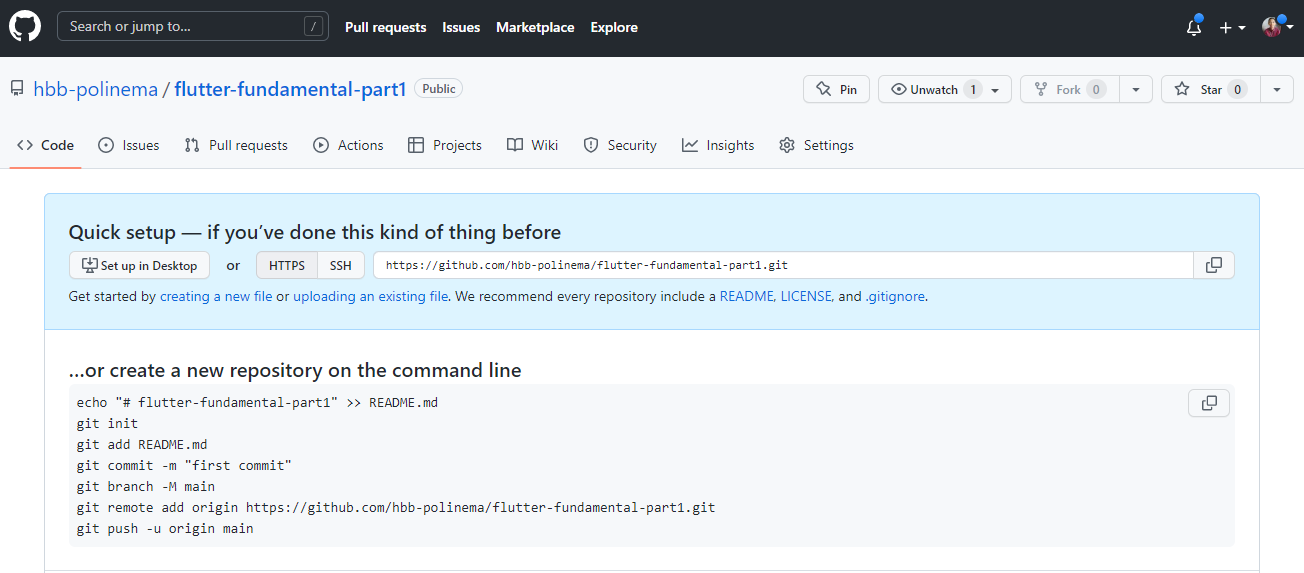
## Langkah 1:

Login ke akun [GitHub](https://github.com/new) Anda, lalu buat repository baru dengan nama “**flutter-fundamental-part1**”



## Langkah 2:

Lalu klik tombol “**Create repository**” lalu akan tampil seperti gambar berikut.



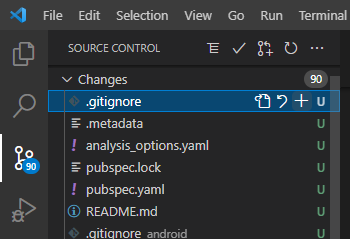
## Langkah 3:

Kembali ke VS code, project flutter hello\_world, buka terminal pada menu **Terminal > New Terminal**. Lalu ketik perintah berikut untuk inisialisasi git pada project Anda.

| git init |
| --- |

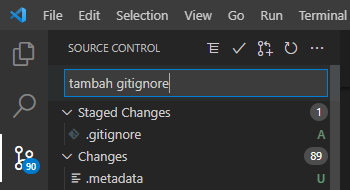
## Langkah 4:

Pilih menu **Source Control** di bagian kiri, lalu lakukan **stages** (+) pada file **.gitignore** untuk mengunggah file pertama ke repository GitHub.



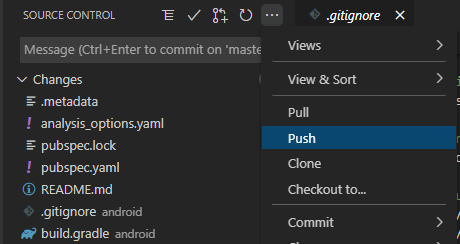
## Langkah 5:

Beri pesan commit “**tambah gitignore**” lalu klik **Commit** (✔)



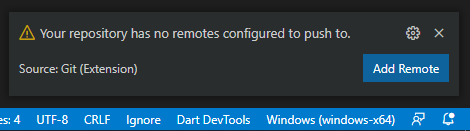
## Langkah 6:

Lakukan push dengan klik bagian menu titik tiga > **Push**



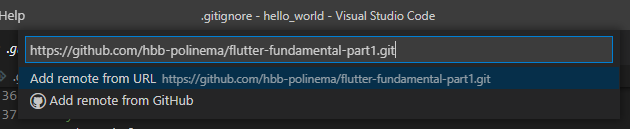
## Langkah 7:

Di pojok kanan bawah akan tampil seperti gambar berikut. Klik “**Add Remote**”

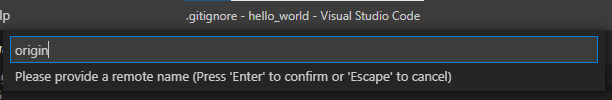


## Langkah 8:

Salin tautan repository Anda dari browser ke bagian ini, lalu klik **Add remote**

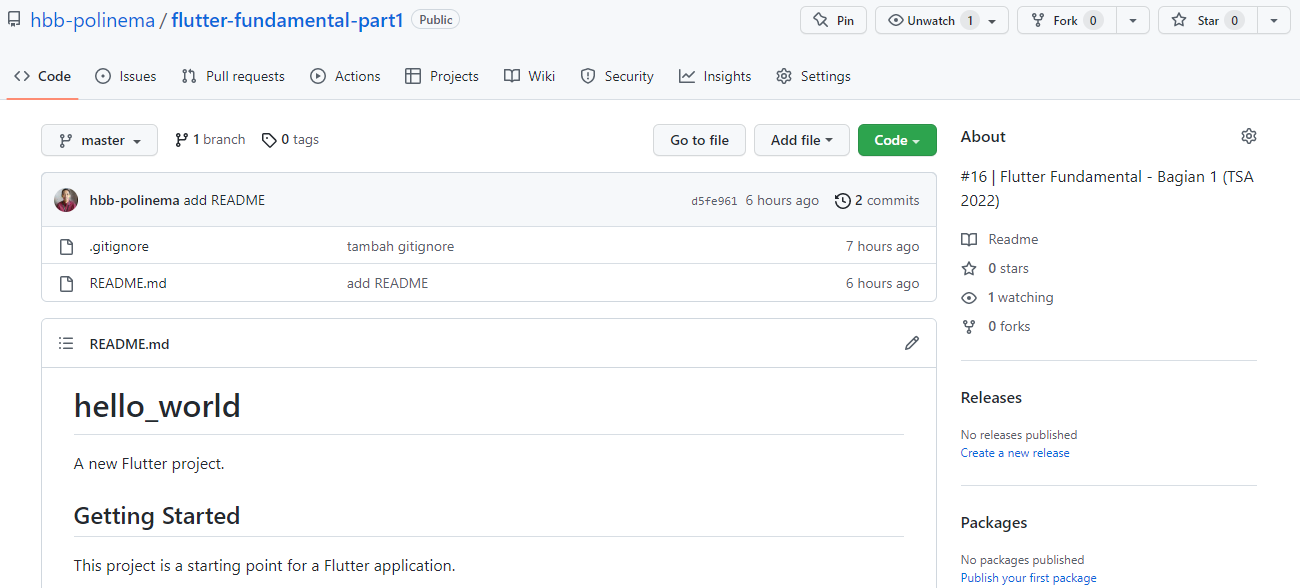


Setelah berhasil, tulis remote name dengan “**origin**”



## Langkah 9:

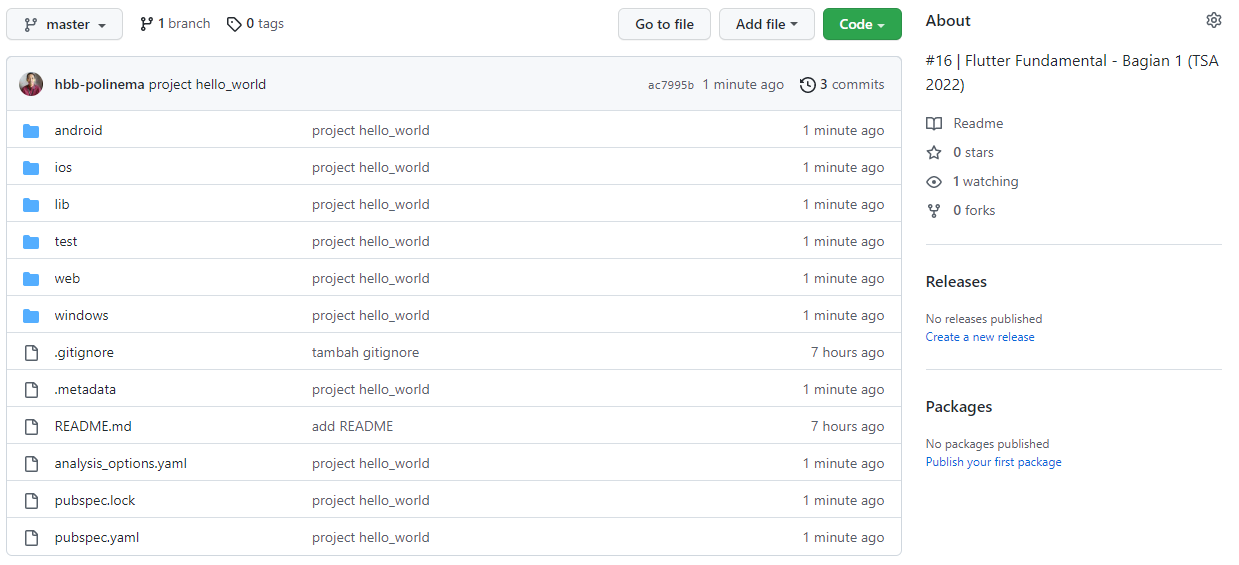
Lakukan hal yang sama pada file **README.md** mulai dari Langkah 4. Setelah berhasil melakukan push, masukkan username GitHub Anda dan password berupa token yang telah dibuat (pengganti password konvensional ketika Anda login di browser GitHub). Reload halaman repository GitHub Anda, maka akan tampil hasil push kedua file tersebut seperti gambar berikut.



| **Perhatian:** Personal access token GitHub dapat Anda buat melalui <https://github.com/settings/tokens/new> atau dapat membaca dokumentasi [GitHub Token](https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token). |
| --- |

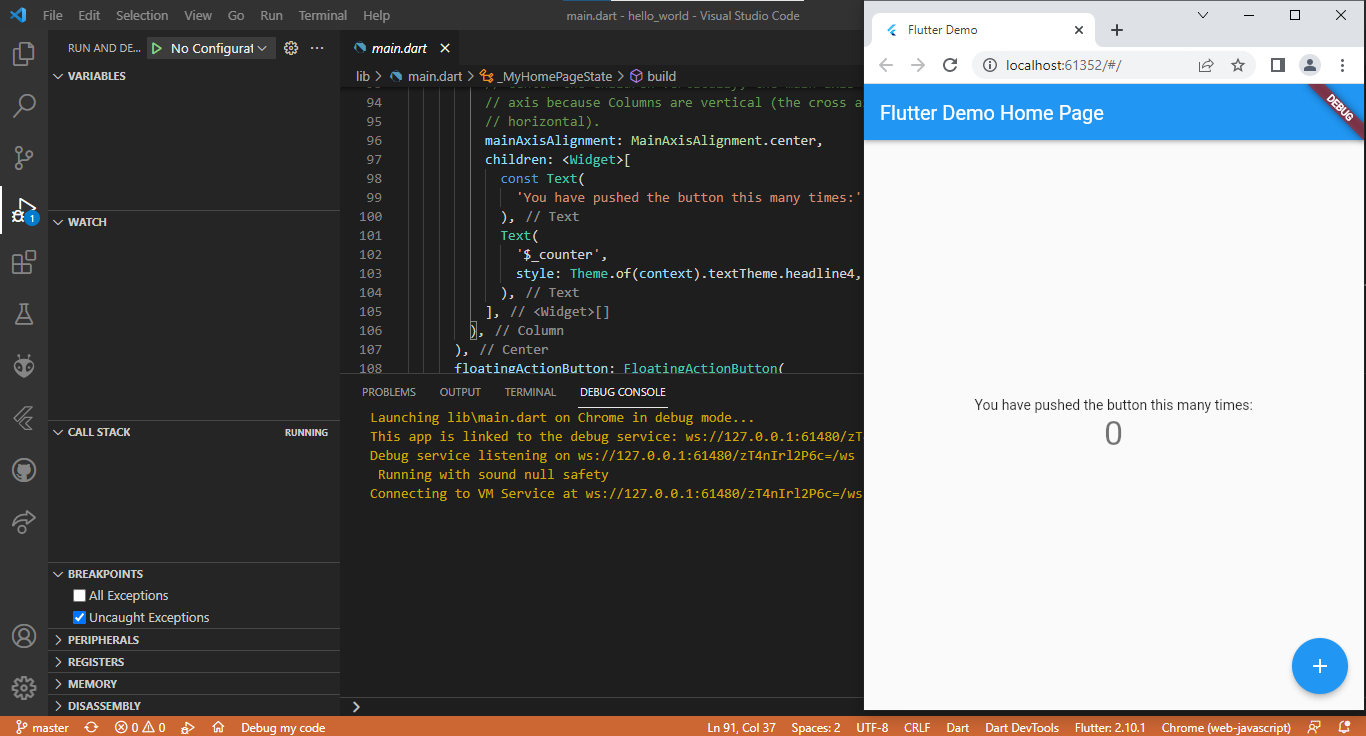
## Langkah 10:

Lakukan push juga untuk semua file lainnya dengan pilih **Stage All Changes**. Beri pesan commit “**project hello\_world**”. Maka akan tampil di repository GitHub Anda seperti berikut.



## Langkah 11:

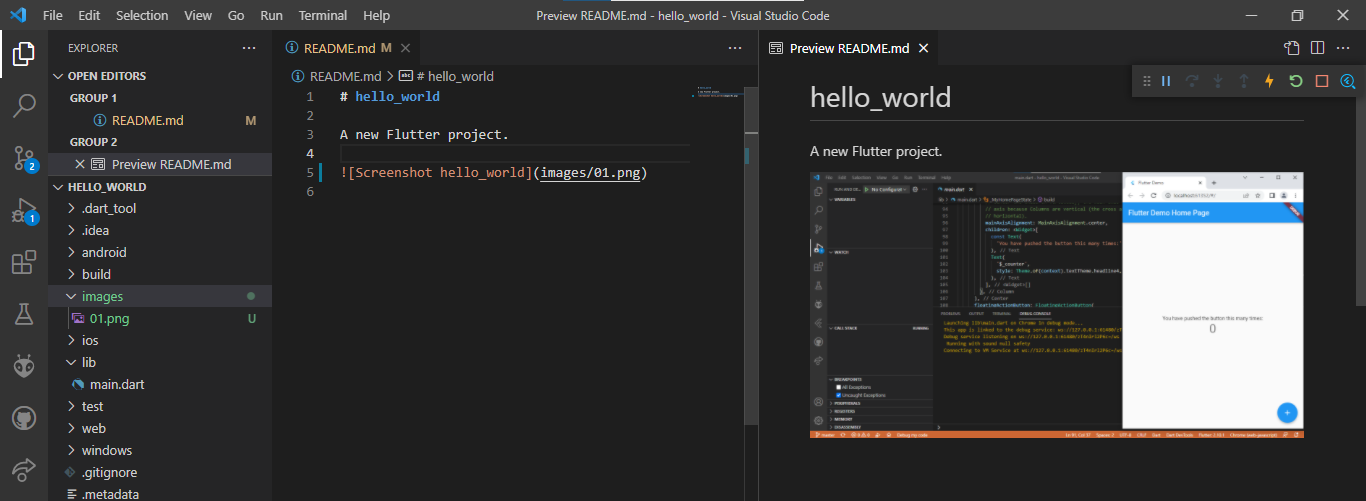
Kembali ke VS Code, ubah platform di pojok kanan bawah ke emulator atau device atau bisa juga menggunakan browser Chrome. Lalu coba running project **hello\_world** dengan tekan **F5** atau **Run > Start Debugging**. Tunggu proses kompilasi hingga selesai, maka aplikasi flutter pertama Anda akan tampil seperti berikut.



| **Perhatian:** Proses Run atau kompilasi untuk pertama kali akan memakan waktu cukup lama (3-5 menit). Namun, proses kompilasi berikutnya menjadi lebih cepat. Berbeda ketika melakukan kompilasi ke device, hal ini akan memakan waktu lebih lama lagi. |
| --- |

## Langkah 12:

Silakan screenshot seperti pada Langkah 11, namun teks yang ditampilkan dalam aplikasi berupa nama lengkap Anda. Simpan file screenshot dengan nama **01.png** pada folder **images** (buat folder baru jika belum ada) di project hello\_world Anda. Lalu ubah isi README.md seperti berikut, sehingga tampil hasil screenshot pada file README.md. Kemudian push ke repository Anda.



| **Perhatian:** Lakukan proses screenshot seperti pada Langkah 12 untuk setiap Laporan Praktikum yang Anda akan buat pada praktikum selanjutnya hingga pertemuan project final. |
| --- |

# Praktikum 3: Menerapkan Widget Dasar

Duration: 30:00

Selesaikan langkah-langkah praktikum berikut ini dengan melanjutkan dari praktikum sebelumnya.

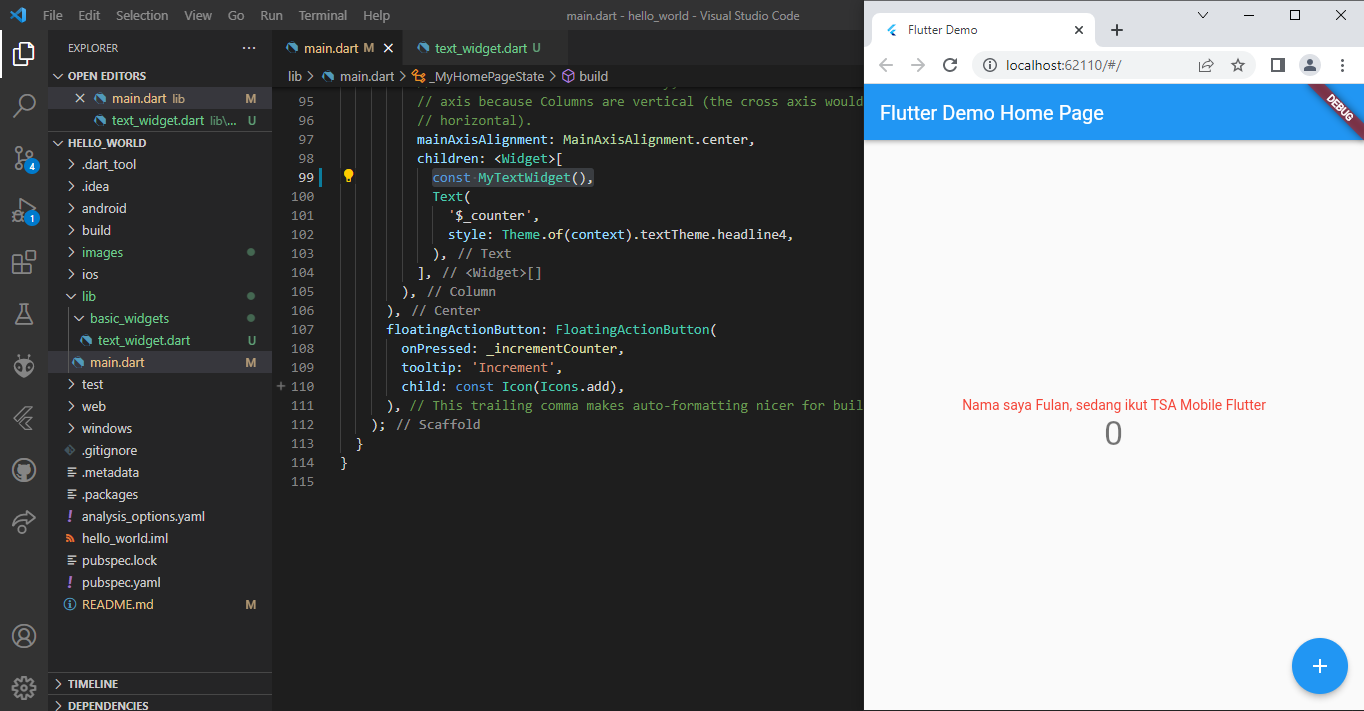
## Langkah 1: Text Widget

Buat folder baru **basic\_widgets** di dalam folder **lib**. Kemudian buat file baru di dalam basic\_widgets dengan nama text\_widget.dart. Ketik atau salin kode program berikut ke project hello\_world Anda pada file text\_widget.dart.

| import 'package:flutter/material.dart';  class MyTextWidget extends StatelessWidget {  const MyTextWidget({Key? key}) : super(key: key);  @override  Widget build(BuildContext context) {  return const Text(  "Nama saya Fulan, sedang belajar Pemrograman Mobile",  style: TextStyle(color: Colors.red, fontSize: 14),  textAlign: TextAlign.center);  }  } |
| --- |

| **Perhatian:** Gantilah teks **Fulan** dengan nama lengkap Anda. |
| --- |

Lakukan import file text\_widget.dart ke main.dart, lalu ganti bagian text widget dengan kode di atas. Maka hasilnya seperti gambar berikut. Screenshot hasil milik Anda, lalu dibuat laporan pada file README.md.



## Langkah 2: Image Widget

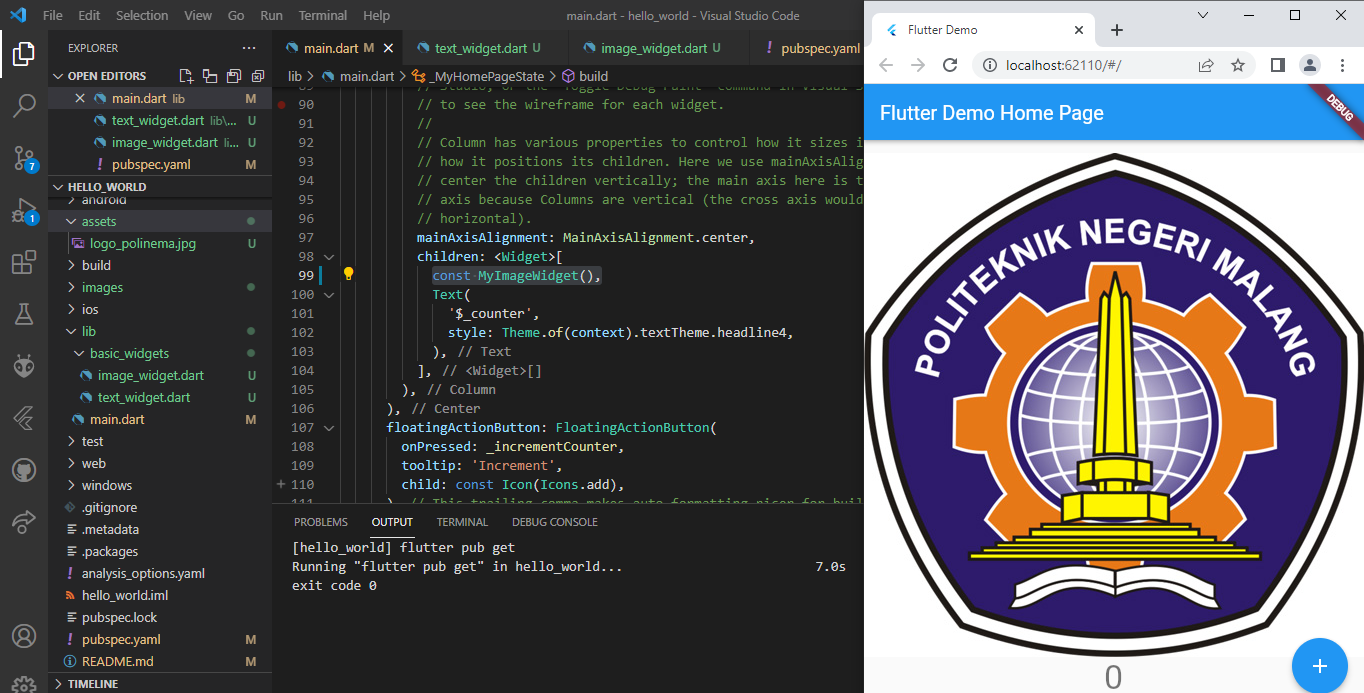
Buat sebuah file image\_widget.dart di dalam folder basic\_widgets dengan isi kode berikut.

| import 'package:flutter/material.dart';  class MyImageWidget extends StatelessWidget {  const MyImageWidget({Key? key}) : super(key: key);  @override  Widget build(BuildContext context) {  return const Image(  image: AssetImage("logo\_polinema.jpg")  );  }  } |
| --- |

Lakukan penyesuaian asset pada file pubspec.yaml dan tambahkan file logo Anda di folder assets project hello\_world.

| flutter:  assets:  - logo\_polinema.jpg |
| --- |

Jangan lupa sesuaikan kode dan import di file main.dart kemudian akan tampil gambar seperti berikut.



# Praktikum 4: Menerapkan Widget Material Design dan iOS Cupertino

Duration: 30:00

Selesaikan langkah-langkah praktikum berikut ini dengan melanjutkan project hello\_world Anda. Lakukan langkah yang sama seperti pada Praktikum 3, yaitu setiap widget dibuat file sendiri lalu import ke main.dart dan screenshot hasilnya.

## Langkah 1: Cupertino Button dan Loading Bar

Buat file di basic\_widgets > loading\_cupertino.dart. Import stateless widget dari material dan cupertino. Lalu isi kode di dalam method Widget build adalah sebagai berikut.

| return MaterialApp(  home: Container(  margin: const EdgeInsets.only(top: 30),  color: Colors.white,  child: Column(  children: <Widget>[  CupertinoButton(  child: const Text("Contoh button"),  onPressed: () {},  ),  const CupertinoActivityIndicator(),  ],  ),  ),  ); |
| --- |

## Langkah 2: Floating Action Button (FAB)

Button widget terdapat beberapa macam pada flutter yaitu ButtonBar, DropdownButton, TextButton, FloatingActionButton, IconButton, OutlineButton, PopupMenuButton, dan ElevatedButton.

Buat file di basic\_widgets > fab\_widget.dart. Import stateless widget dari material. Lalu isi kode di dalam method Widget build adalah sebagai berikut.

| return MaterialApp(  home: Scaffold(  floatingActionButton: FloatingActionButton(  onPressed: () {  // Add your onPressed code here!  },  child: const Icon(Icons.thumb\_up),  backgroundColor: Colors.pink,  ),  ),  ); |
| --- |

## Langkah 3: Scaffold Widget

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

Ubah isi kode main.dart seperti berikut.

| import 'package:flutter/material.dart';  void main() {  runApp(const MyApp());  }  class MyApp extends StatelessWidget {  const MyApp({Key? key}) : super(key: key);  // This widget is the root of your application.  @override  Widget build(BuildContext context) {  return MaterialApp(  title: 'Flutter Demo',  theme: ThemeData(  primarySwatch: Colors.red,  ),  home: const MyHomePage(title: 'My Increment App'),  );  }  }  class MyHomePage extends StatefulWidget {  const MyHomePage({Key? key, required this.title}) : super(key: key);  final String title;  @override  State<MyHomePage> createState() => \_MyHomePageState();  }  class \_MyHomePageState extends State<MyHomePage> {  int \_counter = 0;  void \_incrementCounter() {  setState(() {  \_counter++;  });  }  @override  Widget build(BuildContext context) {  return Scaffold(  appBar: AppBar(  title: Text(widget.title),  ),  body: Center(  child: Column(  mainAxisAlignment: MainAxisAlignment.center,  children: <Widget>[  const Text(  'You have pushed the button this many times:',  ),  Text(  '$\_counter',  style: Theme.of(context).textTheme.headline4,  ),  ],  ),  ),  bottomNavigationBar: BottomAppBar(  child: Container(  height: 50.0,  ),  ),  floatingActionButton: FloatingActionButton(  onPressed: \_incrementCounter,  tooltip: 'Increment Counter',  child: const Icon(Icons.add),  ),  floatingActionButtonLocation: FloatingActionButtonLocation.centerDocked,  );  }  } |
| --- |

## Langkah 4: Dialog Widget

Dialog widget pada flutter memiliki dua jenis dialog yaitu AlertDialog dan SimpleDialog.

Ubah isi kode main.dart seperti berikut.

| class MyApp extends StatelessWidget {  const MyApp({Key? key}) : super(key: key);  @override  Widget build(BuildContext context) {  return const MaterialApp(  home: Scaffold(  body: MyLayout(),  ),  );  }  }  class MyLayout extends StatelessWidget {  const MyLayout({Key? key}) : super(key: key);  @override  Widget build(BuildContext context) {  return Padding(  padding: const EdgeInsets.all(8.0),  child: ElevatedButton(  child: const Text('Show alert'),  onPressed: () {  showAlertDialog(context);  },  ),  );  }  }  showAlertDialog(BuildContext context) {  // set up the button  Widget okButton = TextButton(  child: const Text("OK"),  onPressed: () {  Navigator.pop(context);  },  );  // set up the AlertDialog  AlertDialog alert = AlertDialog(  title: const Text("My title"),  content: const Text("This is my message."),  actions: [  okButton,  ],  );  // show the dialog  showDialog(  context: context,  builder: (BuildContext context) {  return alert;  },  );  } |
| --- |

## Langkah 5: 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 adalah sebagai berikut:

| class MyApp extends StatelessWidget {  const MyApp({Key? key}) : super(key: key);  @override  Widget build(BuildContext context) {  return MaterialApp(  home: Scaffold(  appBar: AppBar(title: const Text("Contoh TextField")),  body: const TextField(  obscureText: false,  decoration: InputDecoration(  border: OutlineInputBorder(),  labelText: 'Nama',  ),  ),  ),  );  }  } |
| --- |

## Langkah 6: 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(const MyApp());  class MyApp extends StatelessWidget {  const MyApp({Key? key}) : super(key: key);  @override  Widget build(BuildContext context) {  return const MaterialApp(  title: 'Contoh Date Picker',  home: MyHomePage(title: 'Contoh Date Picker'),  );  }  }  class MyHomePage extends StatefulWidget {  const MyHomePage({Key? key, required 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<void> \_selectDate(BuildContext context) async {  // Initial DateTime FIinal 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]),  const SizedBox(  height: 20.0,  ),  ElevatedButton(  onPressed: () => {  \_selectDate(context),  // ignore: avoid\_print  print(selectedDate.day + selectedDate.month + selectedDate.year)  },  child: const Text('Pilih Tanggal'),  ),  ],  ),  ),  );  }  } |
| --- |

# Tugas Praktikum

Duration: 90:00

1. Selesaikan Praktikum 1 sampai 4, lalu dokumentasikan dan push ke repository Anda berupa screenshot setiap hasil pekerjaan beserta penjelasannya di file README.md!
2. Pada praktikum 4 mulai dari Langkah 3 sampai 6, buatlah file widget tersendiri di folder basic\_widgets, kemudian pada file main.dart cukup melakukan import widget sesuai masing-masing langkah tersebut!
3. Selesaikan [Codelabs: Your first Flutter app](https://codelabs.developers.google.com/codelabs/flutter-codelab-first#0), lalu buatlah laporan praktikumnya dan push ke repository GitHub Anda!
4. README.md berisi: capture hasil akhir tiap praktikum (*side-by-side*, bisa juga berupa file GIF agar terlihat proses perubahan ketika ada aksi dari pengguna) di browser dan perangkat fisik (*device*) dengan menampilkan NIM dan Nama Anda sebagai ciri pekerjaan Anda. Jika *mode developer* di perangkat HP Anda belum aktif, silakan cari di internet cara mengaktifkannya!
5. Kumpulkan berupa link repository/commit GitHub Anda ke tautan spreadsheet yang telah disepakati oleh dosen!

# Selamat!

Duration: 0:00

Selamat Anda telah menyelesaikan Codelab ini. Anda telah mempelajari terkait Flutter Fundamental untuk dasar-dasar widget.

Pada codelab berikutnya, Anda akan mempelajari tentang Flutter Fundamental Bagian 2 terkait layout, position, navigasi dan route.

## Apa selanjutnya?

Silakan cek beberapa sumber belajar lainnya:

* [FlutLab is a modern Flutter online IDE](https://flutlab.io/)
* [Flutter System Architecture](https://docs.google.com/presentation/d/1cw7A4HbvM_Abv320rVgPVGiUP2msVs7tfGbkgdrTy0I/edit#slide=id.p)
* [Adding a Home Screen widget to your Flutter App](https://codelabs.developers.google.com/flutter-home-screen-widgets#0)

## Referensi

* [Simone Alessandria - Flutter Cookbook\_ 100+ step-by-step recipes for building cross-platform, professional-grade apps with Flutter 3.10.x and Dart 3.x,-Packt Publishing (2023)](https://drive.google.com/file/d/1W9uSLC9wlf9fFudmCWFHTx9dan5SXII0/view)
* [(Mastering Computer Science) Sufyan bin Uzayr - Mastering Flutter\_ A Beginner's Guide-CRC Press (2022)](https://drive.google.com/file/d/1W8l7EHa7ssyA6xqXCvYnBbUfYEJTXAcY/view)
* [Flutter for Beginners Second Edition](https://drive.google.com/file/d/1hJbctx-_O3B6kUeG70bJD5xfbnsWwBXY/view?usp=sharing)
* <https://www.youtube.com/flutterdev>
* <https://github.com/PacktPublishing/Flutter-for-Beginners-Second-Edition>
* Buku Ajar Pemrograman Mobile Dengan Flutter, Jurusan Teknologi Informasi, Politeknik Negeri Malang. 2021.