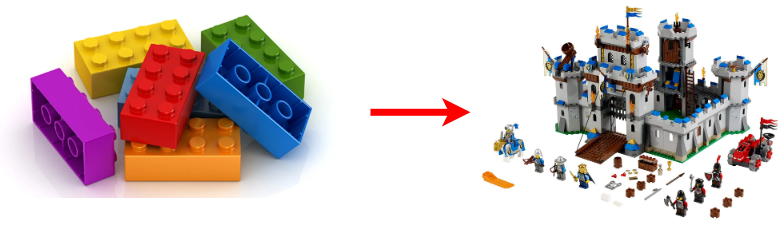
## **Apa itu Widget**

Kita telah berkenalan, menginstal, dan belajar fundamental Flutter. Itu *entrée* alias hidangan pembukanya. Nah, sekarang kita akan menyibak ke *plat du jour* alias menu utamanya. Apa inti dari Flutter? Yap, ***widget*** jawabnya! Sebagian besar yang ada pada Flutter adalah widget. Jadi, relevan jika kita bilang bahwa “Flutter is all about widget.” Text sendiri adalah widget. Button juga widget. Selain itu widget juga dapat dibangun dari kumpulan beberapa widget. Lantas mengapa widget begitu penting dalam flutter? Penasaran? Mari kita bahas!

### **World of components**

Perlu kita ketahui bahwa konsep Widget pada Flutter itu terinspirasi oleh salah satu library JavaScript yang digunakan untuk membangun sebuah website yaitu ReactJS. ReactJS memiliki konsep Component. Mari kita analogikan dengan mainan Lego! Di Lego terdapat *block-block* kecil yang nantinya kita susun untuk membangun sebuah istana Lego. Berarti *component* dalam *programming* adalah sekumpulan *block-block code* yang digunakan untuk membangun sebuah aplikasi.



Widget sama halnya dengan component yang merupakan kumpulan block code untuk membangun aplikasi Flutter. Ketika membangun aplikasi Flutter kita harus berpikir layaknya bermain Lego. Kita harus bisa membuat dan menyusun widget-widget dengan tepat. Tujuannya, agar aplikasi yang kita buat lebih mudah untuk dikembangkan.

### **Bagaimana cara menulis widget**

Sebetulnya pada pembahasan sebelumnya secara tidak sadar kita telah membuat sebuah widget dan menggunakan widget yang telah disediakan.

* class MyApp extends StatelessWidget {
* const MyApp({Key? key}) : super(key: key);
* @override
* Widget build(BuildContext context) {
* return MaterialApp(
* title: 'Flutter Demo',
* theme: ThemeData(
* primarySwatch: Colors.blue,
* ),
* home: Scaffold(
* appBar: AppBar(
* title: const Text('Hello, world!'),
* ),
* body: const Center(
* child: Text('Hello, world!'),
* ),
* ),
* );
* }
* }

Pada kode di atas kita telah membuat sebuah Widget MyApp dan telah menggunakan widget-widget bawaan Flutter di antaranya *MaterialApp*, *Scaffold*, *Center*, dan *Text*. Ketika menggunakan widget, kita tinggal panggil nama widget dan bila ada properti atau parameter pada widget tersebut tinggal kita isikan properti atau parameternya.

* Center(
* child: Text('Hello world!'),
* )

Kode di atas merupakan contoh pemanggilan widget Center. Widget Center ini digunakan untuk membuat widget yang ada di dalamnya berada di posisi tengah (mirip seperti alignment center). Tinggal ketikkan Center lalu tambahkan properti child di dalamnya.

Perlu diketahui bahwa sebagian besar widget bawaan memiliki pola *parent-child*, seperti halnya Center yang memiliki child yang artinya di dalam child bisa terdapat widget lagi. Maka penulisan parent child akan seperti di bawah ini.

* Center( // parent dari Button
* child: TextButton( // child dari Center dan parent dari Text
* child: Text(), // child dari TextButton
* ),
* )

Pada contoh di atas widget Center dan Button hanya dapat memiliki satu anak atau bisa disebut child. Ada pula widget yang dapat memiliki banyak anak atau bisa disebut children, seperti Row, Column, ListView, GridView, dan semacamnya.

* Row(
* children: <Widget>[
* //di dalam children akan berisi banyak widget
* ]
* )

Contoh di atas adalah widget Row yang memiliki children. Di dalam children nantinya kita bisa menambahkan banyak widget. Berbeda dengan child yang diisi langsung dengan sebuah Widget, children akan berisi sebuah list yang di dalamnya diisi dengan banyak widget.

## **StatelessWidget dan StatefulWidget**

Seperti yang kita tahu jantung dari aplikasi Flutter adalah widget. Sebagian besar yang ada pada Flutter merupakan widget. Membuat tombol, menampilkan gambar, text, dan membuat tampilan berada di tengah pada Flutter semuanya menggunakan widget. Kita juga dapat membuat widget sendiri untuk dapat digunakan lain waktu ataupun dibagikan kepada Flutter developer lain (dalam bentuk packages).

Widget pada Flutter memiliki dua jenis, yaitu StatelessWidget dan StatefulWidget. Sebagai developer Flutter, kita harus memahami betul kedua jenis widget tersebut, maka pada bagian ini kita akan mempelajari apa itu StatelessWidget dan StatefulWidget.

### **Apa itu State?**

Sebelum membahas kedua jenis widget tersebut, kita harus berkenalan terlebih dahulu dengan istilah *State*. Kenapa demikian? Widget kita akan terus berurusan dengan State. Lalu apa itu State?

Untuk teman-teman dengan background *frontend web developer*, tentu tak akan asing dengan istilah State ini, terutama menggunakan framework ReactJS. Tetapi untuk Anda tanpa *background* tersebut tidak perlu risau. State tidaklah sulit untuk dimengerti. Jadi State adalah data yang ada pada suatu widget. Widget menyimpan data yang nantinya dapat berubah sesuai interaksi pengguna.

Karena Flutter menggunakan paradigma OOP (Object-Oriented Programming), state biasanya menjadi suatu properti dari sebuah class. Contohnya sebagai berikut:

* class ContohWidget extends StatelessWidget{
* final String \_judul;
* ...
* }

Variabel \_judul di atas merupakan contoh pendeklarasian state pada suatu widget.

### **StatelessWidget**

Setelah mengenal apa itu state, maka yang pertama kita akan bahas adalah StatelessWidget. StatelessWidget adalah widget yang nilai *state*-nya tidak dapat berubah (*immutable*) maka widget tersebut lebih bersifat statis dan memiliki interaksi yang terbatas.

Sekarang kita akan membuat sebuah Widget sederhana:

* class Heading extends StatelessWidget {
* final String text;
* const Heading({Key? key, required this.text}) : super(key: key);
* @override
* Widget build(BuildContext context){
* return Text(
* text,
* style: const TextStyle(
* fontSize: 24.0,
* fontWeight: FontWeight.bold,
* ),
* );
* }
* }

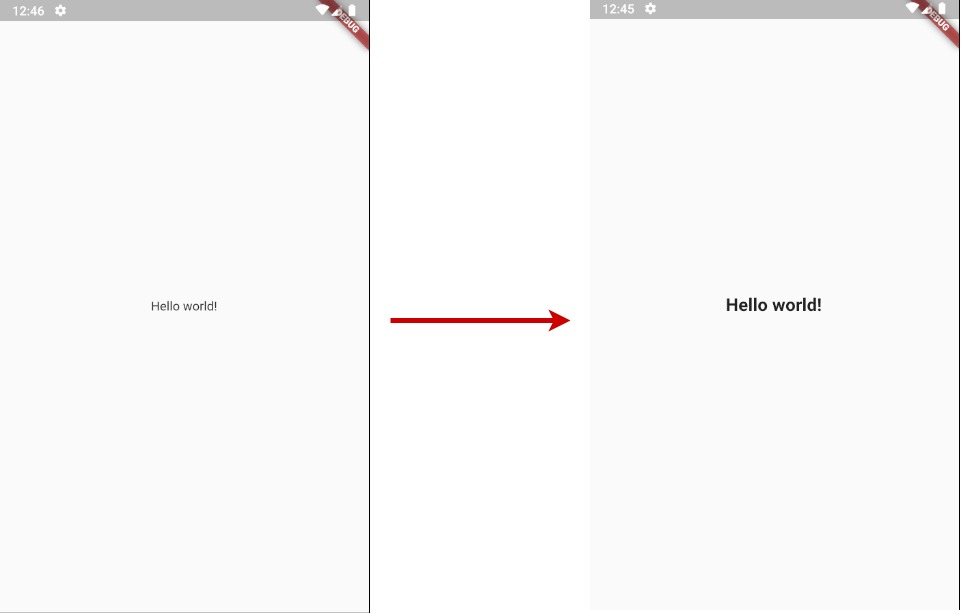
Widget di atas merupakan sebuah widget untuk membuat Heading (sebuah text yang digunakan untuk judul). Kita akan panggil widget yang telah diubah ke kode project pertama kita.

* import 'package:flutter/material.dart';
* void main() => runApp(MyApp());
* class MyApp extends StatelessWidget {
* const MyApp({Key? key}) : super(key: key);
* @override
* Widget build(BuildContext context) {
* return MaterialApp(
* title: 'Flutter Demo',
* theme: ThemeData(
* primarySwatch: Colors.blue,
* ),
* home: const Scaffold(
* body: Center(
* child: Text("Hello world!"),
* ),
* ),
* );
* }
* }
* class Heading extends StatelessWidget {
* final String text;
* const Heading({Key? key, required this.text}) : super(key: key);
* @override
* Widget build(BuildContext context){
* return Text(
* text,
* style: const TextStyle(
* fontSize: 24.0,
* fontWeight: FontWeight.bold,
* ),
* );
* }
* }

Kita coba ubah widget Text yang menampilkan "Hello world!" dengan widget Heading yang kita buat.

* import 'package:flutter/material.dart';
* void main() => runApp(MyApp());
* class MyApp extends StatelessWidget {
* @override
* Widget build(BuildContext context) {
* return MaterialApp(
* title: 'Flutter Demo',
* theme: ThemeData(
* primarySwatch: Colors.blue,
* ),
* home: const Scaffold(
* body: Center(
* **child: Heading( // mengubah widget Text**
* **text:"Hello world!",**
* **),**
* ),
* ),
* );
* }
* }
* class Heading extends StatelessWidget {
* final String text;
* const Heading({Key? key, required this.text}) : super(key: key);
* @override
* Widget build(BuildContext context){
* return Text(
* text,
* style: const TextStyle(
* fontSize: 24.0,
* fontWeight: FontWeight.bold,
* ),
* );
* }
* }

Maka ketika kita ubah Text dengan widget Heading, hasilnya akan berubah. Tulisan "Hello world!" jadi lebih besar.



Sesuai definisi StatelessWidget, *state*-nya tidak dapat berubah (*immutable*), maka state yang ada di dalam kelas tersebut harus dibuat final. Nilainya hanya dapat diisi melalui *constructor class*-nya.

* final String text; // state text bersifat final
* const Heading({Key? key, required this.text}) : super(key: key); // lalu state text masuk ke constructor

### **StatefulWidget**

Kebalikan dari StatelessWidget, StatefulWidget ialah widget yang *state*-nya dapat berubah-ubah nilainya, yang berarti StatefulWidget bersifat dinamis dan memiliki interaksi yang tak terbatas.

Penulisan StatefulWidget sangat berbeda dengan StatelessWidget, berikut penulisannya:

* class ContohStateful extends StatefulWidget {
* final String parameterWidget; // ini parameter widget
* const ContohStateful({Key? key, required this.parameterWidget}) : super(key: key);
* @override
* \_ContohStatefulState createState() => \_ContohStatefulState();
* }
* class \_ContohStatefulState extends State<ContohStateful>{
* String \_dataState; // ini state dari Widget ContohStateful
* @override
* Widget build(BuildContext context){
* // isi sebuah widget
* }
* }

Contoh di atas adalah cara penulisan StatefulWidget. Seperti yang kita lihat, penulisan StatefulWidget lebih panjang. Tetapi yang harus diperhatikan adalah properti dari setiap *class*-nya. Pada class ContohStateful propertinya hanya berupa parameter ketika memanggil ContohStateful, parameter tersebut tidak wajib ada. Sedangkan pada class \_ContohStatefulState, properti \_dataState merupakan state yang sebenarnya. Kita akan mengubah nilai \_dataState.

Misalnya kita coba membuat contoh StatefulWidget sederhana:

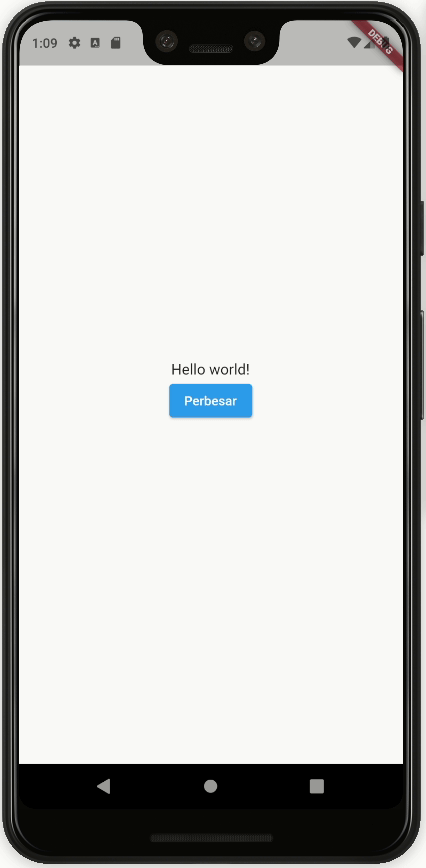
* class BiggerText extends StatefulWidget {
* final String text;
* const BiggerText({Key? key, required this.text}) : super(key: key);
* @override
* \_BiggerTextState createState() => \_BiggerTextState();
* }

* class \_BiggerTextState extends State<BiggerText> {
* double \_textSize = 16.0;
* @override
* Widget build(BuildContext context) {
* return Column(
* mainAxisAlignment: MainAxisAlignment.center,
* children: <Widget>[
* Text(widget.text, style: TextStyle(fontSize: \_textSize)),
* ElevatedButton(
* child: const Text("Perbesar"),
* onPressed: () {
* setState(() {
* \_textSize = 32.0;
* });
* },
* )
* ],
* );
* }
* }

Letakkan kode di atas setelah StatelessWidget Heading yang telah kita buat sebelumnya lalu panggil StatefulWidget BiggerText pada MyApp.

* import 'package:flutter/material.dart';
* void main() => runApp(MyApp());
* class MyApp extends StatelessWidget {
* @override
* Widget build(BuildContext context) {
* return MaterialApp(
* title: 'Flutter Demo',
* theme: ThemeData(
* primarySwatch: Colors.blue,
* ),
* home: const Scaffold(
* body: Center(
* **child: BiggerText(text:"Hello world!"), // Ubah widget Heading ke PerubahanText**
* ),
* ),
* );
* }
* }
* class Heading extends StatelessWidget {
* final String text;
* const Heading({Key? key, required this.text}) : super(key: key);
* @override
* Widget build(BuildContext context) {
* return Text(
* text,
* style: const TextStyle(
* fontSize: 24.0,
* fontWeight: FontWeight.bold,
* ),
* );
* }
* }
* **class BiggerText extends StatefulWidget {**
* **final String text;**
* **const BiggerText({Key? key, required this.text}) : super(key: key);**
* **@override**
* **\_BiggerTextState createState() => \_BiggerTextState();**
* **}**
* **class \_BiggerTextState extends State<BiggerText> {**
* **double \_textSize = 16.0;**
* **@override**
* **Widget build(BuildContext context) {**
* **return Column(**
* **mainAxisAlignment: MainAxisAlignment.center,**
* **children: <Widget>[**
* **Text(widget.text, style: TextStyle(fontSize: \_textSize)),**
* **ElevatedButton(**
* **child: const Text("Perbesar"),**
* **onPressed: () {**
* **setState(() {**
* **\_textSize = 32.0;**
* **});**
* **},**
* **)**
* **],**
* **);**
* **}**
* **}**

Maka hasilnya akan seperti berikut:



Ketika tombol "Perbesar" ditekan, text "Hello world!" akan membesar karena state \_textSize diubah nilainya. Mengubah nilai state harus dilakukan pada fungsi setState seperti berikut:

* setState(() {
* \_textSize = 32.0; // ukuran text diubah menjadi 32
* });

Anda dapat memahami lebih dalam terkait Stateless dan Stateful Widget dengan membaca dokumentasi berikut ini:

* [StatelessWidget Class](https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html)
* [StatefulWidget Class](https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html)

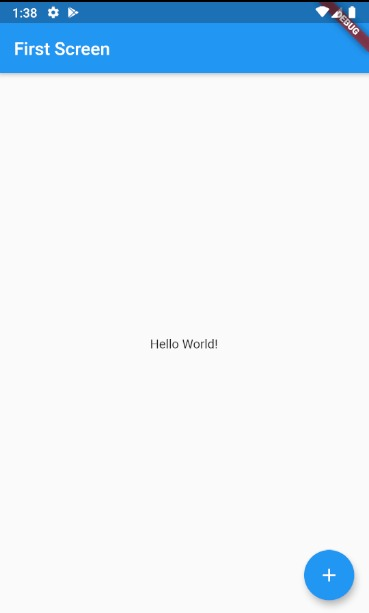
## **Widget - Widget Umum**

Setelah sebelumnya kita belajar mengenai StatelessWidget dan StatefulWidget, kali ini kita akan belajar widget-widget yang umum digunakan dalam pengembangan aplikasi Flutter. Dengan widget-widget tersebut, kita akan belajar bagaimana membuat sebuah tampilan (material design), layout sederhana, button, serta input control.

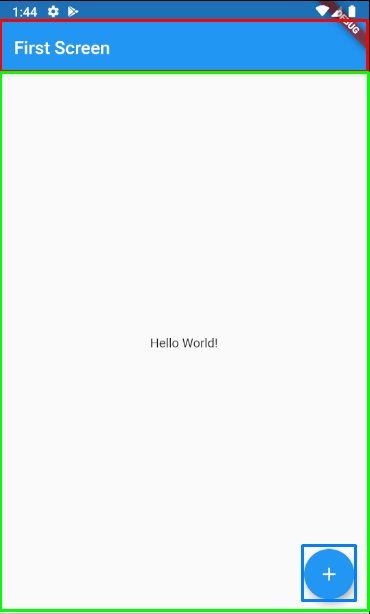
Penasaran seperti bagaimana rupa widget-widget pada Flutter? Yuk, lanjut ke materi berikutnya.

## **Scaffold**

*Scaffold* merupakan sebuah widget yang digunakan untuk membuat tampilan dasar *material design* pada aplikasi Flutter, yang dapat disebut juga dasar sebuah halaman pada aplikasi Flutter. Tampilan dasar tersebut seperti berikut:



Tampilan di atas merupakan implementasi dari Scaffold. Scaffold di atas memiliki 3 bagian yaitu **AppBar**, **Body**, dan **FloatingActionButton**. Ketiga bagian tersebut diilustrasikan seperti berikut:



Pada gambar di atas kotak berwarna merah merupakan AppBar; kotak berwarna hijau merupakan body; dan kotak berwarna biru merupakan FloatingActionButton.

Untuk membuat sebuah Scaffold kita hanya cukup memanggil class Scaffold seperti berikut:

* class FirstScreen extends StatelessWidget {
* const FirstScreen({Key? key}) : super(key: key);
* @override
* Widget build(BuildContext context) {
* return Scaffold();
* }
* }

Pada kode di atas kita membuat sebuah StatelessWidget bernama FirstScreen, yang merupakan widget tampilan kita. Kemudian di dalam method build kita panggil Scaffold.

Jangan lupa untuk memanggil FirstScreen pada Widget MyApp seperti berikut:

* 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 MaterialApp(
* title: 'Flutter Demo',
* theme: ThemeData(
* primarySwatch: Colors.blue,
* ),
* **home: const FirstScreen(),// Panggil FirstScreen di sini**
* );
* }
* }
* class FirstScreen extends StatelessWidget {
* const FirstScreen({Key? key}) : super(key: key);
* @override
* Widget build(BuildContext context) {
* return Scaffold();
* }
* }

Ketika kita menjalankan aplikasi Flutter, pada layar akan hanya muncul tampilan berwarna putih.



### **AppBar**

Setelah kita membuat Scaffold pertama kita pada Widget FirstScreen, sekarang kita akan menambahkan AppBar pada Scaffold. Seperti yang kita tahu AppBar merupakan Header (bagian paling atas) aplikasi atau biasa dikenal dengan toolbar. Pada AppBar umumnya terdapat judul dan ActionButton.

Berikut adalah cara menambahkan AppBar pada Scaffold:

* class FirstScreen extends StatelessWidget {
* const FirstScreen({Key? key}) : super(key: key);
* @override
* Widget build(BuildContext context) {
* return Scaffold(
* **appBar: AppBar(**
* **title: const Text('First Screen'),**
* **),**
* );
* }
* }

Pada kode di atas kita menambahkan parameter appBar pada Scaffold dan menambahkan title pada AppBar tersebut. Title di sini tidak hanya spesifik Text saja, melainkan juga dapat diisi dengan widget lainnya seperti TextField untuk kolom pencarian atau yang lainnya. Setelah menambahkan kode di atas, coba *refresh* atau *hot reload* aplikasi Flutter Anda. Selain menambahkan title kita dapat menambahkan widget-widget actions seperti pada kode berikut:

* class FirstScreen extends StatelessWidget {
* const FirstScreen({Key? key}) : super(key: key);
* @override
* Widget build(BuildContext context) {
* return Scaffold(
* appBar: AppBar(
* title: const Text('First Screen'),
* **actions: <Widget>[**
* **IconButton(**
* **icon: const Icon(**
* **Icons.search,**
* **color: Colors.white,**
* **),**
* **onPressed: () {},**
* **),**
* **],**
* ),
* );
* }
* }

Pada kode di atas kita menambahkan Icon search pada bagian kanan AppBar. Lalu kita juga dapat menambahkan action pada bagian kiri AppBar misalnya untuk tombol yang menampilkan menu (drawer), seperti pada kode berikut:

* class FirstScreen extends StatelessWidget {
* const FirstScreen({Key? key}) : super(key: key);
* @override
* Widget build(BuildContext context) {
* return Scaffold(
* appBar: AppBar(
* title: const Text('First Screen'),
* actions: [
* IconButton(
* icon: const Icon(
* Icons.search,
* color: Colors.white,
* ),
* onPressed: () {},
* ),
* ],
* **leading: IconButton(**
* **icon: const Icon(**
* **Icons.menu,**
* **color: Colors.white,**
* **),**
* **onPressed: () {},**
* **),**
* ),
* );
* }
* }

Tidak seperti pada actions, leading hanya dapat menampung satu widget saja. Secara *default*, *leading* akan berisi tombol untuk kembali ke halaman sebelumnya (jika tersedia), atau tombol untuk menu drawer (jika kita mengatur untuk drawer pada Scaffold tersebut). Untuk melihat hasilnya lakukan refresh atau hot reload pada aplikasi Flutter Anda.



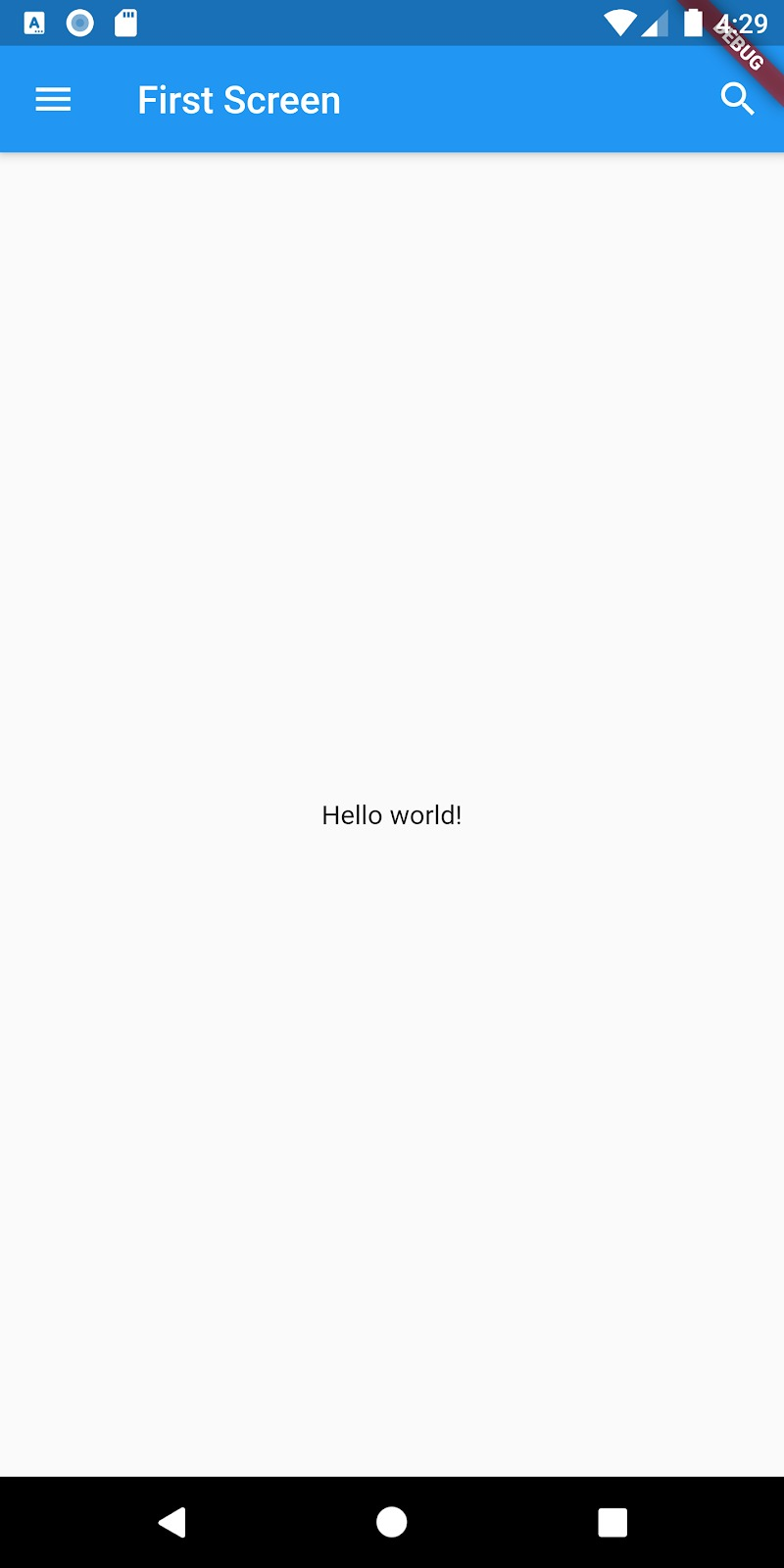
### 

### **Body**

Setelah menambahkan AppBar kita akan menambahkan *body*. Seperti pada ilustrasi sebelumnya, body merupakan bagian utama dari Scaffold dan kita akan banyak menuliskan kode pada bagian body ini. Untuk implementasi body kita akan menambahkan parameter body pada Scaffold seperti berikut:

* class FirstScreen extends StatelessWidget {
* const FirstScreen({Key? key}) : super(key: key);
* @override
* Widget build(BuildContext context) {
* return Scaffold(
* appBar: AppBar(
* title: const Text('First Screen'),
* actions: [
* IconButton(
* icon: const Icon(
* Icons.search,
* color: Colors.white,
* ),
* onPressed: () {},
* ),
* ],
* leading: IconButton(
* icon: const Icon(
* Icons.menu,
* color: Colors.white,
* ),
* onPressed: () {},
* ),
* ),
* **body: const Center(**
* **child: Text('Hello world!'),**
* **),**
* );
* }
* }

Pada kode di atas kita telah menambahkan body yang di dalamnya kita memanggil widget Center yang akan menampilkan Text "Hello World!".



### **FloatingActionButton**

Selanjutnya, kita akan menambahkan sebuah tombol bulat pada bagian kanan bawah seperti ilustrasi sebelumnya yaitu *FloatingActionButton*. FloatingActionButton ini merupakan bagian dari Scaffold yang digunakan untuk menampilkan sebuah tombol aksi yang posisinya *floating* (melayang dan posisinya tetap). Untuk menggunakan FloatingActionButton tambahkan kode Anda seperti berikut:

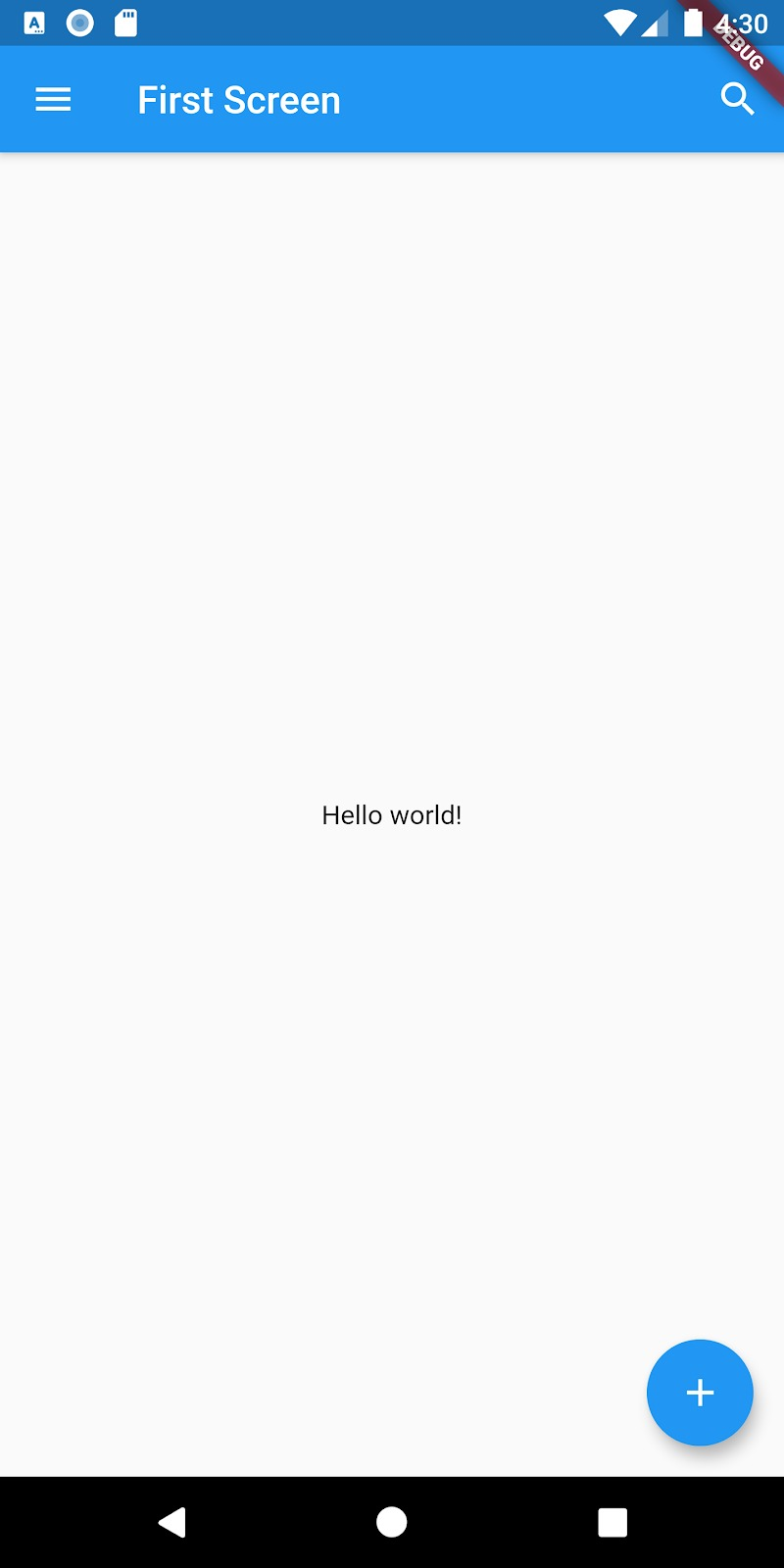
* class FirstScreen extends StatelessWidget {
* const FirstScreen({Key? key}) : super(key: key);
* @override
* Widget build(BuildContext context) {
* return Scaffold(
* appBar: AppBar(
* title: const Text('First Screen'),
* actions: [
* IconButton(
* icon: const Icon(
* Icons.search,
* color: Colors.white,
* ),
* onPressed: () {},
* ),
* ],
* leading: IconButton(
* icon: const Icon(
* Icons.menu,
* color: Colors.white,
* ),
* onPressed: () {},
* ),
* ),
* body: const Center(
* child: Text('Hello world!'),
* ),
* **floatingActionButton: FloatingActionButton(**
* **child: const Icon(Icons.add),**
* **onPressed: () {},**
* **),**
* );
* }
* }

### 

### 

### **Hasil Akhir**

Setelah kita menambahkan AppBar, body, dan FloatingActionButton maka hasil akhirnya akan seperti berikut:



Untuk memahami Scaffold lebih dalam, Anda bisa membaca tautan berikut:

* [Scaffold Class](https://api.flutter.dev/flutter/material/Scaffold-class.html)
* [Scaffold Sample Apps](https://flutter.dev/docs/catalog/samples/Scaffold)

## 

## **Pengenalan Container**

Bagaimana sejauh ini? Semoga materinya dapat Anda praktikkan dengan mulus ya.

Pada bagian sebelumnya kita mempelajari widget Scaffold. Kini saatnya belajar tentang widget Container. Container adalah widget yang digunakan untuk melakukan *styling*, membuat sebuah *shape* (bentuk), dan *layout* pada widget *child*-nya. Sebagai contoh:

* Container(
* color: Colors.blue,
* child: const Text(
* 'Hi',
* style: TextStyle(fontSize: 40),
* ),
* ),

Pada kode di atas kita membuat sebuah Text "Hi" yang dibungkus oleh widget Container dan kita beri parameter color dengan nilai Colors.blue. Kita letakkan Container di dalam parameter body **Scaffold**. Apa hasilnya? Text "Hi" akan memiliki background berwarna biru. Jalankan project Anda untuk menampilkan hasil seperti berikut:  


### **Width & Height**

Kita dapat mengatur lebar (*width*) dan tinggi (*height*) suatu Container seperti berikut:

* Container(
* color: Colors.blue,
* width: 200,
* height: 100,
* child: const Text(
* 'Hi',
* style: TextStyle(fontSize: 40),
* ),
* ),

Kode di atas ketika dijalankan hasilnya akan seperti berikut:  


### **Padding & Margin**

Container menyediakan padding & margin. Padding merupakan jarak antara konten (*child*) dengan Container, sedangkan margin merupakan jarak antara Container dengan bagian luar container.

Penggunaan padding adalah seperti berikut:

* Container(
* color: Colors.blue,
* padding: const EdgeInsets.all(10),
* child: const Text(
* 'Hi',
* style: TextStyle(fontSize: 40),
* ),
* ),

Pada kode di atas kita menambahkan padding pada semua sisi container secara merata dengan nilai 10. Maka jika me-*refresh* aplikasi flutter, akan ada jarak antara Text "Hi" dengan batas (*border*) dari container.  


Lalu penggunaan margin pun sama seperti halnya padding, maka contoh kodenya seperti berikut:

* Container(
* color: Colors.blue,
* margin: const EdgeInsets.all(10),
* child: const Text(
* 'Hi',
* style: TextStyle(fontSize: 40),
* ),
* ),

Maka hasil dari kode di atas Container akan bergeser lebih ke dalam karena ada jarak antara Container dengan bagian luar Container.



## **Dekorasi Container**

Decoration merupakan bagian dari Container untuk *styling*. Pada decoration kita dapat menentukan warna *background* (*solid/gradient color*), *shadow*, *border*, *border radius* (membulatkan sudut), mengatur *shape* (bentuk), dan lain-lain.

### **Color**

Contoh menentukan warna background dari container dengan decoration seperti berikut:

* Container(
* decoration: BoxDecoration(
* **color: Colors.red,**
* ),
* child: const Text(
* 'Hi',
* style: TextStyle(fontSize: 40),
* ),
* ),

Ketika dijalankan maka tampilan aplikasi akan seperti berikut:  


Untuk menggunakan decoration cukup menambahkan parameter decoration pada Container lalu beri nilai BoxDecoration. Pada contoh di atas kita merubah warna Container menjadi merah dengan memberi parameter color pada BoxDecoration. Ada catatan penting ketika menggunakan color pada BoxDecoration, yaitu pastikan tidak memberi parameter color pada Container.

### **Shape**

Contoh selanjutnya pada decoration adalah kita akan mengatur shape (bentuk) dari Container, contohnya sebagai berikut:

* Container(
* decoration: BoxDecoration(
* color: Colors.red,
* **shape: BoxShape.circle,**
* ),
* child: const Text(
* 'Hi',
* style: TextStyle(fontSize: 40),
* ),
* ),

Pada kode di atas kita menambahkan parameter shape dengan nilai BoxShape.circle. Artinya, bentuk dari Container tersebut akan berbentuk lingkaran. BoxShape memiliki opsi *circle* atau *rectangle*.  


### **Shadow**

Untuk menambahkan shadow pada Container kita akan menambahkan parameter boxShadow pada BoxDecoration, seperti berikut:

* Container(
* decoration: BoxDecoration(
* color: Colors.red,
* **boxShadow: const [**
* **BoxShadow(**
* **color: Colors.black,**
* **offset: Offset(3, 6),**
* **blurRadius: 10,**
* **),**
* **],**
* ),
* child: const Text(
* 'Hi',
* style: TextStyle(fontSize: 40),
* ),
* ),

Pada kode di atas parameter boxShadow merupakan sebuah Array. Di dalamnya terdapat BoxShadow yang artinya pada Container kita dapat memberikan banyak bayangan atau shadow.  


### **Border**

Border merupakan batas garis dengan content (child). Begini cara menambahkan border pada container:

* Container(
* decoration: BoxDecoration(
* color: Colors.red,
* **border: Border.all(color: Colors.green, width: 3),**
* ),
* child: const Text(
* 'Hi',
* style: TextStyle(fontSize: 40),
* ),
* ),



Apabila Anda ingin membuat border yang tidak berujung lancip cukup tambahkan parameter borderRadius Pada BoxDecoration seperti berikut:

* Container(
* decoration: BoxDecoration(
* color: Colors.red,
* border: Border.all(color: Colors.green,width: 3),
* **borderRadius: BorderRadius.circular(10),**
* ),
* child: const Text(
* 'Hi',
* style: TextStyle(fontSize: 40),
* ),
* ),



### **Kesimpulan**

Dengan menggunakan Widget Container kita dapat membuat variasi widget yang kita buat. Sebenarnya banyak sekali parameter-parameter yang dapat digunakan pada Container dan juga pada BoxDecoration. Anda dapat mengeksplorasi hal tersebut dengan mencarinya di Google atau pada dokumentasi resmi flutter.

* [Container Class](https://api.flutter.dev/flutter/widgets/Container-class.html)

## **Padding**

Sebelumnya kita telah belajar banyak hal mengenai Container. Kali ini kita akan belajar widget Padding. Seperti namanya widget Padding merupakan sebuah widget yang khusus untuk memberikan padding pada suatu widget.

Contoh penggunaan widget Padding seperti berikut:

* Padding(
* padding: const EdgeInsets.all(30),
* child: const Text('Ini Padding')
* )

Pada kode di atas widget Padding harus memiliki child. Child di sini merupakan sebuah widget yang nantinya akan diberi padding. Parameter padding ditambahkan untuk menentukan besaran padding yang diinginkan.

Anda dapat membaca informasi detail tentang Padding pada dokumentasi berikut:

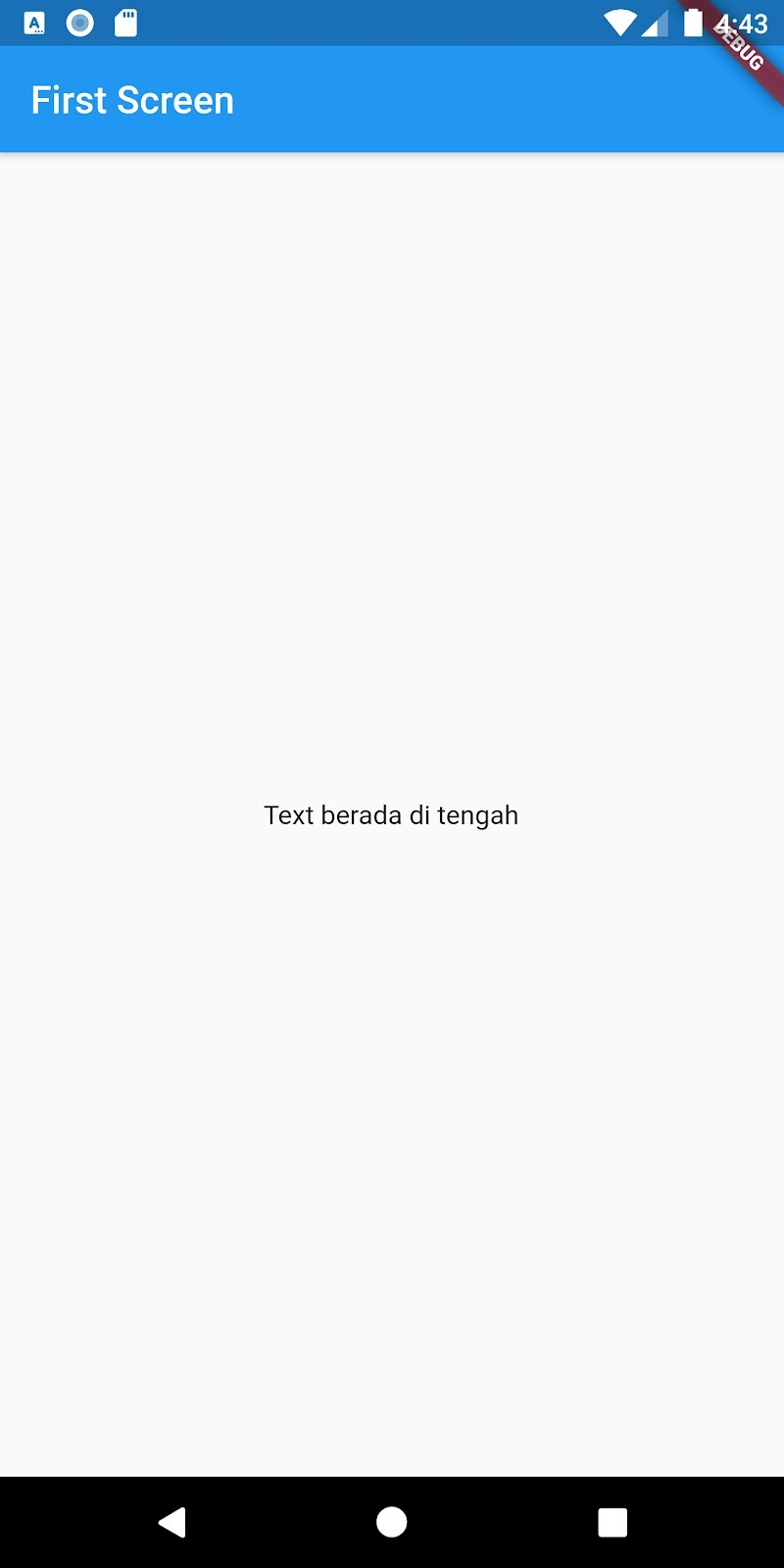
* [Padding Class](https://api.flutter.dev/flutter/widgets/Padding-class.html)

## **Center**

Widget Center merupakan sebuah widget yang digunakan untuk membuat suatu widget berada pada posisi tengah. Penggunaan widget Center sangatlah simpel, yakni seperti berikut:

* Center(
* child: const Text('Text berada di tengah'),
* )

Widget Center hanya membutuhkan parameter child untuk membuat widget di dalamnya berada pada posisi tengah. Hasil dari Center seperti berikut:



## 

## **Row dan Column**

Selanjutnya kita akan mempelajari bagaimana cara membuat widget yang kita gunakan berjajar secara vertikal atau horizontal. Lalu apa yang dimaksud dengan membuat widget yang berjajar? Perhatikan gambar berikut:  


Pada gambar di atas kita memiliki tampilan ikon-ikon yang merupakan kumpulan tombol, di antaranya *share*, *like* dan *dislike*. Tombol-tombol tersebut tersusun berjajar secara horizontal. Nah, untuk membuat berjajar horizontal atau membentuk baris kita menggunakan widget Row. Sedangkan untuk menyusun widget yang membentuk kolom atau vertikal, kita bisa menggunakan widget Column.

### 

### 

### **Widget Row**

Seperti yang dicontohkan sebelumnya, widget Row merupakan suatu widget yang digunakan untuk membuat widget-widget tersusun berjajar secara horizontal. Row memiliki sintaks seperti berikut:

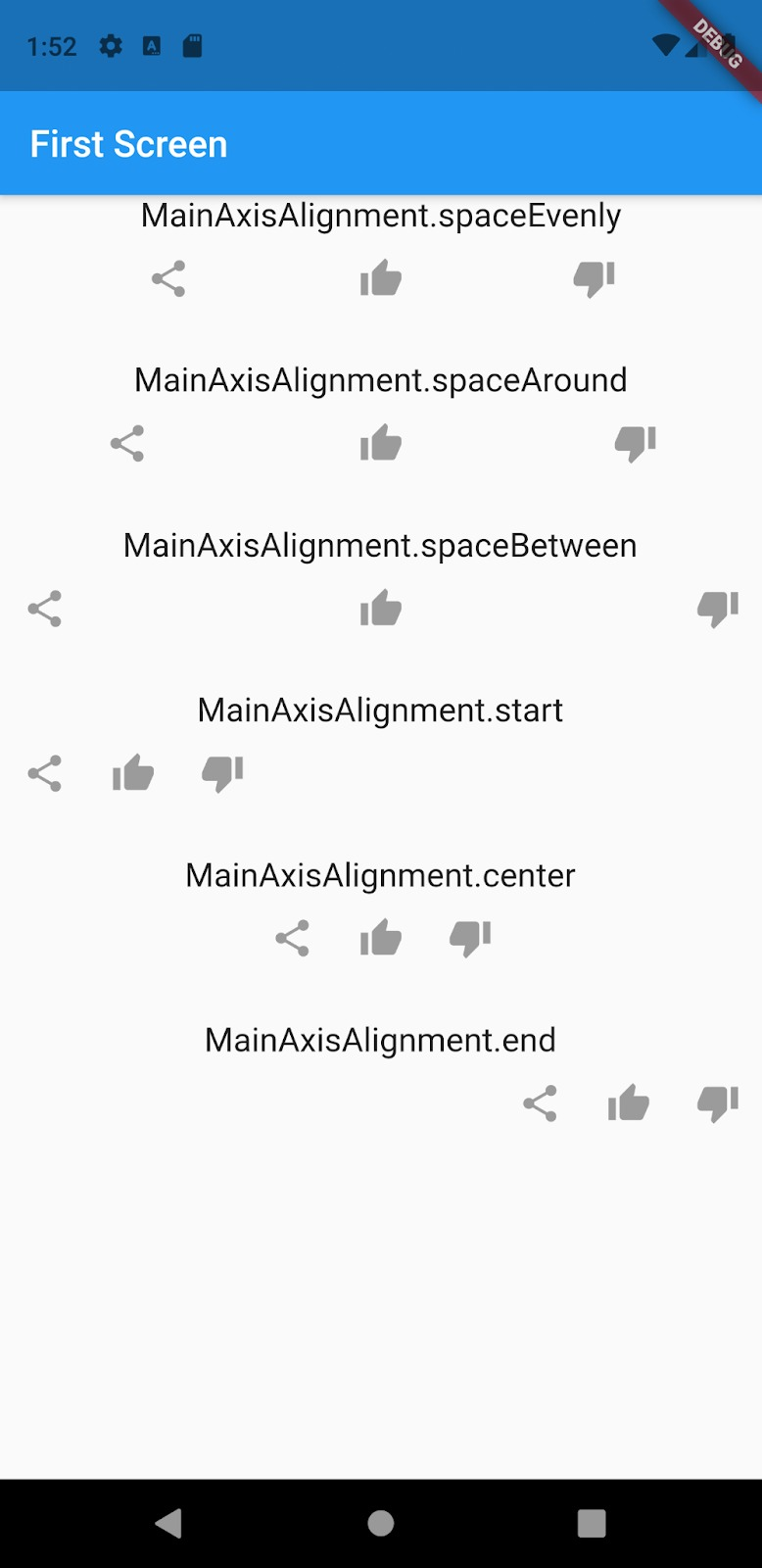
* Row(
* children: <Widget>[
* //di sini berisi widget-widget
* ],
* )

Untuk membuat widget-widget berjajar secara horizontal kita harus memasukkan widget-widget tersebut ke dalam parameter children. Parameter children berisi kumpulan atau list dari widget karena kita dapat menyusun beberapa widget sekaligus di dalamnya. Jika mengacu pada contoh tombol-tombol di atas kodenya seperti berikut:

* Row(
* mainAxisAlignment: MainAxisAlignment.spaceEvenly,
* children: const <Widget>[
* Icon(Icons.share),
* Icon(Icons.thumb\_up),
* Icon(Icons.thumb\_down),
* ],
* )

Seperti yang kita lihat, kita membuat sebuah IconButton berada di dalam parameter children. Kita menambahkan pula mainAxisAlignment yang merupakan parameter *alignment* pada Row. Parameter mainAxisAlignment yang berfungsi untuk mengatur *alignment* horizontal dari Row (*alignment* utama). Selain itu Row juga memiliki parameter *crossAxisAlignment* yang berfungsi untuk mengatur *alignment* secara vertikal. Kedua parameter ini juga berlaku sebaliknya untuk widget Column.  


Berikut ini adalah contoh penerapan mainAxisAlignment pada Row:



### **Widget Column**

Kebalikan dari Row, Column merupakan suatu widget yang digunakan untuk membuat widget-widget tersusun berjajar secara vertikal. Column memiliki sintaks mirip dengan Row, seperti berikut:

* Column(
* children: <Widget>[
* //di sini berisi widget-widget
* ]
* )

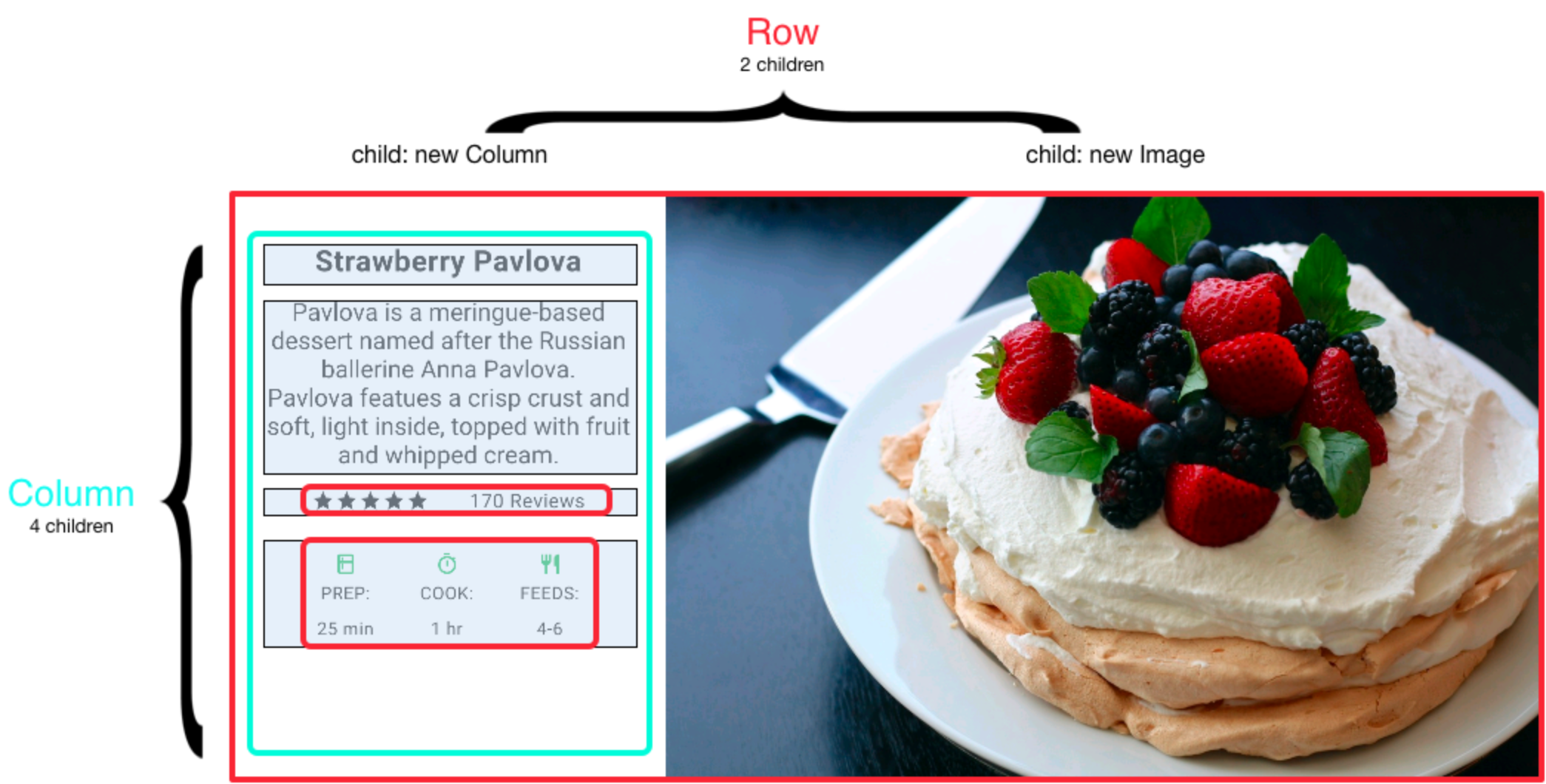
Contoh penerapan Column seperti berikut:

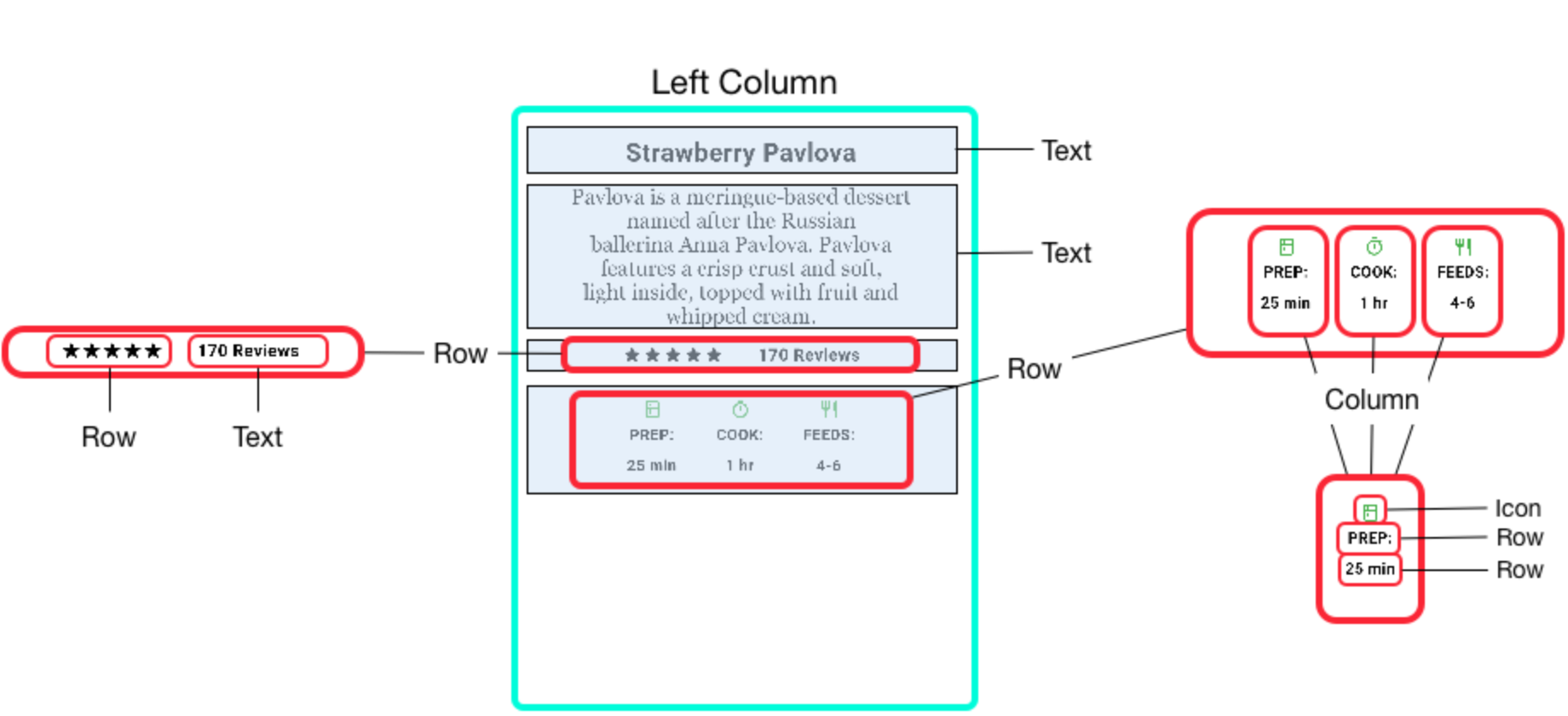
* Column(
* children: const <Widget>[
* Text(
* 'Sebuah Judul',
* style: TextStyle(fontSize: 32, fontWeight: FontWeight.bold),
* ),
* Text('Lorem ipsum dolor sit amet'),
* ],
* )

Maka akan menghasilkan tampilan seperti berikut:  


### **Kesimpulan**

Untuk membuat sebuah widget-widget berjajar kita dapat menggunakan widget *Row* atau *Column*. Sebenarnya penggunaan Row dan Column dapat dipadukan sehingga dapat membuat sebuah layout yang kompleks seperti berikut:





Untuk memahami Row, Column, dan bagaimana menyusun layout dengan Flutter secara mendalam, silakan pelajari dokumentasi berikut:

* [Row Class](https://api.flutter.dev/flutter/widgets/Row-class.html)
* [Column Class](https://api.flutter.dev/flutter/widgets/Column-class.html)
* [Layouts in Flutter](https://flutter.dev/docs/development/ui/layout)