**PEMROGRAMAN MOBILE LANJUT**

**RESUME BUKU**



Dosen Pembimbing:

**Yoga Sahria, S.Kom., M.Kom.**

Disusun Oleh:

**Muhammad Ilham Kusumawardhana**

**5180411151**

**PROGRAM STUDI INFORMATIKA**

**FAKULTAS SAINS & TEKNOLOGI**

**UNIVERSITAS TEKNOLOGI YOGYAKARTA**

**YOGYAKARTA**

**2021**

**JUDUL BUKU : Enjoy Flutter**

**Penulis : - Yusmi Putra Wiraganda**

**- Ari Ramadhani**

**- Rizki Syaputra**

**Penerbit : Udaconding**

** **

1. **Widget Dasar Flutter**
2. Text Widget

Adalah widget yang diunakan untuk menampilkan text ke dalam view / tampilan.

Sintaks dasar :

**Text(‘Halo’)**

Struktur Text Widget

Text widget dibagi jadu dua property, yaitu **string** dan **property ,** dimana string bersifat wajib dan property bersifat opsional

**Text(String, Properti)**

Properti yang ada didalam widget text :

Key **key**,

TextStyle **style**,

StrutStyle **strutStyle**,

TextAlign **textAlign**,

TextDirection **textDirection**,

Locale **locale**,

bool **softWrap**,

TextOverflow **overflow**,

double **textScaleFactor**,

int **maxLines**,

String **semanticsLabel**,

TextWidthBasis **textWidthBasis**,

TextHeightBehavior **textHeightBehavior**

1. Container Widget

Widget ini berguna untuk menambahkan padding, margin, boderdder, dan atau warna background pada suatu widget, maka fungsi container class pada flutter widget.

Container merupakan **single child widget** yang berarti hanya dapat memiliki satu buah child widhet saja, akan tetapi kita bisa menggunakan **row**,**column**, **text**, atau bahkan container lain.

Fungsi property yang ada pada container widget

* **Child**

child digunakan untuk membuat anak atau turunan dari widget, contoh container yang menggunakan child :

**body: Container(**

**child: Text(**

**'Ayo Belajar Flutter',**

**)**

**),**

* **Alignment**

Alignment digunakan untuk mengatur posisi child widget, property alignment diantaranya seperti :

**bottomCenter** untuk memindahakan kebawah bagian tengah.

**bottomLeft** untuk memindahkan ke bawah bagian Kiri.

**bottomRight** untuk memindahkan ke bawah bagian Kanan.

**center** untuk untuk memindahkan ke posisi tengah.

**centerLeft** untuk memindahkan ke tengah bagian Kiri.

**centerRight** untuk memindahkan ke tengah bagian Kanan.

**topCente**r untuk memindahkan ke atas bagian Tengah.

**topLeft** untuk memindahkan ke atas bagian Kiri.

**topRight** untuk memindahkan ke atas bagian Kanan.

Contoh penggunaan alignment :

**body: Container(**

**alignment: Alignment.bottomCenter,**

**child: Text(**

**'Halo',**

**style: TextStyle(**

**fontSize: 15,**

**),**

**)**

**),**

* **Color**

Propoerti ini membuat wadah / container memiliki warna latar belakang.

Contoh penggunaan color :

**body: Container(**

**alignment: Alignment.center,**

**color: Colors.blue,**

**child: Text(**

**'Halo',**

**style: TextStyle(**

**fontSize: 20,**

**color: Colors.white**

**),**

**)**

**),**

Kita juga bisa menggunakan warna custom dengan kode sebagai berikut

color: Color(0xFF42A5F5);

color: Color.fromARGB(0xFF, 0x42, 0xA5, 0xF5);

color: Color.fromARGB(255, 66, 165, 245);

color: Color.fromRGBO(66, 165, 245, 1.0);

* **Height dan Weight**

Secara default ukuran container menyesuaikan dengan body layer aplikasinya , tapi container juga bisa diatur menggunakan property **height** dan **weight.**

* **Margin**

Margin digunakan untuk membuat jarak antara tepi container dengan tepi widget lainya. Property margin bisa digunakan sekaligus pada 4 sisi wadah dengan menggunakan fungsi EdgeInsets.all().

Contoh oenggunaan margin :

**body: Container(**

**margin: EdgeInsets.all(20),**

**height: 200,**

**width: 200,**

**alignment: Alignment.topLeft,**

**color: Colors.purple,**

**child: Text(**

**'Ayo Belajar Flutter',**

**style: TextStyle(**

**fontSize: 20,**

**color: Colors.white**

**),**

**)**

**),**

Kita juga bisa memberi jarak pada 1 sisi wadah saja dengan sintaks :

**margin: EdgeInsets.only(left: 20)**

* **Padding**

Digunakan untuk menambahkan jarak antara container dengan widget yang ada dalamnya. Sama dengan margin penggunaannya bisa dengan menambahakan fungsi EdgeInsets.all() untuk mengatur ukuran padding sama di ke-empat sisinya.

Contoh penggunaan padding :

**body: Container(**

**padding: EdgeInsets.only(left:20),**

**height: 200,**

**width: 200,**

**alignment: Alignment.topLeft,**

**color: Colors.purple,**

**child: Text(**

**'Ayo Belajar Flutter',**

**style: TextStyle(**

**fontSize: 20,**

**color: Colors.white**

**),**

**)**

**),**

* **Transform**

melakukan rotasi pada wadah (container) yang dapat dilakukan dengan melakukan dari berbagai sumbu putar misalnya X,Y dan Z yang dimana kita menggunakan fungsi **Matrix4** untuk melakukan rotasinya.

Contoh penggunaan transform :

**Container(**

**margin: EdgeInsets.all(20),**

**transform: Matrix4.rotationZ(0.1),**

**height: 200,**

**width: 200,**

**color: Colors.purple,**

**),**

* **Decoration**

Yang terakhir disini Anda bisa menghias kotak Container dengan berbagai macam efek dekorasi seperti misalnya mengubah warna border, memberikan gambar, atau membuat efek bayangan pada kotak containernya.

1. **Image Widget**

* **Gambar dari asset local**

Pada flutter kita bisa menambahkan gambar memalui asset local atau gambar yang kita simpan didalam folder projek.

Cara menampilkan gambar dari asset local :

1. Buat Folder baru assets/gambar
2. Masukan Gambar kedalam folder tersebut
3. Daftarkan gambar yang kita masukan tadi ke pubspec.yaml

Lalu masukan gambar yang sudah didaftarkan tadi menggunakan sintaks image.asset(). Contoh :

**Widget build(BuildContext context) {**

**return MaterialApp(**

**home: Scaffold(**

**appBar: AppBar(**

**title: Text("Flutter"),**

**),**

**body: Image.asset('assets/images/Flutter.jpg'),**

**)**

**);**

**}**

Format gambar yang disupport oleh flutter sebagai berikut : JPEG, WebP, GIF, animated WebP/GIF, PNG, BMP, and WBMP

* **Gambar Dari internet**

untuk menampilkan gambar dari internet lebih mudah dari pada cara sebelumnya , karena hanya memerlukan sintaks image.networj(url); , namun menggunakan network image punya beberapa kekurangan yang harus dipertimbangkan yaitu :

untuk me load gambar memerlukan akses internet , dan jika kita tidak mempunyai akses internet gambar akan blank/ kosong

contoh penggunaak network image :

**Image.network('https://cdn.pixabay.com/photo/2019/11/10/17/36/indonesia-4616370\_1280.jpg');**

* **Mengatur Ukuran tinggi dan lebar**

untuk mengatur tinggi dan lebar gambar, kita dapat menggunakan properti **height** untuk tinggi dan **width untuk lebarnya .**

**contoh penggunaan nya :**

**body: Image.asset(**

**'assets/images/paddy-field.jpg',**

**height: 100,**

**width: 200,**

**),**

* **Manipulasi Warna Gambar**

kita juga dapat memanipulasi warna gambar dengan properti **colorBlendMode** penggunaan properti ini biasanya digabungkan dengan properti **color**.

Contoh oenggunaan colorBlend :

**body: Column(children: <Widget>[**

**Image.asset('assets/images/Flutter.jpg',**

**color: Colors.grey,**

**colorBlendMode: BlendMode.hue),**

**]),**

* **Properti Image widget lainya**

|  |  |
| --- | --- |
| **Properti** | **Kegunaan** |
| Contain | Untuk membuat gambar agar memuat memenuhi sesuai dengan ukuran Box |
| Cover | Untuk membuat gambar memenuhi keseluruhan dari box / layar aplikasi baik itu untuk ukuran ketinggian dan juga lebarnya. Apabila gambar memiliki ukuran lebar lebih besar maka gambar akan menjadi ditarik memanjang keatas dan ke samping dengan ukuran yang sama besar memenuhi tinggi aplikasi. |
| Fill | Fungsi ini sama seperti cover yaitu untuk memenuhi gambar ke bagian tinggi dan lebar aplikasi namun perbedaanya disini adalah penggunaan fill akan merubah presisi gambar yang artinya gambar bisa saja memanjang atau melebar untuk memenuhi layar aplikasinya. |
| fitHeight | Jika pada fungsi fill keseluruhan tinggi dan lebar gambar yang akan memenuhi layar, penggunaan fitHeight berguna untuk memenuhi layar hanya pada tinggi gambar saja |
| fitWidth | Untuk yang ini gambar akan memenuhi layar pada lebar gambarnya saja. |

* **Border Radius**

kita akan mencoba dua metode untuk memanipulasi bentuk gambar mejadi lingkaran yaitu dengan widget ClipOval dan widget  BoxDecoration.

1. BoxDecoration

**body: Container(**

**height: 300,**

**width: 300,**

**decoration: BoxDecoration(**

**image: DecorationImage(**

**image: AssetImage('assets/images/paddy-field.jpg'),**

**fit: BoxFit.cover**

**),**

**shape: BoxShape.circle**

**)**

**),**

1. ClipOval

**body: ClipOval(**

**child: Image(**

**width: 300,**

**height: 300,**

**image: AssetImage('assets/images/paddy-field.jpg'),**

**fit: BoxFit.cover),**

**),**

1. **Layouting**

Row widget adalah widget yang digunakan untuk memposisikan widget-widget lainnya secara horizontal. Sedangkan column widget digunakan untuk menempatkan widget secara menurun atau vertical. Jika anda terbiasa dengan XML android maka Row dan column seperti LinearLayout.

Row dan Column merupakan multiple children widget atau yang yang dapat memiliki lebih dari satu widget.

* **Penggunaan Row dan Column**

Mesikpun row widget menempatkan childrennya secara horizontal namun widget ini **tidak** bisa scrolling. Jadi apabila row widget telah melebihi ukuran layar akan muncul error overflow. Jika ingin membuat layout horizontal yang dapat di scroll maka gunakan ListView.

Contoh Row :

**return Scaffold(**

**appBar: AppBar(**

**title: Text("Belajar Layout"),**

**),**

**body: Row(**

**children: [**

**Container(**

**color: Colors.greenAccent,**

**child: FlutterLogo(**

**size: 60.0,**

**),**

**),**

**Container(**

**color: Colors.orangeAccent,**

**child: FlutterLogo(**

**size: 60.0,**

**),**

**),**

**Container(**

**color: Colors.purpleAccent,**

**child: FlutterLogo(**

**size: 60.0,**

**),**

**),**

**],**

**),**

**);**

* Contoh column widget body:

**Column(**

**children: [**

**Container(**

**color: Colors.greenAccent,**

**child: FlutterLogo(**

**size: 60.0,**

**),**

**),**

**Container(**

**color: Colors.orangeAccent,**

**child: FlutterLogo(**

**size: 60.0,**

**),**

**),**

**Container(**

**color: Colors.purpleAccent,**

**child: FlutterLogo(**

**size: 60.0,**

**),**

**),**

**],**

**),**

* **MainAxisAlignment**

properti yang dimiliki oleh row dan column yang berfungsi untuk mengatur posisi widget di dalamnya.

Contoh :

**Row(**

**mainAxisAlignment: MainAxisAlignment.spaceEvenly,**

**…..**

**Column(**

**mainAxisAlignment: MainAxisAlignment.spaceEvenly,**

**……**

Banyak pilihan pada mainAxisAlignment seperti :

|  |  |
| --- | --- |
| Value | Penjelasan |
| MainAxisAlignment.center | Memposisikan widget pada tengah layar |
| MainAxisAlignment.start | Memposisikan widget mengikuti posisi widget pertama |
| MainAxisAlignment.end | Memposisikan widget mengikuti posisi widget terakhir |
| MainAxisAlignment.spaceEvenly | Memberi jarak antara masing-masing child secara merata pada ruang yang tersedia |
| MainAxisAlignment.spaceAround | Memberi jarak antar widget secara merata namun hanya memberikan setengah jarak pada awal dan akhir widget |

* **ListView**

Membuat sebuah List widget yang dapat di scroll pada Flutter cukup mudah dengan menggunakan widget ListView.

Jenis jenis listview :

1. ListView
2. ListView.builder
3. ListView.separated
4. ListView.custom

* **GridView**

Sama seperti listview hanya saja gridview dapat di geser kanan dan ke kiri ,

Contoh sederhana penggunaan gridview :

**body: GridView.count(**

**crossAxisCount: 3,**

**children: <Widget>[**

**FlutterLogo(),**

**FlutterLogo(),**

**FlutterLogo(),**

**FlutterLogo(),**

**FlutterLogo(),**

**],**

**)**

GridView dasarnya merupakan CustomScrollView widget yang dapat di scroll (scrollable) sehingga tidak perlu khawatir apabila jumlah dari widget grid melebihi ukuran layar. Dalam GridView kita juga dapat menggunakan List

Contoh :

**class BelajarGridView extends StatelessWidget {**

**@override**

**Widget build(BuildContext context) {**

**return MaterialApp(**

**debugShowCheckedModeBanner: false,**

**home: Scaffold(**

**appBar: AppBar(**

**title: Text("belajarFlutter.com"),**

**),**

**body: GridView.count(**

**crossAxisCount: 3,**

**children: List.generate(9, (index) {**

**return Container(**

**child: Card(**

**color: Colors.deepPurpleAccent,**

**),**

**);**

**}),**

**)),**

**);**

**}**

**}**

Terdapat lima fungsi yang dapat digunakan dalam penggunaan GridView

* **GridView**

Contoh :

**GridView(**

**gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(crossAxisCount: 2),**

**children: <Widget>[**

**FlutterLogo(),**

**FlutterLogo(),**

**FlutterLogo(),**

**FlutterLogo(),**

**],**

**)**

* **GridView.count**

Contoh :

**GridView.count(**

**crossAxisCount: 2,**

**children: <Widget>[**

**FlutterLogo(),**

**FlutterLogo(),**

**FlutterLogo(),**

**FlutterLogo(),**

**],**

**)**

* **GridView .builder**

Contoh :

**GridView.builder(**

**gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(crossAxisCount: 2),**

**itemBuilder: (\_, index) => FlutterLogo(),**

**itemCount: 4,**

**)**

* **GridView .custom**

Contoh :

**GridView.custom(**

**gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(crossAxisCount: 2),**

**childrenDelegate: SliverChildListDelegate(**

**[**

**FlutterLogo(),**

**FlutterLogo(),**

**FlutterLogo(),**

**FlutterLogo(),**

**],**

**),**

**)**

* **GridView .extent**

Contoh :

**GridView.extent(**

**maxCrossAxisExtent: 400,**

**children: <Widget>[**

**FlutterLogo(),**

**FlutterLogo(),**

**FlutterLogo(),**

**FlutterLogo(),**

**],**

**)**

* **Properti Pada GridView**

|  |  |
| --- | --- |
| **Properti** | **Penjelasan** |
| crossAxisCount | Berfungsi untuk menentukan jumlah kolom (hanya tersedia untuk GridView.count) |
| gridDelegate | Untuk mendelegasikan grid item, biasanya digabungkan dengan penggunaan SliverGrid. Hanya tersedi di method costum dan builder |
| childAspectRatio | Menentukan rasio dari grid item. Contoh childAspectRatio: 16 / 9 |
| scrollDirection | Menentukan arah scroll. Secara default bernilai Axis.vertical Dapat dirubah menjadi : Axis.horizontal |
| crossAxisSpacing | Menentukan jarak spasi antar grid item secara sejajar dengan scrollDirection |
| mainAxisSpacing | Menentukan jarak spasi antar grid item secara berlawanan dengan scrollDirection |
| reverse | Membalik arah srollDirection. Bernilai **Boolean** dengan default value **false** |
| padding | Memberi jarak GridView dengan Widget lain yang bersinggungan |

* **StackWidget**

widget yang memungkinkan kita untuk membuat widget-widget saling bertumpuk.

Stack widget memungkinkan kita untuk menampilkan beberapa lapis widget ke layar. Stack widget juga merupakan [multiple children widget](https://belajarflutter.com/flutter-widgets/multiple-children-widget/) yang artinya memiliki properti **children** sehingga dapat menampung lebih dr satu widget. Urutan dari lapisan widget pada stack dari **bawah ke atas**.

Contoh penggunaan :

**Stack (**

**children: <Widget> [**

**BawahWidget(),**

**TengahWidget(),**

**AtasWidget(),**

**],**

Ukuran dari stack widget diambil dari ukuran item widget terbesarnya. Misal dalam stack widget terdapat 4 widget dimana ukuran widget A (100×100), B (150×200), C (200×200) dan D (50×50), maka ukuran Stack adalah 200×200.

Untuk mengubah posisi item dari stack widget, kita bisa menggunakan **Align**widget

**Align(**

**alignment: Alignment.topRight,**

**child: Container(**

**color: Colors.deepPurple,**

**height: 200.0,**

**width: 200.0,**

**),**

**)**

atau jika ingin lebih leluasa dalam memposisikan widget kita juga dapat menggunakan [Positioned widget](https://api.flutter.dev/flutter/widgets/Positioned-class.html).

**Positioned(**

**right: 40.0,**

**top: 100.0,**

**child: Container(**

**color: Colors.deepPurple,**

**height: 200.0,**

**width: 200.0,**

**),**

**)**

1. **Bloc Pattern**

Bloc Patter pada flutter merupakan **state management** sistem dari google untuk digunakan oleh developer mobile. Bloc ini sangan membantu dalam state untuk membuat akses data dari tempat sentral/pusat ke widget lainya dalam projek flutter.

State management yang umum dikenal pada flutter adalah **BLOC**. **RxDart**, dan **Provider**.

Bloc (Business Logic of Componen). Dimana pada state management ini antara logic dan view terpisah. Seperti MVC. Bentuk hierarki atau gambaran dari Bloc Patern ini adalah sebagai berikut:

Visible to User

Presenter/ViewModel

BLOC

UI Screen

Data Handler

Repository

Data Provider

Network Provider

BLOC tidak akan pernah memiliki refrensi tentang widget di latar UI, Layar UI hanya akan mengamati perubahan yang berasal dari BLOC.

Berikut ini flow Ketika handle data menggunakan **BLOC** pattern pada flutter:

BLOC Component

StreamBuilder

Widget

BLOC Provider

InheritedWidget

**BLOC** memiliki dua komponen sederhana: **Sinks** dan **Stream**, yang keduanya disediakan oleh StreamController. Kita dapat menambahkan stream input event / Data ke dalam **Sink** dan mendengarkannya sebagai stream output data melalui stream.

StreamController dapat diakses melalui library ‘**dart: async**’ atau sebagai PublishSubject, ReplaySubject, atau BehaviourSubject melalui paket **RxDart.**

**Implementasi BloC Patern**

1. Buat projek baru
2. Tambahkan library bloc kedalam projek, yaitu didalam file **pubspec.yaml.**

**Flutter\_bloc : ^0.20.1**

Untuk versi library flutter\_bloc dapat dilihat pada pub.dartlang.org dan selalu di cek disana karena versinya selalu update.

1. Pada saat menggunakan pattern ini, kita perlu membuat beberapa file, diantaranya

|  |  |
| --- | --- |
| Class | Penjelasan |
| Counter\_Provider.dart | Class ini berfungsi untuk proses logic menghitung secara increment. |
| Counter\_Bloc.dart | Class ini berfungsi untuk deklarasi blocnya, deklarasi strea, controller dan counter providernya. |
| Counter.dart | Class ini berfungsi sebagai view untuk menampilkan data counter. |
| Main.dart | Class ini berfungsi sebagai class utama |

1. Penjelasan sintak pada :

**Counter\_Provider.dart**

Didalam file ini terdapat class **CounterProvider** yang nantinya akan memproses bilangan dengan menambahkan bilangan dengan angka 1 setiap kali class dipanggil . menggunakan fungsi **increaseCount()**

**Counter\_bloc.dart**

Pada file ini kita akan membuat sebuah StreamController yang bernama counterController. Pada StreamController ini kita dapat menambahkan **sink** untuk input event nya. Dan juga menampilkan output.

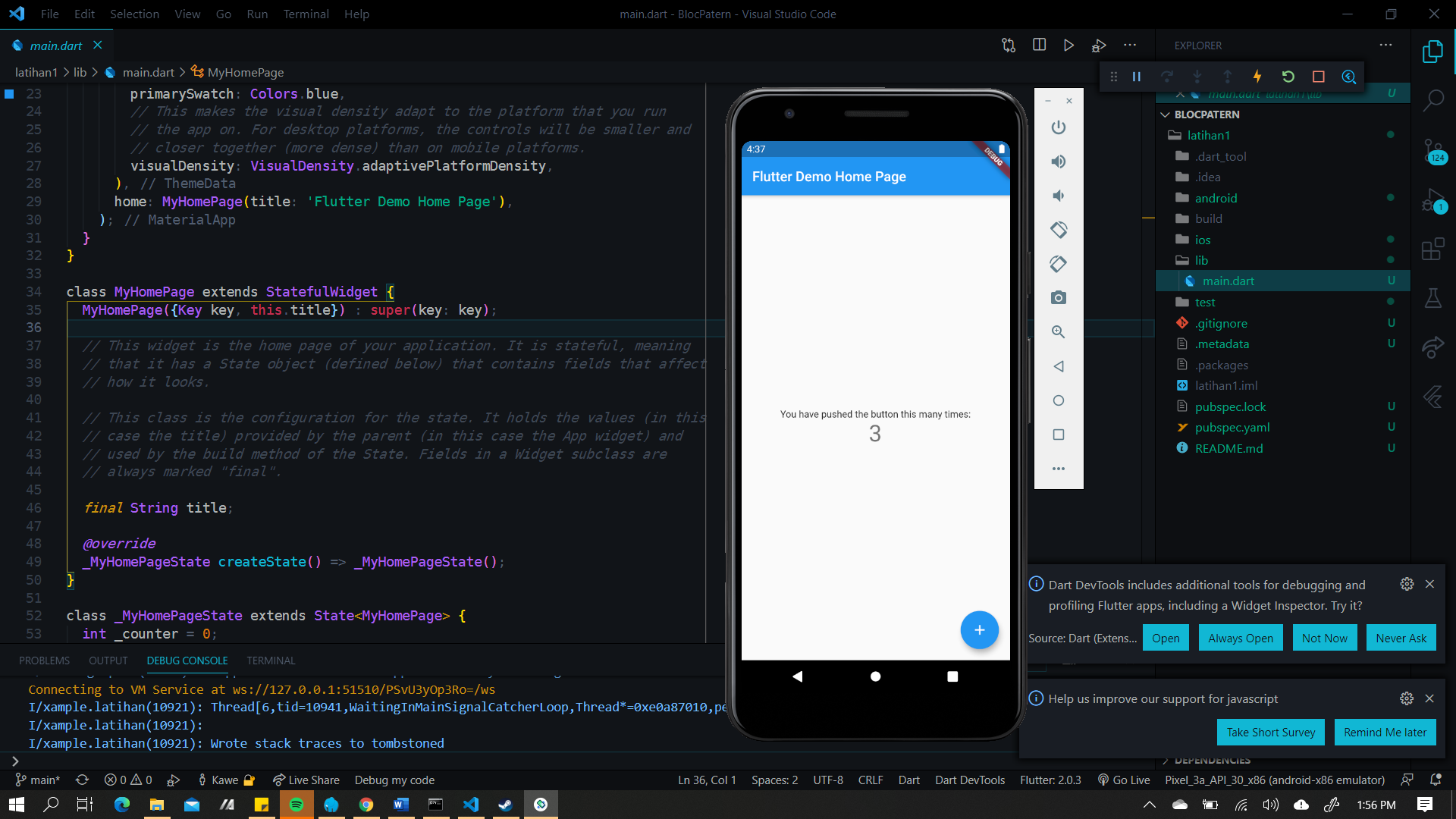
**Counter\_bloc.dart**

Pada file ini kita akan membuat class yang akan menampilkan jumlah berapa kali user menakan tombol. Untuk menampilkan data, kita menggunakan **‘${snapshop.data}’** pada widget text, Data tersebut kita dapat dari **StreamBuilder.**

**Main.dart**

Pada class ini kita akan menampilkan data yang sudah diinput melalui tombol yang ditekan kita langsung menampiikan data melalui class Counter yang dipanggil pada bagian **body : Counter()**

1. Output :



1. **RxDart Flutter**

RxDart Flutter Merupakan sebuah library reactive programming yang digunakan untuk Bahasa pemrograman google dart. RxDart sendiri berasal dari ReactiveX, ReactiveX sendiri merupakan sebuah library yang digunakan untuk program yang memiliki asynchronous dan event bus.

**Implementasi RxDart**

1. Buat Projek baru , dan setelah itu buat lah package baru dengan nama src, kemudian buat satu file dengan nama app.dart didalamnya setelah itu buat 2 folder baru dengan nama blocs dan screens, kemudian pada folder blocs buat 3 file baru dengan nama **bloc.dart**, **provider.dart**, dan **validators.dart** dan pada folder screens buat satu file dengan nama login\_screen.dart
2. Selanjutnya pada file **pubspec.yaml** kita tambahkan library untuk RxDart.

**xdart ^0.19.0**

struktur file pada folder **lib** :

Lib

|----src

|----Blocs

|----bloc.dart

|----Provider.dart

|----Validator.dart

|----Screens

|----Login\_screen.dart

|----App.dart

| Main.dart

1. Pada Main.dart kita perlu mengimport class yang ada pada src/app.dart
2. Pada class didalam app.dart kita akan menampilkan sebuah login form. Login form tersebut kita dapatkan dari class login\_screen.dart yang nantinya akan kitai import.
3. Pada class Bloc.dart kita akan membuat class yang mengexends dari class validator. Pada class ini kita akan menambahkan beberapa kedalam stream.
4. Pada class validator kita akan mengecek format email yang diimput apakah sudah benar atau belum dan juga passwordnya harus minimal 6 karakter.
5. Pada class login\_screen.dart kita hanya akan menampilkan / view data. Class provider dan bloc yang terlah dibuat sebelumnya kita tambahkan kedalam class login ini.
6. Kemudian class provider merupakan class yang menurunkan atau mewarisi widget. Pada class provider ini juga kita akan melakukan import terlebih dahulu class bloc.dart yang sudah dibuat sebelumnya. Setelah itu kita deklarasikan method bloc tersebut.

Ex: final Bloc bloc = Bloc();

Untuk menampung variable widget yang digunakan, kita menggunakan code yang dibawah ini

Provider({Key key, Widget child}) : super(key : key, child : child);