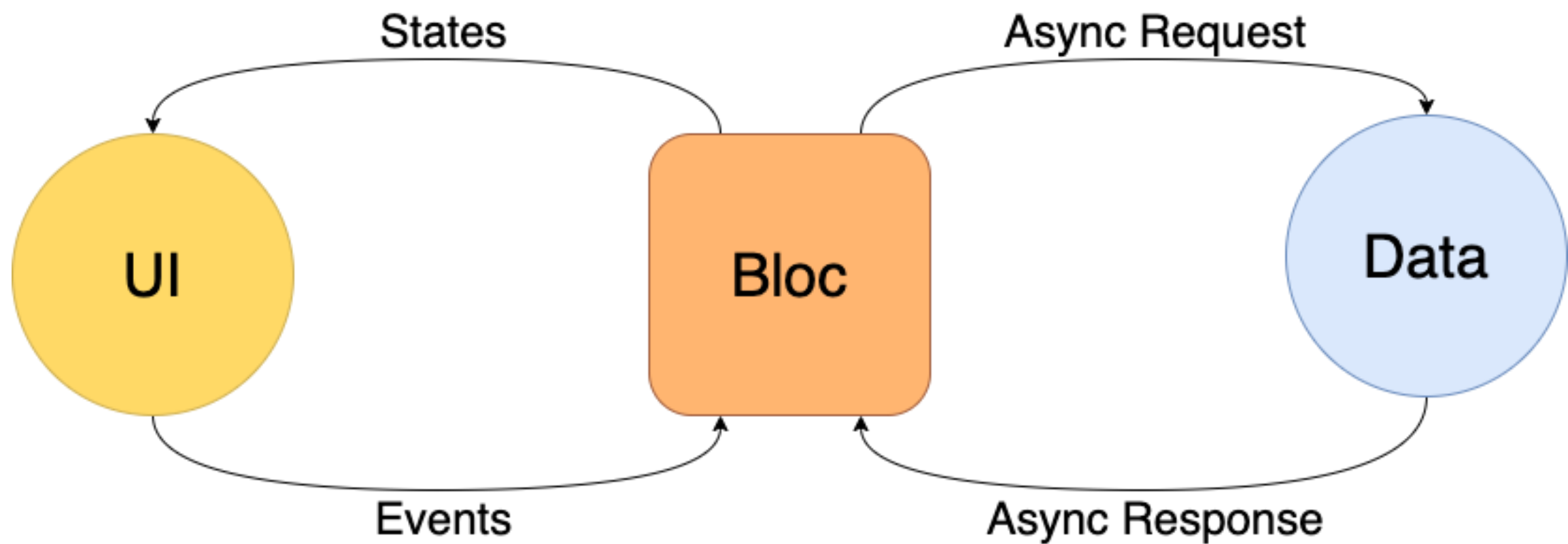


Architecture patterns

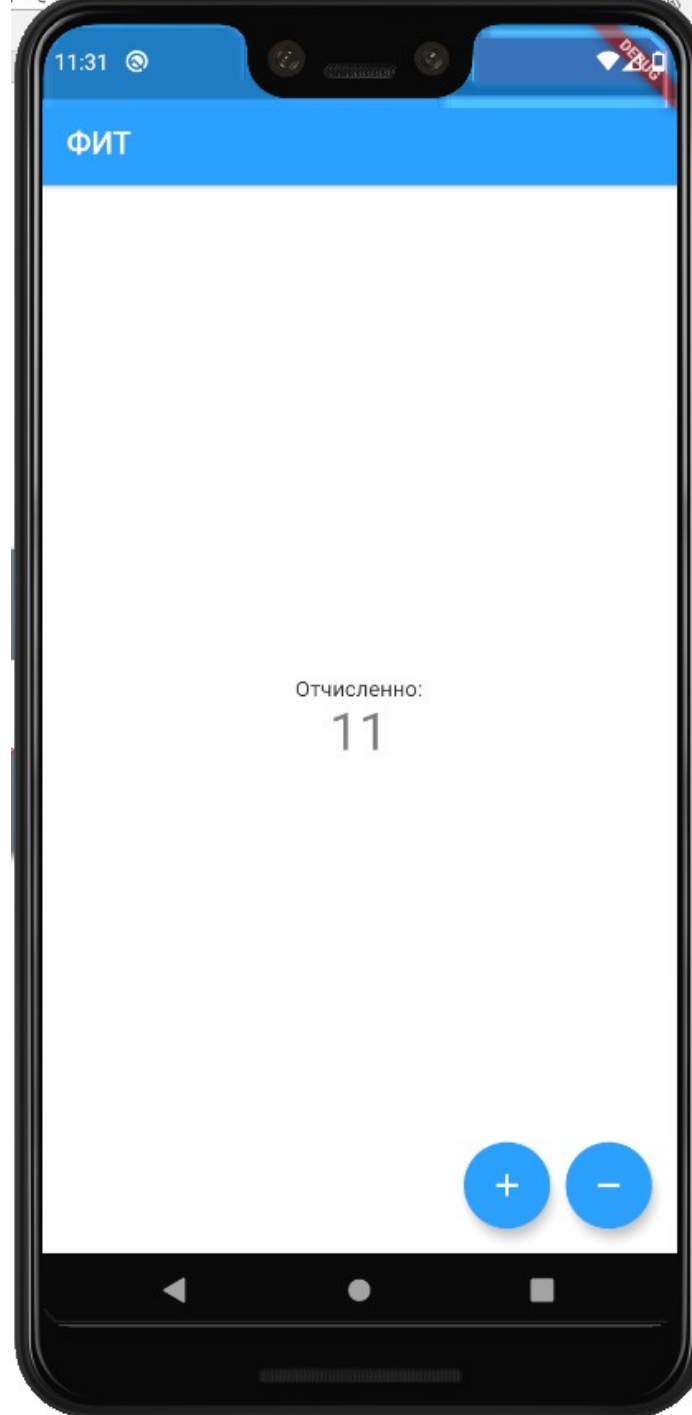
BLoc



Задачи

Убрать всю логику из виджетов

В классе VLoC получать только потоки на вход и выдавать только потоки на выход.



```
class _MyHomePageState extends State<MyHomePage> {  
  int _counter = 0;  
  
  void _incrementCounter(){  
    setState(() {  
      _counter++;  
    });  
  }  
  void _decrementCounter(){  
    setState(() {  
      _counter--;  
    });  
  }  
}
```

```
floatingActionButton: Row(  
  mainAxisAlignment: MainAxisAlignment.end,  
  children: <Widget>[  
    FloatingActionButton(  
      onPressed: _incrementCounter,  
      tooltip: 'Increment',  
      child: Icon(Icons.add),  
    ),  
    SizedBox(width: 10),  
    FloatingActionButton(  
      onPressed: _decrementCounter,  
      tooltip: 'Decrement',  
      child: Icon(Icons.remove),  
    ),  
  ],  
)
```



counter_event.dart

```
abstract class CounterEvent{}  
  
class IncrementEvent extends CounterEvent{}  
class DecrementEvent extends CounterEvent{}
```


counter_block.dart

```
class CounterBloc{
  int _counter =0;
  final _counterStateController = StreamController<int>();
  StreamSink<int> get _inCounter => _counterStateController.sink; //input
  Stream<int> get counter => _counterStateController.stream; //output
  final _counterEventController = StreamController<CounterEvent>();

  Sink<CounterEvent> get counterEventSink => _counterEventController.sink; //input

  CounterBloc(){
    _counterEventController.stream.listen(_mapEventToState);
  }

  void _mapEventToState(CounterEvent event) {
    if(event is IncrementEvent){
      _counter++;
    }else{
      _counter--;
    }
    _inCounter.add(_counter);
  }

  void dispose(){
    _counterEventController.close();
    _counterStateController.close();
  }
}
```

widget.dart

```
class _MyHomePageBlocState extends
State<MyHomePageBlock> {
  final _block = CounterBloc();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
...

```

```

body: Center(
  child: StreamBuilder(
    stream: _block.counter,
    initialData: 0,
    builder: (BuildContext context, AsyncSnapshot<int> snapshot) {
      return Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          Text('Отчисленно:'),
          Text('${snapshot.data}',
            style: Theme
              .of(context)
              .textTheme
              .headline4,)
        ],
      );
    },
  ),
),
floatingActionButton: Row(
  mainAxisAlignment: MainAxisAlignment.end,
  children: <Widget>[
    FloatingActionButton(
      onPressed: ()=>_block.counterEventSink.add(IncrementEvent()),
      tooltip: 'Increment',
      child: Icon(Icons.add),
    ),
  ],
),

```

...


Block Library

flutter_bloc 3.2.0



Published Jan 30, 2020 •  [bloclibrary.dev](#)  307 likes

FLUTTER ANDROID IOS WEB

VS Code




bloc felixangelov.bloc


Felix Angelov |  17,104 |  | Repository | License

Support for the bloc library and provides tools for effectively creating blocs for both Flutter and AngularDart apps.

[Disable](#) [Uninstall](#)

Android Studio

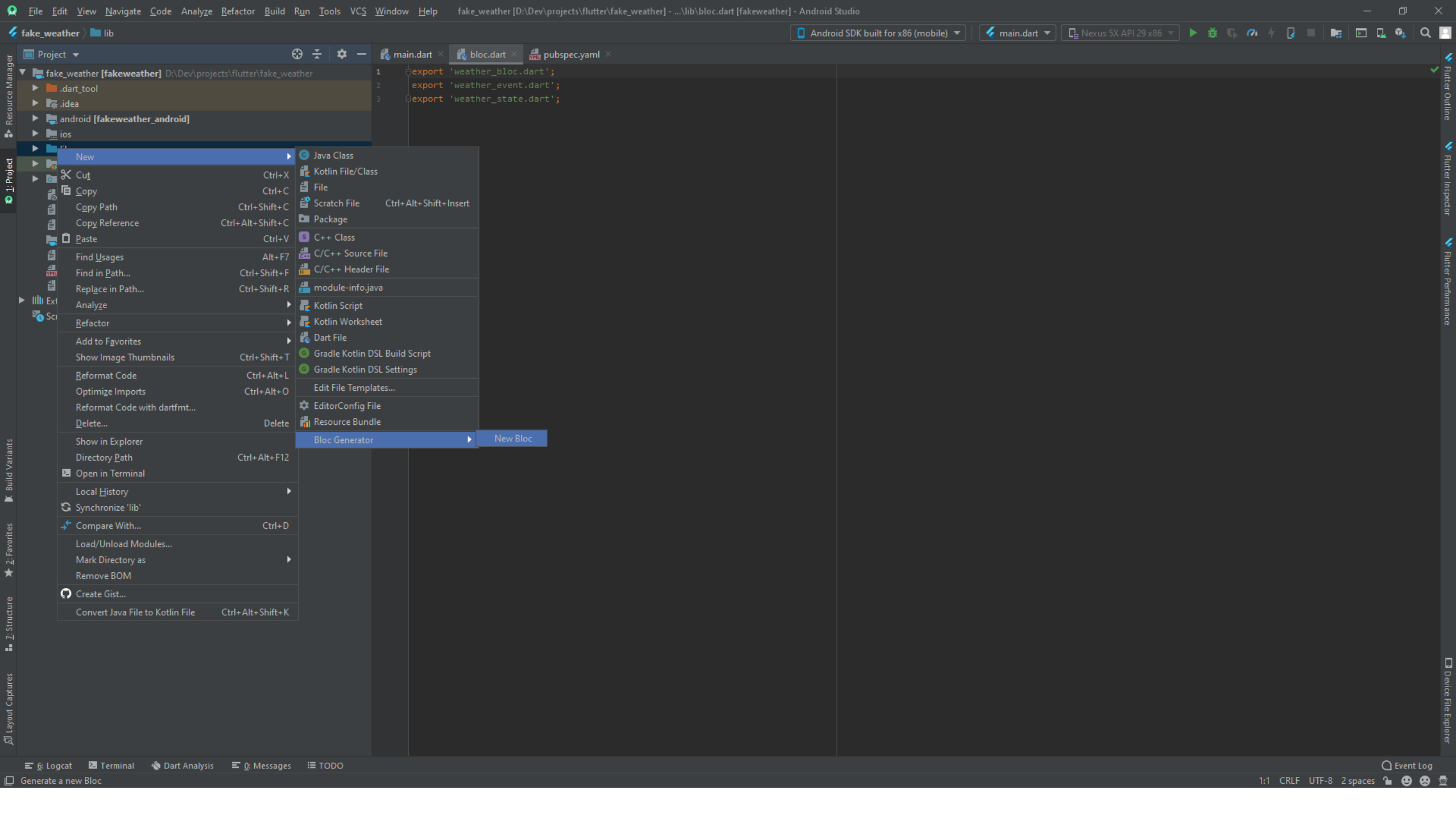


Bloc Code Generator 

1.6.0 Bloc

Pubspec.yaml

```
dependencies:  
  flutter:  
    sdk: flutter  
  flutter_bloc: ^0.15.1  
  equatable: ^0.2.6
```





weather.dart

```
import 'package:equatable/equatable.dart';
import 'package:meta/meta.dart';

class Weather extends Equatable {
  final String cityName;
  final double temperature;

  Weather({
    @required this.cityName,
    @required this.temperature,
  }) : super([cityName, temperature]);
}
```


Weather_event.dart

```
import 'package:equatable/equatable.dart';
import 'package:meta/meta.dart';

@immutable
abstract class WeatherEvent extends Equatable {
  WeatherEvent([List props = const []]) : super(props);
}

class GetWeather extends WeatherEvent {
  final String cityName;
  GetWeather(this.cityName) : super([cityName]);
}
```

Weather_state.dart

```
@immutable
abstract class WeatherState extends Equatable {
  WeatherState([List props = const []]) : super(props);
}

class WeatherInitial extends WeatherState {}

class WeatherLoading extends WeatherState {}

class WeatherLoaded extends WeatherState {
  final Weather weather;

  WeatherLoaded(this.weather) : super([weather]);
}
```

weather_bloc.dart

```
import 'dart:async';
import 'dart:math';
import 'package:bloc/bloc.dart';
import 'package:fakeweather/weather.dart';
import './bloc.dart';

class WeatherBloc extends Bloc<WeatherEvent, WeatherState> {
  @override
  WeatherState get initialState => WeatherInitial();

  @override
  Stream<WeatherState> mapEventToState(WeatherEvent event,) async* {
    if (event is GetWeather) {
      yield WeatherLoading();
      final weather = await _fetchWeatherFromFakeApi(event.cityName);
      yield WeatherLoaded(weather);
    }
  }

  Future<Weather> _fetchWeatherFromFakeApi(String cityName) {
    return Future.delayed(
      Duration(seconds: 1),
      () {
        return Weather(
          cityName: cityName,
          // Random 20 - 35.99
          temperature: 20 + Random().nextInt(15) + Random().nextDouble(),
        );
      },
    );
  }
}
```

<https://bloclibrary.dev/>

MVC

Installing

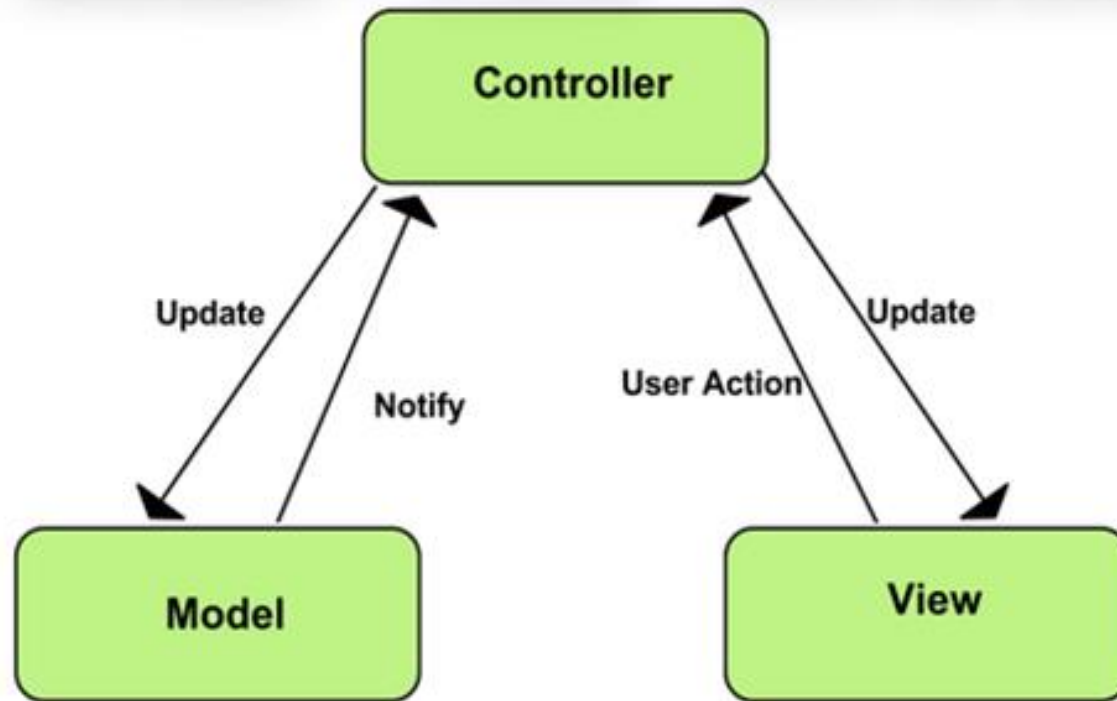
mvc_pattern 6.6.4+2 

Published Jan 26, 2021 •  andrioussolutions.com

FLUTTER | ANDROID | IOS | LINUX | MACOS | WEB | WINDOWS

```
dependencies: mvc_pattern: ^6.6.4+2
```

Model - View - Controller



Widget = View

```
class _MyHomePageState extends State<MyHomePage> {  
  final Controller _con = Controller.con;  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        // Here we take the value from the MyHomePage object that was created by  
        // the App.build method, and use it to set our appbar title.  
        title: Text(widget.title),),  
      body: Center(  
        child: Column(  
          mainAxisAlignment: MainAxisAlignment.center,  
          children: <Widget>[  
            Text(  
              widget.title,),  
            Text(  
              '${_con.counter}',  
              style: Theme.of(context).textTheme.display1, ),],),),  
      floatingActionButton: FloatingActionButton(  
        onPressed: () {  
          setState(  
            _con.incrementCounter );},  
        tooltip: 'Increment',  
        child: Icon(Icons.add),),); }  
}
```


Model

```
class Model {  
    static int get counter => _counter;  
    static int _counter = 0;  
    static int _incrementCounter() => ++_counter;  
}
```

Controller

```
class Controller extends ControllerMVC {  
  factory Controller() {  
    if (_this == null) _this = Controller._();  
    return _this;  
  }  
  static Controller _this;  
  
  Controller._();  
  
  static Controller get con => _this;  
  
  int get counter => Model.counter;  
  void incrementCounter() {  
    Model._incrementCounter();  
  }  
}
```

Redux

Installing

dependencies: flutter_redux: ^0.7.0

flutter_redux 0.7.0 

Published Oct 18, 2020 •  brianegan.com

FLUTTER | ANDROID | IOS | LINUX | MACOS | WEB | WINDOWS

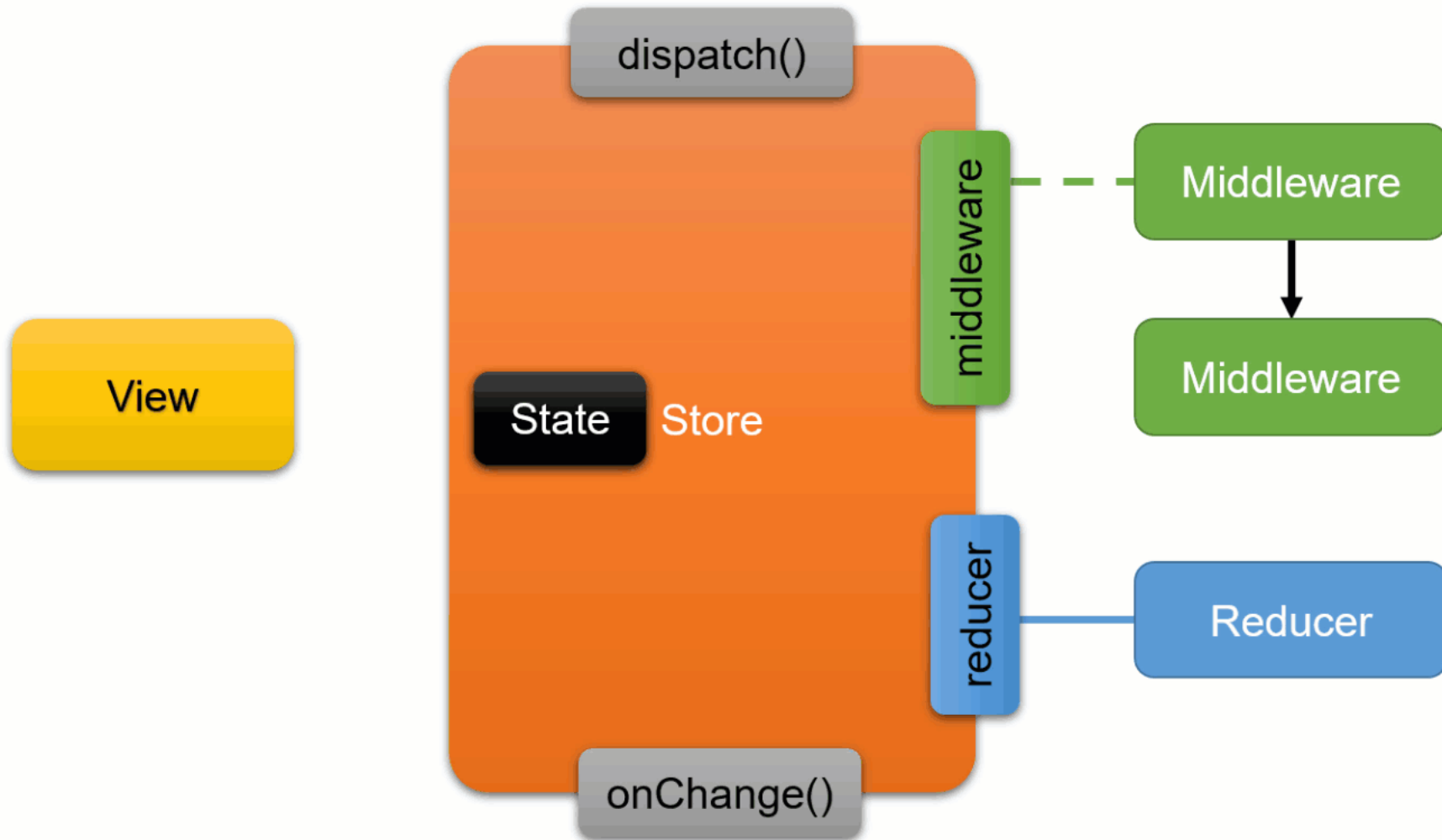
Redux widgets

StoreProvider - базовый виджет. Он передаст Redux Store всем потомкам, которые его запросят.

StoreBuilder - виджет-потомок, который получает Store от StoreProvider и передает его в функцию строителя виджетов.

StoreConnector - виджет-потомок, который получает Store от ближайшего предка StoreProvider, преобразует Store в ViewModel с заданной функцией конвертера и передает ViewModel в функцию построителя. Каждый раз, когда Магазин генерирует событие изменения, виджет автоматически перестраивается. Не нужно управлять подписками!

REDUX



Action

```
enum Actions { Increment }
```

Reducer

```
int counterReducer(int state, dynamic action) {  
    if (action == Actions.Increment) {  
        return state + 1;  
    }  
  
    return state;  
}
```


Store

```
void main() {  
  final store = new Store<int>(counterReducer, initialState: 0);  
  
  runApp(new FlutterReduxApp(  
    title: 'Flutter Redux Demo',  
    store: store,  
  ));  
}
```

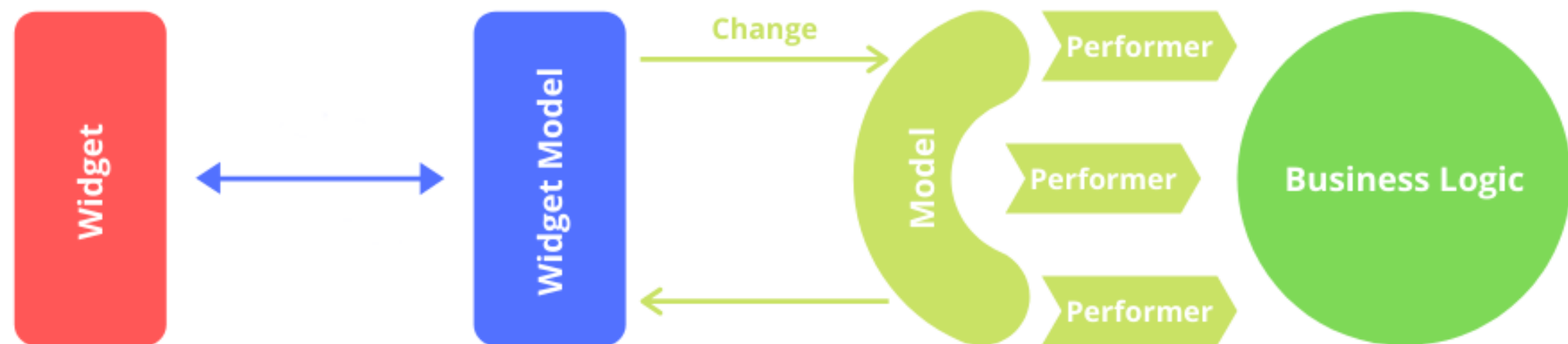
View

```
class FlutterReduxApp extends StatelessWidget {  
  final Store<int> store;  
  
  final String title;  
  
  FlutterReduxApp({Key key, this.store, this.title}) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    return new StoreProvider<int>(  
      store: store,  
      child: new MaterialApp(  
        theme: new ThemeData.dark(),  
        title: title,  
        home: new Scaffold(  
          appBar: new AppBar(  
            title: new Text(title),
```

```
body: new Center(  
  child: new Column(  
    mainAxisAlignment: MainAxisAlignment.center,  
    children: [  
      new Text(  
        'You have pushed the button this many times:',  
      ),  
      new StoreConnector<int, String>(  
        converter: (store) => store.state.toString(),  
        builder: (context, count) {  
          return new Text(  
            count,  
            style: Theme.of(context).textTheme.display1,  
          );  
        }],  
      ),  
    ],  
  ),  
),
```

```
floatingActionButton: new StoreConnector<int, VoidCallback>(
  converter: (store) {
    // Return a `VoidCallback`, which is a fancy name for a function
    // with no parameters. It only dispatches an Increment action.
    return () => store.dispatch(Actions.Increment);
  },
  builder: (context, callback) {
    return new FloatingActionButton(
      // Attach the `callback` to the `onPressed` attribute
      onPressed: callback,
      tooltip: 'Increment',
      child: new Icon(Icons.add),
    );
  },
),), );}}
```

MWWM



Элементы:

Widget-UI — та самая вёрстка.

WidgetModel — логика этого самого UI и его состояния.

Model — контракт с сервисным слоем. На данный момент это экспериментальная часть. Исторически у нас использовались Interactor'ы напрямую в WM.

```
class RepositorySearchWm extends WidgetModel {  
  
    RepositorySearchWm(  
        WidgetModelDependencies baseDependencies, //1  
        Model model, //2  
    ) : super(baseDependencies, model: model); //3  
  
}
```

- 1 - WidgetModelDependencies - это набор необходимых зависимостей. По умолчанию есть ErrorHandler, который дает возможность разместить логику обработки ошибок в одном месте. Вы должны предоставить реализацию обработчика.
- 2 - Модель является контрактом со служебным слоем. На данный момент это дополнительная функция. Можно пользоваться услугами напрямую, но это не рекомендуется.
- 3 - не пренебрегайте предоставлением модели суперклассу, если вы не хотите использовать модель.

Добавьте виджет, просто создав StatefulWidget и заменив родительский класс на CoreMwwmWidget.

```
class RepositorySearchScreen extends CoreMwwmWidget {
```

```
//...
```

```
@override
```

```
State<StatefulWidget> createState() {  
  return _RepositorySearchScreenState();  
}  
}
```

Добавить конструктор

```
RepositorySearchScreen({  
    WidgetModelBuilder wmBuilder, // need to testing  
}) : super(  
    widgetModelBuilder: wmBuilder ??  
    (ctx) => RepositorySearchWm(  
        // provide args,  
    ),  
);
```

Заменить родительский класс State на WidgetState

```
class _RepositorySearchScreenState extends WidgetState<RepositorySearchWm>
```