Contents

[Url’s 2](#_Toc527355231)

[Flutter environment setup 2](#_Toc527355232)

[Theory 3](#_Toc527355233)

[Widgets 3](#_Toc527355234)

[New application 3](#_Toc527355235)

[Important widgets to see: 4](#_Toc527355236)

# Url’s

<https://www.raywenderlich.com/116-getting-started-with-flutter>

## Company’s site. Rating 8

<https://flutter.io/bootstrap-into-dart/>

## Good basic level material design

<https://proandroiddev.com/flutter-from-zero-to-comfortable-6b1d6b2d20e>

## Best set of many examples and libraries so far. Very very good and best collection of ready to use examples

<https://solido.github.io/awesome-flutter/>

## Expansion Tile example

This is nested tiles. One row of listview opens another set of clickable rows.

<https://flutter.io/catalog/samples/expansion-tile-sample/>

## Propagation of data between widgets

<https://www.didierboelens.com/2018/06/widget---state---context---inheritedwidget/>

## Very good example of Advanced concept with case study covering all features of Flutter

<http://takyam-git.github.io/flutter.github.io/tutorial/>

## Stream example

https://medium.com/@ayushpguptaapg/using-streams-in-flutter-62fed41662e4

# Flutter environment setup

Android studio is best for Flutter development work as on October 2018

1. Download flutter SDk
2. Unzip it
3. Copy it in say c:\flutter
4. Add in path c:\flutter\bin
5. Install Android studio; create a project in Android studio. Creating a project will create a device which will be used in Vcode later. Android studio can be closed.
6. In Vcode add extension for flutter project
7. In Vcode command pallette select flutter new project. This will create a new project with all its files for Android and IPhone.
8. In lib folder main.dart is your program file which will be compiled for apk.
9. When you run debug it will create a debug build which will be too big. The debug build runs in a device. The device is automatically selected.
10. You can create a release build: In the launch.json set like below:

"configurations": [

{

"name": "Flutter",

"request": "launch",

"type": "dart",

"flutterMode": "debug"

}

]

Set the flutterMode as “release “

In command line of your project ‘flutter build android’. This will create small release apk in the android build folder.

1. Give ‘flutter upgrade’ to upgrade to latest version.
2. Run ‘flutter doctor’. Its cool.

## Starting emulator from command line

See this url

<https://stackoverflow.com/questions/4974568/how-do-i-launch-the-android-emulator-from-the-command-line>

# Cli commands

## Create new projects through cli

Flutter create <project-directory-name>

## Build

Flutter build apk

# Theory

* Flutter uses dart. The dart code is single threaded, and it uses async / await pattern

Almost every item in flutter is a widget. Widget is immutable. The widget concept is not taken from Dart.

* Streams can be used as rxjs in flutter.
* You can create a config.dart or globals.dart. Even if you import the file many times, it is imported only once. Import of a file is lazy. It is imported only when it is used. Once it is imported it is stored in RAM and is not imported again when it is re-referenced or re-imported.

## Widgets

Stateless: These have all configurations within themselves. They don’t depend on others. Like static image in an image view. There is no data binding.

Stateful: They depend on **state** object. They maintain their dynamic information by interacting with state object. There is data binding

Everything in Flutter from button, container, style is widget.

# New application

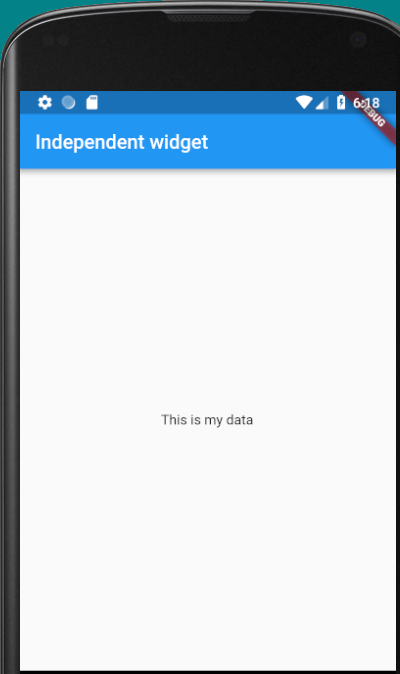
Pubspec.yaml is like package.json file. In lib folder main.dart is the program file.

Application architecture is simple:

* You create a main method which instantiates a MyApp class which extends widget
* The MyApp class overrides build method. This returns a widget
* The UI is mainly wiring up of widgets. If any property of a widget is also a widget you can place any widget in that place.

It is a good practice to keep topmost widget as MaterialApp and inside that Scaffold widget.

**import 'package:flutter/material.dart'**;  
  
**void** main() => runApp(**new** MyApp());  
  
**class** MyApp **extends** StatelessWidget {  
 @override  
 Widget build(BuildContext context) {  
 Widget matWidget = MaterialApp(  
 title: **'test'**,  
 home: Scaffold(  
 appBar: AppBar(title: Text(**'Independent widget'**)),  
 body: Center(  
 child: Text(**'This is my data'**),  
 )));  
 **return** (matWidget);  
 }  
}



# Important widgets to see:

For study of layouts in flutter use this url

<https://proandroiddev.com/flutter-layout-cheat-sheet-5363348d037e>

Try out

* MaterialApp, Scaffold, Center, Text, Row, Column widgets. In rows and columns check mainAxisAlignment and crossAxisAlignment
* Check ListView, Card, Stack, Positioned, Expanded, Container, ConstrainedBox, Sizedbox, dataTable

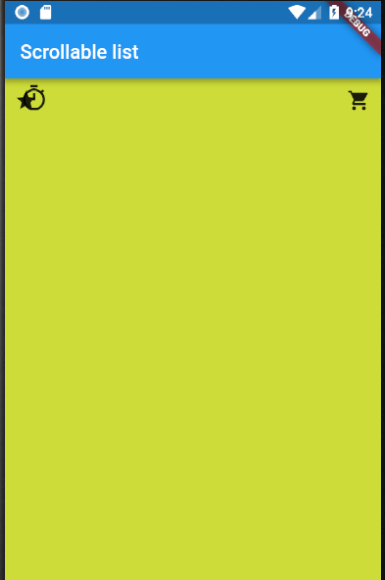
# Tips

* Use flutter build apk to build a release build
* By proper use of Row and column widget you can create good layouts
  + crossAxisAlignment.stretch can be used to place a set of controls in middle of screen
  + See:
    - MainAxisSize.max, MainAxisSize.min
  + Use IntrinsicWidth to make width of all buttons same as maximum width button
* Stack can be used to put one widget over another / overlapping. By combination of Stack->material+Icon1\_icon2 etc. You can create shopping cart with items.

body: Stack(  
 fit: StackFit.**expand**,  
 children: <Widget>[  
 Material(color: Colors.*lime*,),  
 Positioned(left: 10.0,top: 10.0,child: Icon(Icons.*star*),),  
 Positioned(right: 10.0,top:10.0, child: Icon(Icons.*shopping\_cart*),),  
 Positioned(left:15.0, top:5.0, child: Icon(Icons.*timer*,size: 30.0,))  
 ],  
)

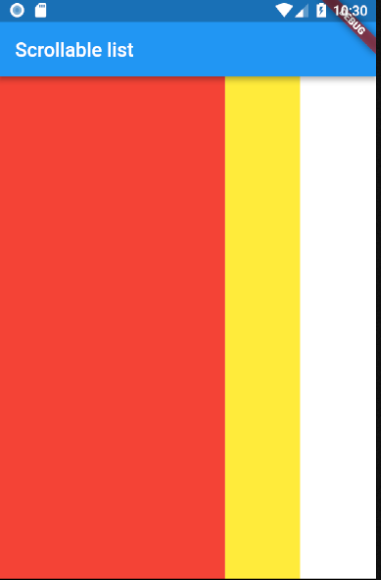
Hereinabove the Material widget is used as canvas and Positioned widgets are used to place absolute positions in it.

LayoutBuilder can be used to ascertain the maximum size of the layout and icons can be placed at maximum position.



* Expanded works with Flex / Flexbox and is good for distribution of space between multiple widgets.

body: Row(  
 children: <Widget>[  
 Expanded(  
 child: Container(  
 decoration: BoxDecoration(color: Colors.*red*),  
 ),  
 flex: 3,  
 ),  
 Expanded(  
 child: Container(  
 decoration: BoxDecoration(color: Colors.*yellow*),  
 ),  
 flex: 1,  
 ),  
 Expanded(  
 child: Container(  
 decoration: BoxDecoration(color: Colors.*white*),  
 ), flex: 1,  
 )  
 ],  
)



* ConstrainedBox can be used to wrap another widget and make it occupy any size in absolute, relative or ratio terms.
* In Container widget use double.infinity to attain size of container. Otherwise it takes the minimum size of its child widget.
  + Decoration is placed in background and foregroundDecoration is placed on top of widget.
  + Container’s transform property can be used to tilt the widget in it.
* Sizedbox can be used to provide custom spacing between multiple widgets

Column(  
 children: <Widget>[  
 Icon(Icons.*star*, size: 50.0),  
 **const** SizedBox(height: 30.0,),  
 Icon(Icons.*star*, size: 50.0),  
 Icon(Icons.*star*, size: 50.0)  
 ],  
)



* + Sized box can also be used to show / hide a widget based on Boolean value.
* SafeArea widget can be used to avoid painting the status bar etc.

# Stateful widgets in Flutter

If the data in your widget does not change you can do away with Stateless widgets. But if data which is displayed by widget changes then this widget needs to be redrawn and hence you need a Stateful widget. In Flutter the widgets need to be very inexpensive affair. That means the addition and removal of widgets should not affect the UI much. Keeping this in mind the formation of Stateful widget in Flutter is done in two stages. 1) Create Stateful Widget class (extends StatefulWidget), 2) Create the new State class (extends State)

The Stateful widget just needs to override the createState method. The new state class needs to incorporate all the variables and functions which affect the state along with it needs to override the build method.

## Create Stateful widget class

This just overrides the createState method and returns the new state which is MyState(). You need not write new keyword in Dart 2.

**class** MyStatefull **extends** StatefulWidget {  
 @override  
 MyState createState() {  
 **return** MyState();  
 }  
}

## Create State class

Here you create state variables and methods working on these variables. When the value of state variable changes you must call the setState method. This method takes a function argument which is called back immediately. So, the variable changing methods should call setState method passing it the function which incorporates the logic of variable value change. You must all override build method here only. The build method can have buttons etc. which call the methods who change the state of course also calling in setState in between. Finally in main method you create a StateLess widget which instantiates the StateFull widget. Note that the build method of MyState returns Scaffold widget and not MaterialApp widget because at runtime the Scaffold widget fits into MaterialApp widget.

**class** MyState **extends** State {  
 int **\_counter** = 0;  
  
 **void** increment() {  
 setState(() {  
 **\_counter**++;  
 });  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 Widget myWidget =

Scaffold(  
 appBar: AppBar(  
 title: Text(**'Testing stateful widgets'**),  
 ),  
 body: Container(  
 margin: EdgeInsets.all(20.0),  
 child: Row(  
 mainAxisAlignment: MainAxisAlignment.**center**,  
 children: <Widget>[  
 SizedBox(  
 width: 40.0,  
 child: Text(**'**$**\_counter'**),  
 ),  
 SizedBox(width: 20.0),  
 RaisedButton(  
 child: Text(**'Click me'**),  
 onPressed: increment,  
 )  
 ],  
 ),  
 ),  
 );  
**return** (myWidget);  
 }  
}

## Create Main and StateLess widget

**void** main() => runApp(**new** MyApp());  
  
**class** MyApp **extends** StatelessWidget {  
 @override  
 Widget build(BuildContext context) {  
 Widget material = MaterialApp(  
 title: **'abc'**,  
 home: MyStatefull(),  
 );  
 **return** (material);  
 }  
}

# Gesture detector can work on any widget

class MyButton extends StatelessWidget {

@override

Widget build(BuildContext context) {

return GestureDetector(

onTap: () {

print('MyButton was tapped!');

},

child: Container(

height: 36.0,

padding: const EdgeInsets.all(8.0),

margin: const EdgeInsets.symmetric(horizontal: 8.0),

decoration: BoxDecoration(

borderRadius: BorderRadius.circular(5.0),

color: Colors.lightGreen[500],

),

child: Center(

child: Text('Engage'),

),

),

);

}

}

# Examples

## Navigation to pages using routes

Shown in MProbeFlutter project

## Null aware expressions

* ??

var a = b?? 2;

if b is null then a = 2. You can put expression or closure also after ??

var z; var b

var a = b ?? z=3;

var a = b ?? (){z=3;b =1; }()

* ??=

var a;

a ??= 2;

here if a is null then set a = 2.

* ?.

obj?.method()

Use ?. when you want to call a method / getter on an object IFF that object is *not* null (otherwise, return null).

Note that method() is a method in obj. So if obj is null then return obj otherwise call obj.method() and return its value;