Flutter Tutorial

Beginner to Expert

Table of Content

Introduction

Flutter is an open-source UI software development kit created by Google. It is used to develop applications for Android, iOS, Windows, Mac, Linux, Google Fuchsia and the web.

Flutter is Google's mobile UI framework that can quickly build high-quality native user interfaces on iOS and Android. Flutter works with existing code. Flutter is being used by more and more developers and organizations around the world, and Flutter is completely free and open source . At present, some modules of the company are developed using Flutter
.
The major components of Flutter include:

Dart platform
Flutter engine
Foundation library
Design-specific widgets

Dart Platform
Flutter apps are written in the Dart language and make use of many of the language's more advanced features

You can refer Dart Language at Dart

Flutter Installation

Flutter is supporting HybridApp development on different Os.
To set up the flutter on each individual os by this Flutter official Tutorial

In this section we will learn how to install Flutter SDK in Windows system.

Step 1: Download Flutter SDK from Official website The Latest version is 1.12
Step 2: Unzip and archive file in specific folder, like c:\flutter\
Step 3: Update the system path to include flutter bin directory
Step 4: Open terminal and move to installed directory and run

```
flutter doctor
```

flutter doctor is tool to check all requirements for the flutter is installed or not and show the details

The result of the above command will give you

```
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel master, v1.14.6-pre.51, on Microsoft Windows
[Version 6.3.9600], locale en-IN)
[√] Android toolchain - develop for Android devices (Android SDK
version 28.0.3)
[√] Chrome - develop for the web
[√] Android Studio (version 3.3)
[!] Android Studio (version 3.4)
```

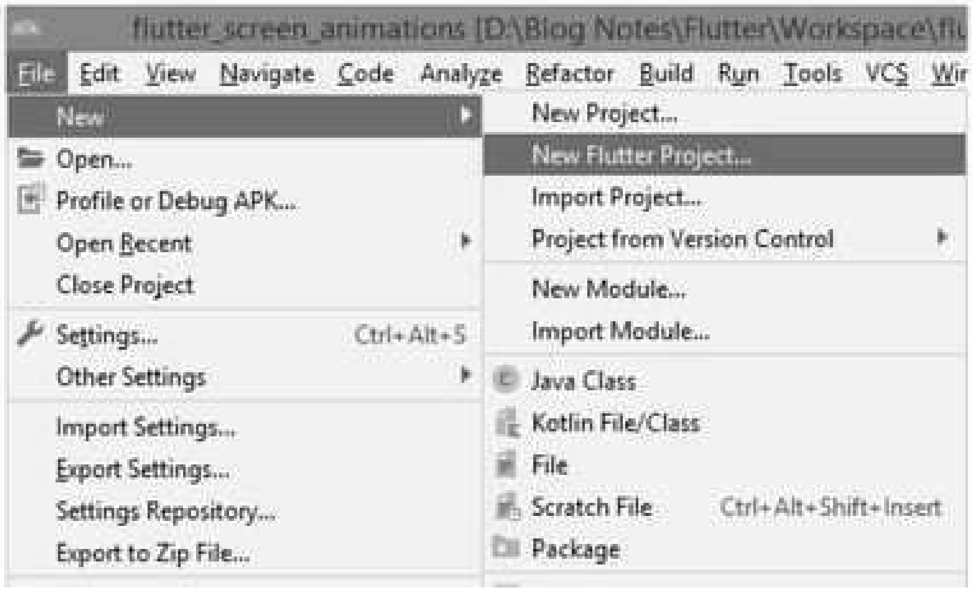
```
    X Flutter plugin not installed; this adds Flutter specific
functionality.
    X Dart plugin not installed; this adds Dart specific functionality.
[√] Connected device (2 available)

! Doctor found issues in 1 category.
```

Step 5: Install Android Studio
Step 6: Check Latest Android SDK installed or not, if not install it
Step 7: Open Android studio and install Dart and Flutter Plugins for Android studio.

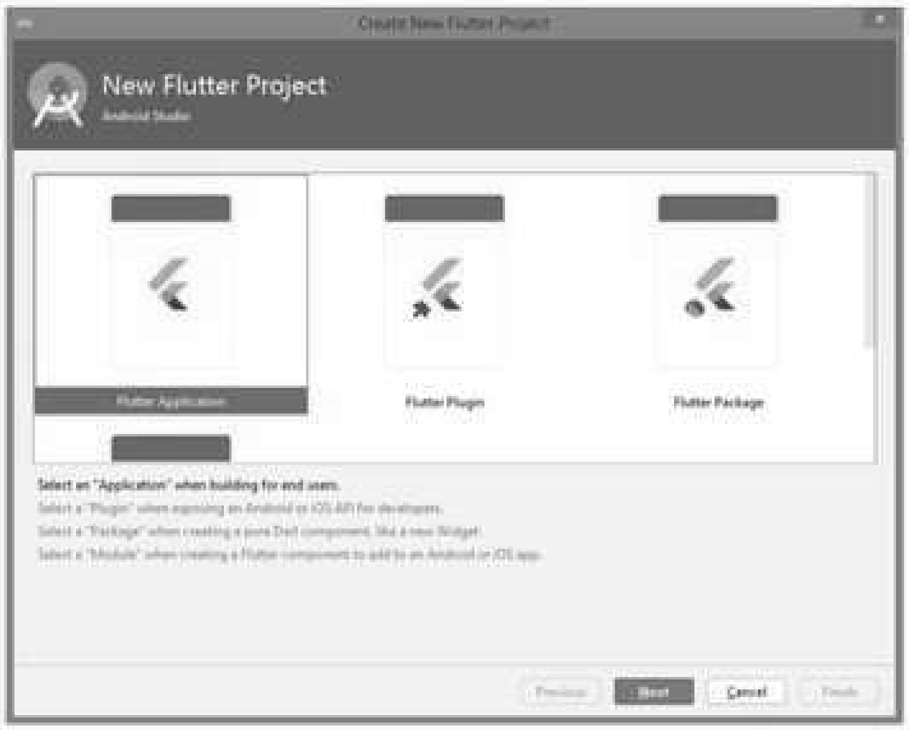
- Click File > Settings > Plugins.
- Select the Flutter plugin and click Install.
- Click Yes when prompted to install the Dart plugin.
- Restart Android studio

Flutter – Creating Simple Application in Android Studio

Now Open File -> Create new Flutter Project



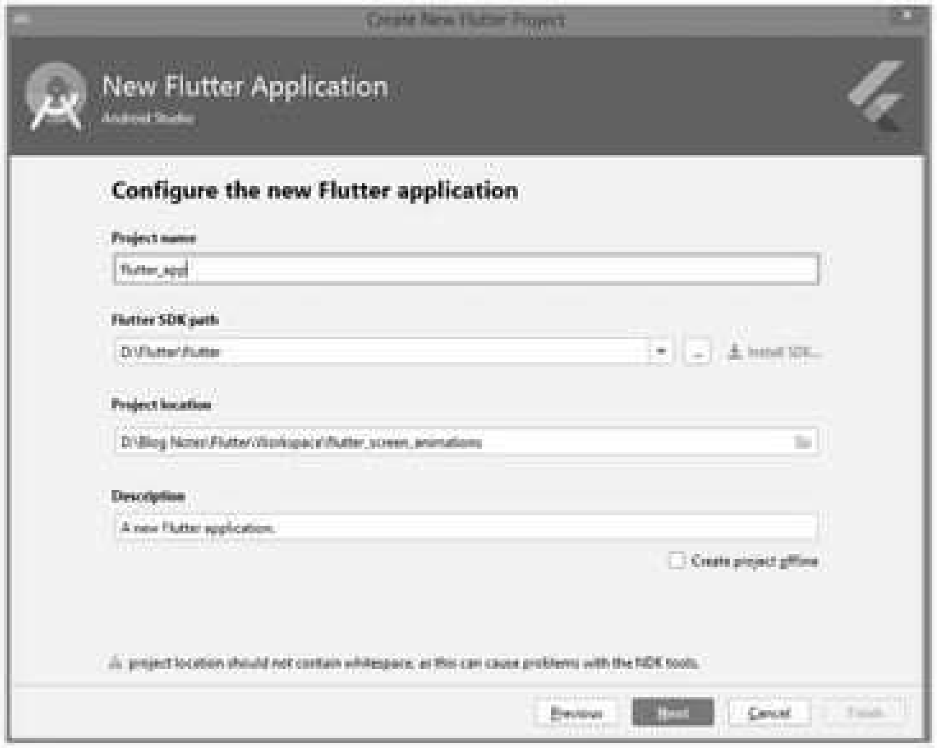
It will prompt below screen



Select an "Application" when building for end users.
Select a "Plugin" when exposing an Android or iOS API for developers.
Select a "Package" when creating a pure Dart component, like a new Widget.
Select a "Module" when creating a Flutter component to add to an Android or iOS app.

Select Flutter application and press Next

Now it will ask below details



Configure the new Flutter application

Project name: flutter_app

Flutter SDK path: D:\Flutter\Flutter

Project location: D:\Blog Notes\Flutter\Workspace\flutter_screen_animations

Description: A new Flutter application.

☐ Create project offline

⚠ project location should not contain whitespace, as this can cause problems with the NDK tools.

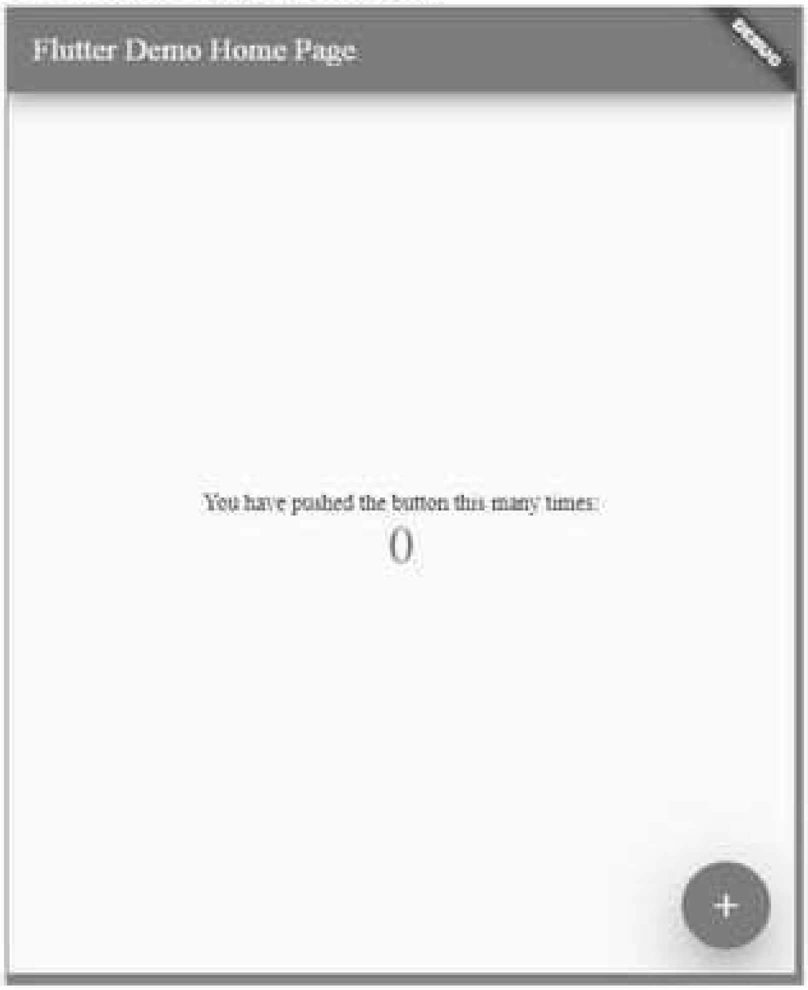
- Enter your Project Name
- Set Flutter SDk path (It is installation path)
- Set the Project Location
- Press Next Button

Enter Domain name and Press Finish

Now it will create a project with auto generated code

Now connect real device/Emulator and run the application

The output will be like this



Application Folder Structure

To understand flutter fully we need to understand the first Flutter folder structure.

- **android** - Auto generated source code to create android application
- **ios** - Auto generated source code to create ios application
- **web**- Auto generated source code to create web application
- **lib** - Main folder containing Dart code written using flutter framework
- **lib/main.dart** - Entry point of the Flutter application
- **test** - Folder containing Dart code to test the flutter application
- **test/widget_test.dart** - Sample code
- **.gitignore** - Git version control file
- **.metadata** - auto generated by the flutter tools
- **.packages** - auto generated to track the flutter packages
- **.iml** - project file used by Android studio
- **pubspec.yaml** - Used by Pub, Flutter package manager
- **pubspec.lock** - Auto generated by the Flutter package manager, Pub
- **README.md** - Project description file written in Markdown format

Dart Basics

Dart is an open-source general-purpose programming language. It was originally developed by Google.
Dart is an object-oriented language with C-style syntax. It supports programming concepts like interfaces, classes, unlike other programming languages Dart doesn't support arrays.
Dart collections can be used to replicate data structures such as arrays, generics, and optional typing.
The following code shows a simple Dart program

```
void main() {
  print('Hello, World!');
}
```

Every Dart application contains main() functions, from here code will execute.

Variables

Variable is named storage location and Data types simply refers to the type and size of data associated with variables and functions.
Dart uses a var keyword to declare the variable.
The syntax of var is defined below

```
var name = 'Flutter';
```

Dart provide us various built in data types
Numbers

As Other Programming languages to Java or C++ Dart does not have anything like float or long. Dart offers just two types of number

Int

Double

Strings : It represents a sequence of characters. String values are specified in either
single or double quotes

Booleans : Dart uses the bool keyword to represent Boolean values – true and false

Lists & Maps : It is used to represent a collection of objects

Runes : Represents string Unicode coded characters (UTF-32 code points), etc

Symbols : Use a Symbol literal to obtain the symbol's symbol object, which is to add a # symbol in front of the identifier

```
String name = 'Flutter';
```

Here String is Data type, name is variable name and Flutter is value of variable

```
var name = 'Flutter';
```

The type of the name variable is inferred as String.
The compiler will check we have var keyword not String so type depends on value which in this case is String

Example:

```
void main() {
  String fName = 'Chandu';
 var lName='Mouli';
 int intValue = 123;
 print(fName);
 print(lName);
```

```
  print(intValue);
}
```

Output will be

Chandu

Mouli

123

List : Declare a list is very simple, you can simply use square brackets [] to define the list. The following are common operations for lists

```
main(List<String> args) {
  var list = [1,2,3,4];

  print(list); //Output: [1, 2, 3, 4]
  //Length
  print(list.length);

  //Selecting single value
  print(list[1]);    //Output: 2

  //Adding a value
  list.add(10);

  //Removing a single instance of value
  list.remove(3);

  //Remove at a particular position
  list.removeAt(0);
}
```

If we want to define a compile-time constant list, for example, the contents of the list are immutable, you can use keywords const

```
main(List<String> args) {
  var list = const [1,2,3,4];
}
```

Defining a map

We can define maps by using curly braces {}

```
main(List<String> args) {
        var map = {
                'key1': 'value1',
                'key2': 'value2',
                'key3': 'value3'
        };

        //Fetching the values
        print(map['key1']);    //Output: value1
        print(map['test']);    //Output: null

        //Add a new value
        map['key4'] = 'value4';

        //Length
        print(map.length);

        //Check if a key is present
        map.containsKey('value1');

        //Get entries and values
        var entries = map.entries;
        var values = map.values;
}
```

We can also define Map by constructor

```
main(List<String> args) {
  var squares = new Map();
  squares[4] = 16;
}
```

Functions

Functions in dart are similar to those in JavaScript.
It consists of the function name, return value, and parameters.

```
main(List<String> args) {
  var name = fullName('Chandu', 'Mouli');
  print(name);
}

String fullName(String firstName, String lastName) {
  return "$firstName $lastName";
}
```

return type is an option, if we can remove return type and function looks like below

```
main(List<String> args) {
  var name = fullName('Chandu', 'Mouli');
  print(name);
}
 fullName(String firstName, String lastName) {
  return "$firstName $lastName";
}
```

If the function having single line we can write it as

```
main(List<String> args) {
  var name = fullName('Chandu', 'Mouli');
  print(name);
}

fullName(String firstName, String lastName) => "$firstName
$lastName";
```

Named parameter

Dart provides Named parameters while calling the functions

```
main(List<String> args) {
  var name = fullName(firstName: 'Chandu', lastName: 'Mouli');
  print(name);
}

fullName({String firstName, String lastName}) {
  return "$firstName $lastName";
}
```

Default value parameter

If we didn't pass parameter to function call we can define a default value to the parameters

```
main(List<String> args) {
  var name = fullName(firstName: 'Chandu');
  print(name);
}
fullName({String firstName, String lastName = "Shekar"}) {
  return "$firstName $lastName";
}
```

Control flow

If - else This is similar to other programing languages If - else

```
main(List<String> args) {
  var day = 6;

  if (number > 7) {
    print('Not a Week Day');
  } else if (number < 100) {
    print('Week Day');
  }
}
```

Loops

For Loop

```
main(List<String> args) {
  for (int i = 0; i < 10; i++) {
    print('$i');
  }
}
```

While loop

```
main(List<String> args) {
 int i = 0;
 while(i < 10) {
  print('$i');
  i++;
 }
}
```

Do-while loop

```
main(List<String> args) {
 int i = 0;
 do {
  print('$i');
  i++;
 } while (i < 10);
}
```

Switch

```
main(List<String> args) {
 int day = 5;
 switch(age) {
  case 1:
   print('Sunday.');
   break;
  case 2:
      print('Monday.');
   break;
```

```
    case 3:
     print('Tuesday');
     break;
    case 4:
     print('Wednesday');
     break;
    case 5:
     print('Thursday');
     break;
        case 6:
     print('Friday');
     break;
        case 7:
     print('Saturday');
     break;
  }
}
```

Exception handling

Similar to other programing languages dart also we can handle exceptions by
Try, catch blocks and throw exceptions by throw keyword

```
main(List<String> args) {
  divide(10, 0);
}
divide(int a, int b) {
  if (b == 0) {
    throw new IntegerDivisionByZeroException();
  }
  return a / b;
}
```

Let's catch the exception pass catch block

```
main(List<String> args) {
 try {
   divide(10, 0);
 } on IntegerDivisionByZeroException {
   print('Division by zero.');
 }
}
divide(int a, int b) {
 if (b == 0) {
   throw new IntegerDivisionByZeroException();
 }
 return a / b;
}
```

Flutter - Widgets

Now you are not having knowledge on flutter basics then go with Technical overview

What is a widget?

Everything within a flutter application is widget. From basic "text","Buttons" to "Screen Layouts".
In flutter application everything is designed by widget only.
These widgets are arranged in hierarchical order to be displayed onto the screen.
In the Flutter widgets most of widgets are Container widgets unlike Text widget

Widgets are two types

- Stateless Widget
- Stateful widget

All Widgets are categorized into below groups

1. Platform specific widgets
2. Layout widgets
3. State maintenance widgets
4. Platform independent / basic widgets

Platform specific widgets

Flutter provides platform specific widgets like Android and Ios

Android specific widgets are designed based on Material design rules
These widgets are called Material Widgets

Ios specific widgets are designed based on Human Interface Guidelines by Apple,
These widgets are called Cupertino widgets

Material Widgets

Scaffold
AppBar
BottomNavigationBar
TabBar
TabBarView
ListTile
RaisedButton
FloatingActionButton
FlatButton
IconButton
DropdownButton
PopupMenuButton
ButtonBar
TextField
Checkbox
Radio
Switch
Slider
Date & Time Pickers

SimpleDialog
AlertDialog

Cupertino Widgets

CupertinoButton
CupertinoPicker
CupertinoDatePicker
CupertinoTimerPicker
CupertinoNavigationBar
CupertinoTabBar
CupertinoTabScaffold
CupertinoTabView
CupertinoTextField
CupertinoDialog
CupertinoDialogAction
CupertinoFullscreenDialogTransition
CupertinoPageScaffold
CupertinoPageTransition
CupertinoActionSheet
CupertinoActivityIndicator
CupertinoAlertDialog
CupertinoPopupSurface

Layout Widgets

Layout widgets are widgets which will arrange the widget on the screen.
Examples: Container,Column,Row,Stack....

Single Child Widgets

Container
Padding
ConstrainedBox
Baseline
FractinallySizedBox
IntrinsicHeight
IntrinsicWidth
LimitedBox
OffStage
OverflowBox
SizedBox
SizedOverflowBox
Transform
CustomSingleChildLayout
Align

Multi Child Widgets

Row
Column
ListView
GridView
Expanded
Table
Flow
Stack

Example of widgets and Layout widgets

First create a simple visible widget

```
Text("Text widget")
```

If we want to add this visible widget to the Screen we need to put this widget inside the layout widget.
Let's create layout widget

```
Container(
child: Text("Text Widget")
)
```

Now let;s add this Lyoutwidget to the screen by

```
class Sample1 extends StatelessWidget{
        @override
        Widget build (BuildContext context)
        {
                return Container(
                        child: Text("Text Widget");
                )
        }
}
```

Image

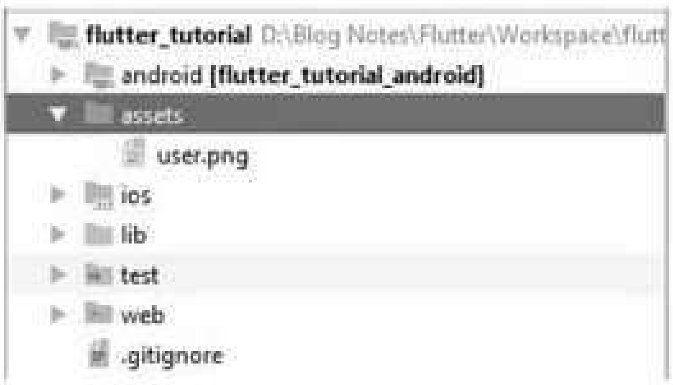
Image widget used to show an image. When displaying
an image, you specify the image source in the constructor:
image provider
asset,
network,
file,
memory

Load Network Image

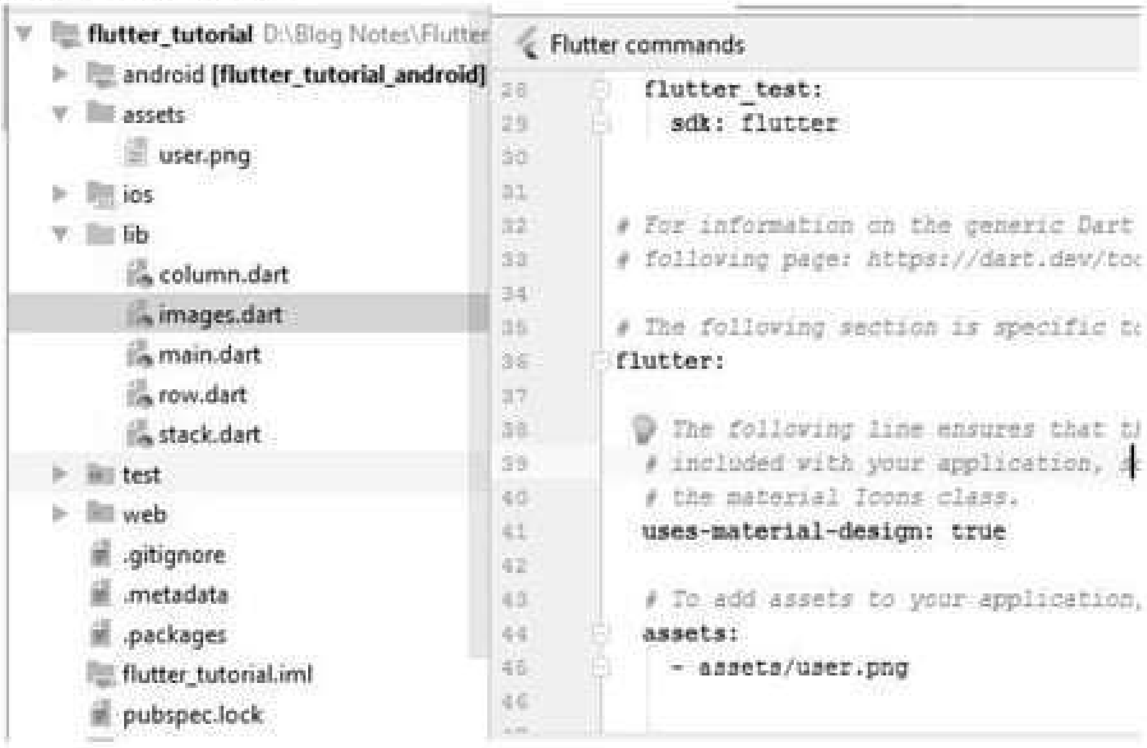
```
class ImageWidget extends StatelessWidget{
 @override
 Widget build(BuildContext context) {
  // TODO: implement build
  return MaterialApp(
    home: SafeArea(
      child: Scaffold(
        body: Center(child:
Image.network("https://cdn.pixabay.com/photo/2016/07/0
3/16/06/global-warming-1494965_960_720.jpg",width:
200,))
      ),
    ),
  );
 }

}
```

Load Image from Assets

To load images from assets first we need to create an Assets folder inside the application.
It could be like below


flutter_tutorial D:\Blog Notes\Flutter\Workspace\flutt
android [flutter_tutorial_android]
assets
user.png
ios
lib
test
web
.gitignore

Now after adding image into assets folder we need to set the path inside pubspec.yaml file



```
  flutter_test:
    sdk: flutter


# For information on the generic Dart
# following page: https://dart.dev/to

# The following section is specific t
flutter:

  # The following line ensures that t
  # included with your application, s
  # the material Icons class.
  uses-material-design: true

  # To add assets to your application,
  assets:
    - assets/user.png
```

Next run the below command in terminal to configure image

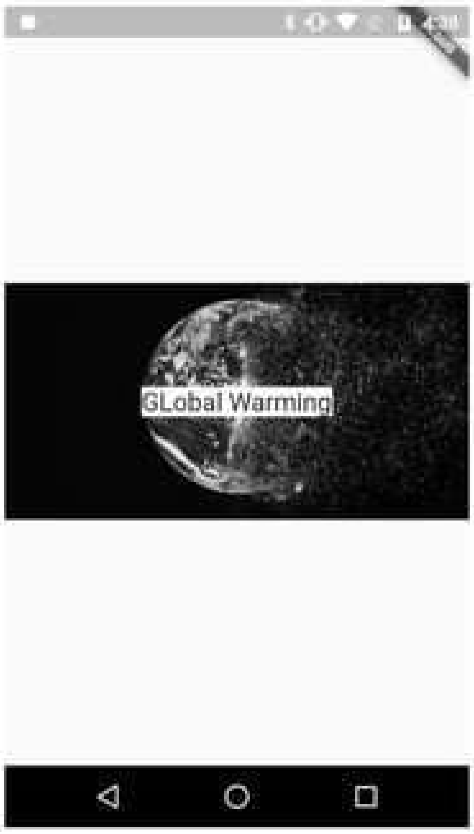
```
flutter packages get
```

Now lets create sample

```
class ImageWidget extends StatelessWidget{
 @override
 Widget build(BuildContext context) {
   // TODO: implement build
   return MaterialApp(
     home: SafeArea(
       child: Scaffold(
         body: Center(child: Image.asset("assets/user.png",width:
200,))
       ),
     ),
   );
 }
}
```

While Loading the images there is no option to show placeholder with above way,
Then how to show Placeholder while loading the image.
With FadeInImage widget we can achieve to show placeholder image
Replace above code with

```
Scaffold(
 body: Center(child: FadeInImage.assetNetwork(placeholder:
"assets/user.png",
   image:
"https://cdn.pixabay.com/photo/2016/07/03/16/06/globa
l-warming-1494965_960_720.jpg",width: 200,))
)
```



Icon

The icon widget allows us to quickly build icon widgets using a pre-built list of material icons, available in the Icons class.
We can specify the icon size and color

```
class IconWidget extends StatelessWidget{
 @override
 Widget build(BuildContext context) {
  // TODO: implement build
  return MaterialApp(
   home: SafeArea(
    child: Scaffold(
        body: Center(child: Icon(Icons.email,color: Colors.pink,size:
48,))
  ))
   ,
  );
 }

}
```

Buttons

We can't imagine a programming language without click events. Similarly other languages flutter provided buttons to handle click events.
We have different types of buttons in flutter
FlatButton
RaisedButton
IconButton
OutlineButton
DropdownButton
BackButton
CloseButton
FloatingActionButton

All these buttons click event is handled by **onPressed()**

```
onPressed: (){
}
```

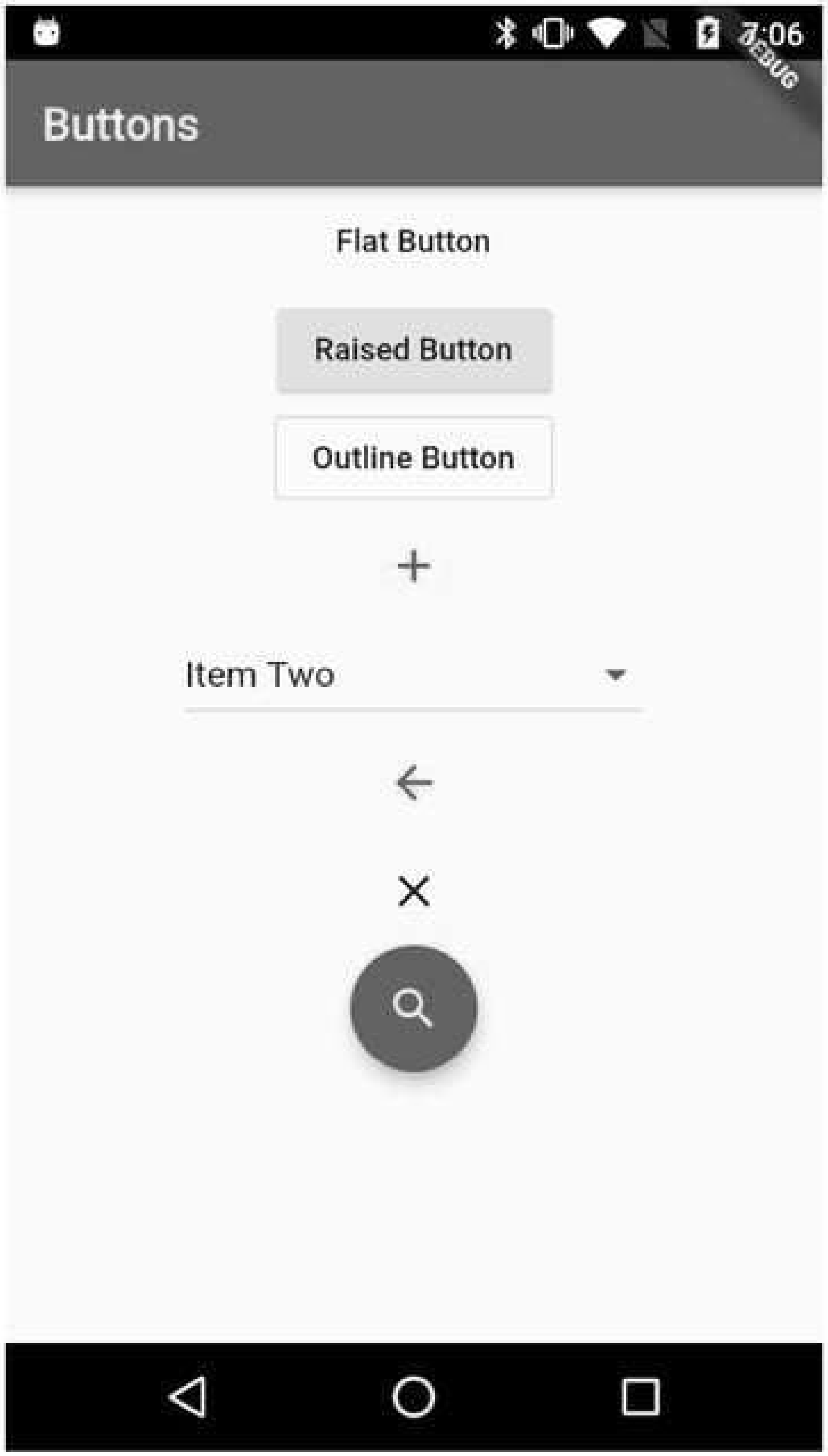
FlatButton : This type of button doesn't have any border, When we click on it it will show press effect
RaisedButton : When we need to show some decoration we can use this button

IconButton : It is a material button, Flashes background circle when clicked on
OutlineButton : A bordered button whose elevation increases and whose background becomes opaque when the
button is pressed
DropDownButton : It is a Material widget, Which is used for selecting from a list of items
It similar to Spinner in Android
CloseButton : An IconButton setup for use as a close button to close modals (or any other closeable content).
Flashes background circle when clicked on
FloatingActionButton: It is a Material widget button, A button that hovers in a layer above content

Button Examples

```
class ButtonWidget extends StatefulWidget{
 @override
 State<StatefulWidget> createState() {
  // TODO: implement createState
  return ButtonsWidgetState();
 }

}

class ButtonsWidgetState extends State<ButtonWidget>{
 var selected_item="Please choose a location";
 List<String>list=[
  "Please choose a location",
  "Item One",
  "Item Two",
  "Item Three",
  "Item Four",
 ];
 @override
 Widget build(BuildContext context) {
  // TODO: implement build
  return MaterialApp(
   home: SafeArea(child: Scaffold(
    appBar: AppBar(title: Text("Buttons"),backgroundColor:
Colors.pink,),
    body: Container(
     child: Center(
      child: Column(
       children: <Widget>[
        FlatButton(onPressed: (){
         debugPrint('Button Clicked ');
        }, child: Text("Flat Button")),

        RaisedButton(onPressed: (){
         debugPrint('Button Clicked ');
        },child: Text("Raised Button"),),

        OutlineButton(onPressed: (){
         debugPrint('Button Clicked ');
        },child: Text("Outline Button"),highlightedBorderColor:
Colors.pink,),

        IconButton(onPressed: (){
         debugPrint('Button Clicked ');
        },icon: Icon(Icons.add),color: Colors.pink,),

        DropdownButton(


         items:list.map((value){
          return DropdownMenuItem(child: Text(value),value:
value);
         }).toList(),
        hint: Text("Please choose a location"),
         value: selected_item,
         onChanged: (value){
```

```
                     selected_item=value;
                     setState(() {
                     });
                     debugPrint('Changed: ${value}');
                   },
                 )
                 BackButton(onPressed: (){
                   debugPrint('Button Clicked ');
                 },color: Colors.pink,),
                 CloseButton(),
                 FloatingActionButton(onPressed: (){
                   debugPrint('Button Clicked ');
                 }, child: Icon(Icons.search),backgroundColor: Colors.pink,)
               ],
             ),
           ),
         ),
       }),
     );
   }
}
```


7:06
DEBUG
Buttons
Flat Button
Raised Button
Outline Button
Item Two

Multi Child Layouts

Flutter - Linear Layout

In Android we have a linear layout to arrange childs in horizontal and vertical, similarly in flutter we can arrange by Row, Column widgets.

Example

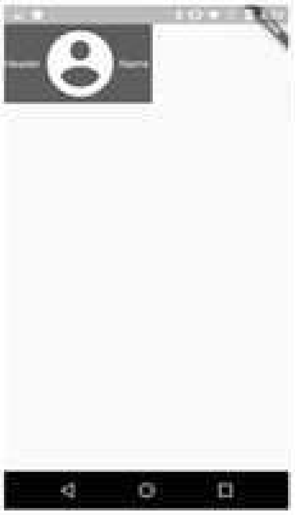
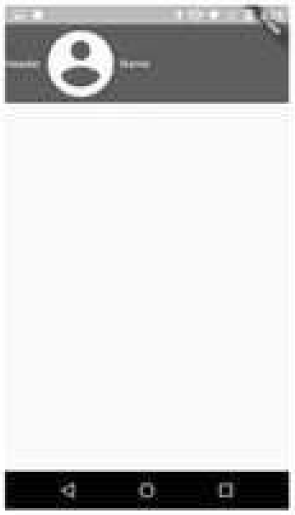
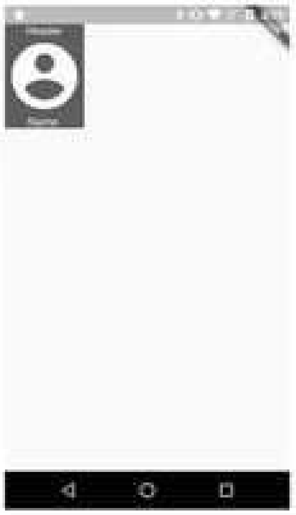
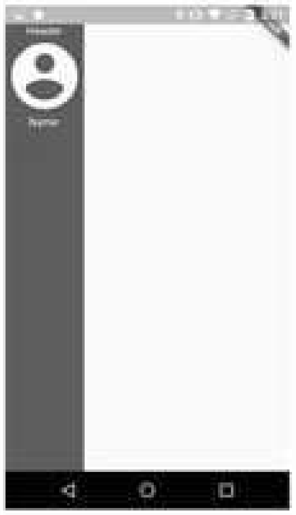
Horizontal Arrangement by **Row**

```
class RowWidget extends StatelessWidget {
 // This widget is the root of your application.
 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   title: 'Flutter Tutorial',
   theme: ThemeData(
    primarySwatch: Colors.blue,
   ),
   home: Scaffold(
    body: SafeArea(child:
    Container(
     color: Colors.brown,
     child: Row(
      mainAxisSize: MainAxisSize.min,
      crossAxisAlignment: CrossAxisAlignment.start,
      children: <Widget>[
       Text("Header",style: TextStyle(color: Colors.white),),
       Icon(Icons.account_circle,size: 100,color: Colors.white,),
       Text("Name ",style: TextStyle(color: Colors.white))
      ],
     ),
    )),
   ),
    );
  }
}
```

Vertical Arrangement by **Column**

```
class ColumnWidget extends StatelessWidget {
 // This widget is the root of your application.
 @override
 Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Flutter Tutorial',
    theme: ThemeData(
     primarySwatch: Colors.blue,
    ),
    home: Scaffold(
     body: SafeArea(child:
     Container(
      color: Colors.brown,
      child: Column(
       mainAxisSize: MainAxisSize.max,
       children: <Widget>[
        Text("Header",style: TextStyle(color: Colors.white),),
        Icon(Icons.account_circle,size: 100,color: Colors.white,),
        Text("Name ",style: TextStyle(color: Colors.white))
       ],
      ),
     )),
    ),
  );
 }
}
```

We can see the arrangement of children horizontal/vertical in below screen

Row wrap_content	Row match_parent	Column wrap_content	Column match_parent
			

How to set the Gravity for these widgets

We can set the Gravity by using **CrossAxisAlignment**

How it will work for Row and Column widgets
If we set the property for the Row it will align based on Vertical direction(center,start,end...)
If we set the property for Column it will align based on Horizontal direction(center,start,end...)

Framelayout in Flutter

Flutter uses Stack widgets to control child widgets at a layer. Child widgets can completely or partially cover the base widgets.

Stack control positions its children relative to the edges of its box. This class is useful if you just want to overlap multiple child widgets.

```
class StackWidget extends StatelessWidget {
// This widget is the root of your application.
 @override
 Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Flutter Tutorial',
    theme: ThemeData(
      primarySwatch: Colors.blue,
    ),
    home: Scaffold(
      body: SafeArea(child:
      Center(
        child: Stack(
          alignment: const Alignment(0, 0),
          children: <Widget>[

Image.network("https://cdn.pixabay.com/photo/2017/04/23
/19/17/climate-change-2254711_960_720.jpg"),

            Container(
              decoration: BoxDecoration(
                color: Colors.white,
              ),
              child: Text('GLobal Warming',style: TextStyle(fontSize:
20),),
            ),
          ],
          ),
         ])),
        ),
      );
    }
  }
```

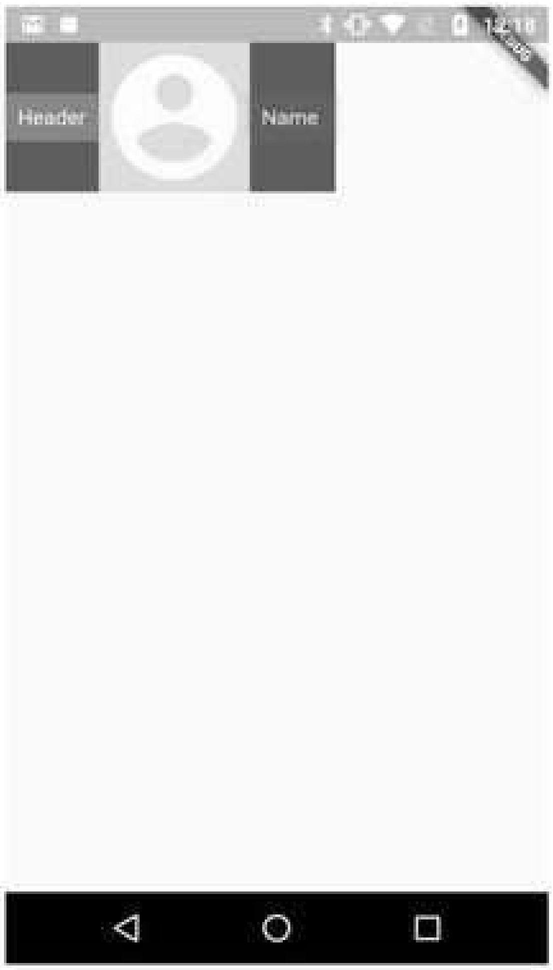
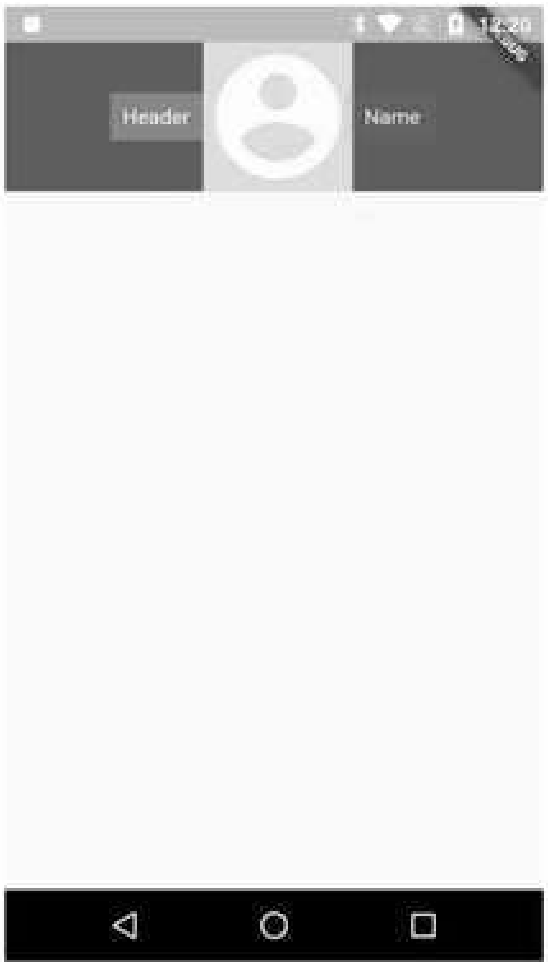
Flex Widget

The Flex Widget is similar to Row and Column widget.
We can use it as Row and Column by specifying the **direction** property.

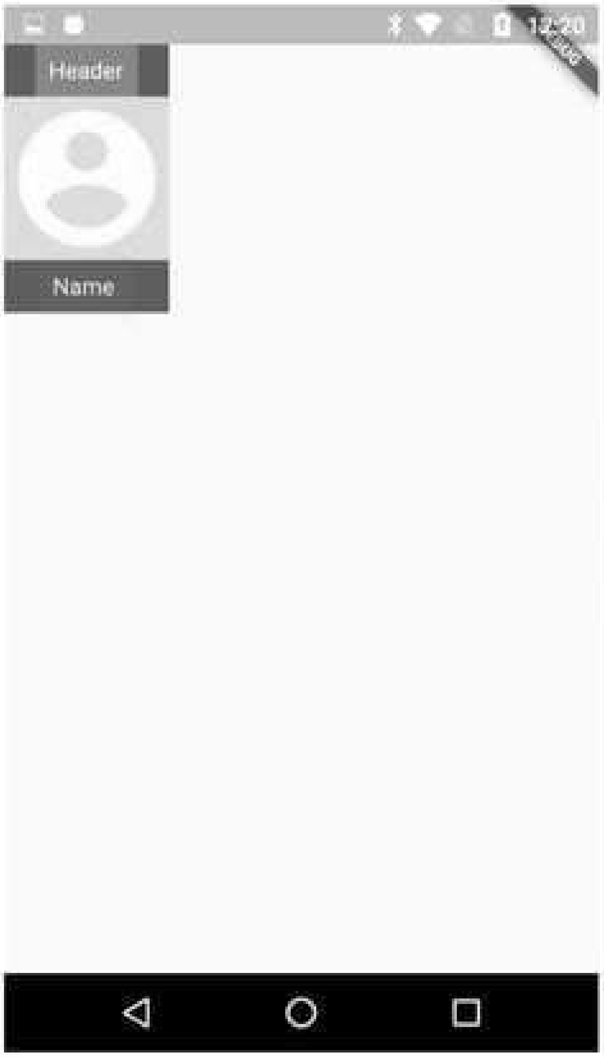
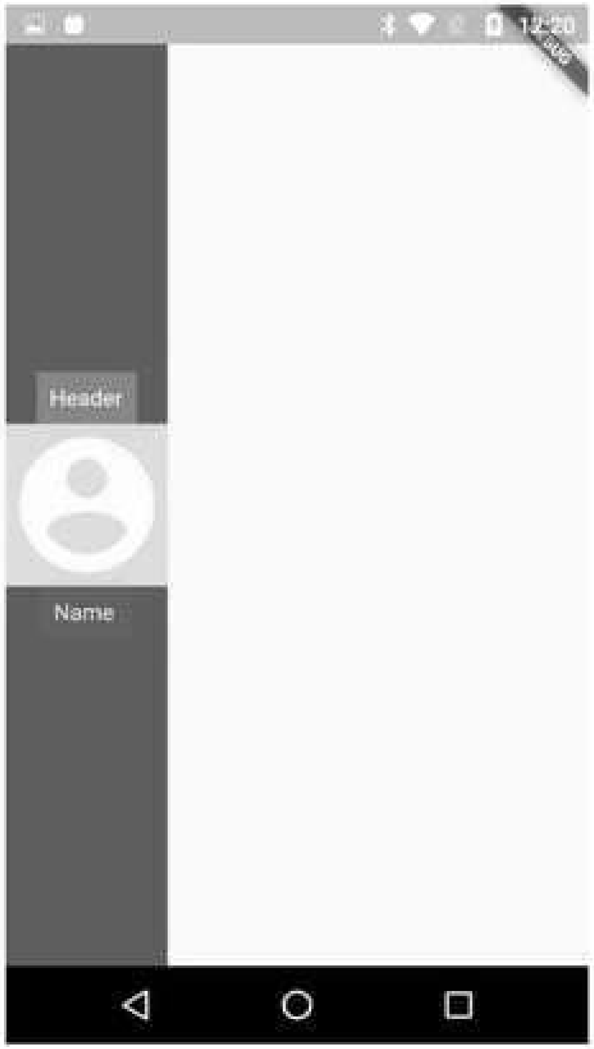
```
class FlexWidget extends StatelessWidget {
 // This widget is the root of your application.
 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   title: 'Flutter Tutorial',
   theme: ThemeData(
    primarySwatch: Colors.blue,
   ),
   home: Scaffold(

    body: SafeArea(child:
      Container(
       color: Colors.brown,
       child: Flex(
        direction: Axis.vertical,
        mainAxisSize: MainAxisSize.min,
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
         Container(child: Padding(
          padding: const EdgeInsets.all(8.0),
          child: Text("Header",style: TextStyle(color: Colors.white),),
            ),color: Colors.green,),
            Container(child: Icon(Icons.account_circle,size: 100,color:
Colors.white,),color: Colors.yellow,),
            Container(child: Padding(
              padding: const EdgeInsets.all(8.0),
              child: Text("Name ",style: TextStyle(color: Colors.white)),
            ),color: Colors.pink,)
          ],
        ),
      ),
    ),
  ),
 );
}
}
```

Change different alignment and check the result

Like Row Widget

Direction Horizontal wrap_content	Direction Horizontal match_parent
 	

Like Column Widget

Direction Vertical wrap_content	Direction Vertical match_parent
	

Weight Property like Android in Flex widget

If we add more items inside flex , to fit all these we can use Expandable widget to set each child flex ,

```
Flex(
 direction: Axis.horizontal,
 mainAxisSize: MainAxisSize.min,
 mainAxisAlignment: MainAxisAlignment.center,

 children: <Widget>[
  Flexible(
   flex: 1,
   child: Container(child: Padding(
    padding: const EdgeInsets.all(8.0),
    child: Text("Header",style: TextStyle(color: Colors.white),),
   ),color: Colors.green,),
  ),

  Flexible(flex: 1,child: Container(child:
Icon(Icons.account_circle,size: 100,color: Colors.white,),color:
Colors.yellow,)),
  Flexible(flex: 1,
   child: Container(child: Padding(
    padding: const EdgeInsets.all(8.0),
    child: Text("Name ",style: TextStyle(color: Colors.white)),
   ),color: Colors.pink,),
  ),
 ],
)
```

Listview

If we have more items to make them scrollable if the screen of the user device is smaller than the content of the control. In Flutter, the easiest way is to use ListView

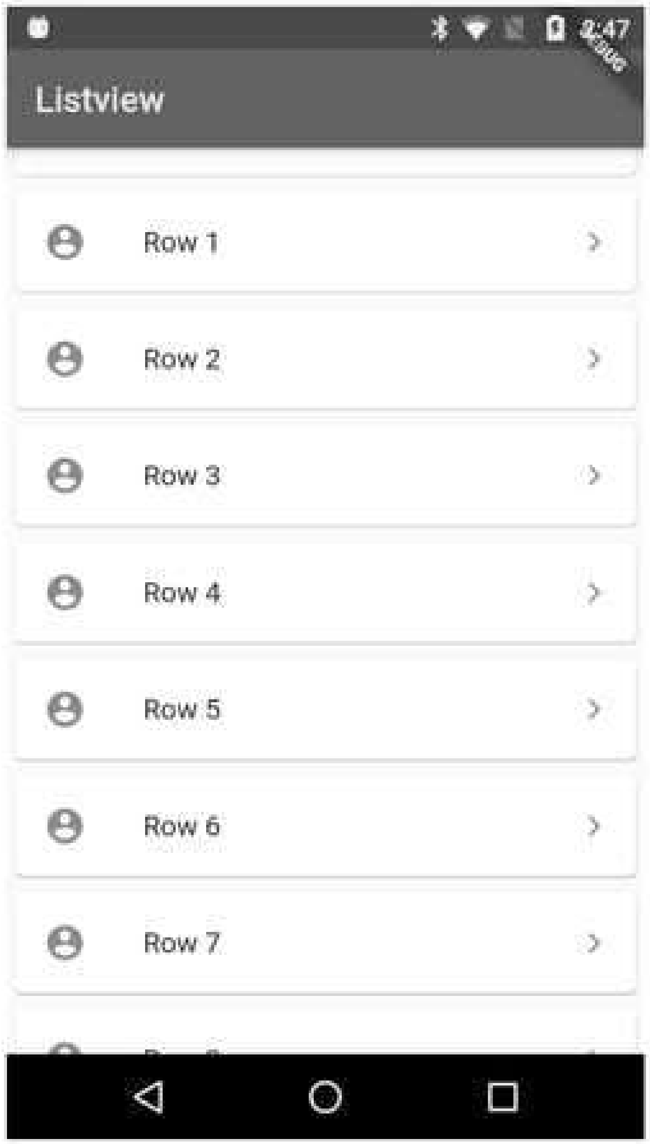
Here simple listview example

```
class ListViewWidget extends StatefulWidget {
 ListViewWidget({Key key}) : super(key: key);

 @override
 ListViewWidgetState createState() => ListViewWidgetState();
}

class ListViewWidgetState extends State<ListViewWidget> {
 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   home: Scaffold(
    appBar: AppBar(
     backgroundColor: Colors.pink,
     title: Text("Listview"),
    ),
    body: _getListData(),
   ),
  );
 }

 _getListData() {


  List<Widget> widgets = [];
  for (int i = 0; i < 100; i++) {
   widgets.add( Card(
    margin: EdgeInsets.all(5),
          child: ListTile(
            title: Text("Row $i"),
            leading: Icon(Icons.account_circle),
            trailing: Icon(Icons.arrow_forward_ios,size: 14,),
          ),
        ));
      }
      return ListView(children: widgets);
    }
  }
```

How will we handle the item click events?

ListTile has the property of onTap() function, with this we can handle the Click events of each child item.

```
onTap: (){
        _scaffoldKey.currentState.showSnackBar(SnackBar(content:
Text("Clicked on Child  $i")));
      },
```

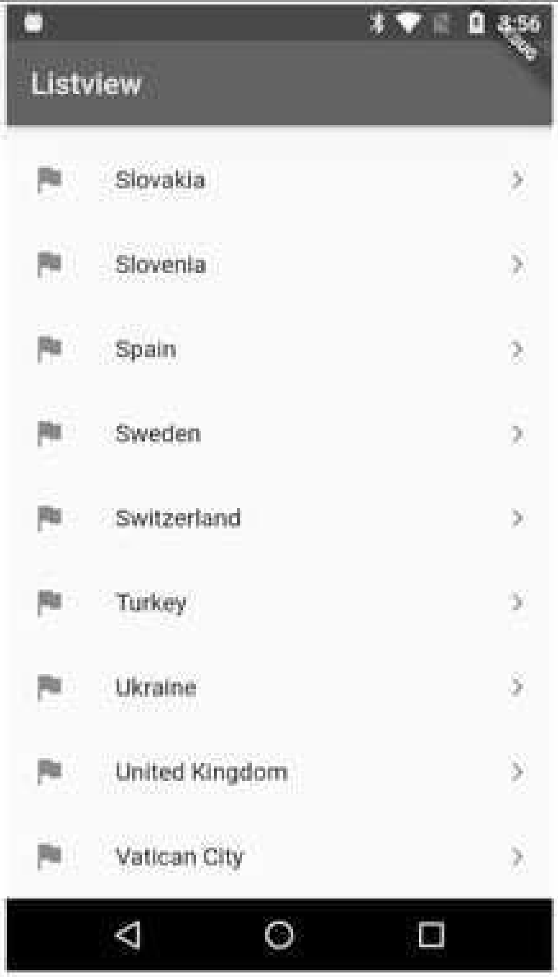


Dynamic ListView

The above example shows all static static widgets data. If we want to show dynamic data then we need to use
ListView.Builder()

```
class ListViewWidget extends StatefulWidget {
  ListViewWidget({Key key}) : super(key: key);
  @override
  ListViewWidgetState createState() => ListViewWidgetState();
}

class ListViewWidgetState extends State<ListViewWidget> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          backgroundColor: Colors.pink,
          title: Text("Listview"),
        ),
        body: _getDynamicList(),
      ),
    );
  }
_getDynamicList()
  {
    var countries = ['Albania', 'Andorra', 'Armenia', 'Austria',
    'Azerbaijan', 'Belarus', 'Belgium', 'Bosnia and Herzegovina', 'Bulgaria',
    'Croatia', 'Cyprus', 'Czech Republic', 'Denmark', 'Estonia', 'Finland',
    'France', 'Georgia', 'Germany', 'Greece', 'Hungary', 'Iceland', 'Ireland',
    'Italy', 'Kazakhstan', 'Kosovo', 'Latvia', 'Liechtenstein', 'Lithuania',
    'Luxembourg', 'Macedonia', 'Malta', 'Moldova', 'Monaco', 'Montenegro',
    'Netherlands', 'Norway', 'Poland', 'Portugal', 'Romania', 'Russia',
    'San Marino', 'Serbia', 'Slovakia', 'Slovenia', 'Spain', 'Sweden',
    'Switzerland', 'Turkey', 'Ukraine', 'United Kingdom', 'Vatican City'];

    return ListView.builder(
      itemCount: countries.length,
     itemBuilder: (ctx,index){
      return ListTile(
        onTap: (){

          _scaffoldKey.currentState.showSnackBar(SnackBar(content:
Text("Clicked on Country  ${countries[index]}")));
```

```
      },
      title: Text(countries[index]),
      leading: Icon(Icons.flag),
      trailing: Icon(Icons.arrow_forward_ios,size: 14,),
    );
  });
 }
}
```



Slovakia
Slovenia
Spain
Sweden
Switzerland
Turkey
Ukraine
United Kingdom
Vatican City

Listview.separated

```
      class ListViewBuilderWidget extends StatefulWidget{

       @override
```

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

}


class _ListViewBuilderWidget  extends
State<ListViewBuilderWidget>{

 @override
 Widget build(BuildContext context) {
  return Scaffold(

    appBar: new AppBar(
     title: new Text("ListviewBuilder Widget"),
    ),

    body: ListView.separated(
      itemCount: 100,
      itemBuilder: (BuildContext context, int index) {
       return ListTile(title: Text(" $index - ", style: TextStyle(color:
Colors.blue),));
      },

      separatorBuilder: (BuildContext context, int index) {
       return Divider(color: Colors.blue, height: 10,);
      }
    ),
  );
 }

}
```

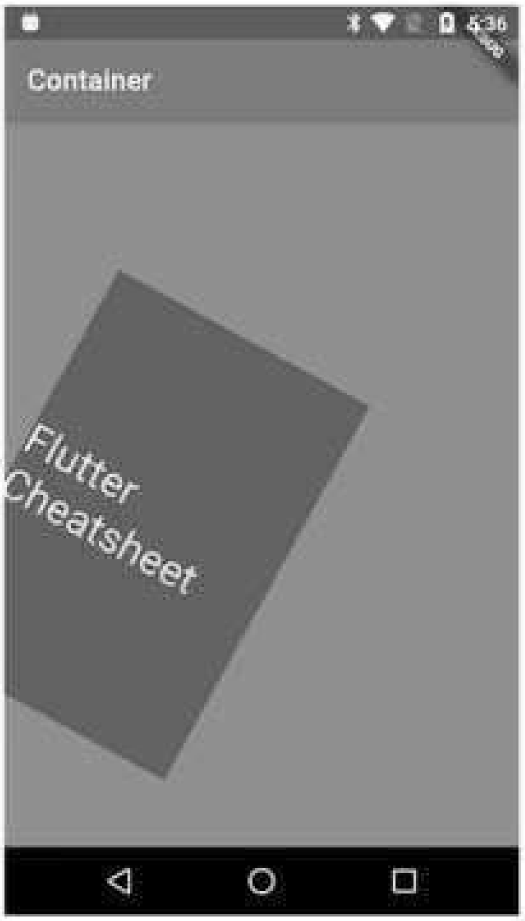
Examples of Single Child Layout Widgets

Container

A convenience widget that combines common painting, positioning, and sizing widgets. Often used to contain wrap child widgets and apply styling

Container having the below properties
Color Property
Child Property
Alignment Property
Constraints Property
Margin Property
Padding Property
Decoration Property
ForegroundDecoration Property
Transform Property

```
class ContainerWidget extends StatelessWidget{
 @override
 Widget build(BuildContext context) {
  // TODO: implement build
  return MaterialApp(
   home: Scaffold(
    appBar: AppBar(title: Text("Container"),),
    body: Container(
     color: Color.fromARGB(255, 66, 165, 245),
     child: Container(
      color: Colors.pink,
      alignment: Alignment.center,
      constraints: BoxConstraints(
       maxHeight: 300,
            maxWidth: 200,
            minWidth: 150,
            minHeight: 150,
           ),
           child: Container(
            child: Text("Flutter Cheatsheet",
             style: TextStyle(
               fontSize: 30.0,
              color: Colors.white,
 ),),),
           transform: Matrix4.rotationZ(0.5),
 ),
          alignment: Alignment.center,
        ),),);
  }}
```



Card

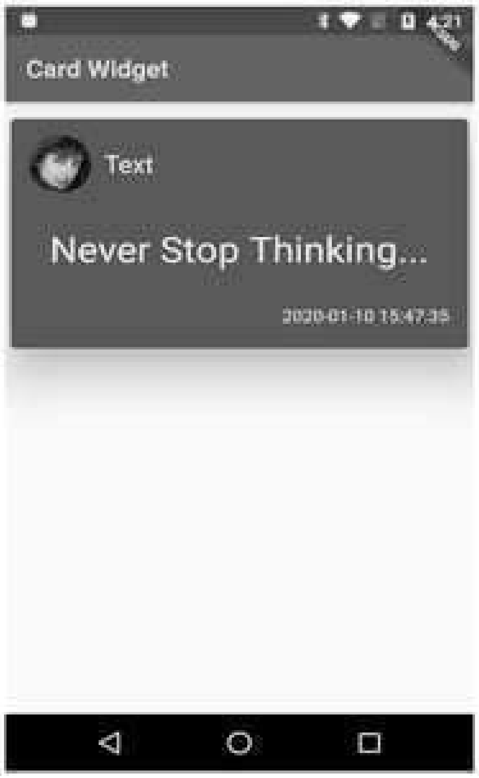
A card-like widget. Similar to Android's CardView, the card has slightly rounded corners and shadows

Card Widget attribute

color : container background color
elevation : Z-axis height, to achieve the shadow effect.
shape : defines the shape of the container
margin : margin
clipBehavior : the way to clip content

Example:

```
class CardWidget extends StatelessWidget{
 @override
 Widget build(BuildContext context) {
  // TODO: implement build
  return MaterialApp(
   home: Scaffold(
    appBar: AppBar(title:Text("Card Widget"),backgroundColor:
Colors.pink,),
     body: Container(
      alignment: Alignment.topCenter,
      margin: EdgeInsets.only(top: 10.0),
      child: SizedBox(
       width: 400.0,
       height: 200.0,
       child: Card(
        color: Colors.purple,
        elevation: 30.0,
        child: Padding(
         padding: EdgeInsets.all(
          14.0,
),
child: Column(
 children: <Widget>[
  Row(
   children: <Widget>[
    CircleAvatar(

     backgroundImage: NetworkImage(

"https[redacted]/photo,[redacted]
[redacted]"),
     radius: 24.0,
    ),
    Container(
     margin: EdgeInsets.only(left: 10.0),
     child: Text(
      "Text",
      style:
      TextStyle(color: Colors.white, fontSize: 20.0),
     ),
    ),
   ],
  ),
  Container(
   margin: EdgeInsets.only(top: 30.0),
   child: Text(
    "Never Stop Thinking...",
    style: TextStyle(color: Colors.white, fontSize: 30.0),
   ),
  ),
  Container(
   alignment: Alignment.bottomRight,
   margin: EdgeInsets.only(top: 30.0),
   child: Text(
    "2020-01-10 15:47:35",
    style: TextStyle(color: Colors.white, fontSize: 14.0),
   ),
  ),
                    ],
                  ),
                ),
              ),
            ),
          ),
          ),
        );
      }

    }
```



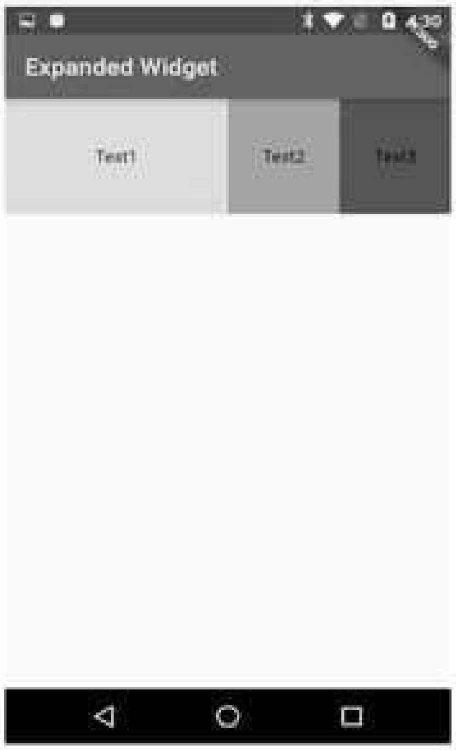
Expanded

The Expanded component allows Row, Column, Flex and other sub-components to expand in the direction of their main axis and fill the available space. Similar usage of widget properties in Android

- Expanded will be distributed as full as possible in the main axis direction of Row, Column, or Flex
- If Column contains two childs and two widgets are expanded, both share the
  available vertical space evenly.
- If only one is expanded, the expanded one takes up all the available vertical space.
- If neither is expanded, the available vertical space goes unfilled

Example:

```
class ExpandedWidget extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  // TODO: implement build
  return MaterialApp(
    home: Scaffold(
     appBar: AppBar(backgroundColor: Colors.pink,title:
Text("Expanded Widget"),),
    body: new Row(
     children: <Widget>[
      new Expanded(
       flex: 2,
       child: new Container(
        child: new Text('Text1', textAlign: TextAlign.center),
        height: 100,
        alignment: AlignmentDirectional.center,
        color: Colors.yellow,
       ),
      ),
      new Expanded(
          flex: 1,
          child: new Container(
            child: new Text('Text2', textAlign: TextAlign.center),
            height: 100,
            alignment: AlignmentDirectional.center,
            color: Colors.lightGreen,
          ),
        ),
        new Expanded(
          flex: 1,
          child: new Container(
            child: new Text('Text3', textAlign: TextAlign.center),
            height: 100,
            alignment: AlignmentDirectional.center,
            color: Colors.deepPurple,
          ),
        ),
      ],
    ),
  ));
 }
}
```



Flexible

It is actually Expanded inheritance Flexible. Use Flexible widgets to Row, Column or Flex provided in the main shaft expands to fill the available space in flexibility (e.g., horizontal filling sub assembly Row or perpendicular filled Column), but Expandeddifferent, Flexible is not required to fill the available space sub components.

Flexible The control must be Row, Column or Flex a descendant of Row, the path from the control to its closure , Column or Flex the path must contain only Stateless Widgets or Stateful Widget these, and cannot be other types of widgets (for example Render ObjectWidget)

Center

This widget is used to center a Widget within its parent Widget.

GestureDetector

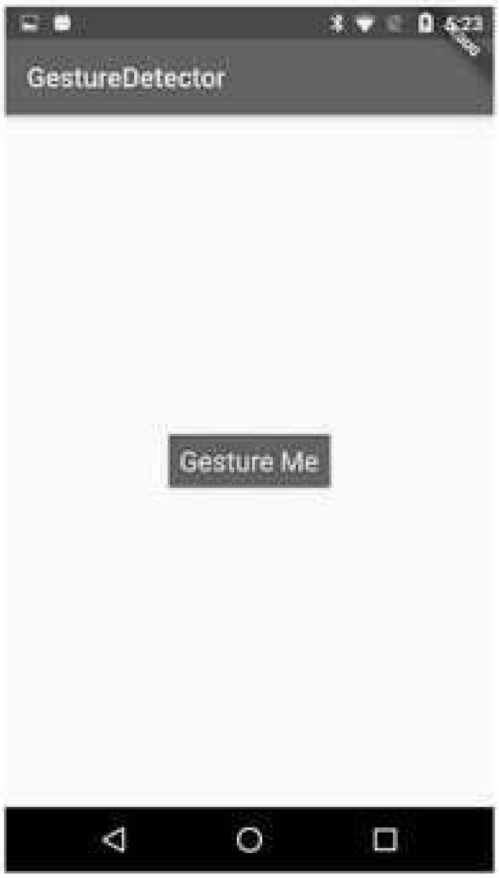
GestureDetector is a widget that detects gestures. If the child property of GestureDetector is not empty, GestureDetector sets its size to the size of the child.
If the child property is empty, it sets its size to the size of the parent component
We have different type of gestures, below are few of them

- onTapDown,
- onTapUp,
- onTap,
- onTapCancel,
- onForcePressPeak,
- onForcePressUpdate,
- onForcePressEnd,
- onPanDown,
- onPanStart,
- onPanUpdate,
- onPanEnd,
- onPanCancel,
- onScaleStart,
- onScaleUpdate,
- onScaleEnd

- onSecondaryTapDown,
- onSecondaryTapUp,
- onSecondaryTapCancel,
- onDoubleTap,
- onLongPress,
- onLongPressStart,
- onLongPressMoveUpdate,
- onLongPressUp,
- onLongPressEnd,

Example

```
class GesterDetectorWidget extends StatelessWidget{
 GlobalKey<ScaffoldState>_scaffoldstate=GlobalKey();
 @override
 Widget build(BuildContext context) {
  // TODO: implement build
  return MaterialApp(
    home: Scaffold(
     key: _scaffoldstate,
    appBar: AppBar(title:Text("Card Widget"),backgroundColor:
Colors.pink,),
  body: Container(
  child: GestureDetector(

   child: Center(


     child: Container(
      color: Colors.pink,
      child: Padding(
       padding: const EdgeInsets.all(8.0),
       child: Text("Gesture Me",style: TextStyle(fontSize:
20,color: Colors.white),),
      ),
     )),
    onTap: (){

_scaffoldstate.currentState.showSnackBar(SnackBar(content:
Text("onTap Event")));
    },
    onDoubleTap: (){

_scaffoldstate.currentState.showSnackBar(SnackBar(content:
Text("onDoubleTap Event")));
    },
    onLongPress: (){

_scaffoldstate.currentState.showSnackBar(SnackBar(content:
Text("onLongPress Event")));
    },
  ),
  )
  )
  );
 }

}
```



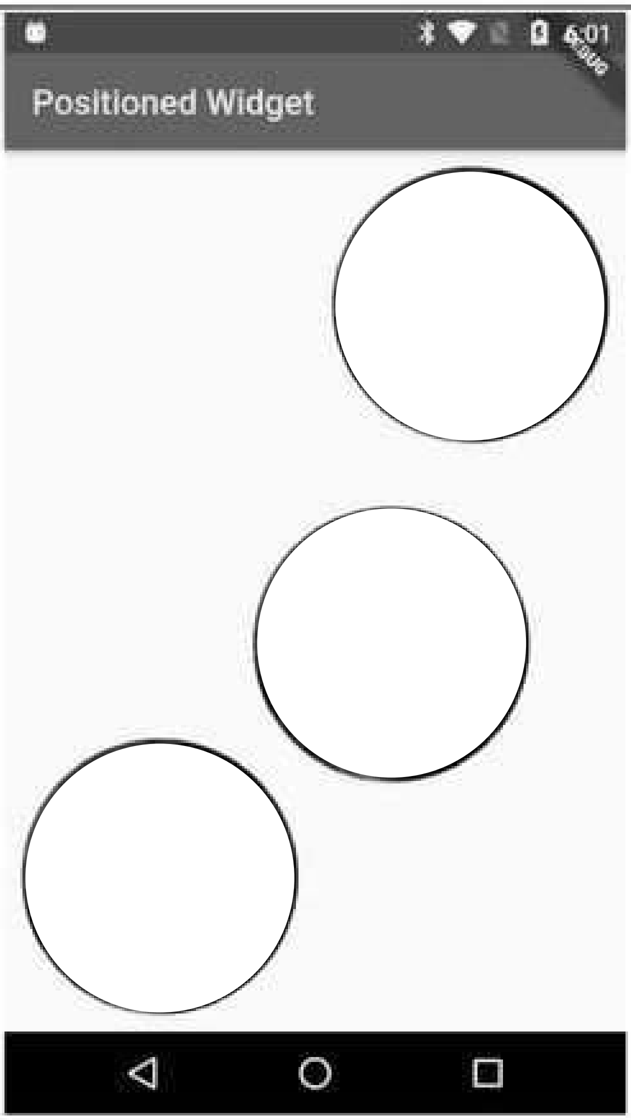
Positioned

This use controls the position of the widget, through which he can place a component at will, a bit like an absolute layout

```
Positioned({
        Key key,
        this.left,
        this.top,
        this.right,
        this.bottom,
        this.width,
        this.height,
        @required Widget child,
})
```

Example

```
class PositionedWidget extends StatelessWidget{
 GlobalKey<ScaffoldState>_scaffoldstate=GlobalKey();
 @override
 Widget build(BuildContext context) {
  final size = MediaQuery.of(context).size;
  return MaterialApp(
    home: Scaffold(
      key: _scaffoldstate,
      appBar: AppBar(title:Text("Positioned
Widget"),backgroundColor: Colors.pink,),
      body:Container(
       width: size.width,
       height: size.height,
       child: Stack(
        children: <Widget>[
         Positioned(
          child: CircleAvatar(
           backgroundImage:
NetworkImage("https://cdn.pixabay.com/[redacted]"),
           radius: 80.0,
          ),
          right:10, top: 10,
         ),
        Positioned(
          child: CircleAvatar(
           backgroundImage:
NetworkImage("https://cdn.pixabay.com/photo/2016/10/17/17/4
1/priyanka-chopra-1748248_960_720.jpg"),
           radius: 80.0,
          ),
           left: size.width / 2 * 0.8,
           top: size.height / 2 * 0.7,
            ),
              Positioned(
                child: CircleAvatar(
                  backgroundImage:
NetworkImage("https://cdn.pixabay.com/photo/2016/10/17/17/4
1/                    -1748248_960_720.jpg"),
                  radius: 80.0,
                ),
                left: 10,
                bottom: 10,
            )],
          ), ),) );
 }}
```



SafeArea

A widget that insets its child by sufficient padding to avoid intrusions by the operating system

SafeArea is mainly used to ensure that the view will not be covered by system components, such as the status bar, etc
SafeArea Constructor is like below

```
const SafeArea({
 Key key,
 this.left = true,
 this.top = true,
 this.right = true,
 this.bottom = true,
 this.minimum = EdgeInsets.zero,
 this.maintainBottomViewPadding = false,
 @required this.child,
})
```

SingleChildScrollView

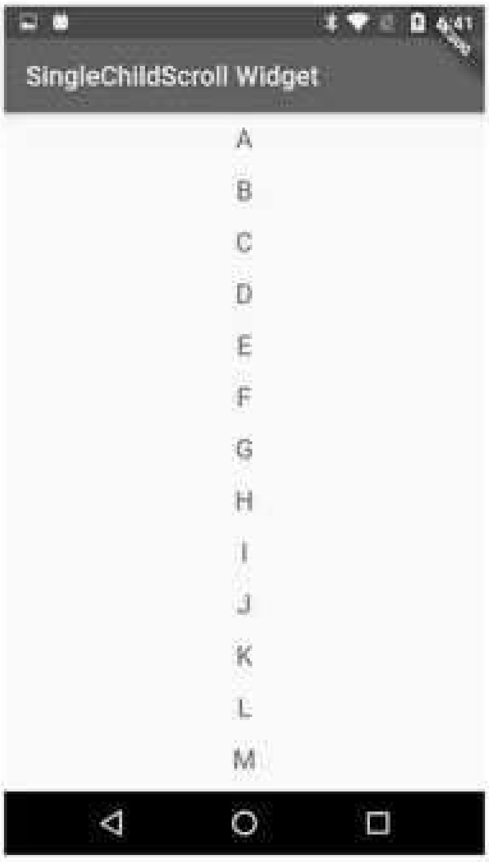
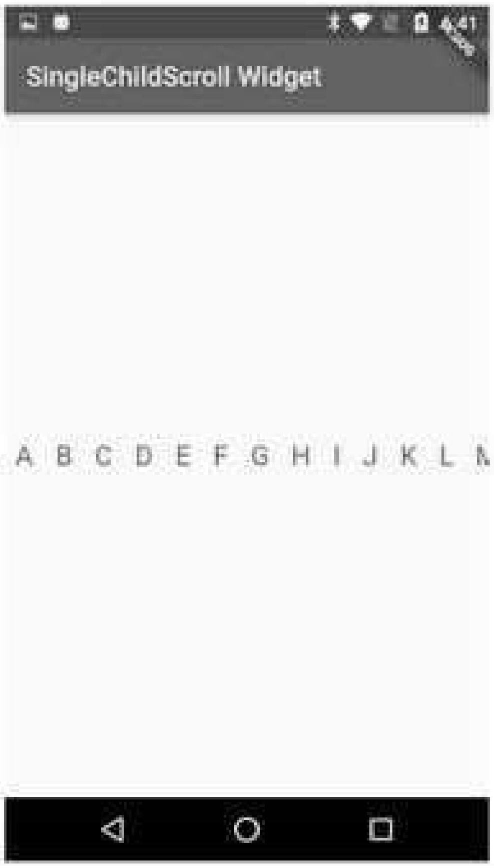
This Widget is used to show a child Widget even if there is not enough space to view the entirety of the child Widget

SingleChildScrollView is similar to scrollview in Android, and it can only contain one child element

```
const SingleChildScrollView({
   Key key,
   this.scrollDirection = Axis.vertical,
        this.reverse = false,
        this.padding,
        bool primary,
        this.physics,
        this.controller,
        this.child,
        this.dragStartBehavior = DragStartBehavior.down,
      })
```

key : the unique identifier of the current element (similar to id in Android)
scrollDirection : scroll direction, default is vertical
reverse : whether to slide in the opposite direction of the reading direction.
padding : padding distance
primary : Whether to use the default Primary ScrollController in the widget tree. When the sliding direction is vertical (scrollDirection value is Axis.vertical) and the controller is not specified, the primary defaults to true
physics : This property accepts a ScrollPhysics object, which determines how the scrollable widget responds to user operations, such as the user continuing to perform an animation after lifting the finger, or how to display it when sliding to the boundary.
controller : This property accepts a ScrollController object. The main role of ScrollController is to control the scroll position and listen for scroll events
child : child element

```
class SingleChildScrollViewWidget extends StatelessWidget{
 var alphabets="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
 @override
 Widget build(BuildContext context) {
  // TODO: implement build
  return MaterialApp(
    home: Scaffold(
    appBar: AppBar(title:Text("SingleChildScroll
Widget"),backgroundColor: Colors.pink,),
  body: horizontalScroll()
    ));
 }


 horizontalScroll()
 {
  return SingleChildScrollView(
   scrollDirection: Axis.horizontal,
   child: Center(
    child: Row(
     children: alphabets.split("").map((a)=>Padding(
      padding: const EdgeInsets.all(8.0),
      child: Text(a,style: TextStyle(fontSize: 20,color:
Colors.pink),),
     )).toList(),
    ),
   ),
  );
 }
 verticalScroll()
 {
  return SingleChildScrollView(
   scrollDirection: Axis.vertical,
   child: Center(
    child: Column(
     children: alphabets.split("").map((a)=>Padding(
      padding: const EdgeInsets.all(8.0),
      child: Text(a,style: TextStyle(fontSize: 20,color:
Colors.pink),),
     )).toList(),
    ),
   ),
  );
 }
}
```

Vertical Direction	Horizontal Direction
	

Themes

When we build a Flutter app, we build a root Widget. That Widget usually returns a MaterialApp, which builds the foundations for the app. One of the constructor arguments for MaterialApp is the Theme object. This object specifies the colors to be used in the application's Widgets. As you can see below the user can pass in Theme data into the MaterialApp constructor using a ThemeData object

Change Dynamic theme
Find example here http://rrtutors.com/description/27

Scaffold

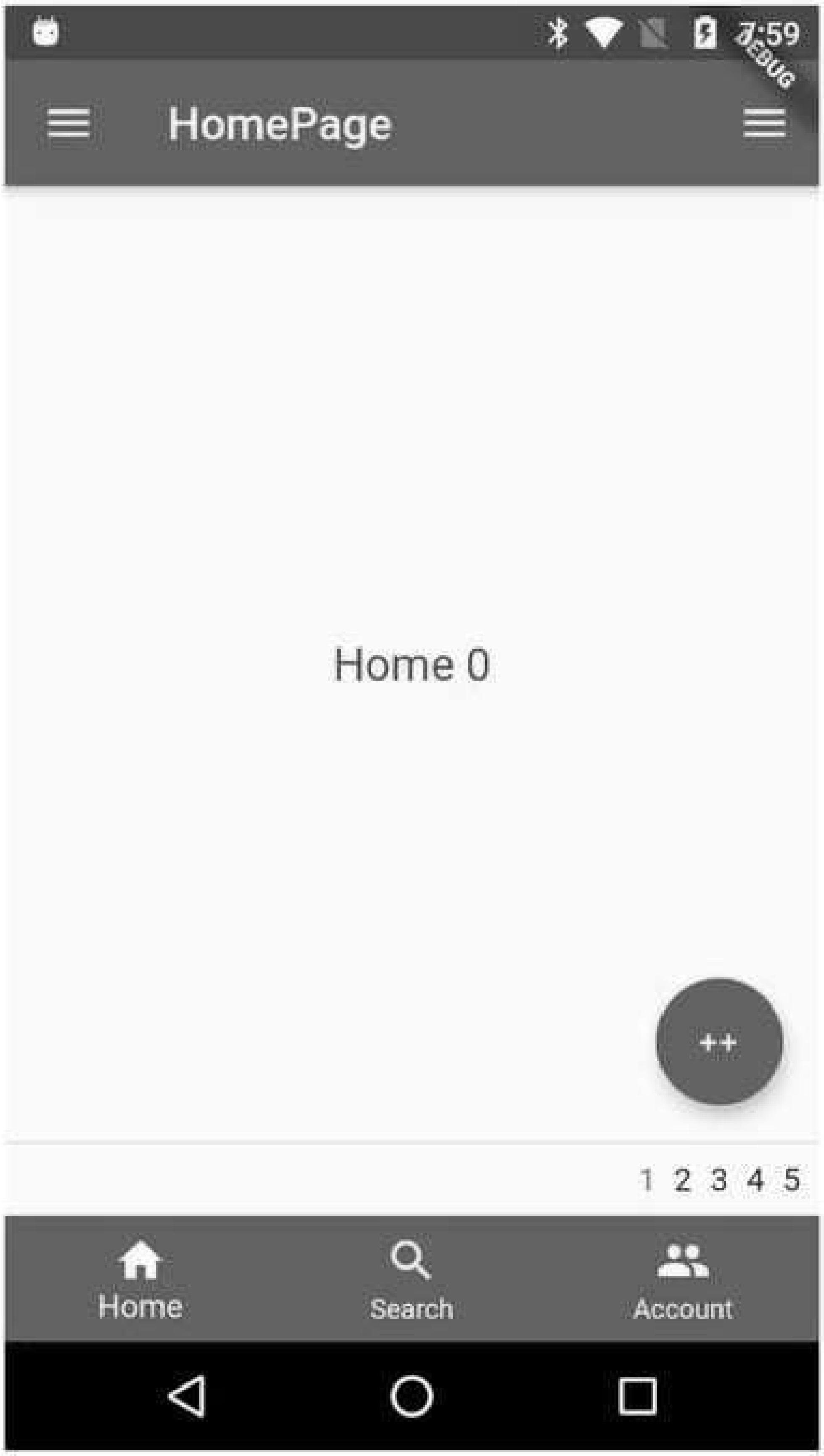
Scaffold is the page display framework
Scaffold has different attributes to handle the Pages

appBar: An AppBar displayed at the top of the interface, which is the ActionBar and Toolbar in Android
body: the main content widget displayed in the current interface
floatingActionButton: FAB defined in paper and ink design, the main function button of the interface
persistentFooterButtons: Buttons that are fixed to the bottom, such as OK and Cancel buttons below the dialog box
drawer: sidebar control
backgroundColor: The background color of the content. The default value is ThemeData.scaffoldBackgroundColor.
bottomNavigationBar: the navigation bar displayed at the bottom of the page

resizeToAvoidBottomPadding: similar to
android: windowSoftInputMode = "adjustResize" in Android,
controls whether the content body of the interface is rearranged to avoid the bottom being covered, for example, when the keyboard is displayed, the re-layout is to avoid covering the content with the keyboard. The default value is true

```
class ScffoldHomePage extends StatefulWidget {
 @override
 State<StatefulWidget> createState() {
  return ScffoldHomePageState();
 }
}
```

```
class ScffoldHomePageState extends State<ScffoldHomePage> {

 num index =0;


 List <Widget> pageWidgetList =[
  Home(),
  SearchScreen(),
  ProfileScreen(),
 ];

 @override
 Widget build(BuildContext context) {

  return Scaffold(

   appBar: AppBar(
    title: Text("HomePage"),
    backgroundColor: Colors.pink,
   ),

   body:pageWidgetList[index],

   floatingActionButton: FloatingActionButton(
    child: Text("++"),

    onPressed: () {

    },

    tooltip: "Click tooltips",

    backgroundColor: Colors.pink,

    focusColor: Colors.green,

    hoverColor: Colors.purpleAccent,
      splashColor: Colors.deepPurple,

      foregroundColor: Colors.white,

      elevation: 0.0,

      highlightElevation: 20.0,
    ),

    floatingActionButtonLocation:
FloatingActionButtonLocation.endFloat,

    persistentFooterButtons: <Widget>[
      Text(
        "1",
        style: TextStyle(color: Colors.blue),
      ),
      Text("2"),
      Text("3"),
      Text("4"),
      Text("5"),
    ],

    drawer: Container(
      color: Colors.grey,
      width: 120,
      child: FlatButton(
        child: Text("Close Left Swipe"),
        onPressed: () {
          Navigator.of(context).pop();
        },
      ),
    ),

    endDrawer: Container(
      color: Colors.orange,
      width: 200,
    height: 800,
    child: FlatButton(
     child: Text("Close Right Swipe",style: TextStyle(color:
Colors.white),),
     onPressed: () {
      Navigator.of(context).pop();
     },
    ),
   ),

   bottomNavigationBar:new BottomNavigationBar(

    backgroundColor: Colors.pink,
    items: <BottomNavigationBarItem>[

    BottomNavigationBarItem(icon:Icon(Icons.home,color:
index==0?Colors.white:Colors.white,),title: Text("Home",style:
TextStyle(color: index==0?Colors.white:Colors.white),) ),
    BottomNavigationBarItem(icon:Icon(Icons.search,color:
index==1?Colors.white:Colors.white,),title: Text("Search",style:
TextStyle(color: index==1?Colors.white:Colors.white),)  ),
    BottomNavigationBarItem(icon:Icon(Icons.people,color:
index==2?Colors.white:Colors.white,),title:
Text("Account",style: TextStyle(color:
index==2?Colors.white:Colors.white),)  ),
   ],
onTap: (flag) {
     print("flag $flag");
     index = flag;
     setState(() {});
    },
    currentIndex: index,
   ) ,
  );
 }
}
```


7:59
DEBUG
HomePage
Home 0
++
1 2 3 4 5
Home
Search
Account

Dialogs

A material design dialog, Dialogs are temporary windows that appear as overlays over the existing application

Show dialog is simple

```
showDialog(context: context,
        builder: (context) => Center(child: Text("Dialog")));
```

In flutter we have multiple ways to show dialogs

1. Alert Dialog
2. Custom Dialog
3. Full-Screen Dialog

Simple AlertDialog

```
showDialog(
  context: context,
  builder: (BuildContext context){
    return AlertDialog(
      title: Text("Alert Dialog"),
      content: Text("Dialog Content"),
    );
  }
)
```

This is a simple alert dialog with title and message. We can also add buttons to handle the events

Add buttons to Alert Dialogs

This widget, there is a parameter called action. It accepts an array of widgets and we can provide multiple buttons to that.
Those Buttons will appear in the bottom right corner of the dialog

```
actions:[
    FlatButton(
      child: Text("Close"),
    )
  ]
```

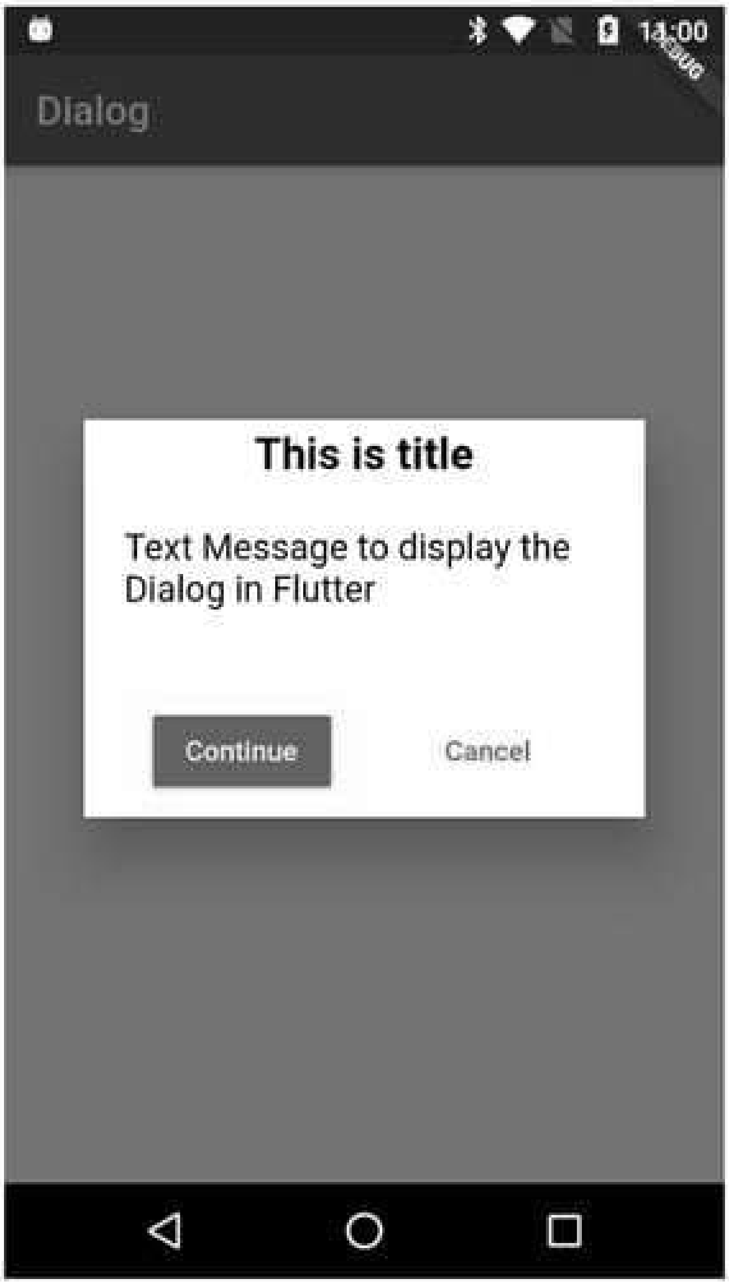
How to close Dialog

We can close the Displayed Dialog by calling the Navigator.of(context).pop();

```
FlatButton(
       child: Text("Close"),
       onPressed: (){
         Navigator.of(context).pop();
       },
     )
```

Example

```
class MyDialog extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(title: Text("Dialog"),backgroundColor:
Colors.pink,),
     body: Center(
        child: FlatButton(
          onPressed: () {
            showMyDialog(context);
          },
          child: Text(
            'Show Me',
            style: TextStyle(
                fontSize: 16.0,
                fontWeight: FontWeight.bold,
                color: Colors.white),
          ),
          color: Colors.pink,
        ),
      ),
    );
  }
  void showMyDialog(BuildContext context) {
    showDialog(
        context: context,
        barrierDismissible: false,
        builder: (BuildContext context) {
          return Dialog(
            child: _contentWidget(context),
            insetAnimationCurve: Curves.fastOutSlowIn,
            insetAnimationDuration: Duration(milliseconds: 100),
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.all(
                Radius.circular(8.0),
              ), ),);
        });
  }
  Widget _contentWidget(BuildContext context) {
    return Center(
      widthFactor: 2.0,
      heightFactor: 1.0,
      child: Container(
        width: 300.0,
        height: 200.0,
    color: Colors.white,
    child: Column(
     children: <Widget>[
       Expanded(
        child: Container(
          padding: EdgeInsets.only(top: 5.0),
          child: Text('This is title',style: TextStyle(color:
Colors.black,fontWeight: FontWeight.bold,fontSize: 22.0),),),
        flex: 1,
       ),
       Expanded(
        child: Container(
          alignment: Alignment.topLeft,
          margin: EdgeInsets.all(20.0),
          child: Text('Text Message to display the Dialog in
Flutter',style: TextStyle(fontSize: 18.0,color: Colors.black),),
        ),
        flex: 3,
       ),
       Expanded(
        child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          children: <Widget>[
           RaisedButton(onPressed: (){
            Navigator.of(context).pop();
           },
            child: Text('Continue',style: TextStyle(color:
Colors.white)),color: Colors.pink,),
           FlatButton(onPressed: (){
            Navigator.of(context).pop();
           },
            child: Text('Cancel',style: TextStyle(color:
Colors.pink),)),
          ],
        ),
        flex: 2,
       ),
     ],
            ),
          ),
        );
      }
    }
```


14:00
Dialog
This is title
Text Message to display the Dialog in Flutter
Continue
Cancel

ExpansionPanelList & ExpansionPanel

These two widgets are designed to work together to present a list of expandable panels to the user
We have to manage the state of what was expanded / collapsed and rebuild the ExpansionPanelList & ExpansionPanels everytime the state changes

ExpansionPanel

Shrink the panel. It has a title and a body that can be expanded or collapsed. The body of the panel is only visible when expanded.
The shrink panel is only used as a child of ExpansionPanelList. Example implementation, please use ExpansionPanelList

```
ExpansionPanel({
  @required this.headerBuilder,
  @required this.body,
  this.isExpanded = false,
  this.canTapOnHeader = false,
})
```

ExpansionPanelList

A material expansion panel list that lays out its children and animates expansions
Lays out the child ExpansionPanels

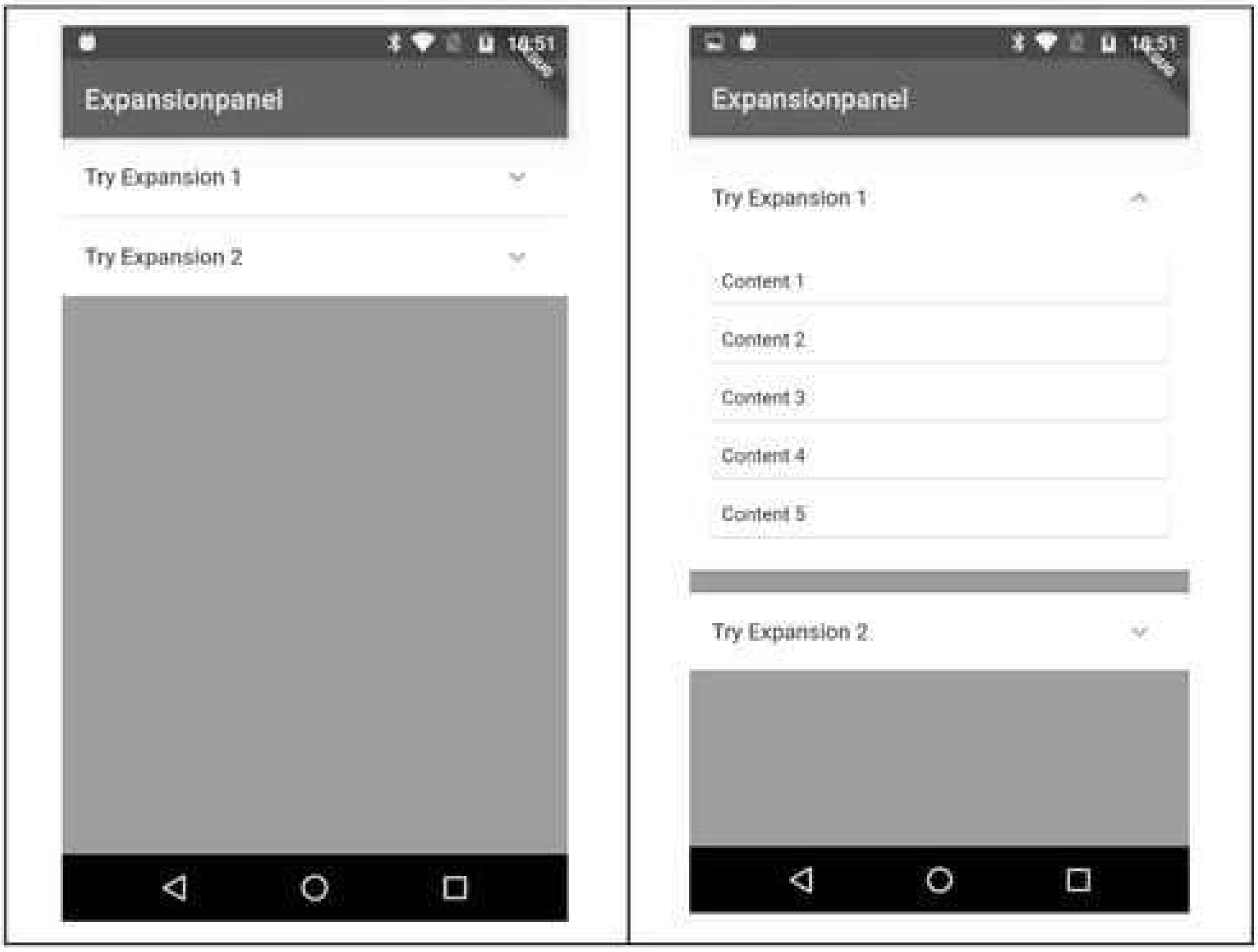
```
const ExpansionPanelList({
  Key key,
  this.children = const <ExpansionPanel>[],
  this.expansionCallback,
  this.animationDuration = kThemeAnimationDuration,
})
```

There are only three parameters that we need to use:

children: Needless to say, it is ExpansionPanel
expansionCallback: expansion callback, here will return the index of the click
animationDuration: the duration of the animation

Example:

```
class ExpansionWidget extends StatefulWidget{
 @override
 State<StatefulWidget> createState() {
  // TODO: implement createState
  return ExpansionWidgetState();
 }

}
class ExpansionWidgetState extends State<ExpansionWidget>{
 List<bool>listExpans=List();

 @override
 void initState() {
  // TODO: implement initState
  super.initState();
  listExpans.add(false);
  listExpans.add(false);
 }
 @override
 Widget build(BuildContext context) {
  // TODO: implement build
  return Scaffold(
   backgroundColor: Colors.grey,
   appBar: AppBar(title:
Text("Expansionpanel"),backgroundColor: Colors.pink,),
   body: SingleChildScrollView(
    child: Container(
        alignment: Alignment.center,
        child: Column(
         children: <Widget>[
          ExpansionPanelList(
           children : <ExpansionPanel>[
            ExpansionPanel(
             headerBuilder:(context, isExpanded){
              return ListTile(
               title: Text('Try Expansion 1'),
              );
             },
             body: Padding(
              padding: EdgeInsets.fromLTRB(15, 0, 15, 15),
              child: ListBody(
               children: <Widget>[
                Card(
                 margin:EdgeInsets.fromLTRB(0, 0, 0, 10),
                 child: Padding(padding: EdgeInsets.all(8),child:
Text('Content 1'),),
                ),
                Card(
                 margin:EdgeInsets.fromLTRB(0, 0, 0, 10),
                 child: Padding(padding: EdgeInsets.all(8),child:
Text('Content 2'),),
                ),
                Card(
                 margin:EdgeInsets.fromLTRB(0, 0, 0, 10),
                 child: Padding(padding: EdgeInsets.all(8),child:
Text('Content 3'),),
                ),
                Card(
                 margin:EdgeInsets.fromLTRB(0, 0, 0, 10),

                 child: Padding(padding: EdgeInsets.all(8),child:
Text('Content 4'),),
                ),
                Card(
                 margin:EdgeInsets.fromLTRB(0, 0, 0, 10),
              child: Padding(padding: EdgeInsets.all(8),child:
Text('Content 5'),),
             ),
            ],
           ),
          ),
          isExpanded: listExpans[0],
          canTapOnHeader: true,
         ),

         ExpansionPanel(

          headerBuilder:(context, isExpanded){
           return ListTile(
            title: Text('Try Expansion 2 '),
           );
          },
          body: Padding(
           padding: EdgeInsets.fromLTRB(15, 0, 15, 15),
           child: ListBody(
            children: <Widget>[
             Card(
              margin:EdgeInsets.fromLTRB(0, 0, 0, 10),
              child: Padding(padding: EdgeInsets.all(8),child:
Text('Content 1'),),
             ),
             Card(
              margin:EdgeInsets.fromLTRB(0, 0, 0, 10),
              child: Padding(padding: EdgeInsets.all(8),child:
Text('Content 2'),),
             ),
             Card(
              margin:EdgeInsets.fromLTRB(0, 0, 0, 10),
              child: Padding(padding: EdgeInsets.all(8),child:
Text('Content 3'),),
             ),
             Card(
              margin:EdgeInsets.fromLTRB(0, 0, 0, 10),
                    child: Padding(padding: EdgeInsets.all(8),child:
Text('Content 4'),),
                  ),
                  Card(
                    margin:EdgeInsets.fromLTRB(0, 0, 0, 10),
                    child: Padding(padding: EdgeInsets.all(8),child:
Text('Content 5'),),
                  ),
                ],
              ),
            ),
            isExpanded: listExpans[1],
            canTapOnHeader: true,
          ),

        ],
        expansionCallback:(panelIndex, isExpanded){
          setState(() {
            listExpans[panelIndex] = !isExpanded;
          });
        },
        animationDuration : kThemeAnimationDuration,
      ),
    ],
   ),
  ),
 )
);


}

}
```



GridView

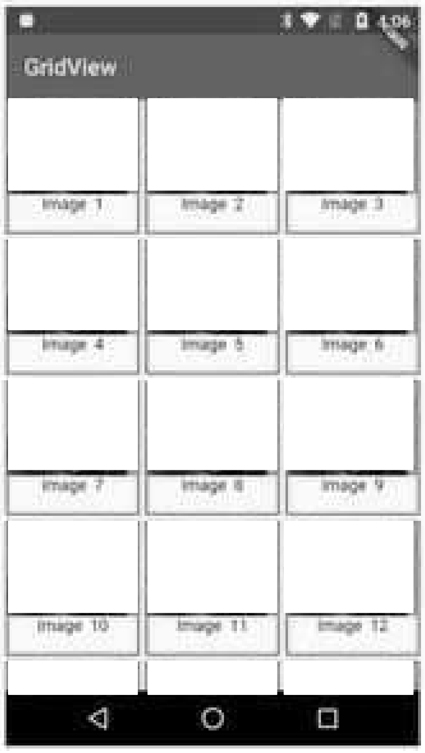
GridView displays the values of a data source in a table where each column represents a field and each row represents a record

Constructors
GridView.builder()
GridView.count()
GridView.custom()
GridView.extent()

```
GridView.builder({
 Key key,
 Axis scrollDirection = Axis.vertical,
 bool reverse = false,
 ScrollController controller,
 bool primary,
 ScrollPhysics physics,
 bool shrinkWrap = false,
 EdgeInsetsGeometry padding,
 @required this.gridDelegate,
 @required IndexedWidgetBuilder itemBuilder,
 int itemCount,
 bool addAutomaticKeepAlives = true,
 bool addRepaintBoundaries = true,
 bool addSemanticIndexes = true,
 int semanticChildCount,
})
```

Example

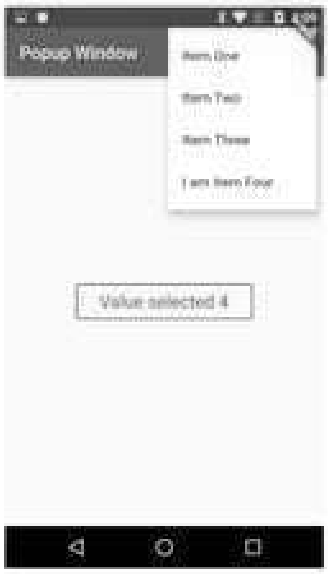
```
class GridViewWidget extends StatelessWidget {
 List list = new List();
 GridViewWidget() {
  for (int i = 1; i < 20; i++) {
   int j = (i % 9) + 1;
   var temp = {
    "imageurl":
"https://cdn.pixabay.com/photo/2016/10/17/17/41/
              -1748248_960_720.jpg",
    "title": "Image  $i"
   };
   list.add(temp);
  }
 }
 @override
 Widget build(BuildContext context) {
    // TODO: implement build
    return Scaffold(
      appBar: AppBar(
       title: Text("GridView"),
       backgroundColor: Colors.pink,
      ),
      body: GridView.builder(
        gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
          crossAxisCount: 3,
          mainAxisSpacing: 5,
          crossAxisSpacing: 5,
        ),
        itemCount: list.length,
        itemBuilder: (BuildContext context, int index) {
          return Container(
            decoration: BoxDecoration(
               border: Border.all(
              color: Colors.red,
              width: 2,
            )),
            child: Column(
             mainAxisSize: MainAxisSize.max,
              mainAxisAlignment: MainAxisAlignment.end,
              children: <Widget>[
                Image.network(
                 list[index]['imageurl'],
                 fit: BoxFit.cover,
                ),
                Expanded(child: Text(list[index]['title'])),
              ],
            ),
          );
        }));
  }
  }
```



PopupMenu

Why use this PopupMenu? Because most message interfaces have a setting option in the upper right corner, and this option is most commonly implemented through PopupMenu.

Look at the effect map



```
class PopupWidget extends StatefulWidget{
 @override
 State<StatefulWidget> createState() {
  // TODO: implement createState
  return PopupWidgetState();
 }

}

class PopupWidgetState extends State<PopupWidget>
{
 int _value=1;

 @override
 Widget build(BuildContext context) {
  // TODO: implement build
  return Scaffold(
    appBar: AppBar(title: Text("Popup
Window"),backgroundColor: Colors.pink,

    actions: <Widget>[
     _NomalPopMenu()
    ],),
    body: Container(
      child: Center(
       child: Container(
          decoration:ShapeDecoration(shape: OutlineInputBorder(

          )),
          width: 200,
          height: 40,
         child: Center(child: Text("Value selected $_value",style:
TextStyle(color: Colors.pink,fontSize: 20),)),
       ),
      ),
    ),
  );
```