Widgets are building blocks with which you compose the user interface for your app. A button would be a widget, so would a text output be.

You also have “invisible” widgets which you don’t see directly on the screen but which you use to structure and control other (visible) widgets. For example, there is a “Column” widget in Flutter, with which you can order other widgets (e.g. buttons or text widgets) in a column, sitting above each other.

You build a “widget tree”, which means, that you have one top-most widget (often the “MaterialApp” widget) which then takes other widget(s) as arguments - and so on.

You also build your own widgets in Flutter - actually, your entire app is wrapped in one core widget that you build. Your widgets than are composed from other built-in or custom widgets.

A basic Flutter app could look like this:

Widget build(BuildContext context) { // each widget has a build() method

  return MaterialApp(

    home: Scaffold(

      body: Column(

        children: [

          Text(‘This gets drawn on the screen!’),

          RaisedButton(child: Text(‘Tap me!’), onPressed: () {}),

        ],

      ),

    ),

  );

}

In this example, you can also see the widget tree: We have widgets nested into each other - MaterialApp, Scaffold, Column, Text and RaisedButton are all widgets built-into Flutter. And we pass them all into each other as named arguments in their constructors. For example, the Column() widget has a children argument which takes a list of other widgets (Text and RaisedButton in this example).

A Flutter app starts by first executing the main() function in your main.dart file - this is done automatically (you don’t have to do anything for that to happen).

Inside of the main() function, you need to call the runApp() function which is builtinto Flutter (that’s why you also need to import the Flutter package into the file where you run runApp(): import ‘package:flutter/material.dart’ enables this function).

runApp() expects to get a (custom) widget which in turn is a Dart class that extends either StatelessWidget or StatefulWidget - classes which you can extend since they’re made available by the Flutter material.dart file/ package.

You instantiate your custom widget and pass the instance/ object as an argument to runApp().

Your custom widget then needs to have a build() method which returns a Widget in turn. All custom widgets need to have this build() method - StatelessWidget / StatefulWidget (which you have to extend) force you to have one.

The Widget returned by your build() method is then what Flutter draws onto the screen of your app. It calls the build() method for you, you don’t do that manually.

* runApp(MyApp())

void runapp(Widget app)

Takes a widget as argument, MyApp() in this case, and runs the app

* MyApp()- Calls build method
* State- the state of the widget is the information of the objects that its properties are holding at the time of its creation
* StatelessWidget – Widgets that cannot change their state dynamically, ie widgets do not get redrawn during runtime of the app. UI gets re-rendered only when input data changes and not when local state changes.

class MyApp extends StatelessWidget {

//…

}

* StatefulWidget- Widget that can change their state dynamically when user interacts during runtime of the app, ie UI gets re-rendered when either input data or local state changes.

It consists of two classes MyApp and MyAppState

class MyApp extends StatefulWidget {         /\* This class gets recreated when external

                                              data changes \*/

  @override

  State<StatefulWidget> createState() {

    // TODO: implement createState

    return \_MyAppState();

  }

}

class \_MyAppState extends State<MyApp> {     /\* The data inside this class

                                                does not get rebuilt \*/

  var \_questionIndex = 0;

  void \_answerQuestion() {

    setState(() //Calls build again and re renders UI {

      \_questionIndex = \_questionIndex + 1; //contains data which is to be altered, in

    }); this case it is \_questionIndex

    print(\_questionIndex);

  }

//…

}

* Stateless vs Stateful

Flutter has two types of widgets you can build.

Widgets which extend StatelessWidget are meant to present data. They return a widget (tree) which simply outputs some data. You can’t change internal data (i.e. class properties) in your widget’s class and re-render the UI upon such a change.

Widgets which extend StatefulWidget also return a widget (tree) but there, you can also change some internal data (class properties) in a way such that the user interface is re-built - i.e. your changes are reflected in your app’s user interface.

* build method

Flutter calls the build() method when drawing your widgets onto the app’s screen. The build() method returns a widget => The content Flutter should draw onto the screen.

The build method is called any time when setState is called, when widget's **dependencies** update, or any of the parent widgets (like Materialapp, Scaffold) are rebuilt.

[Widget](https://api.flutter.dev/flutter/widgets/Widget-class.html) build(

[BuildContext](https://api.flutter.dev/flutter/widgets/BuildContext-class.html) context

)

Widget : return type of the build method

* BuildContext:

BuildContext is the object type and context is the object here

BuildContext contains information about the location in the tree at which this widget is being built. BuildContext objects are passed to WidgetBuilder functions in this case it is StatelessWidget.build

* MaterialApp

A class

Wraps widgets for material design application

Takes arguments for the MaterialApp constructor

A Material app starts with the [MaterialApp](https://api.flutter.dev/flutter/material/MaterialApp-class.html) widget, which builds a number of useful widgets at the root of your app, including a [Navigator](https://api.flutter.dev/flutter/widgets/Navigator-class.html)(helps in smooth transition between screens of application)

Properties:

color

darkTheme

home- This is the route that is displayed first when the application is started normally

title

theme

return MaterialApp(

      home: Scaffold(

        appBar: AppBar(

          title: Text('My First App'),

        ),

        body: Column(

          children: [

//…

],

        ),

      ),

);

* Scaffold

Takes a number of different widgets as named arguments

Occupies the available space ie the whole screen.

Provide a [framework](https://www.geeksforgeeks.org/software-framework-vs-library/) to implement the basic material design layout of the application.

Properties

appBar- horizontal bar displayed at the top of the Scaffold widget

backgroundColor

body- displays main content in Scaffold. Signifies place below appBar and the widgets inside the body are positioned at the top-left of the available space by default.

* AppBar

AppBar is also a built-in class or widget in flutter.

App bars are typically used in the [Scaffold.appBar](https://api.flutter.dev/flutter/material/Scaffold/appBar.html) property, which places the app bar as a fixed-height widget at the top of the screen

* SilverAppBarwidget

gives scrollable functionality to the app bar.

Properties of AppBar

actions

title

backgroundColor

elevation

shape

centerTitle

bottomOpacity

foregroundColor

* Text

The [Text](https://api.flutter.dev/flutter/widgets/Text-class.html) widget displays a string of text with single style. The string might break across multiple lines or might all be displayed on the same line depending on the layout constraints.

Properties

data- text to display

style

textAlign

NOTE: hover over any widget or press Ctrl + space inside any widget to see the arguments it can take

---------------------------------------------------------------------------------------------------------------------------------------------------------

OLD BUTTON WIDGETS VS NEW BUTTON WIDGETS

Old Buttons New Buttons

* RaisedButton ElevatedButton
* FlatButton TextButton
* OutlineButton OutlinedButton

Advantage of ElevatedButton over RaisedButton – It automatically picks the theme color if the color is not mentioned

Differences in using Old Buttons and New Buttons:

RaisedButton and ElevatedButton

RaisedButton(

            child: Text('RaisedButton'),

            color: Color.fromARGB(255, 255, 164, 59),

            textColor: Colors.black,

            onPressed: () {}, //or onPressed: null,

          ),

ElevatedButton(

            child: Text('ElevatedButton'),

            style: ButtonStyle(

              backgroundColor: MaterialStateProperty.all(Colors.blue),

              foregroundColor: MaterialStateProperty.all(Colors.white),

            ),

            /\* or

            ElevatedButton(

              style: ElevatedButton.styleFrom(

                primary:Colors.blue,

                onPrimary:Colors.white,

              ),

            ), \*/

            onPressed: null,

          ),

FlatButton and TextButton

FlatButton(

            child: Text('FlatButton'),

            color: Colors.blue,

            textColor: Colors.black,

            onPressed: () {

              print('Pressed Flat Button');

            },

          ),

TextButton(

            child: Text('TextButton'),

            style: ButtonStyle(

              backgroundColor: MaterialStateProperty.all(Colors.blue),

              foregroundColor: MaterialStateProperty.all(Colors.white),

            ),

            /\* or

              style: ElevatedButton.styleFrom(

                primary:Colors.blue,

                onPrimary:Colors.white,

              ), \*/

            onPressed: () {

              print('Pressed Text Button');

            },

          ),

Similarly for OutlineButton and OutlinedButton

**SECTION -3**

DART DEVTOOLS

Ctrl+shift+P -> Dart: Open DevTools

* Click on any widget in the widget tree to learn more about it
* Debug Page -> adds helper lines in the UI
* Perfoemance Overlay -> shows the performance
* Debug barrier -> click to remove the debug barrier from the screen
* Repaint Rainbow -> when activated, uses green borders to show what all has been repainted

The more often something gets repainted, the more often color around it changes

**SECTION - 4**

SOME IMPORTANT WIDGETS

* App/Page Setup
* MaterialApp/CupertinoApp

Typically the root widget in your app

Does a lot of “behind-the-scenes” setup work for your app

Allows you to configure a global theme for your app

Sets up navigation behavior (e.g. animations) for your app

* Scaffold/CupertinoPageScaffold

Typically used as a frame for a page in your app

Provides a background, app bar, navigation tabs, etc

Only use one scaffold per page

* Layout
* Container
* Row
* Column
* Row/Column Children
* Flexible
* Expanded

Widgets to wrap around child to configure how much space should child widget consume

* Content Containers
* Stack

Used to position items on top of each other (along the Z axis)

Widgets can overlap

You can position items in absolute space (i.e. in a coordinate space) via the Positioned() widget

Eg an image with text in foreground

* Card

A container with some default styling (shadow, background color, rounded corners)

Can take one child (can be anything)

Typically used to output a single piece / group of information

* Repeat Elements
* ListView –

Used to output lists of items

Like a Column() but scrollable (Column is not)

Can be laid out vertically (default) and horizontally

Use ListView.builder() to get optimized item rendering for very long lists

* GridView

Combination of rows and columns which is scrollable

* ListTile-

A pre-styled container / Row() that allows you to achieve a typical “list-item look”

Offers various slots for widgets (e.g. at the beginning, a title, at the end)

Not a must-use but can be handy for a default list-item look

**ListView(children: [] )**is a very important widget in a flutter. It is used to create the list of children but when we want to create a list recursively without writing code again and again then ListView.builder() is used instead of ListView.  **ListView.builder()** creates a scrollable, linear array of widgets.

* Content Types
* Text

A widget that simply outputs some text on the screen

Text can be styled (font family, font weight, font size etc)

Text behavior can be controlled (e.g. clipping if it’s too long)

* Image

Used to render an image on the screen

Supports different sources (included in app, web image, …)

Can be configured to size itself in different ways into a wrapping container

* Icon

Renders an Icon onto the screen

Flutter ships with many default Material (Android) and iOS icons which you can use

There also is an IconButton() widget in case you need a button with an icon

* User Input
* TextField- allows user to enter text
* Custom widgets/buttons
* GestureDetector (invisible)
* InkWell (partly invisible)

Combining Widgets

A card by default assumes the size of its child. If we wrap Text in Container, the size of card will depend on the size of container

If we wrap Card inside a container, the size of card will depend on the size of Container, i.e. it will take the size of its parent and not child.