Flutter QA

1. **Current Varsion of Flutter: 3.10**
2. **Flutter Useful command:**
   1. **where flutter** => will give the path of flutter SDK
   2. **flutter doctor** => Help to check the system & software / depandancy installtion
   3. **flutter doctor** --android-licenses => To agree the Android Licence
   4. **flutter create my\_app\_name** => to create a flutter app by using command
   5. **open -a Simulator** => to check the open / connect simulater or device
   6. **flutter run** => To run the app
   7. **flutter get** => to
   8. **Flutter run – release** => to build the release apk

**What is Dart:**

Dart is an open-source general-purpose programming language. It is 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 language supports the following data types −**

* **Numbers** − It is used to represent numeric literals – Integer and 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 and Maps** − It is used to represent a collection of objects. A simple List can be defined as below –
* **Dynamic** − If the variable type is not defined, then its default type is dynamic. The following example illustrates the dynamic type variable −
* void main() {

* //Variables
* var myName = ‘Abhishek’
* //Final & Constant
* final a = 12;
* const pi = 3.14;
* //List
* var list = [1,2,3,4,5];
* print(list);

* //Map
* var mapping = {'id': 1,'name':'Dart'};
* print(mapping);
* //Funtion
* void main() {
* add(3,4);
* }

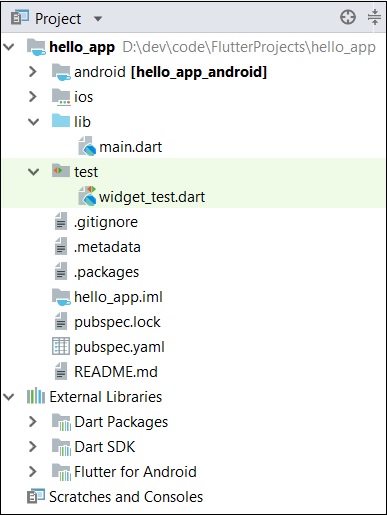
* //Loop
* for( var i = 1 ; i <= 10; i++ ) {
* if(i%2==0) {
* print(i);
* }
* }

* }

**Create The Project from Android Studio:**

Android Studio creates a fully working flutter application with minimal functionality. Let us check the structure of the application and then, change the code to do our task.

The structure of the application and its purpose is as follows −



Various components of the structure of the application are explained here −

* **android** − Auto generated source code to create android application
* **ios** − Auto generated source code to create ios application
* **lib** − Main folder containing Dart code written using flutter framework
* **ib/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

**Replace the dart code in the *lib/main.dart file* with the below code −**

import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {

// This widget is the root of your application.

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Hello World Demo Application',

theme: ThemeData(

primarySwatch: Colors.blue,

),

home: MyHomePage(title: 'Home page'),

);

}

}

class MyHomePage extends StatelessWidget {

MyHomePage({Key key, this.title}) : super(key: key);

final String title;

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text(this.title),

),

body: Center(

child:

Text(

'Hello World',

)

),

);

}

}

Let us understand the dart code line by line.

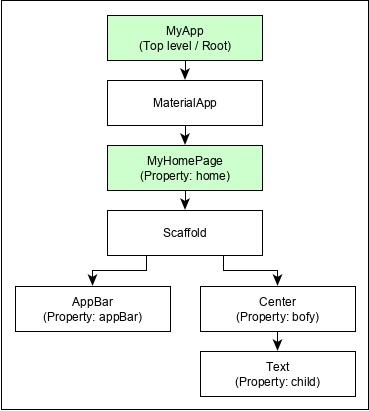
* **Line 1** − imports the flutter package, *material*. The material is a flutter package to create user interface according to the Material design guidelines specified by Android.
* **Line 3** − This is the entry point of the Flutter application. Calls *runApp* function and pass it an object of *MyApp* class. The purpose of the *runApp* function is to attach the given widget to the screen.
* **Line 5-17** − Widget is used to create UI in flutter framework. *StatelessWidget* is a widget, which does not maintain any state of the widget. *MyApp* extends *StatelessWidget* and overrides its *build method*. The purpose of the *build* method is to create a part of the UI of the application. Here, *build* method uses *MaterialApp*, a widget to create the root level UI of the application. It has three properties - *title, theme* and *home*.
  + *title* is the title of the application
  + *theme* is the theme of the widget. Here, we set *blue* as the overall color of the application using *ThemeData* class and its property, *primarySwatch*.
  + home is the inner UI of the application, which we set another widget, **MyHomePage**
* **Line 19 - 38** − *MyHomePage* is same as *MyApp* except it returns *Scaffold* Widget. *Scaffold* is a top level widget next to *MaterialApp* widget used to create UI conforming material design. It has two important properties, *appBar* to show the header of the application and body to show the actual content of the application. *AppBar* is another widget to render the header of the application and we have used it in *appBar* property. In *body* property, we have used *Center* widget, which centers it child widget. *Text* is the final and inner most widget to show the text and it is displayed in the center of the screen.

**Widgets**

The core concept of the Flutter framework is **In Flutter, Everything is a widget**. Widgets are basically user interface components used to create the user interface of the application.

In *Flutter*, the application is itself a widget. The application is the top- level widget and its UI is build using one or more children (widgets), which again build using its children widgets. This **composability** feature helps us to create a user interface of any complexity.

For example, the widget hierarchy of the hello world application (created in previous chapter) is as specified in the following diagram −



Here the following points are worth notable −

* ***MyApp*** is the user created widget and it is build using the Flutter native widget, *MaterialApp*.
* ***MaterialApp*** has a home property to specify the user interface of the home page, which is again a user created widget, *MyHomePage*.
* *MyHomePage* is build using another flutter native widget, *Scaffold*
* ***Scaffold*** has two properties – *body* and *appBar*
* ***body*** is used to specify its main user interface and *appBar* is used to specify its header user interface
* *Header UI* is build using flutter native widget, *AppBar* and *Body UI* is build using *Center* widget.
* The *Center* widget has a property, *Child*, which refers the actual content and it is build using *Text* widget

## Widget Build Visualization

In *Flutter*, widgets can be grouped into multiple categories based on their features, as listed below −

* Platform specific widgets
* Layout widgets
* State maintenance widgets
* Platform independent / basic widgets

**Platform specific widgets**

Flutter has widgets specific to a particular platform - Android or iOS.

**Android** specific widgets are designed in accordance with ***Material design*** *guideline* by Android OS. Android specific widgets are called as *Material widgets*.

**iOS** specific widgets are designed in accordance with *Human Interface Guidelines* by Apple and they are called as ***Cupertino* widgets**.

**Some of the most used material(Android) widgets are as follows −**

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

**Some of the most used *Cupertino(IOS)* widgets are as follows −**

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

### Layout widgets

In Flutter, a widget can be created by composing one or more widgets. To compose multiple widgets into a single widget, *Flutter* provides large number of widgets with layout feature. For example, the child widget can be centered using *Center* widget.

Some of the popular layout widgets are as follows −

* **Container** − A rectangular box decorated using *BoxDecoration* widgets with background, border and shadow.
* **Center** − Center its child widget.
* **Row** − Arrange its children in the horizontal direction(**Left To Righ**t - X-Axis).
* **Column** −Arrange its children in the vertical direction(**Top To Buttom - Y - Axis**).
* **Stack** − Arrange one above the another.

### State maintenance widgets

In Flutter, all widgets are either derived from *StatelessWidget* or *StatefulWidget*.

Widget derived from ***StatelessWidget* does not have any state information** but it may contain widget derived from *StatefulWidget*. The dynamic nature of the application is through interactive behavior of the widgets and the state changes during interaction. For example, tapping a counter button will increase / decrease the internal state of the counter by one and reactive nature of the *Flutter* widget will auto re-render the widget using new state information

**Type of Layout Widgets**

Layout widgets can be grouped into two distinct category based on its child −

* Widget supporting a single child
* Widget supporting multiple child

## Single Child Widgets

In this category, widgets will have only one widget as its child and every widget will have a special layout functionality.

* **Container** − Generic, single child, box based container widget with alignment, padding, border and margin along with rich styling features.
* **Center** − Simple, Single child container widget, which centers its child widget.
* **FittedBox** − It scales the child widget and then positions it according to the specified fit.
* **AspectRatio** − It attempts to size the child widget to the specified aspect ratio
* ConstrainedBox
* Baseline
* FractinallySizedBox
* IntrinsicHeight
* IntrinsicWidth
* LiimitedBox
* OffStage
* OverflowBox
* SizedBox
* SizedOverflowBox
* Transform
* CustomSingleChildLayout

## Multiple Child Widgets

In this category, a given widget will have more than one child widgets and the layout of each widget is unique.

For example, *Row* widget allows the laying out of its children in horizontal direction, whereas *Column* widget allows laying out of its children in vertical direction. By composing *Row* and *Column*, widget with any level of complexity can be built.

Let us learn some of the frequently used widgets in this section.

* **Row** − Allows to arrange its children in a horizontal manner(**Left To Right**).
* **Column** − Allows to arrange its children in a vertical manner(**Top To uttom**).
* **ListView** − Allows to arrange its children as list.
* **GridView** − Allows to arrange its children as gallery.
* **Expanded** − Used to make the children of Row and Column widget to occupy the maximum possible area.
* **Table** − Table based widget.
* **Flow** − Flow based widget.
* **Stack** − Stack based widget.

**How to Move one Page to another page:**

//To navigae to another page

Navigator.push(

context,

MaterialPageRoute(builder: (context) => const SecondRoute()),

);

//To navigae to another page with data passing

Navigator.push(

context,

MaterialPageRoute(builder: (context) => const SecondRoute(‘Abhishek’)),

);

//For back to Previous Page

Navigator.of(context).pop();

# Flutter Scaffold

The Scaffold is a widget in Flutter used to implements the basic material **design visual layout structure**. It is quick enough to create a general-purpose mobile application and contains almost everything we need to create a functional and responsive [Flutter](https://www.javatpoint.com/flutter) apps

It provides many widgets or APIs for showing Drawer, SnackBar, BottomNavigationBar, AppBar, FloatingActionButton, body of app, and many more.

**constructor and properties**

**const** Scaffold({

Key key,

**this**.appBar,

**this**.body,

**this**.floatingActionButton,

**this**.floatingActionButtonLocation,

**this**.persistentFooterButtons,

**this**.drawer,

**this**.endDrawer,

**this**.bottomNavigationBar,

**this**.bottomSheet,

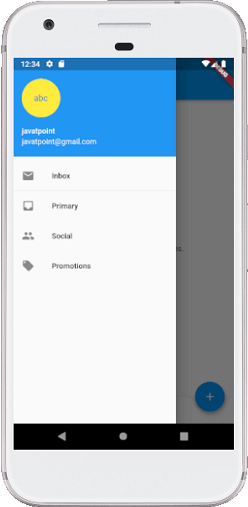
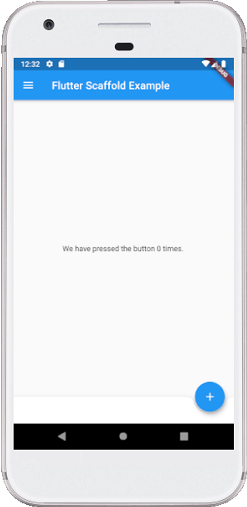
**this**.floatingActionButtonAnimator,

**this**.backgroundColor,

**this**.resizeToAvoidBottomPadding = **true**,

**this**.primary = **true**,

})



**AppBar**

***AppBar*** is usually the topmost component of the app (or sometimes the bottom-most), it contains the toolbar and some other common action buttons.

### Key Properties of Appbar Widget:

* **actions:**This property takes in a list of widgets as a parameter to be displayed after the title if the *AppBar* is a row*.*
* **title:** This property usually takes in the main widget as a parameter to be displayed in the AppBar.
* **backgroundColor:** This property is used to add colors to the background of the *Appbar.*
* **elevation:**This property is used to set the z-coordinate at which to place this app bar relative to its parent.
* **shape:**This property is used to give shape to the *Appbar*and manage its shadow.

      appBar: AppBar(

        title: const Text("GeeksforGeeks"),

        actions: <Widget>[

          IconButton(

            icon: const Icon(Icons.comment),

            tooltip: 'Comment Icon',

            onPressed: () {},

          ), //IconButton

          IconButton(

            icon: const Icon(Icons.settings),

            tooltip: 'Setting Icon',

            onPressed: () {},

          ), //IconButton

        ], //<Widget>[]

        backgroundColor: Colors.greenAccent[400],

        elevation: 50.0,

        leading: IconButton(

          icon: const Icon(Icons.menu),

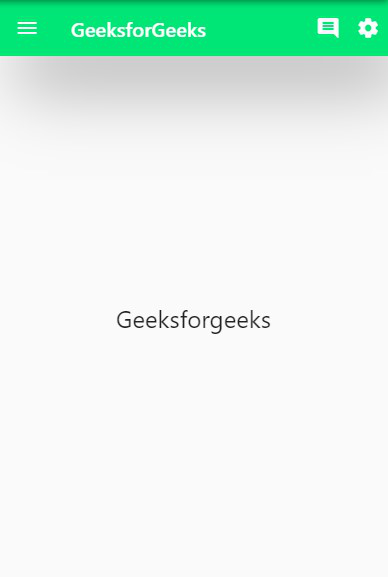
          tooltip: 'Menu Icon',

          onPressed: () {},

        ),

        systemOverlayStyle: SystemUiOverlayStyle.light,

      ),



**Body**

**body:** It is the other primary and required property of this widget, which will **display the main content in the Scaffold**. It signifies the place below the appBar and behind the floatingActionButton & drawer. The widgets inside the body are positioned at the top-left of the available space by default. See the below code:

 body: Center(

     child: Text("Welcome to Javatpoint",

         style: TextStyle( color: Colors.black, fontSize: 30.0,

         ),

         ),

**BottomNavigationBar**

This property is like a **menu that displays a navigation bar** at the bottom of the Scaffold. It can be seen in most of the mobile applications. This property allows the developer to add multiple icons or texts in the bar as items. It should be rendered below the body and persistentFooterButtons. See the below code:

bottomNavigationBar: BottomNavigationBar(

  currentIndex: 0,

  fixedColor: Colors.grey,

  items: [

    BottomNavigationBarItem(

      title: Text("Home"),

      icon: Icon(Icons.home),

    ),

    BottomNavigationBarItem(

      title: Text("Search"),

      icon: Icon(Icons.search),

    ),

  ],

  onTap: (**int** itemIndex){

    setState(() {

      \_currentIndex = itemIndex;

    });

  },

),

**Drawer**

**drawer:** It is a **slider panel** that is displayed at the side of the body. Usually, it is hidden on the mobile devices, but the user can swipe it left to right or right to left to access the drawer menu. It uses the **Drawer widget properties** slides in a **horizontal direction** from the Scaffold edge to show navigation links in the application. An appropriate **icon** for the drawer is set automatically in an appBar property. The **gesture** is also set automatically to open the drawer. See the following code

drawer: Drawer(

        child: ListView(

        children: **const** <Widget>[

        DrawerHeader(

            decoration: BoxDecoration(

            color: Colors.red,

            ),

            child: Text(

            'Welcome to Javatpoint',

            style: TextStyle(

                color: Colors.green,

                fontSize: 30,

            ),

              ),

        ),

        ListTile(

            title: Text('1SendBox'),

        ),

        ListTile(

            title: **new** Text("All Mail Inboxes"),

            leading: **new** Icon(Icons.mail),

        ),

        Divider(

            height: 0.2,

        ),

        ListTile(

            title: **new** Text("Primary Box"),

        ),

        ],

         ),

  ),

# Container

The container in Flutter is a **parent widget that can contain multiple child widgets** and manage them efficiently through width, height, padding, background color, etc. It is a widget that combines common painting, positioning, and sizing of the child widgets.

A basic container has a margin, border, and padding properties surrounding its child widget, as shown in the below image:

Container({Key key,

           AlignmentGeometry alignment,

           EdgeInsetsGeometry padding,

           Color color,

**double** width,

**double** height,

           Decoration decoration,

           Decoration foregroundDecoration,

           BoxConstraints constraints,

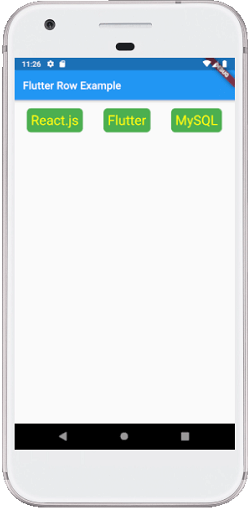
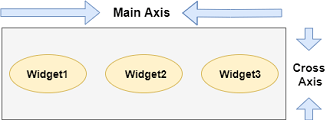
           Widget child,

           Clip clipBehavior: Clip.none

});

**ROW**:

**This widget arranges its children in a horizontal direction on the screen**. In other words, it will expect child widgets in a horizontal array. If the child widgets need to fill the available horizontal space, we must wrap the children widgets in an Expanded widget..

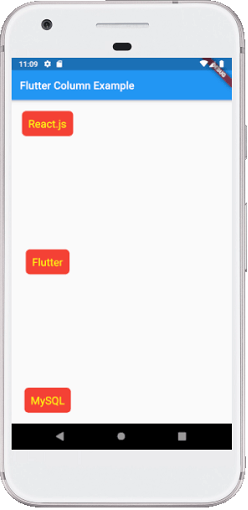
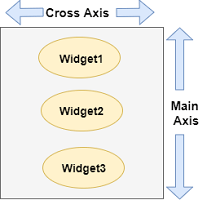


We can align the row's children widget with the help of the following properties:

* **start:** It will place the children from the starting of the main axis.
* **end:** It will place the children at the end of the main axis.
* **center:** It will place the children in the middle of the main axis.
* **spaceBetween:** It will place the free space between the children evenly.
* **spaceAround:** It will place the free space between the children evenly and half of that space before and after the first and last children widget.
* **spaceEvenly:** It will place the free space between the children evenly and before and after the first and last children widget.

**Column**

**This widget arranges its children in a vertical direction on the screen**. In other words, it will expect a vertical array of children widgets. If the child widgets need to fill the available vertical space, we must wrap the children widgets in an Expanded widget.



### Drawbacks OF Row and Column Widget:

**Both(Row & Column) does not have horizontal scrolling**

**Stack**

The stack is a widget in Flutter that contains a list of widgets and positions them on top of the other. In other words, the stack allows developers ***to overlap multiple widgets into a single screen*** and renders them from bottom to top. Hence, the **first widget** is the **bottommost** item, and the **last widget** is the **topmost** item

### Key Points Related to Stack Widget

The following are the key points of the [Flutter](https://www.javatpoint.com/flutter) stack widget:

* The child widget in a stack can be either **positioned** or **non-positioned**.
* Positioned items are wrapped in Positioned widget and must have a one non-null property
* The non-positioned child widgets are aligned itself. It displays on the screen based on the stack's alignment. The default position of the children is in the top left corner.
* We can use the alignment attribute to change the alignment of the widgets.
* Stack places the children widgets in order with the first child being at the bottom and the last child being at the top. If we want to reorder the children's widget, it is required to rebuild the stack in the new order. By default, the **first widget of each stack has the maximum size** compared to other widgets.

child: Stack(

            fit: StackFit.passthrough,

            overflow: Overflow.visible,

            children: <Widget>[

              // Max Size Widget

              Container(

                height: 300,

                width: 400,

                color: Colors.green,

                child: Center(

                  child: Text(

                    'Top Widget: Green',

                    style: TextStyle(color: Colors.white, fontSize: 20),

                  ),

                ),

              ),

              Positioned(

                top: 30,

                right: 20,

                child: Container(

                  height: 100,

                  width: 150,

                  color: Colors.blue,

                  child: Center(

                    child: Text(

                      'Middle Widget',

                      style: TextStyle(color: Colors.white, fontSize: 20),

                    ),

                  ),

                ),

              ),

              Positioned(

                top: 30,

                left: 20,

                child: Container(

                  height: 100,

                  width: 150,

                  color: Colors.orange,

                  child: Center(

                    child: Text(

                      'Bottom Widget',

                      style: TextStyle(color: Colors.white, fontSize: 20),

                    ),

                  ),

                )

              ),

            ],

          ),

        )

      ),

    );

**Output:**

When we run the app, we should get the UI of the screen similar to the below screenshot:

