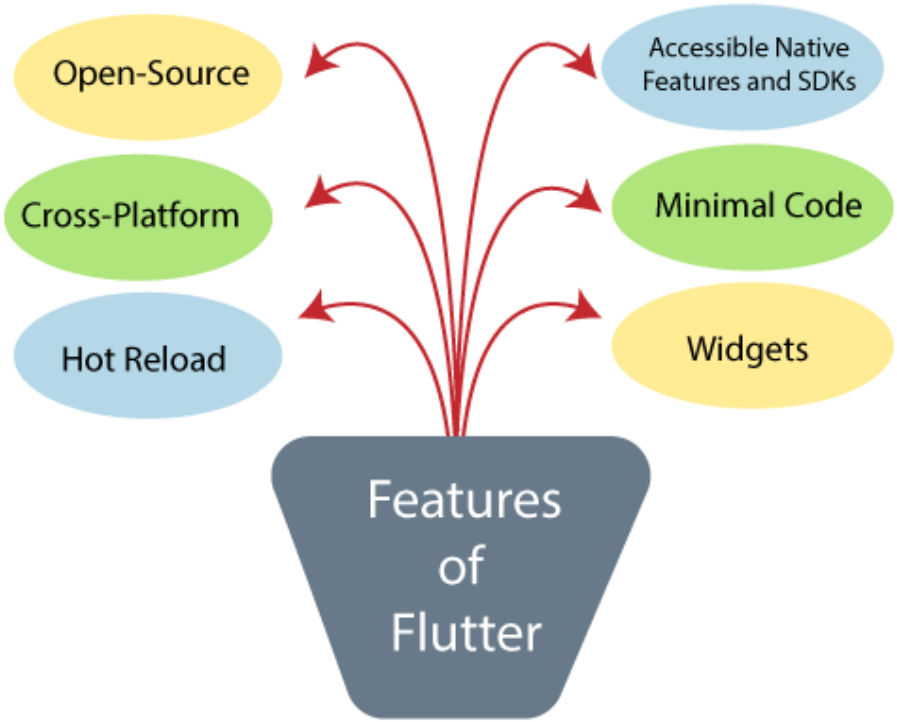
FLUTTER

Flutter is a UI toolkit for creating fast, beautiful, natively compiled applications for mobile, web, and desktop with one programing language and single codebase. It is free and open-source. It was initially developed from **Google** and now manages by European Computer Manufacturers Association (**ECMA)** standard. Flutter apps use Dart programming language for creating an app. The **dart programming** shares several same features as other programming languages, such as Kotlin and Swift, and can be trans-compiled into JavaScript code.

Flutter is mainly optimized for 2D mobile apps that can run on both Android and iOS platforms. We can also use it to build full-featured apps, including camera, storage, geolocation, network, third-party SDKs, and more.

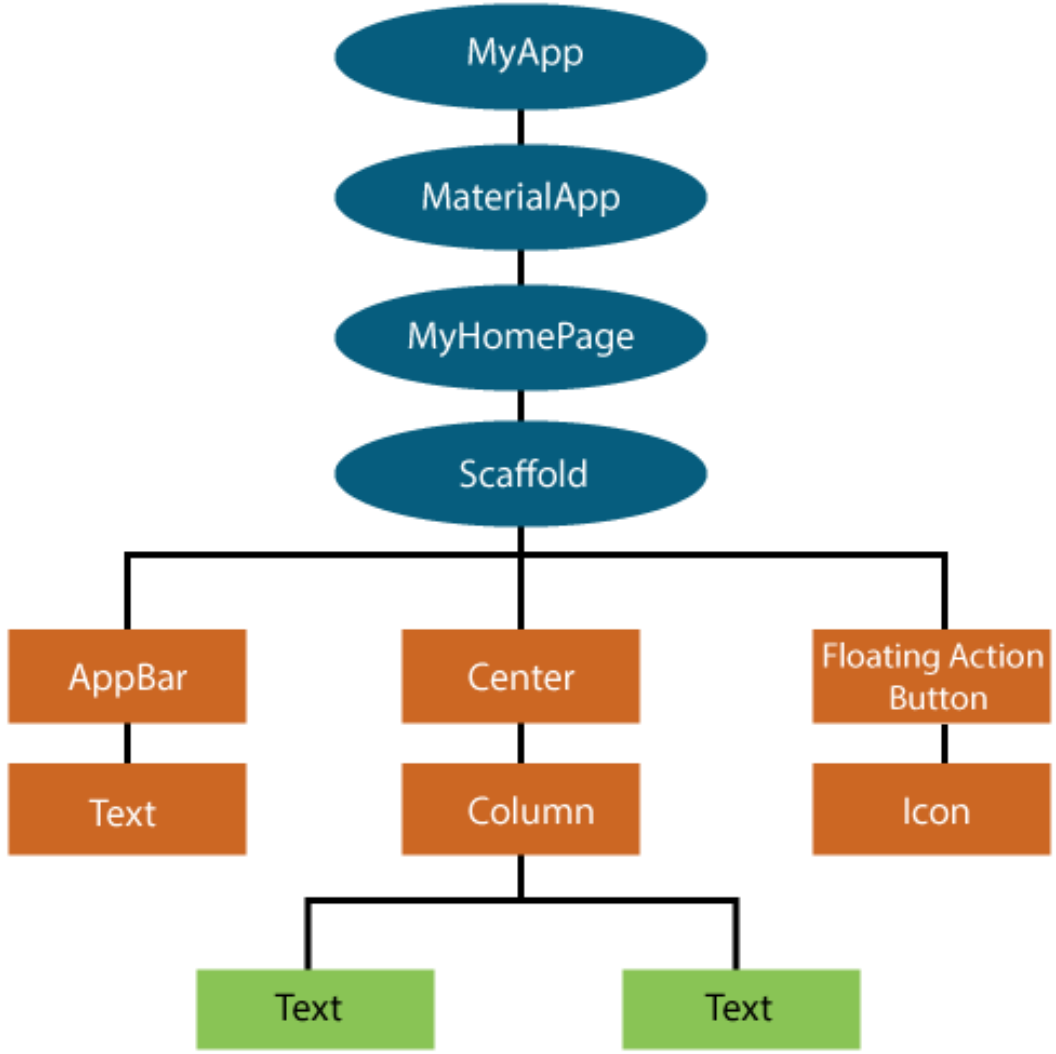
Features of Flutter

Flutter gives easy and simple methods to start building beautiful mobile and desktop apps with a rich set of material design and widgets.



Flutter Widgets

Whenever you are going to code for building anything in Flutter, it will be inside a widget. The central purpose is to build the app out of widgets. It describes how your app view should look like with their current configuration and state. When you made any alteration in the code, the widget rebuilds its description by calculating the difference of previous and current widget to determine the minimal changes for rendering in UI of the app.
Widgets are nested with each other to build the app. It means the root of your app is itself a widget, and all the way down is a widget also. For example, a widget can display something, can define design, can handle interaction, etc.



We can create the Flutter widget like this:

```
Class ImageWidget extends StatelessWidget {
    // Class Stuff
}
```

What is a widget?

Widgets can be compared to LEGO blocks; by adding blocks together, you create an object, and by adding different kinds of blocks, you can alter the look and behavior of the object.

- Widgets are the building blocks of a Flutter app, and each widget is an immutable declaration of the user interface. In other words, widgets are configurations (instructions) for different parts of the UI.

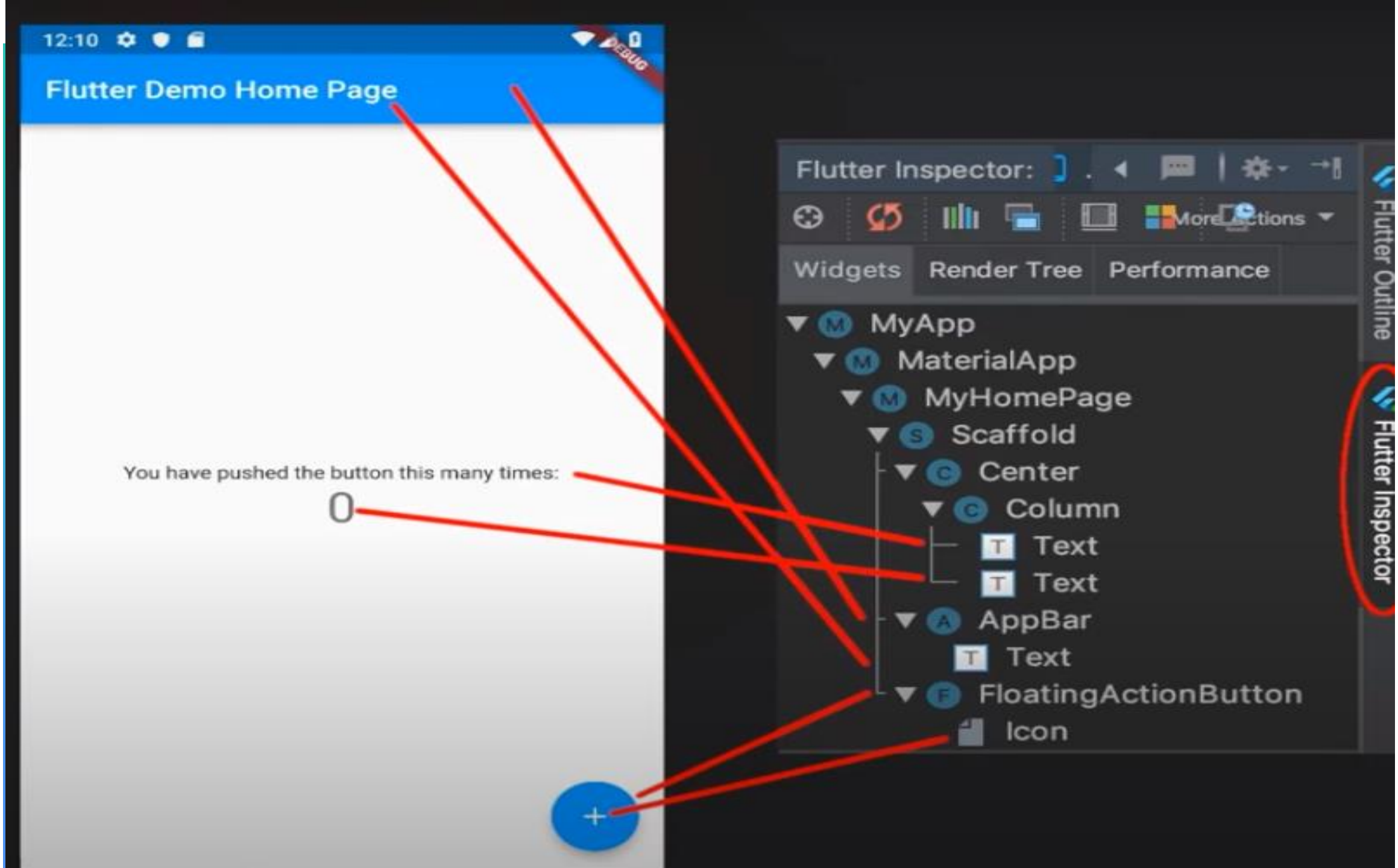
- Placing the widgets together creates the widget tree. For example, say an architect draws a blueprint of a house; all of the objects like walls, windows, and doors in the house are the widgets, and all of them work together to create the house or, in this case, the application.

Category of Widgets

There are mainly 14 categories in which the flutter widgets are divided.
They are mainly separated on the basis of the functionality they provide in the flutter app.

1. Accessibility: These are the set of widgets that make a flutter app more easil accessible.
2. Animation and Motion: These widgets add animation to other widgets.
3. Assets, Images, and Icons: These widgets take charge of assets such as display images and show icons.
4. Async: These provide async functionality in the flutter application.
5. Basics: These are the bundle of widgets which are absolutely necessary for the development of any flutter application.
6. Cupertino: These are the ios designed widgets.
7. Input: This set of widgets provide input functionality in a flutter application.

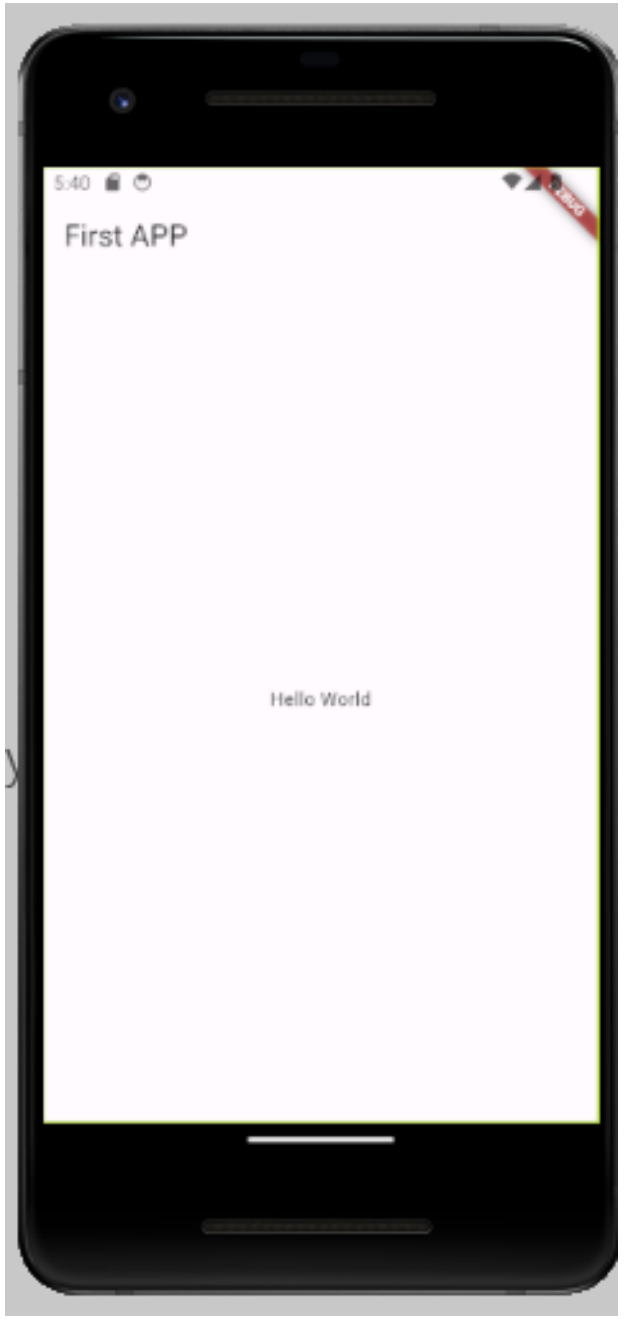
8. Interaction Models: These widgets are here to manage touch events and route users to different views in the application.
9. Layout: This bundle of widgets helps in placing the other widgets on the screen as needed.
10. Material Components: This is a set of widgets that mainly follow material design by Google.
11. Painting and effects: This is the set of widgets which apply visual changes to their child widgets without changing their layout or shape.
12. Scrolling: This provides sacrollability of to a set of other widgets that are not scrollable by default.
13. Styling: This deals with the theme, responsiveness, and sizing of the app.
14. Text: This display text.


12:10
DEBUG
Flutter Demo Home Page
You have pushed the button this many times:
0
Flutter Inspector:
More Actions
Widgets
Render Tree
Performance
MyApp
MaterialApp
MyHomePage
Scaffold
Center
Column
Text
Text
AppBar
Text
FloatingActionButton
Icon
Flutter Outline
Flutter Inspector

```dart
import 'package:flutter/material.dart';

void main() {
  runApp(MyHomePage());
}

class MyHomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text("First APP"),
        ),
        body: Center(
          child: Text('Hello World'),
        ),
      ),
    );
  }
}
```



Types of Widget

Visible widget

The visible widgets are related to the user input and output data. Some of the important types of this widget are:

Text

A Text widget holds some text to display on the screen. We can align the text widget by using **textAlign** property, and style property allow the customization of Text that includes font, font weight, font style, letter spacing, color, and many more. We can use it as like below code snippets.

```
new Text(
Hello, Javatpoint!',
textAlign: TextAlign.center,
style: new TextStyle(fontWeight: FontWeight.bold),
)
```

Button

This widget allows you to perform some action on click. Flutter does not allow you to use the Button widget directly; instead, it uses a type of buttons like a TextButton and a ElevatedButton. We can use it as like below code snippets.

```
// TextButton  Example
new TextButton (
  child: Text("Click here"),
  onPressed: () {
    // Do something here
  },
),

//ElevatedButton Example
new ElevatedButton (
  child: Text("Click here"),
  elevation: 5.0,
  onPressed: () {
    // Do something here
  },
),
```

Image

This widget holds the image which can fetch it from multiple sources like from the asset folder or directly from the URL. It provides many constructors for loading image, which are given below:

- **Image:** It is a generic image loader, which is used by **ImageProvider**.
- **asset:** It load image from your project asset folder.
- **file:** It loads images from the system folder.
- **memory:** It load image from memory.
- **network:** It loads images from the network.

To add an image in the project, you need first to create an assets folder where you keep your images and then add the below line in **pubspec.yaml** file.

```
assets:
  - assets/
```

```
class MyHomePage extends StatelessWidget {
  // This widget is the home page of your application.
  final String title;

  @override
  Widget build(BuildContext context) {
    return  MaterialApp (
    home: scaffold(
      appBar: AppBar(
        title: Text(this.title),
     ),
      body: Center(
        child: Image.asset('assets/computer.png'),
      ),
     ),
    );
  }
}
```

Icon

This widget acts as a container for storing the Icon in the Flutter. The following code explains it more clearly.

```
Icon( Icons.thumb_up,
 color: Colors.blue,
 size: 100, ),
```

Invisible widget

The invisible widgets are related to the layout and control of widgets. It provides controlling how the widgets actually behave and how they will look onto the screen. Some of the important types of these widgets are:

Column

A column widget is a type of widget that arranges all its children's widgets in a vertical alignment. It provides spacing between the widgets by using
the **mainAxisAlignment** and **crossAxisAlignment** properties. In these properties, the main axis is the vertical axis, and the cross axis is the horizontal axis.

```
new Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: <Widget>[
    new Text(
      "VegElement",
    ),
    new Text(
      "Non-vegElement"
    ),
  ],
),
```

Row

The row widget is similar to the column widget, but it constructs a widget horizontally rather than vertically. Here, the main axis is the horizontal axis, and the cross axis is the vertical axis.

```
new Row(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: <Widget>[
    new Text(
      "VegElement",
    ),
    new Text(
      "Non-vegElement"
    ),
  ],
),
```

Center

This widget is used to center the child widget, which comes inside it. All the previous examples contain inside the center widget.

```
Center(
  child: new column(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: <Widget>[
      new Text(
        "VegElement",
      ),
      new Text(
        "Non-vegElement"
      ),
    ],
  ),
),
```

Padding

This widget wraps other widgets to give them padding in specified directions. You can also provide padding in all directions. We can understand it from the below example that gives the text widget padding of 6.0 in all directions.

```
Padding(
 padding: const EdgeInsets.all(6.0),
  child: new Text(
    "Element 1",
  ),
),
```

Scaffold

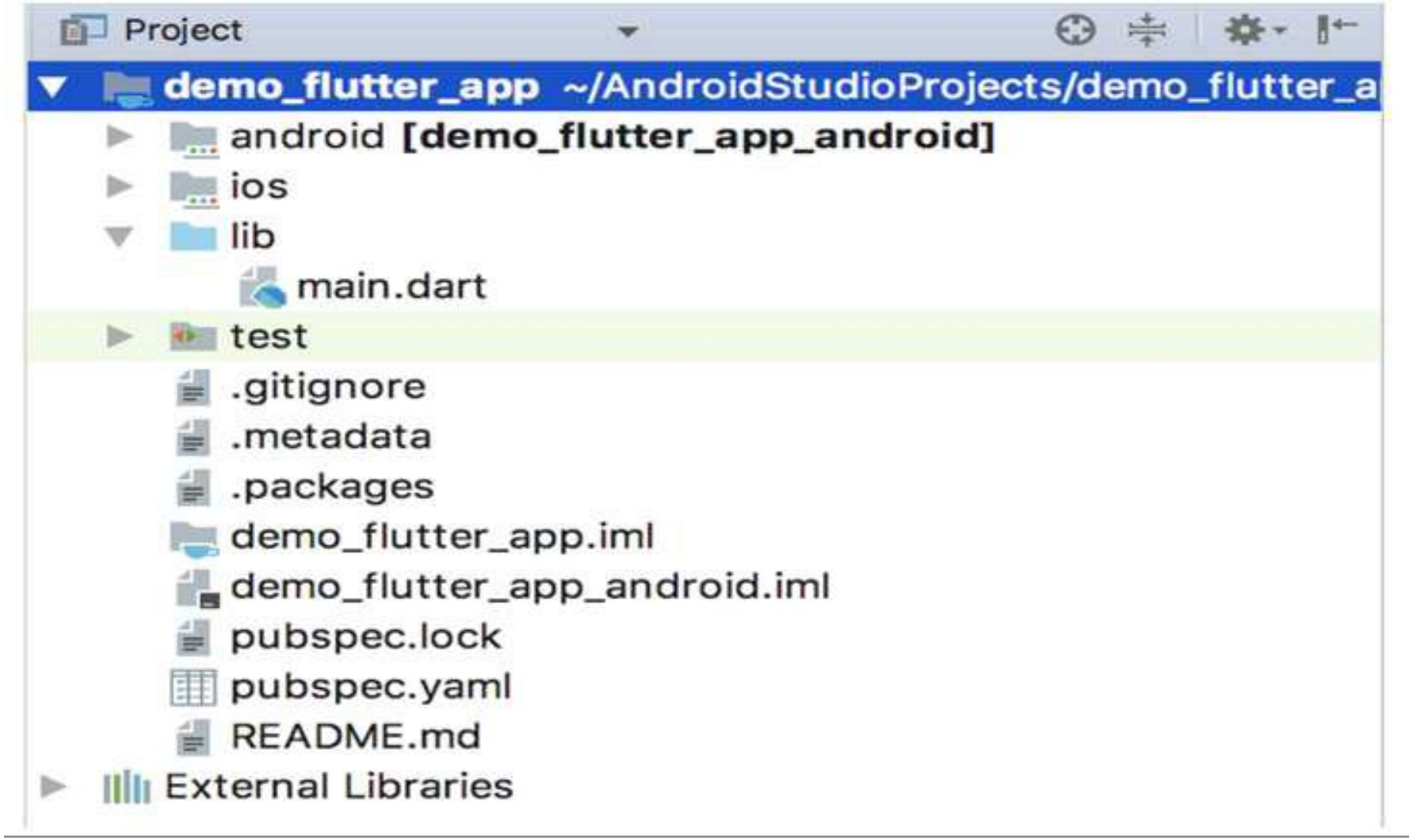
This widget provides a framework that allows you to add common material design elements like AppBar, Floating Action Buttons, Drawers, etc.

Stack

It is an essential widget, which is mainly used for **overlapping** a widget, such as a button on a background gradient.

Exploring 1st Flutter Project

let’s understand the files in the project.



Project Directories

- To make a basic app, you only need to focus on the lib directory and the pubspec.yaml file.

- The “lib” directory holds all the main dart code used to run your app.

- The “pubspec.yaml” file contains all of the packages you’ve imported. (For Android Developers: This is equivalent to your gradle files where you add in dependencies)

Android” and “ios” directories

- The “android” and “ios” directories hold a complete Android and iOS app respectively with all their respective files.

- For example if you go into “android” you’ll find a complete Android project including a manifest.xml, activities, gradle files, etc. If you want to write any platform-specific code or add permissions, you’ll have to edit these projects.

Lib and test directory

- The “lib” directory holds all your .dart files and most if not all your code will rest over here.

- The “test” directory is for writing tests in Dart similar to Instrumented tests in Android using Espresso. Tests help you verify a component works without actually having do it yourself.

The pubspec.yaml File

- Before moving on to main.dart, which contains the main app code, we should take a brief look at the pubspec.yaml file

- It is important to understand that the pubspec.yaml file is not unique to Flutter apps: it is a feature of Dart packages and, as such, also contains all the information needed to make it a Flutter app.

The pubspec.yaml File

firstapp/pubspec.yaml

```
name: flutter_example_name
description: Example description of a flutter app that makes
             great things happen.
```

In the first part we'll specify some metadata about the Flutter app. In this section, two attributes are set:

1. The name, which is the Dart package name and it is the default app name that appears on the home screen or the app drawer. The one drawback with name is that it must be all lowercase and be a valid Dart identifier: it can't start with a digit, it can't contain any character other than letters and underscores (no spaces allowed) and it can't be a Dart keyword (like class, if, try, etc.).
2. The description, which should be a brief, sentence or two explanation of the Flutter app.

The pubspec.yaml File

- The rest of the file specifies the dependencies

firstapp/pubspec.yaml

```
dependencies:
  flutter:
    sdk: flutter

dev_dependencies:
  flutter_test:
    sdk: flutter

flutter:
  uses-material-design: true
```

- Since we are not using any third-party packages, we are just specifying the Flutter SDK itself as a dependency and, with the lines:

flutter:

uses-material-design: true

We make sure that Material Design (Google/Android style) assets like icons are included.