Chapter 2: Introduction

2.1. What is Mobile Application Development?

Mobile app development is a process for building mobile applications for smartphones and digital assistants, most commonly for Android and iOS. The software can be preinstalled on the device or can be downloaded and installed by the user later. The programming and markup languages used for this kind of software development include Java, Swift, Dart, C#, and HTML5.

Types of mobile applications Development

a. **Native App Development:-** The app developed exclusively for a single platform using a native-to-the-operating-system language is a Native App. A native app is usually written in one programming language for a particular operating system. For instance, to develop a Native app for Android, Java, or Kotlin, while for iOS, Objective-C or Swift can be used.

   Native apps acquire all possible advantages of the device and the operating system's features. The apps leverage direct access to the hardware of the devices such as GPS, Camera, Microphone, Offline access, and many more. As a result, they are fast and more reliable, render high performance, and have a better user experience.

b. **Cross-Platform (Multi platform) Mobile Apps:-** Multi Platform mobile development is an approach that allows you to build a single mobile application that runs smoothly on several operating systems. In cross-platform apps, some or even all of the source code can be shared. This means that developers can create and deploy mobile assets that work on both Android and iOS without having to recode them for each individual platform.

   The need by mobile developers to reach the largest possible user base regardless of their preferred platform has given rise to more value being seen in a cross-platform mobile app. The cross-platform dev approach uses a native rendering engine. They offer seamless functionality, easy implementation, and cost-effective production. Cross-Platform apps have low performance as compared to natives but are far better than hybrids. Customization is also a pain since it is limited to the framework you use.

c. **Hybrid Mobile Apps:-** Hybrid apps are coded with web-based technologies, like HTML, CSS, and JavaScript. The resulting app can usually be found on Apple App Store or Google Play Store by searching for the name or URL of the website it's based on. What makes them "hybrid" is that they use WebViews to present their content — think of them as mini-browser windows inside your mobile application.

d. **Progressive Web Apps (PWAs):-** A Progressive web app is a website that acts like a native mobile app. To some extent, progressive web apps or PWAs resemble hybrid ones. As PWAs operate in a browser, there's no need to download them from an app store. Still one can access the app on the home screen. Their progression is based on the user experience that is optimized for each platform.

Though PWA works offline, provides GPS access, push notifications, and many more for Android, for iOS (latest 12.2 release), there are limitations such as no camera access, no default launch image, no installation API, and other issues.

There are no dedicated languages or frameworks for Progressive Web Apps, so there is no need to hire a special type of developer. They can be done in Angular or React. This type of app coding is most popular in e-commerce projects.

2.2. What is Flutter?

Flutter is Google's free and open-source UI framework for creating native mobile applications. Released in 2017, Flutter allows developers to build mobile applications for both iOS and Android with a single codebase and programming language. This capability makes building iOS and Android apps simpler and faster.
The Flutter framework consists of both a software development kit (SDK) and a widget-based UI library. This library includes various reusable UI elements, such as sliders, buttons, and text Inputs.

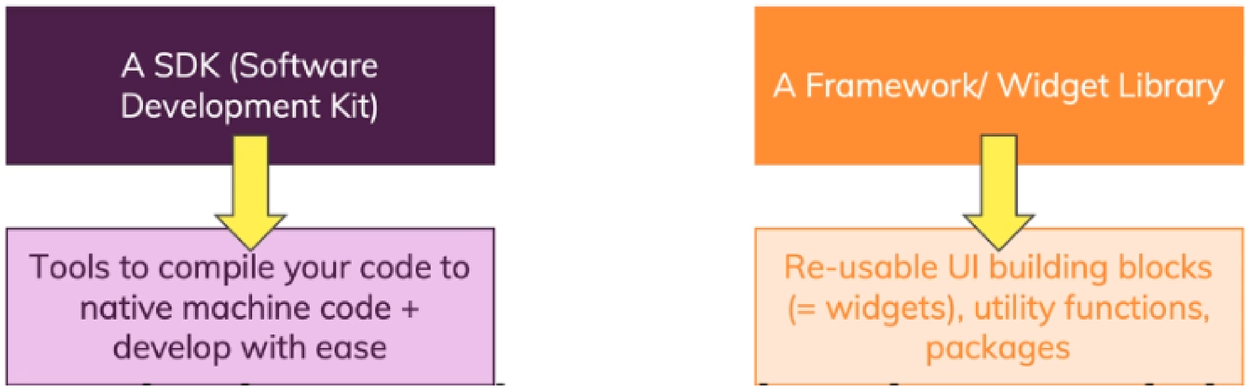

A SDK (Software Development Kit)
Tools to compile your code to native machine code + develop with ease
A Framework/ Widget Library
Re-usable UI building blocks (= widgets), utility functions, packages

Figure 2. Flutter SDK

Developers building mobile applications with the Flutter framework will do so using a programming language called Dart. With a syntax like JavaScript, Dart is a typed object programming language that focuses on front-end development.

Why use Flutter?

Here are some advantages of Flutter:

- ➔ Be highly productive
- ➔ It is used for iOS app development and Android app development from a single codebase.
- ➔ Do more with less code, even on a single OS, with a modern, expressive language and a declarative approach
- ➔ Prototype and iterate easily
- ➔ Experiment by changing code and reloading as your app runs (with hot reload)
- ➔ Fix crashes and continue debugging from where the app left off
- ➔ Create beautiful, highly-customized user experiences
- ➔ Benefit from a rich set of Material Design and Cupertino (iOS-flavor) widgets built using Flutter's own framework
- ➔ Realize custom, beautiful, brand-driven designs, without the limitations of OEM widget sets

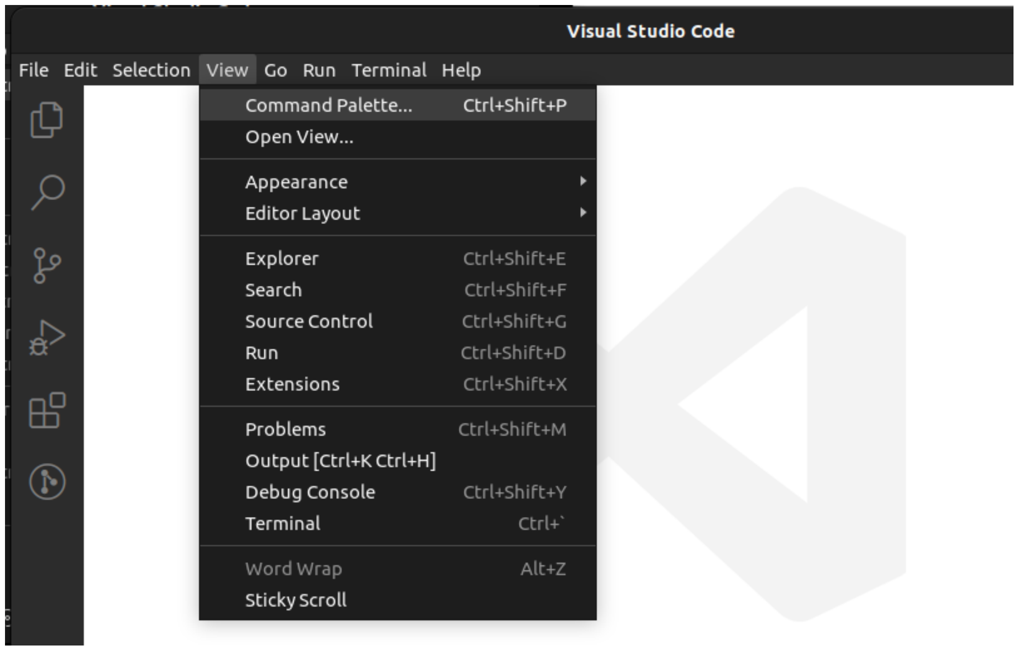
2.3. Flutter architecture and SDK

Flutter is a UI toolkit from Google that allows users to build natively compiled applications for the web, desktop, and mobile devices. It is also embedded in one single codebase, meaning that while Flutter embraces different platforms, it still originates from a single codebase.

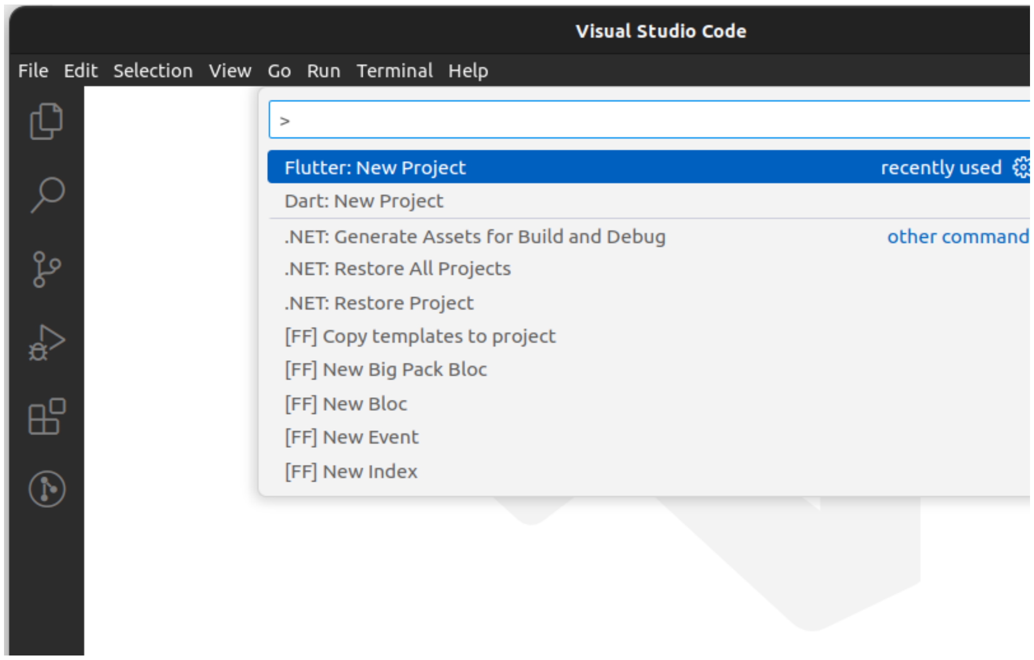
2.4. Creating Flutter Project

Lets see the two ways of how to create a new Flutter project. You can do it with the IDE **Visual Studio Code** or by using the **command-line tool from the Flutter SDK**. You need to have the Flutter SDK installed, otherwise none of these tips will work.

1. Visual Studio Code (VS Code)
    - ➔ Got **View** => **Command Palettes….**

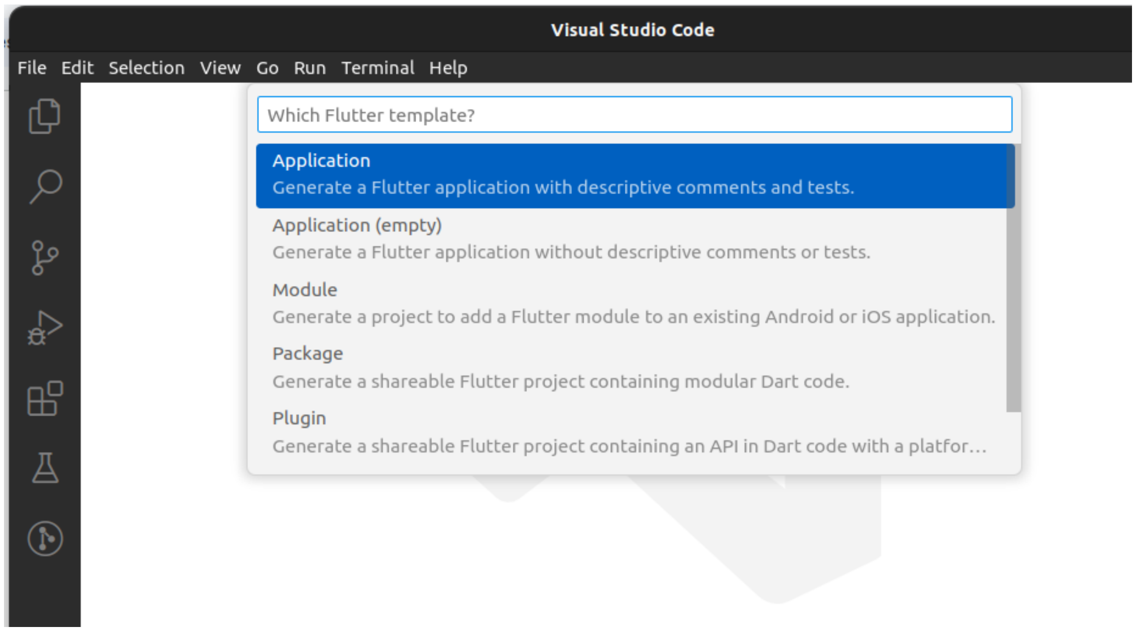


➔ Enter or select: **Flutter: New Project**



```
>
Flutter: New Project                                recently used
Dart: New Project
.NET: Generate Assets for Build and Debug           other command
.NET: Restore All Projects
.NET: Restore Project
[FF] Copy templates to project
[FF] New Big Pack Bloc
[FF] New Bloc
[FF] New Event
[FF] New Index
```

➔ Select the project template **Application**



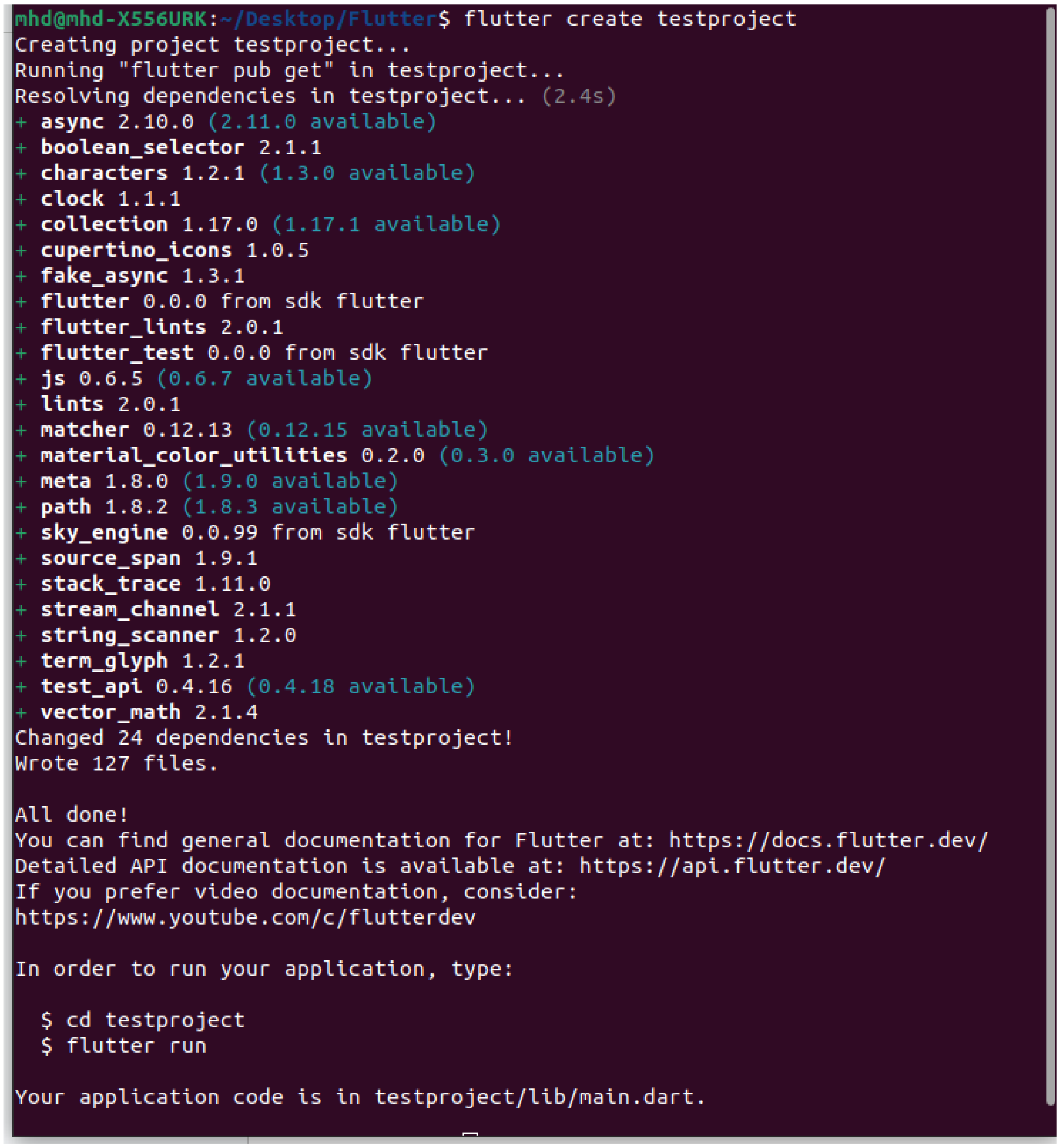
```
Which Flutter template?
Application
Generate a Flutter application with descriptive comments and tests.
Application (empty)
Generate a Flutter application without descriptive comments or tests.
Module
Generate a project to add a Flutter module to an existing Android or iOS application.
Package
Generate a shareable Flutter project containing modular Dart code.
Plugin
Generate a shareable Flutter project containing an API in Dart code with a platfor…
```

➔ Finally, Your project will be created at the desired location and Visual Studio Code will open it immediately.

2. Command line

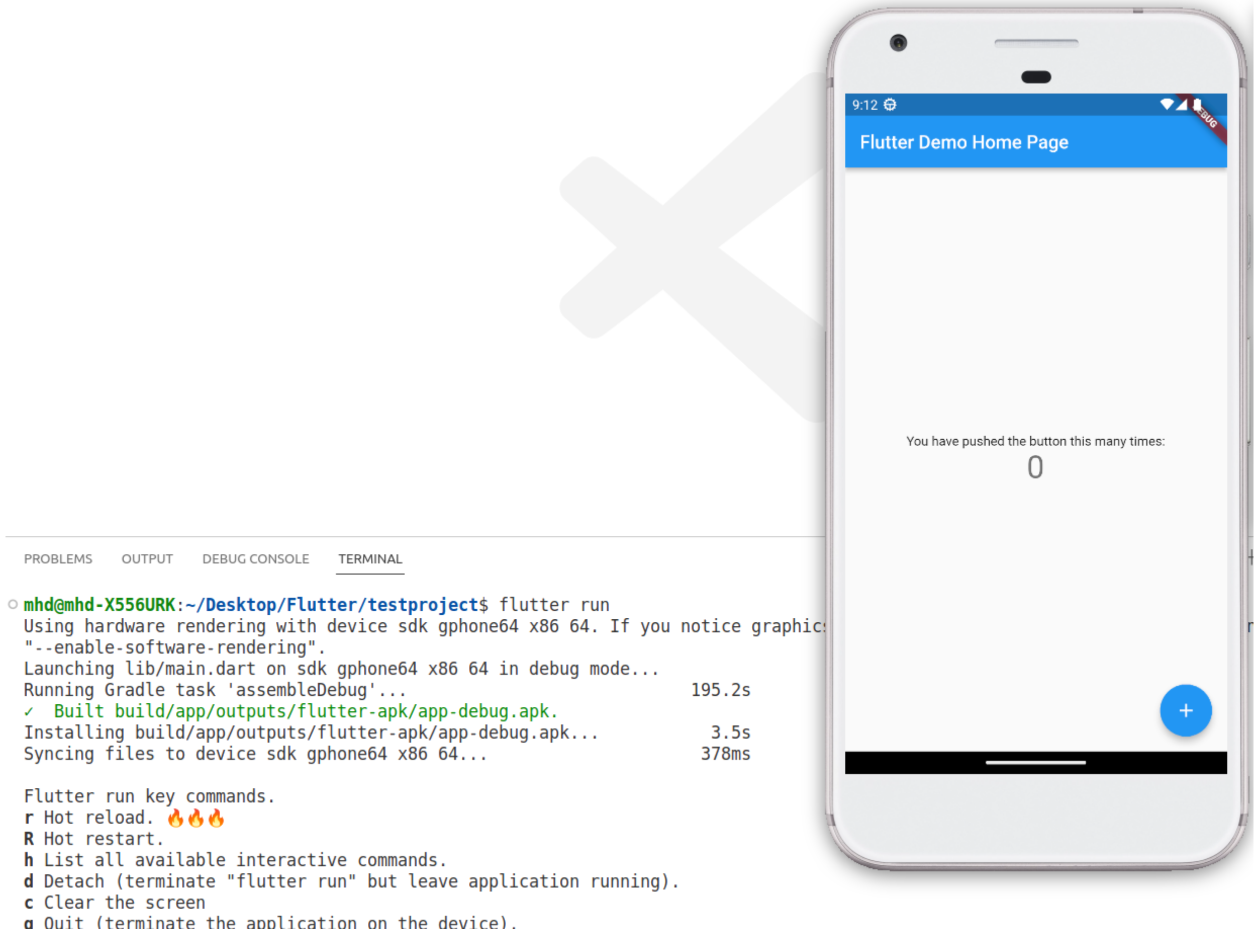
➔ Open any Command-line tool

➔ Run the command **flutter create testproject.** It will create a folder called **testproject** at the current location and put all the app code inside this folder.



```
mhd@mhd-X556URK:~/Desktop/Flutter$ flutter create testproject
Creating project testproject...
Running "flutter pub get" in testproject...
Resolving dependencies in testproject... (2.4s)
+ async 2.10.0 (2.11.0 available)
+ boolean_selector 2.1.1
+ characters 1.2.1 (1.3.0 available)
+ clock 1.1.1
+ collection 1.17.0 (1.17.1 available)
+ cupertino_icons 1.0.5
+ fake_async 1.3.1
+ flutter 0.0.0 from sdk flutter
+ flutter_lints 2.0.1
+ flutter_test 0.0.0 from sdk flutter
+ js 0.6.5 (0.6.7 available)
+ lints 2.0.1
+ matcher 0.12.13 (0.12.15 available)
+ material_color_utilities 0.2.0 (0.3.0 available)
+ meta 1.8.0 (1.9.0 available)
+ path 1.8.2 (1.8.3 available)
+ sky_engine 0.0.99 from sdk flutter
+ source_span 1.9.1
+ stack_trace 1.11.0
+ stream_channel 2.1.1
+ string_scanner 1.2.0
+ term_glyph 1.2.1
+ test_api 0.4.16 (0.4.18 available)
+ vector_math 2.1.4
Changed 24 dependencies in testproject!
Wrote 127 files.

All done!
You can find general documentation for Flutter at: https://docs.flutter.dev/
Detailed API documentation is available at: https://api.flutter.dev/
If you prefer video documentation, consider:
https://www.youtube.com/c/flutterdev

In order to run your application, type:

  $ cd testproject
  $ flutter run

Your application code is in testproject/lib/main.dart.
```

2.5. Run Flutter App



```
mhd@mhd-X556URK:~/Desktop/Flutter/testproject$ flutter run
Using hardware rendering with device sdk gphone64 x86 64. If you notice graphic
"--enable-software-rendering".
Launching lib/main.dart on sdk gphone64 x86 64 in debug mode...
Running Gradle task 'assembleDebug'...                          195.2s
✓  Built build/app/outputs/flutter-apk/app-debug.apk.
Installing build/app/outputs/flutter-apk/app-debug.apk...         3.5s
Syncing files to device sdk gphone64 x86 64...                   378ms

Flutter run key commands.
r Hot reload. 🔥🔥🔥
R Hot restart.
h List all available interactive commands.
d Detach (terminate "flutter run" but leave application running).
c Clear the screen
q Quit (terminate the application on the device).
```

2.6. Introduction to Widget

In Flutter, a widget is a basic building block for creating user interfaces. Widgets are like building blocks or Lego pieces that can be combined to create a complex user interface. Every UI element in Flutter is a widget, whether it is a simple text label or a complex animated widget.

Category of Widgets:

There are mainly 14 categories in which the flutter widgets are divided. They are mainly segregated on the basis of the functionality they provide in a flutter application.

1. **Accessibility**: These are the set of widgets that make a flutter app more easily 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 that are absolutely necessary for the development of any flutter application.
6. **Cupertino**: These are the iOS designed widgets.
7. **Input**: This set of widgets provides 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 that apply visual changes to their child widgets without changing their layout or shape.
12. **Scrolling**: This provides scrollability 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 displays text.

Types of Widget

There are broadly two types of widgets in the flutter:

1. Stateless Widget
2. Stateful Widget

State: The State is the information that can be read synchronously when the widget is built and might change during the lifetime of the widget.

In other words, the state of the widget is the data of the objects that its properties (parameters) are sustaining at the time of its creation (when the widget is painted on the screen). The state can also change when it is used for example when a *CheckBox* widget is clicked a check appears on the box.

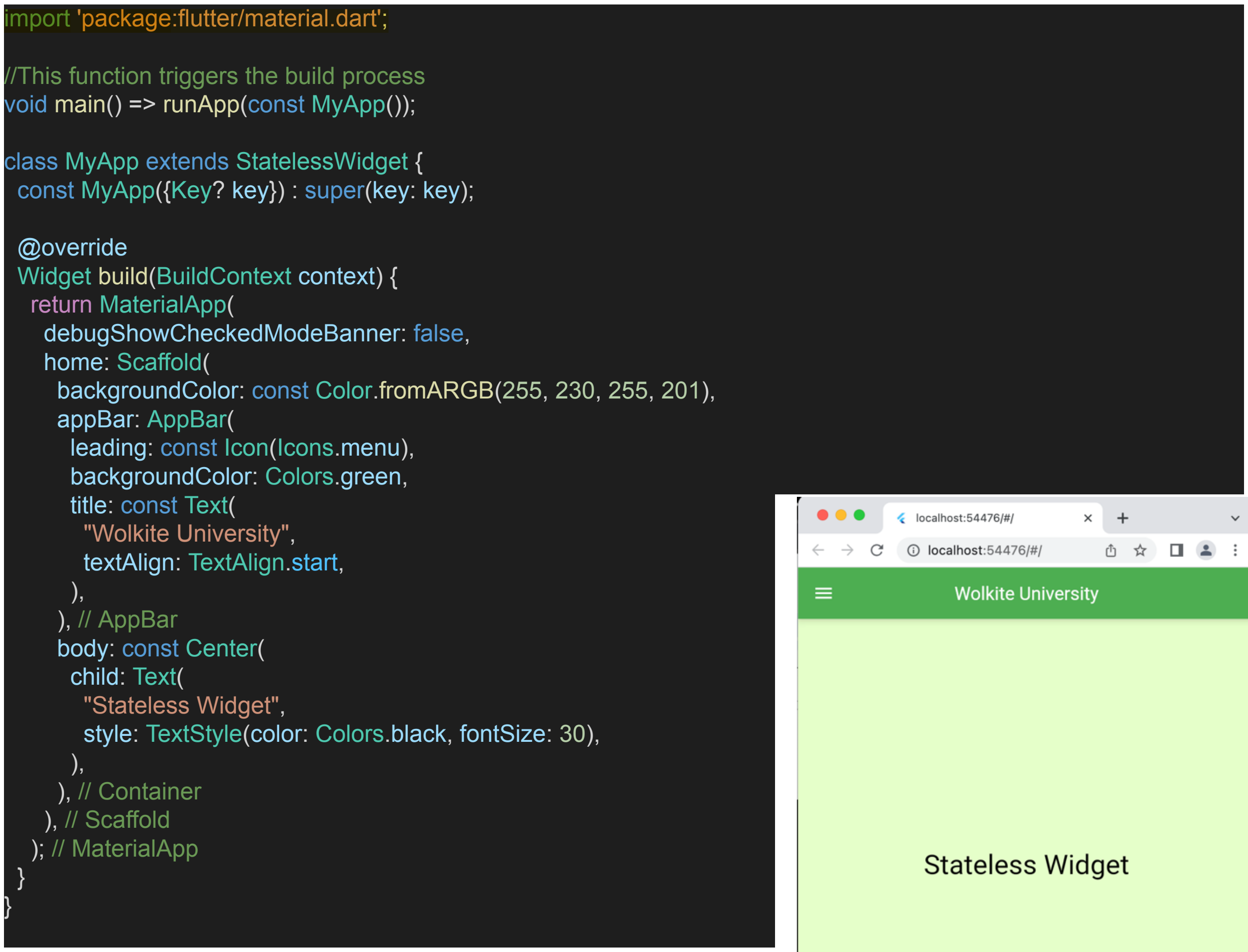
Stateless Widgets: The widgets whose state cannot be altered once they are built are called stateless widgets. These widgets are immutable once they are built i.e any amount of change in the variables, icons, buttons, or retrieving data cannot change the state of the app. Below is the basic structure of a *stateless widget. Stateless* widget overrides the *build*() method and returns a widget. For example, we use *Text* or the *Icon* in our flutter application where the state of the widget does not change in the *runtime*. It is used when the UI depends on the information within the object itself. Other examples can be *Text*, *RaisedButton*, *IconButtons*.

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

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

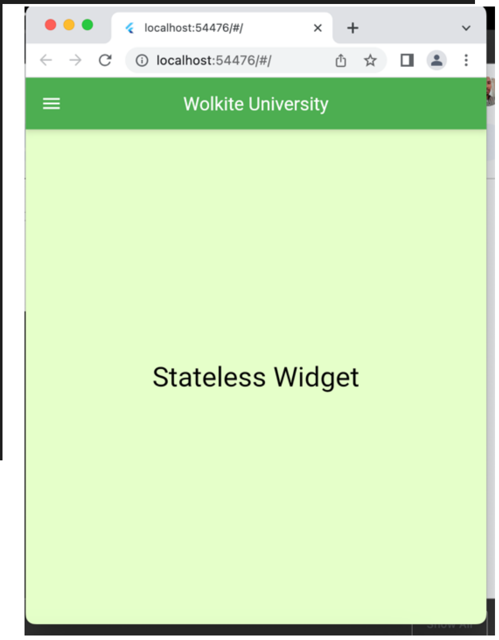
The name of the stateless widget is MyApp which is being called from the *runApp()* and extends a stateless widget. Inside this MyApp a build function is overridden and takes BuildContext as a parameter. This BuildContext is unique to each and every widget as it is used to locate the widget inside the widget tree.

The build function contains a container which is again a widget of Flutter inside which we will design the UI of the app. In the stateless widget, the build function is called only once which makes the UI of the screen.

Example: Stateless Widget



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

//This function triggers the build process
void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Scaffold(
        backgroundColor: const Color.fromARGB(255, 230, 255, 201),
        appBar: AppBar(
          leading: const Icon(Icons.menu),
          backgroundColor: Colors.green,
          title: const Text(
            "Wolkite University",
            textAlign: TextAlign.start,
          ),
        ), // AppBar
        body: const Center(
          child: Text(
            "Stateless Widget",
            style: TextStyle(color: Colors.black, fontSize: 30),
          ),
        ), // Container
      ), // Scaffold
    ); // MaterialApp
  }
}
```



Stateful Widgets: The widgets whose state can be altered once they are built are called stateful Widgets. These states are mutable and can be changed multiple times in their lifetime. This simply means the state of an app can change multiple times with different sets of variables, inputs, data. Below is the basic structure of a stateful widget. Stateful widget overrides the **createState()** method and returns a State. It is used when the UI can change dynamically. Some examples can be CheckBox, RadioButton, Form, TextField.

Classes that inherit “Stateful Widget” are immutable. But the State is mutable which changes in the runtime when the user interacts with it.

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

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

class MyApp extends StatefulWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  // ignore: library_private_types_in_public_api
  _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

So let us see what we have in this code snippet. The name of the Stateful Widget is MyApp which is called from the runApp() and extends a stateful widget. In the MyApp class, we override the create state function. This createState() function is used to create a mutable state for this widget at a given location in the tree. This method returns an instance for the respected state subclass. The other class which is _MyAppState extends the state, which manages all the changes in the widget. Inside this class, the build function is overridden which takes the BuildContext as the parameter. This build function returns a widget where we design the UI of the app. Since it is a stateful widget the build function is called many times which creates the entire UI once again with all the changes.

Example: Stateful Widget

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

//This function triggers the build process
void main() => runApp(const MyApp());

// StatefulWidget
```

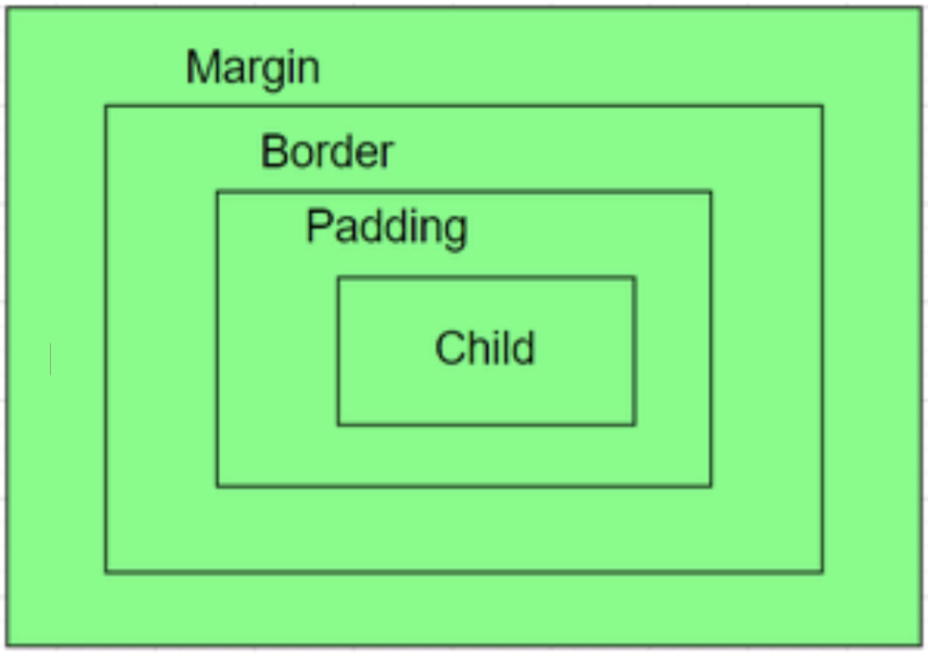
```
class MyApp extends StatefulWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
// ignore: library_private_types_in_public_api
  _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Scaffold(
        backgroundColor: Color.fromARGB(255, 230, 255, 201),
        appBar: AppBar(
          leading: const Icon(Icons.menu),
          backgroundColor: Colors.green,
          title: const Text(
            "Wolkite University",
            textAlign: TextAlign.start,
          ),
        ), // AppBar
        body: const Center(
          child: Text(
            "StateFul Widget",
            style: TextStyle(color: Colors.black, fontSize: 30),
          ),
        ), // Container
      ), // Scaffold
    ); // MaterialApp
  }
}
```

Description of the widgets used are as follows:

- Scaffold - Implements the basic material design visual layout structure.
- App-Bar - To create a bar at the top of the screen.
- Text - To write anything on the screen.
- Container - To contain any widget.
- Center - To provide center alignment to other widgets.

Container class in Flutter

Container class in flutter is a convenience widget that combines common painting, positioning, and sizing of widgets. A Container class can be used to store one or more widgets and position them on the screen according to our convenience. Basically, a container is like a box to store contents. A basic container element that stores a widget has a **margin**, which separates the present container from other contents. The total container can be given a **border** of different shapes, for example, rounded rectangles, etc. A container surrounds its child with **padding** and then applies additional constraints to the padded extent (incorporating the width and height as constraints, if either is non-null).



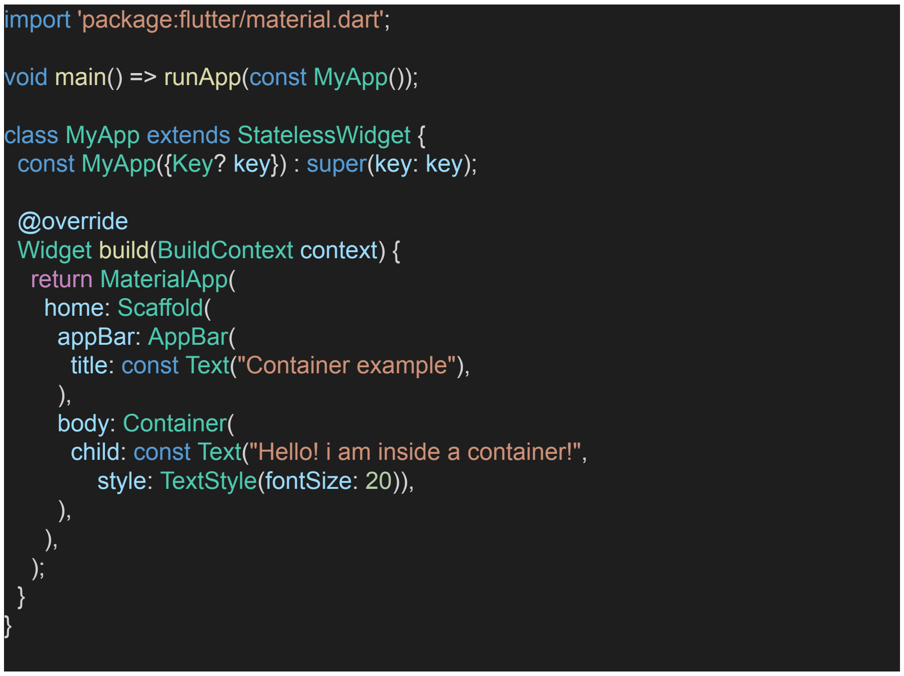
Constructor of Container Class

Syntax:

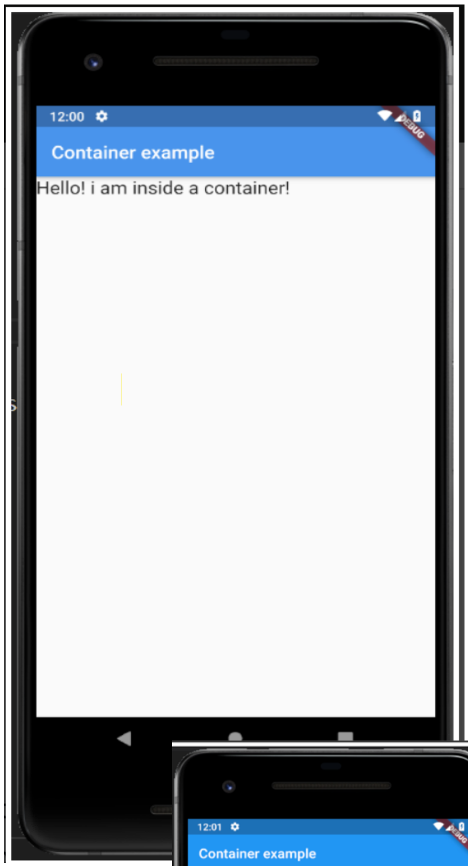
```
Container({Key key,
          AlignmentGeometry alignment,
          EdgeInsetsGeometry padding,
          Color color,
          Decoration decoration,
          Decoration foregroundDecoration,
          double width,
          double height,
          BoxConstraints constraints,
          EdgeInsetsGeometry margin,
          Matrix4 transform,
          Widget child,
        Clip clipBehavior: Clip.none});
```

Properties of Container Class:

1. child: Container widget has a property 'child:' which stores its children. The child class can be any widget. Let us take an example, taking a text widget as a child.

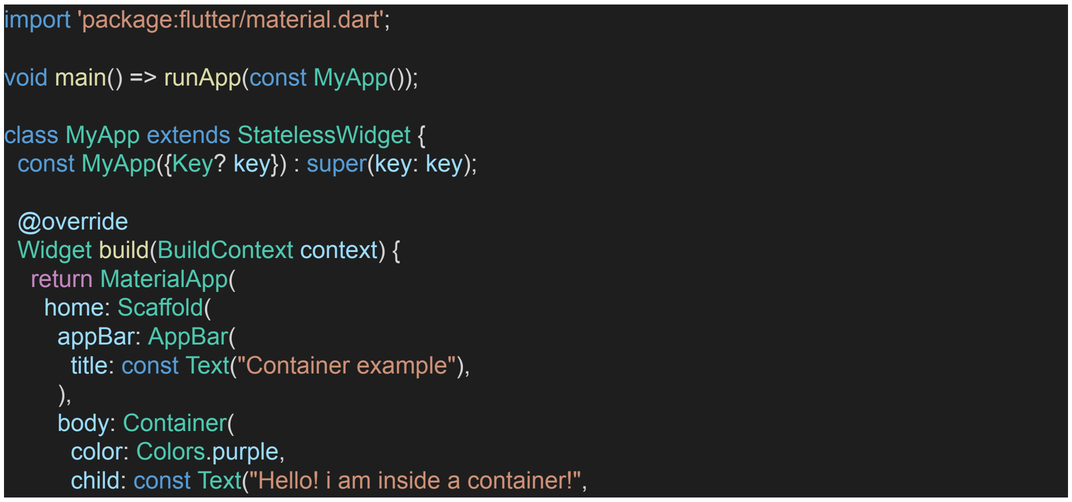


```
import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text("Container example"),
        ),
        body: Container(
          child: const Text("Hello! i am inside a container!",
              style: TextStyle(fontSize: 20)),
        ),
      ),
    );
  }
}
```

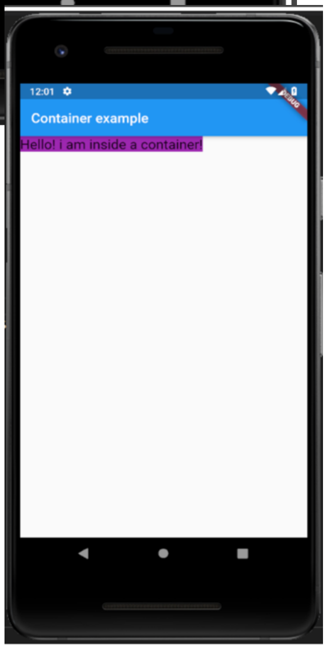


Hello! i am inside a container!

2. color: The color property sets the background color of the entire container. Now we can visualize the position of the container using a background color.



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

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text("Container example"),
        ),
        body: Container(
          color: Colors.purple,
          child: const Text("Hello! i am inside a container!",
```

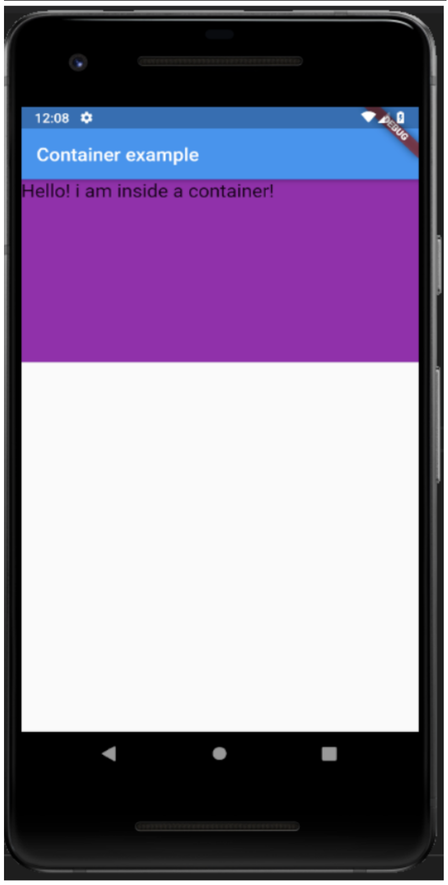


Hello! i am inside a container!

```
            style: TextStyle(fontSize: 20)),
        ),
      ),
    );
  }
}
```

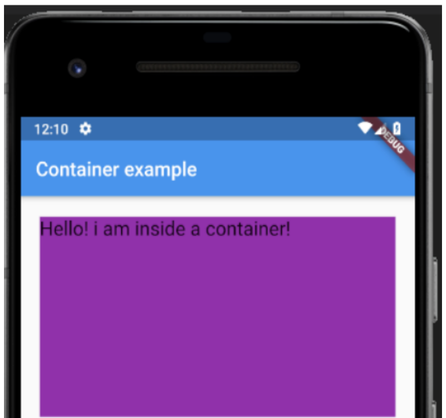
3. height and width: By default, a container class takes the space that is required by the child. We can also specify the height and width of the container based on our requirements.

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

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text("Container example"),
        ),
        body: Container(
          height: 200,
          width: double.infinity,
          color: Colors.purple,
          child: const Text("Hello! i am inside a container!",
              style: TextStyle(fontSize: 20)),
        ),
      ),
    );
  }
}
```


12:08
Container example
Hello! i am inside a container!

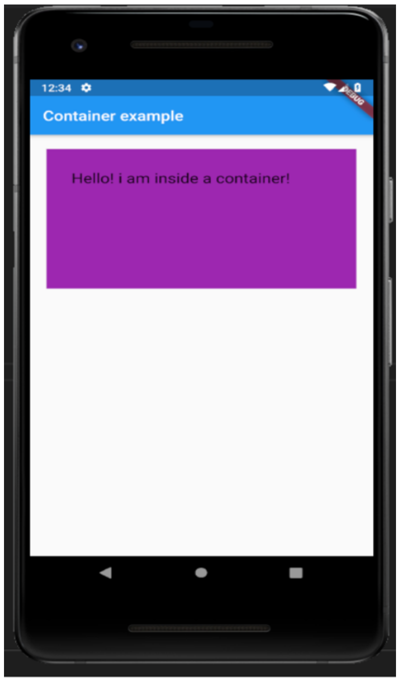
4. margin: The margin is used to create an empty space around the container. Observe the white space around the container. Here EdgeInsets.geometry is used to set the margin .all() indicates that the margin is present in all four directions equally.

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

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
```


12:10
Container example
Hello! i am inside a container!

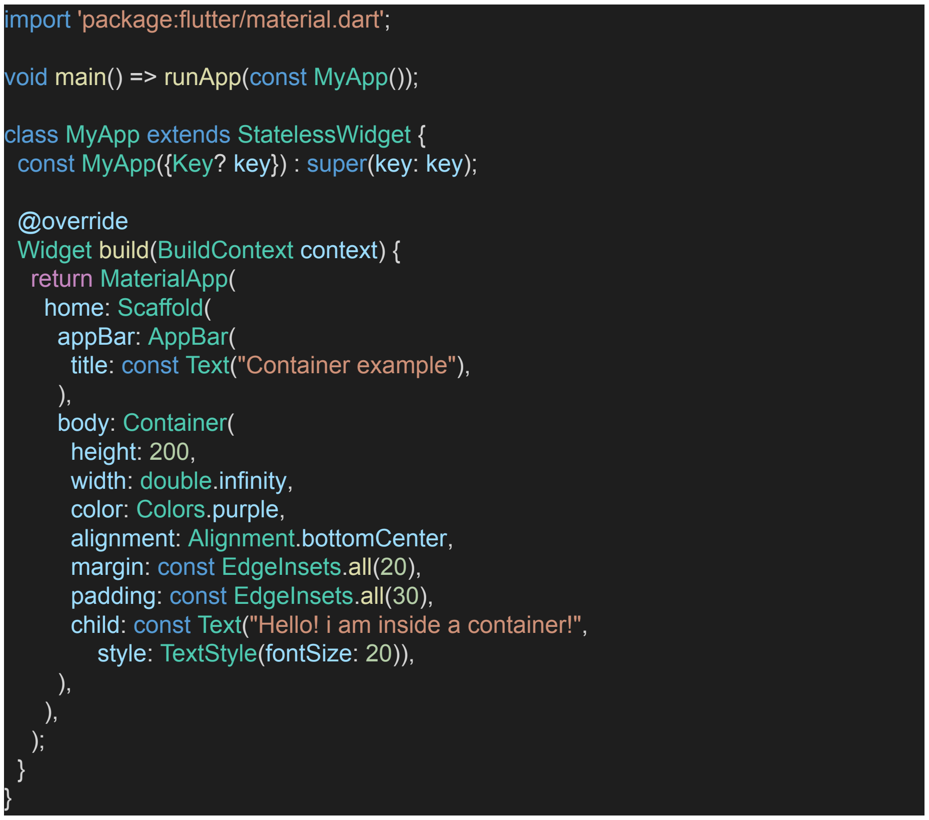
```
 Widget build(BuildContext context) {
   return MaterialApp(
     home: Scaffold(
       appBar: AppBar(
         title: const Text("Container example"),
       ),
       body: Container(
         height: 200,
         width: double.infinity,
         color: Colors.purple,
         margin: const EdgeInsets.all(20),
         child: const Text("Hello! i am inside a container!",
             style: TextStyle(fontSize: 20)),
       ),
     ),
   );
 }
}
```

5. padding: The padding is used to give space from the border of the container from its children. Observe the space between the border and the text.

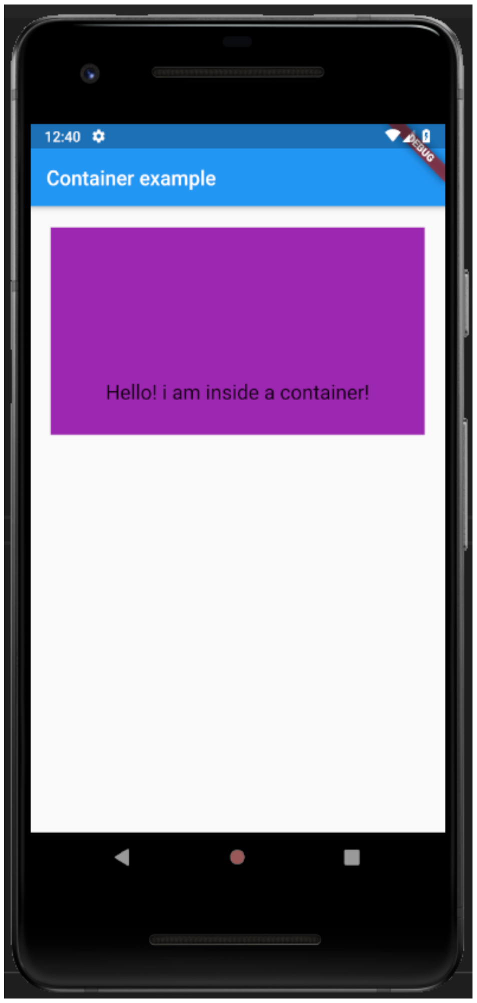

12:34
Container example
Hello! i am inside a container!

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

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

class MyApp extends StatelessWidget {
 const MyApp({Key? key}) : super(key: key);

 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     home: Scaffold(
       appBar: AppBar(
         title: const Text("Container example"),
       ),
       body: Container(
         height: 200,
         width: double.infinity,
         color: Colors.purple,
         margin: const EdgeInsets.all(20),
         padding: const EdgeInsets.all(30),
         child: const Text("Hello! i am inside a container!",
             style: TextStyle(fontSize: 20)),
       ),
     ),
   );
 }
}
```

6. alignment: The alignment is used to position the child within the container. We can align in different ways: bottom, bottom center, left, right, etc. here the child is aligned to the bottom center.

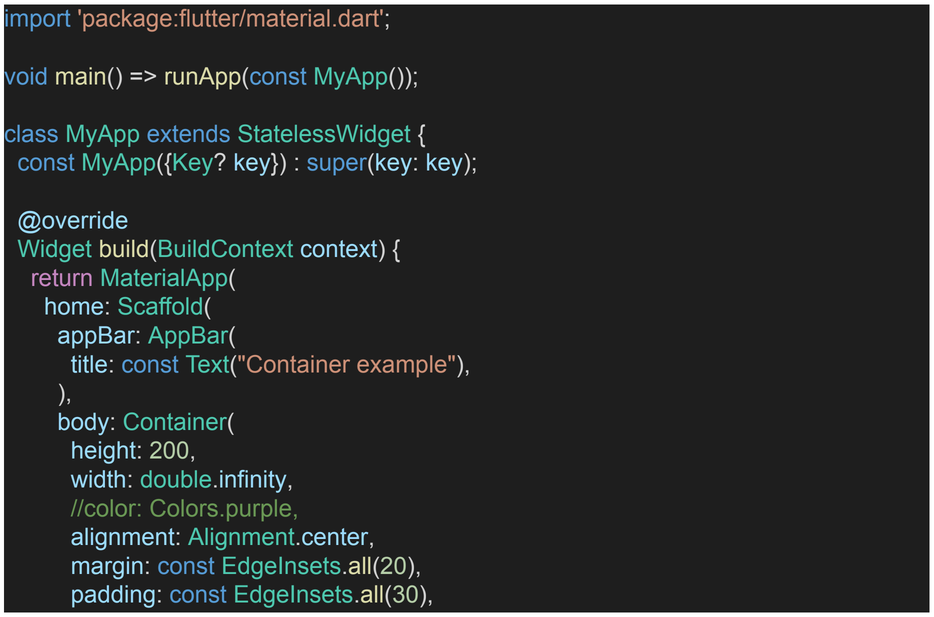


```
import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text("Container example"),
        ),
        body: Container(
          height: 200,
          width: double.infinity,
          color: Colors.purple,
          alignment: Alignment.bottomCenter,
          margin: const EdgeInsets.all(20),
          padding: const EdgeInsets.all(30),
          child: const Text("Hello! i am inside a container!",
              style: TextStyle(fontSize: 20)),
        ),
      ),
    );
  }
}
```

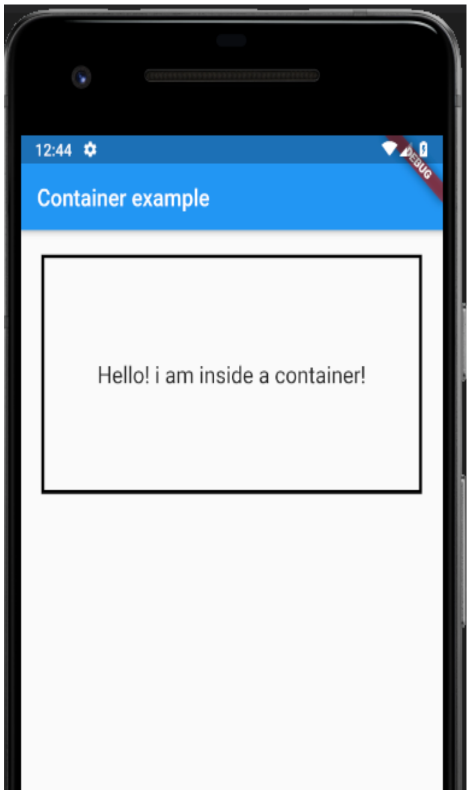


Hello! i am inside a container!

7. Decoration: The decoration property is used to decorate the box(e.g. give a border). This paints behind the child. Whereas foreground Decoration paints in front of a child. Let us give a border to the container. But, both color and border color cannot be given.



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

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text("Container example"),
        ),
        body: Container(
          height: 200,
          width: double.infinity,
          //color: Colors.purple,
          alignment: Alignment.center,
          margin: const EdgeInsets.all(20),
          padding: const EdgeInsets.all(30),
```

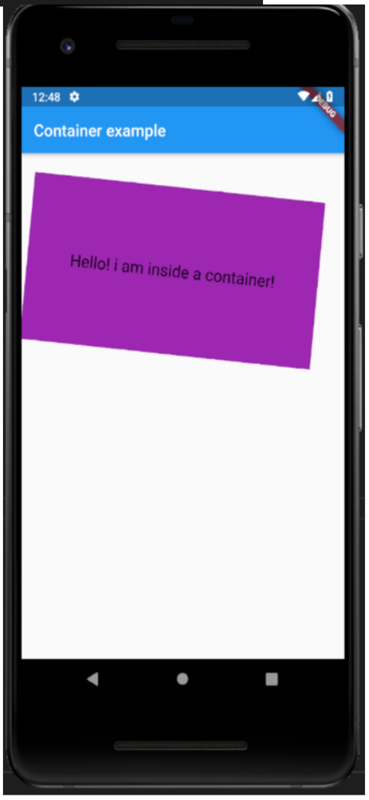


Hello! i am inside a container!

```
        decoration: BoxDecoration(
          border: Border.all(color: Colors.black, width: 3),
        ),
        child: const Text("Hello! i am inside a container!",
            style: TextStyle(fontSize: 20)),
      ),
    ),
  );
 }
}
```

8. Transform: This property of the container helps us to rotate the container. We can rotate the container in any axis, here we are rotating in the z-axis.

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

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

class MyApp extends StatelessWidget {
 const MyApp({Key? key}) : super(key: key);

 @override
 Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        title: const Text("Container example"),
      ),
      body: Container(
        height: 200,
        width: double.infinity,
        color: Colors.purple,
        alignment: Alignment.center,
        margin: const EdgeInsets.all(20),
        padding: const EdgeInsets.all(30),
        transform: Matrix4.rotationZ(0.1),
        child: const Text("Hello! i am inside a container!",
            style: TextStyle(fontSize: 20)),
      ),
    ),
  );
 }
}
```



9. Constraints: When we want to give additional constraints to the child, we can use this property.

10. ClipBehaviour: This property takes in *Clip* Enum as the object. This decides whether the content inside the container will be clipped or not.

11. Foreground Decoration: This parameter holds *Decoration* class as the object. It controls the decoration in front of the Container widget.

Scaffold class in Flutter

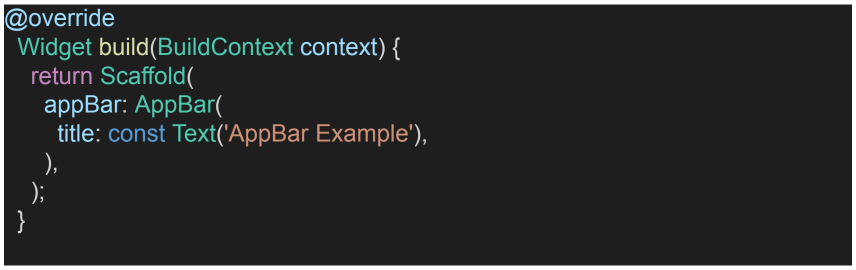
Scaffold is a class in flutter which provides many widgets or we can say APIs like Drawer, Snack-Bar, Bottom-Navigation-Bar, Floating-Action-Button, App-Bar, etc. Scaffold will expand or occupy the whole device screen. It will occupy the available space. Scaffold will provide a framework to implement the basic material design layout of the application.

Constructor of the Scaffold class:

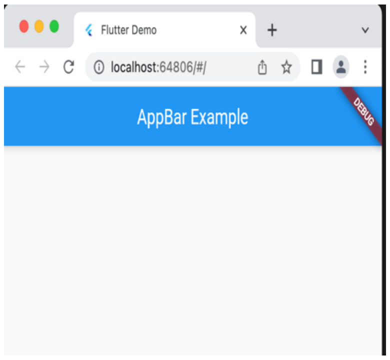
```
const Scaffold({
    Key key,
    this.appBar,
    this.body,
    this.floatingActionButton,
    this.floatingActionButtonLocation,
    this.floatingActionButtonAnimator,
    this.persistentFooterButtons,
    this.drawer,
    this.endDrawer,
    this.bottomNavigationBar,
    this.bottomSheet,
    this.backgroundColor,
    this.resizeToAvoidBottomPadding,
    this.resizeToAvoidBottomInset,
    this.primary = true,
    this.drawerDragStartBehavior
          = DragStartBehavior.start,
    this.extendBody = false,
    this.drawerScrimColor,
})
```

Properties of Scaffold Class:

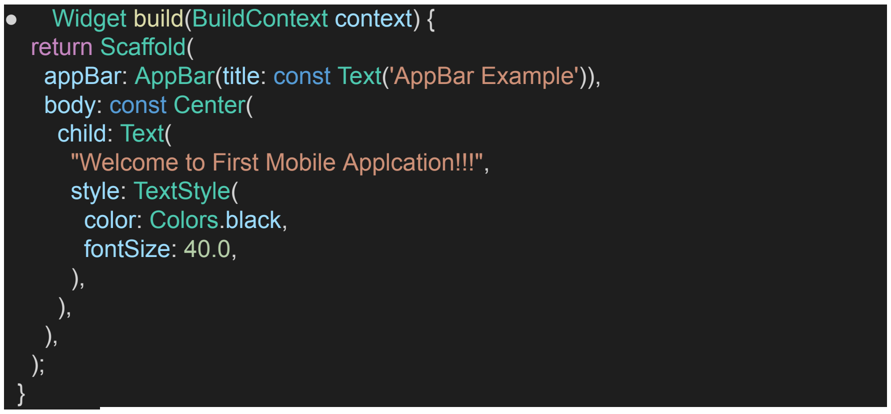
- **app-Bar**: It displays a horizontal bar which is mainly placed at the top of the Scaffold. appBar uses the widget AppBar which has its own properties like elevation, title, brightness, etc.



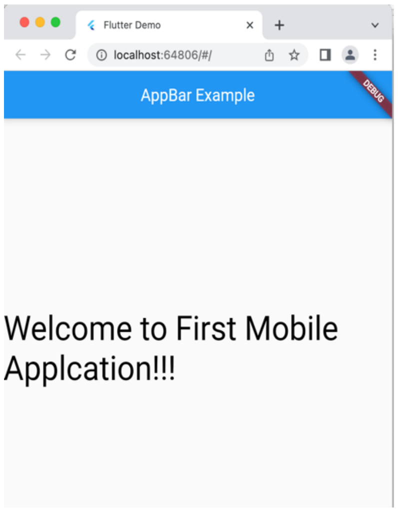
```
@override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(
       title: const Text('AppBar Example'),
     ),
   );
 }
```



- **body:** It will display the main or primary content in the Scaffold. It is below the *appBar* and under the *floatingActionButton*. The widgets inside the body are at the left-corner by default.

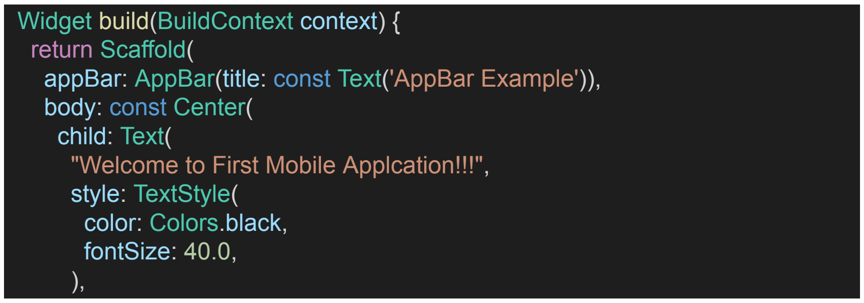


```
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(title: const Text('AppBar Example')),
     body: const Center(
       child: Text(
         "Welcome to First Mobile Applcation!!!",
         style: TextStyle(
           color: Colors.black,
           fontSize: 40.0,
         ),
       ),
     ),
   );
 }
```

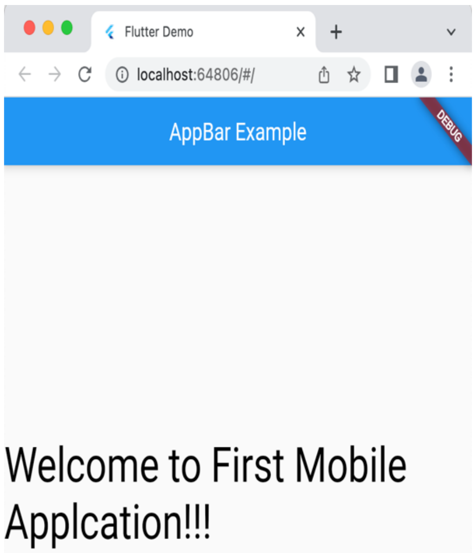


Welcome to First Mobile Applcation!!!

- **floatingActionButton:** *FloatingActionButton* is a button that is placed at the right bottom corner by default. *FloatingActionButton* is an icon button that floats over the content of the screen at a fixed place. If we scroll the page its position won't change, it will be fixed.



```
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: const Text('AppBar Example')),
    body: const Center(
      child: Text(
        "Welcome to First Mobile Applcation!!!",
        style: TextStyle(
          color: Colors.black,
          fontSize: 40.0,
        ),
```

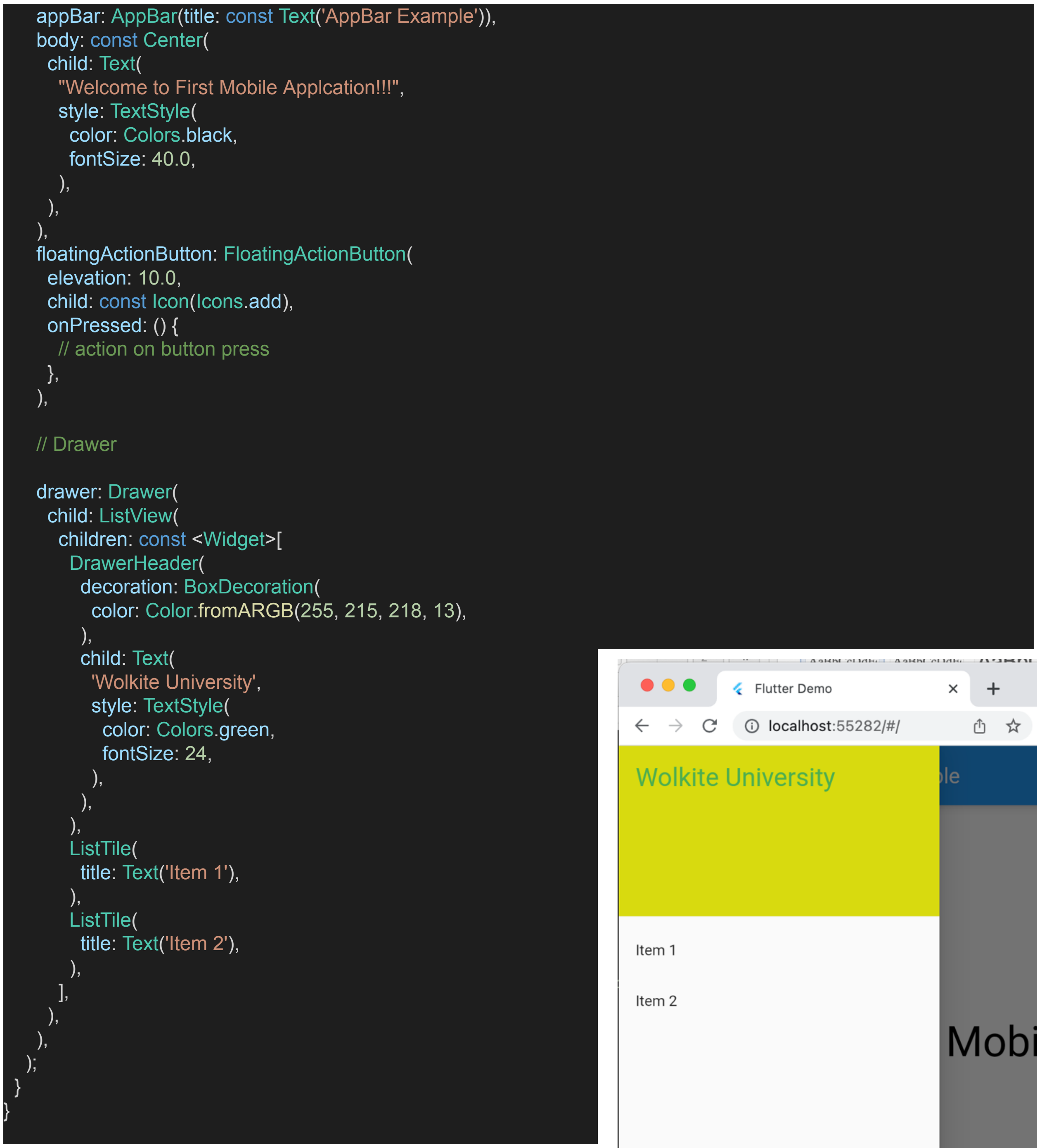


Welcome to First Mobile Applcation!!!

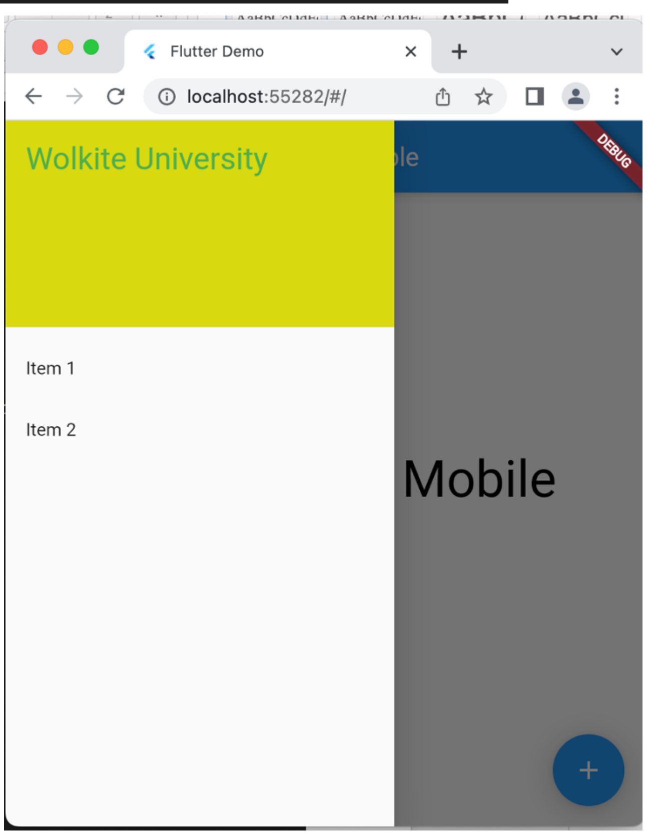
```
        ),
      ),
      floatingActionButton: FloatingActionButton(
        elevation: 10.0,
        child: const Icon(Icons.add),
        onPressed: () {
          // action on button press
        },
      ),
    );
  }
}
```

- **drawer:** *drawer* is a slider menu or a panel which is displayed at the side of the Scaffold. The user has to swipe left to right or right to left according to the action defined to access the drawer menu. In the Appbar, an appropriate icon for the drawer is set automatically at a particular position. The gesture to open the drawer is also set automatically. It is handled by the Scaffold.

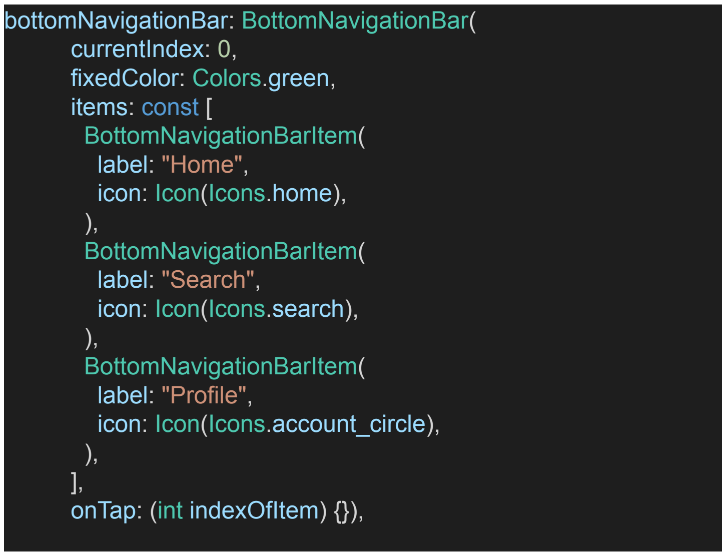
```
import 'package:flutter/material.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: const MyHomePage(title: 'IT 3rd First Lab'),
    );
  }
}

class MyHomePage extends StatefulWidget {
  const MyHomePage({Key? key, required this.title}) : super(key: key);

  final String title;

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  Widget build(BuildContext context) {
    return Scaffold(
```



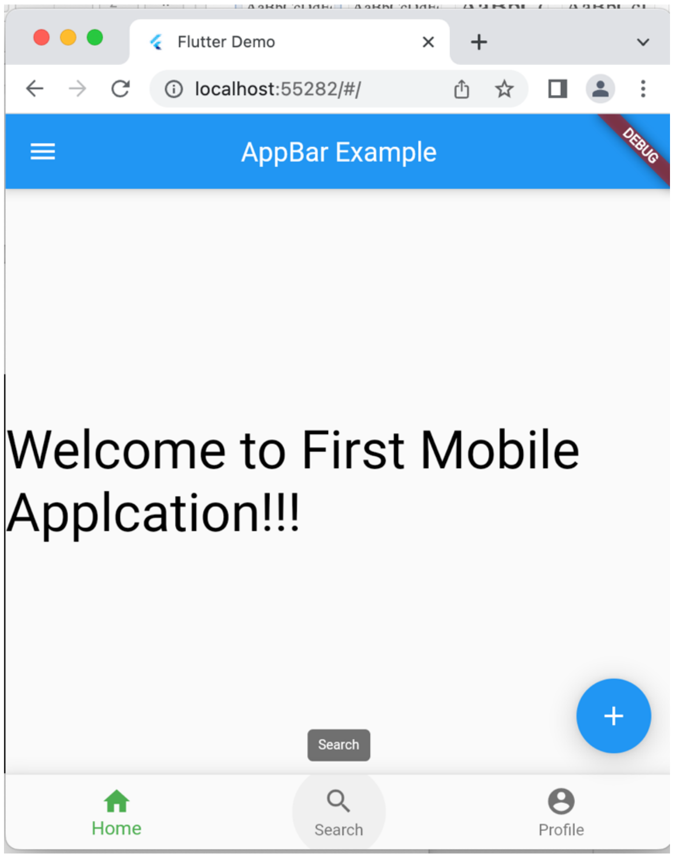
```
    appBar: AppBar(title: const Text('AppBar Example')),
    body: const Center(
      child: Text(
        "Welcome to First Mobile Applcation!!!",
        style: TextStyle(
          color: Colors.black,
          fontSize: 40.0,
        ),
      ),
    ),
    floatingActionButton: FloatingActionButton(
      elevation: 10.0,
      child: const Icon(Icons.add),
      onPressed: () {
        // action on button press
      },
    ),

    // Drawer

    drawer: Drawer(
      child: ListView(
        children: const <Widget>[
          DrawerHeader(
            decoration: BoxDecoration(
              color: Color.fromARGB(255, 215, 218, 13),
            ),
            child: Text(
              'Wolkite University',
              style: TextStyle(
                color: Colors.green,
                fontSize: 24,
              ),
            ),
          ),
          ListTile(
            title: Text('Item 1'),
          ),
          ListTile(
            title: Text('Item 2'),
          ),
        ],
      ),
    ),
  );
 }
}
```



- **bottomNavigationBar:** *bottomNavigationBar* is like a menu at the bottom of the Scaffold. We have seen this navigation bar in most of the applications. We can add multiple icons or texts or both in the bar as items.



```
bottomNavigationBar: BottomNavigationBar(
      currentIndex: 0,
      fixedColor: Colors.green,
      items: const [
        BottomNavigationBarItem(
          label: "Home",
          icon: Icon(Icons.home),
        ),
        BottomNavigationBarItem(
          label: "Search",
          icon: Icon(Icons.search),
        ),
        BottomNavigationBarItem(
          label: "Profile",
          icon: Icon(Icons.account_circle),
        ),
      ],
      onTap: (int indexOfItem) {}),
```



Welcome to First Mobile Applcation!!!