Chapter 3: Flutter Widgets & Styling

3.1. Material App Widget

MaterialApp is a predefined class or widget in a flutter. It is likely the main or core component of a flutter app. The MaterialApp widget provides a wrapper around other Material Widgets. We can access all the other components and widgets provided by Flutter SDK. Text widget, DropdownButton widget, AppBar widget, Scaffold widget, ListView widget, StatelessWidget, StatefulWidget, IconButton widget, TextField widget, Padding widget, ThemeData widget, etc. are the widgets that can be accessed using MaterialApp class. There are many more widgets that are accessed using MaterialApp class. Using this widget, we can make an attractive app that follows the Material Design guidelines.

Constructor of MaterialApp class

Here is the constructor of MaterialApp class:

```
const MaterialApp(
{Key key,
GlobalKey<NavigatorState> navigatorKey,
Widget home,
Map<String, WidgetBuilder> routes: const <String,
WidgetBuilder>{},
String initialRoute,
RouteFactory onGenerateRoute,
InitialRouteListFactory onGenerateInitialRoutes,
RouteFactory onUnknownRoute,
List<NavigatorObserver> navigatorObservers: const
<NavigatorObserver>[],
TransitionBuilder builder,
String title: '',
GenerateAppTitle onGenerateTitle,
Color color,
ThemeData theme,
ThemeData darkTheme,
ThemeData highContrastTheme,
ThemeData highContrastDarkTheme,
ThemeMode themeMode: ThemeMode.system,
Locale locale,
Iterable<LocalizationsDelegate> localizationsDelegates,
LocaleListResolutionCallback localeListResolutionCallback,
LocaleResolutionCallback localeResolutionCallback,
Iterable<Locale> supportedLocales: const <Locale>[Locale('en',
'US')],
bool debugShowMaterialGrid: false,
bool showPerformanceOverlay: false,
bool checkerboardRasterCacheImages: false,
bool checkerboardOffscreenLayers: false,
bool showSemanticsDebugger: false,
bool debugShowCheckedModeBanner: true,
Map<LogicalKeySet, Intent> shortcuts,
Map<Type, Action<Intent>> actions}
)
```

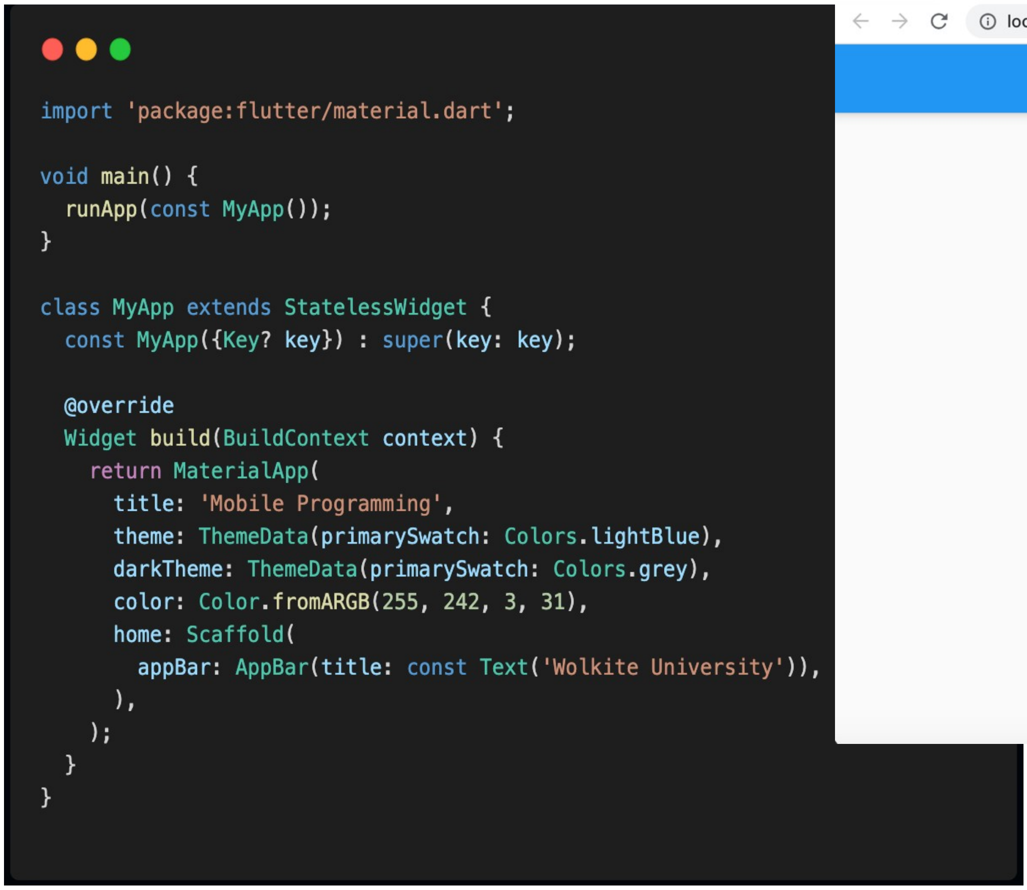
Properties of MaterialApp widget:

- **action:** This property takes in *Map<Type, Action<Intent>>* as the object. It controls intent keys.
- **backButtonDispatcher:** It decided how to handle the back button.
- **checkerboardRasterCacheImage:** This property takes in a boolean as the object. If set to true it turns on the checkerboarding of raster cache images.
- **color:** It controls the primary color used in the application.
- **darkTheme:** It provided theme data for the dark theme for the application.
- **debugShowCheckedModeBanner:** This property takes in a *boolean* as the object to decide whether to show the debug banner or not.
- **debugShowMaterialGird:** This property takes a boolean as the object. If set to true it paints a baseline grid material app.
- **highContrastDarkTheme:** It provided the theme data to use for the high contrast theme.

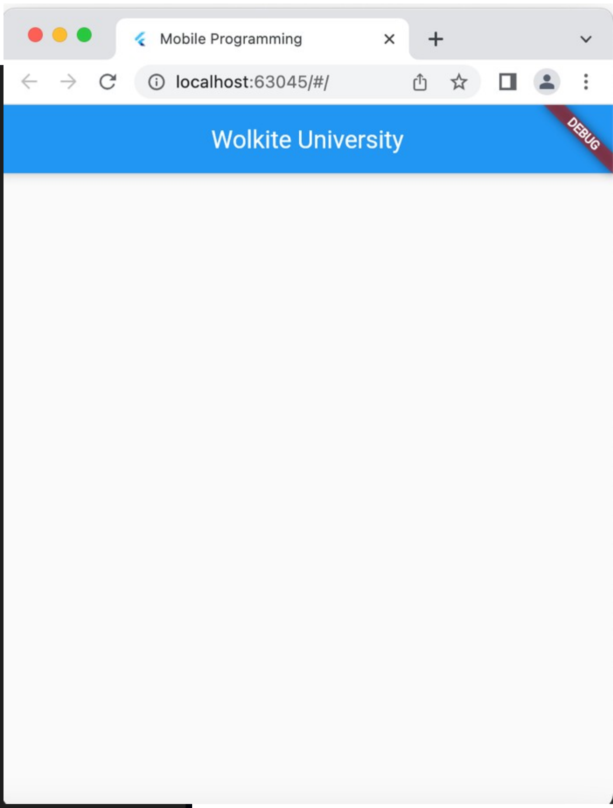
- **home:** This property takes in *widget* as the object to show on the default route of the app.
- **initialRoute:** This property takes in a *string* as the object to give the name of the first route in which the navigator is built.
- **locale:** It provides a locale for the *MaterialApp*.
- **localizationsDelegate:** This provides a delegate for the locales.
- **navigatorObserver:** It takes in *GlobalKey<NavigatorState>* as the object to generate a key when building a navigator.
- **navigatorObserver:** This property holds *List<NavigatorObserver>* as the object to create a list of observers for the navigator.
- **onGenerateInitialRoutes:** This property takes in *InitialRouteListFactory typedef* as the object to generate initial routes.
- **onGeneratRoute:** The *onGenerateRoute* takes in a *RouteFactory* as the object. It is used when the app is navigated to a named route.
- **onGenerateTitle:** This property takes in *RouteFactory typedef* as the object to generate a title string for the application if provided.
- **onUnknownRoute:** The *onUnknownRoute* takes in *RouteFactory typedef* as the object to provide a route in case of failure in other metheod.
- **routeInformationParse:** This property holds *RouteInformationParser<T>* as the object to the routing information from the routeInformationProvider into a generic data type.
- **routeInformationProvider:** This property takes in *RouteInformationProvider* class as the object. It is responsible for providing routing information.
- **routeDelegate:** This property takes in *RouterDelegate<T>* as the object to configure a given widget.
- **routes:** The *routes* property takes in L*ogicalKeySet* class as the object to control the app's topmost level routing.
- **shortcuts:** This property takes in *LogicalKeySet* class as the object to decide the keyboard shortcut for the application.
- **showPerformanceOverlay:** The *showPerformanceOverlay* takes in a *boolean* value as the object to turn on or off performance overlay.

- **showSemantisDebugger:** This property takes in a *boolean* as the object. If set to true, it shows some accessible information.
- **supportedLocales:** The *supportedLocales* property keeps hold of the locals used in the app by taking in *Iterable<E>* class as the object.
- **theme:** This property takes in *ThemeData* class as the object to describe the theme for the MaterialApp.
- **themeMode:** This property holds *ThemeMode enum* as the object to decide the theme for the material app.
- **title:** The *title* property takes in a *string* as the object to decide the one-line description of the app for the device.

Here is very simple code in dart language to make a screen that has an AppBar title as GeeksforGeeks.



```dart
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(
      title: 'Mobile Programming',
      theme: ThemeData(primarySwatch: Colors.lightBlue),
      darkTheme: ThemeData(primarySwatch: Colors.grey),
      color: Color.fromARGB(255, 242, 3, 31),
      home: Scaffold(
        appBar: AppBar(title: const Text('Wolkite University')),
      ),
    );
  }
}
```



⇒ Here we can see that the text defined in the title of the AppBar is displayed at the top.

⇒ The default theme color is green as we defined.

⇒ runApp() calls the widget ***MyApp*** that returns the MaterialApp widget which specifies theme, localization, title, home widget and more.

3.2. Scaffold Widget

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.

Properties of Scaffold Class:

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

```
Widget build(BuildContext context)
{
return Scaffold(
  appBar: AppBar(
  title: const Text('Mobile Application Development'),
  ),
```

- **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.

Example:

```
Widget build(BuildContext context) {
return Scaffold(
  appBar: AppBar(title: const Text('Mobile Application Development')),
  body: const Center(
  child: Text(
    "Content inside body",
    style: TextStyle(
    color: Colors.black,
    fontSize: 40.0,
    ),
  ),
  ),
);
}
```

- **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('Mobile Application Development')),
  body: const Center(
   child: Text(
    "Welcome to Wolkite University!!",
    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* 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.

```
drawer: Drawer(
 child: ListView(
  children: const <Widget>[
   DrawerHeader(
    decoration: BoxDecoration(
     color: Colors.green,
    ),
    child: Text(
     'GeeksforGeeks',
     style: TextStyle(
      color: Colors.green,
      fontSize: 24,
     ),
    ),
   ),
   ListTile(
    title: Text('Item 1'),
   ),
   ListTile(
    title: Text('Item 2'),
   ),
  ],
 ),
),
```

•

3.3. Container & SizedBox Widget

Container Widget

- refer content form chapter 2

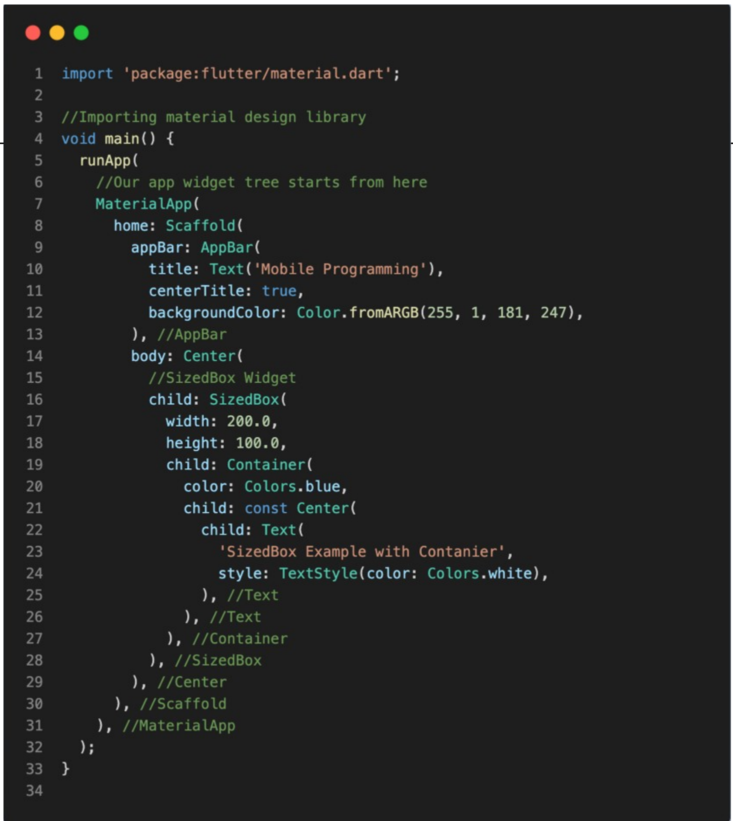
SizedBox Widget

SizedBox is a built-in widget in flutter SDK. It is a simple box with a specified size. It can be used to set size constraints to the child widget, put an empty SizedBox between the two widgets to get some space in between, or something else. It is somewhat similar to a **Container widget** with fewer properties.

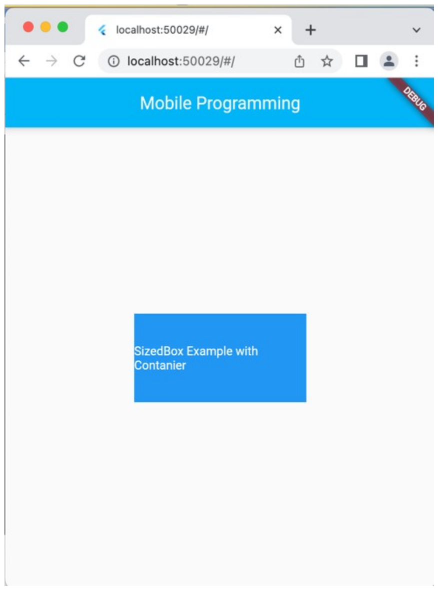
It draws a simple box with the mentioned height and width or a child widget inside.

Constructor of SizedBox class

```
const SizedBox(
{Key key,
double width,
double height,
Widget child})
```



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

//Importing material design library
void main() {
  runApp(
    //Our app widget tree starts from here
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Mobile Programming'),
          centerTitle: true,
          backgroundColor: Color.fromARGB(255, 1, 181, 247),
        ), //AppBar
        body: Center(
          //SizedBox Widget
          child: SizedBox(
            width: 200.0,
            height: 100.0,
            child: Container(
              color: Colors.blue,
              child: const Center(
                child: Text(
                  'SizedBox Example with Contanier',
                  style: TextStyle(color: Colors.white),
                ), //Text
              ), //Text
            ), //Container
          ), //SizedBox
        ), //Center
      ), //Scaffold
    ), //MaterialApp
  );
}
```



3.4. Card Widget

Card is a build-in widget in flutter which derives its design from Google’s Material Design Library. The functionality of this widget on screen is, that it is a bland space or panel with round corners and a slight elevation on the lower side. It comes with many properties like color, shape, shadow color, etc which lets developers customize it the way they like. Below we will go through all its properties and an example to see its implementation.

Constructor of Card class:

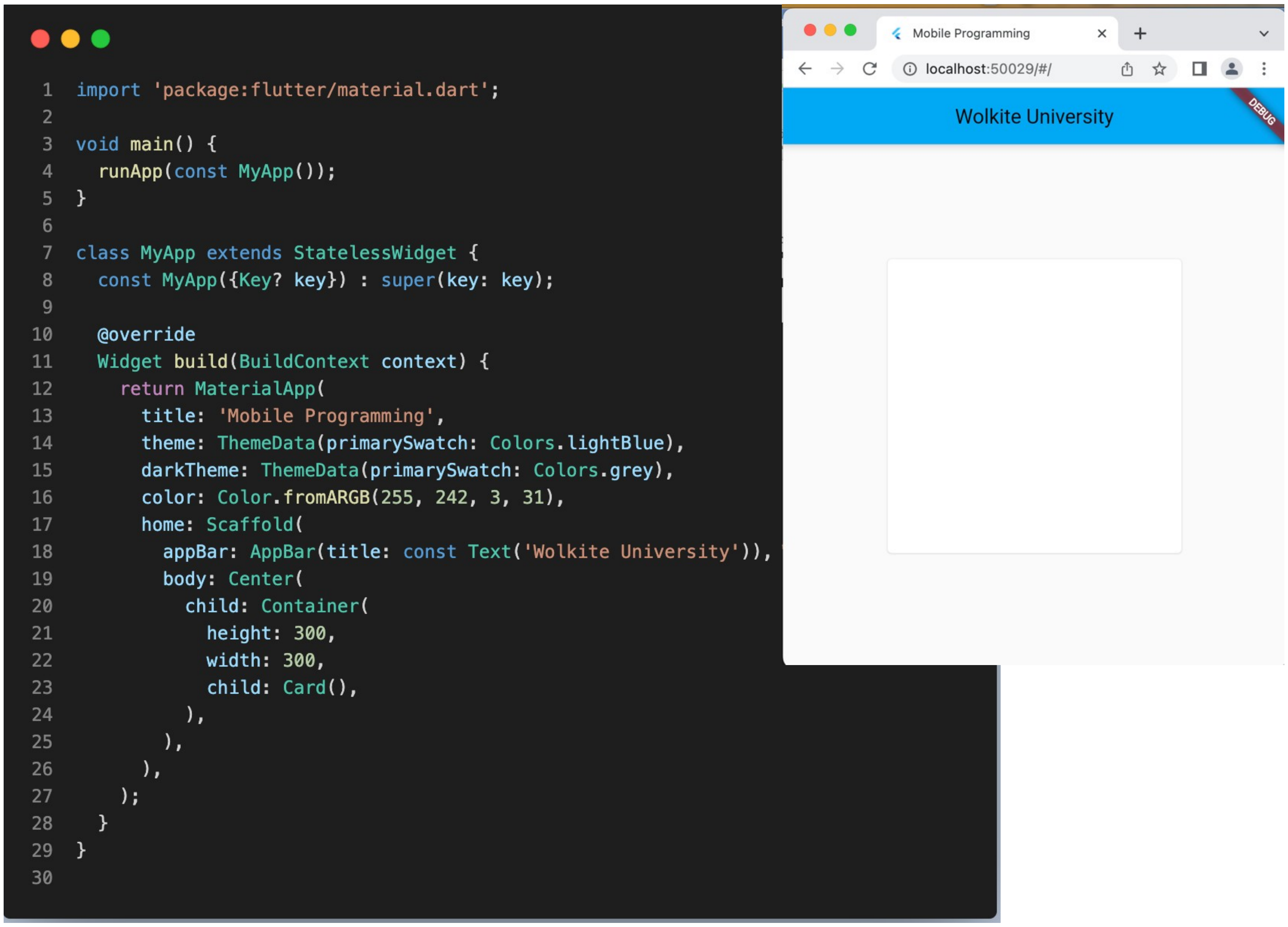
```
const Card(
{Key key,
Color color,
Color shadowColor,
double elevation,
ShapeBorder shape,
bool borderOnForeground: true,
EdgeInsetsGeometry margin,
Clip clipBehavior,
Widget child,
bool semanticContainer: true}
)
```

Properties of Card Widget:

- **borderOnForeground:** This property takes in a *boolean* value as an object to decide whether to print a border or not.
- **child:** This property takes in a widget as an object to show inside the *Card* widget.
- **clipBehavior:** This property decides whether the content inside the Card will be clipped or not. It takes *Clip* enum as an object.
- **color:** This property assigns background color to the card by taking in the *Color* class as the object.
- **elevation:** This property takes in a *double* value as the object to decide the z-coordinate where the card should be placed.
- **margin:** This property takes in *EdgeInsetsGeometry* as the object to add empty space around the *Card*.
- **semanticContainer:** This property takes in a *boolean* as the object. This controls whether the Card widget with all its child widget will be seen as a single widget or not.

- **shadowColor:** This property takes in Color class as the object to assign a color to the shadow, which appears beneath the card. By default, the color is set to black.
- **shape:** This property takes *ShapeBorder* class as the object to decide the shape of the *Card* widget.

Example:



```
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(
      title: 'Mobile Programming',
      theme: ThemeData(primarySwatch: Colors.lightBlue),
      darkTheme: ThemeData(primarySwatch: Colors.grey),
      color: Color.fromARGB(255, 242, 3, 31),
      home: Scaffold(
        appBar: AppBar(title: const Text('Wolkite University')),
        body: Center(
          child: Container(
            height: 300,
            width: 300,
            child: Card(),
          ),
        ),
      ),
    );
  }
}
```

Changing the Color

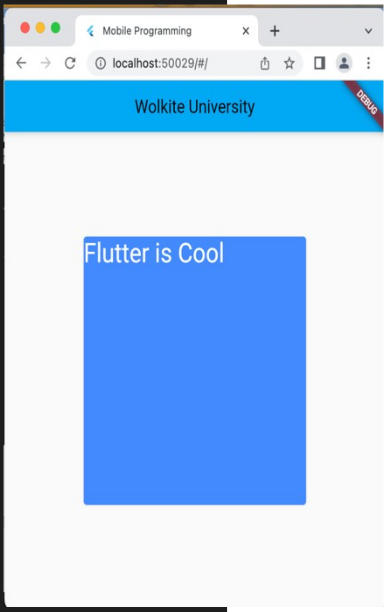
Card widget has white color from scratch. Let’s change it! We can achieve this with the help of one of its property, color.

```
Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Mobile Programming',
     theme: ThemeData(primarySwatch: Colors.lightBlue),
     darkTheme: ThemeData(primarySwatch: Colors.grey),
     color: Color.fromARGB(255, 242, 3, 31),
   home: Scaffold(
     appBar: AppBar(title: const Text('Wolkite University')),
     body: Center(
       child: Container(
         height: 300,
         width: 300,
         child: const Card(
           color: Colors.blueAccent,
         ),
       ),
     ),
   ),
  );
 }
```

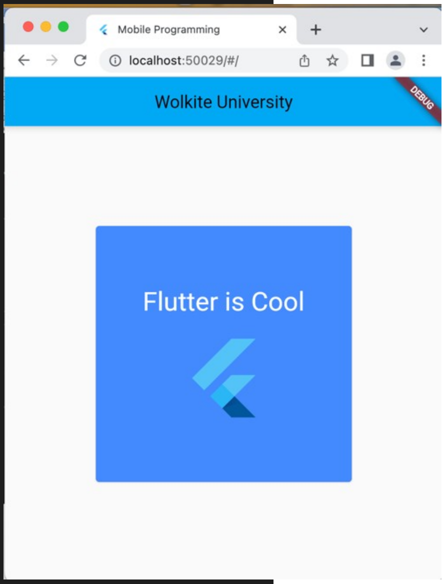
Adding Some Widgets

Card has been made but it doesn't hold something in it. Let's do it!
With the card's child property, let's add some text in it with the help of the Text widget

```
Widget build(BuildContext context) {
  return MaterialApp(
   title: 'Mobile Programming',
   theme: ThemeData(primarySwatch: Colors.lightBlue),
   darkTheme: ThemeData(primarySwatch: Colors.grey),
   color: Color.fromARGB(255, 242, 3, 31),
   home: Scaffold(
     appBar: AppBar(title: const Text('Wolkite University')),
     body: Center(
       child: Container(
         height: 300,
         width: 300,
         child: const Card(
           color: Colors.blueAccent,
           child: Text(
             Flutter is Cool,
             style: TextStyle(color: Colors.white, fontSize: 30),
           ),
         ),
       ),
     ),
   ),
  );
 }
```



```
Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Mobile Programming',
     theme: ThemeData(primarySwatch: Colors.lightBlue),
     darkTheme: ThemeData(primarySwatch: Colors.grey),
     color: Color.fromARGB(255, 242, 3, 31),
     home: Scaffold(
       appBar: AppBar(title: const Text('Wolkite University')),
       body: Center(
         child: Container(
           height: 300,
           width: 300,
           child: Card(
             color: Colors.blueAccent,
             child: Column(
               crossAxisAlignment: CrossAxisAlignment.center,
               mainAxisAlignment: MainAxisAlignment.center,
               children: [
                 const Text(
                   'Flutter is Cool',
                   style: TextStyle(color: Colors.white, fontSize: 30),
                 ),
                 SizedBox(height: 20),
                 Container(height: 100, width: 100, child: FlutterLogo()),
               ],
             ),
           ),
         ),
       ),
     ),
   );
  }
```



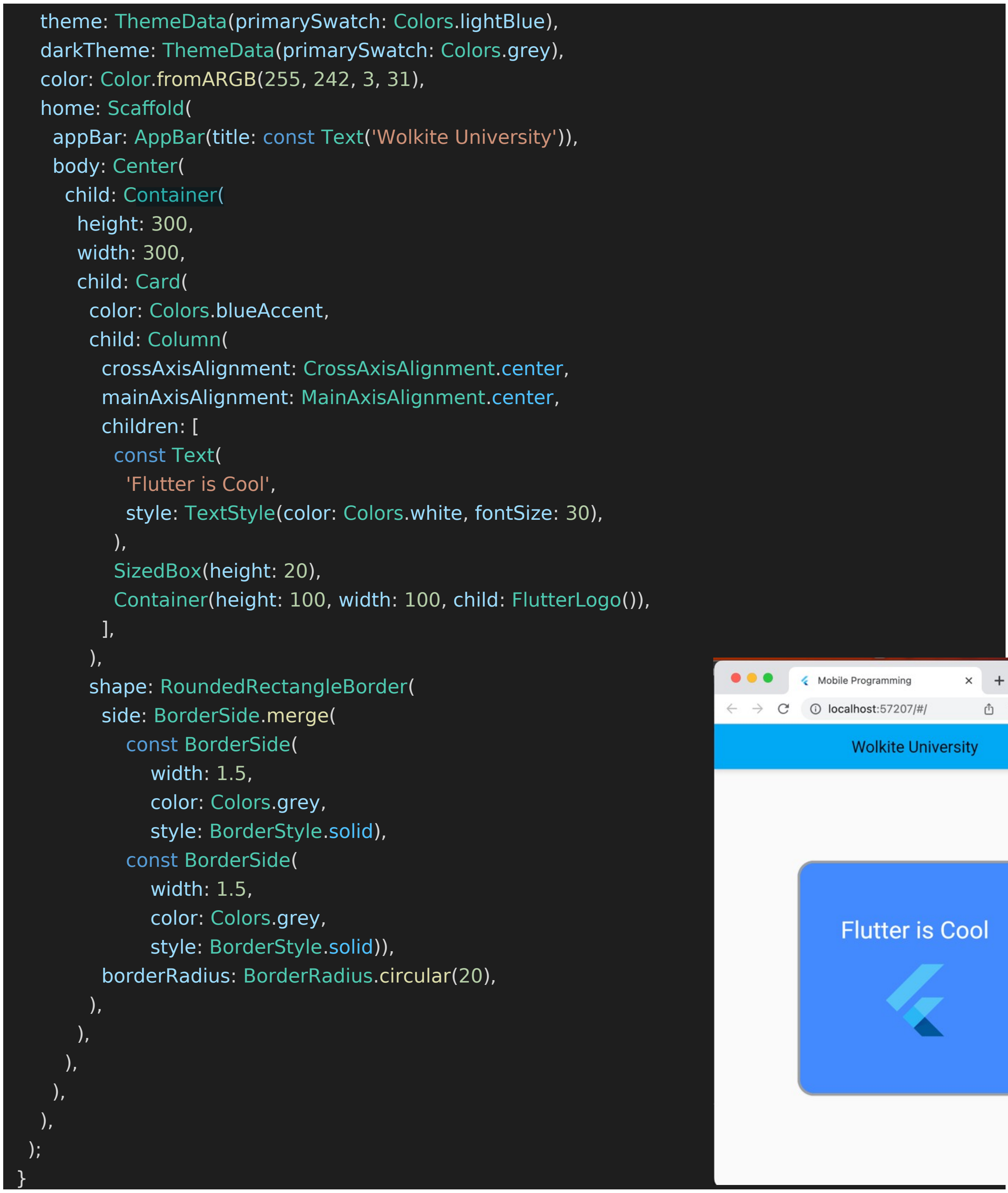
Wolkite University

Flutter is Cool

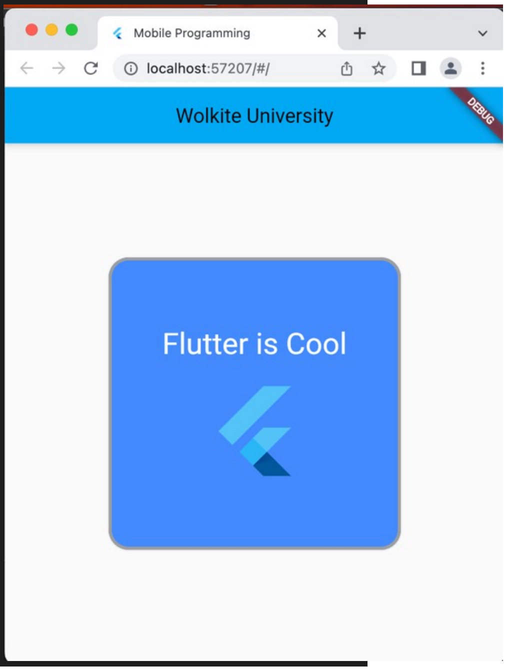
Changing the Shape

Now, let’s try to change the shape of our premade Card. For that, we can use the shape property of the card. Let’s change it to a ***RoundedRectangleBorder*** with a border and radius.

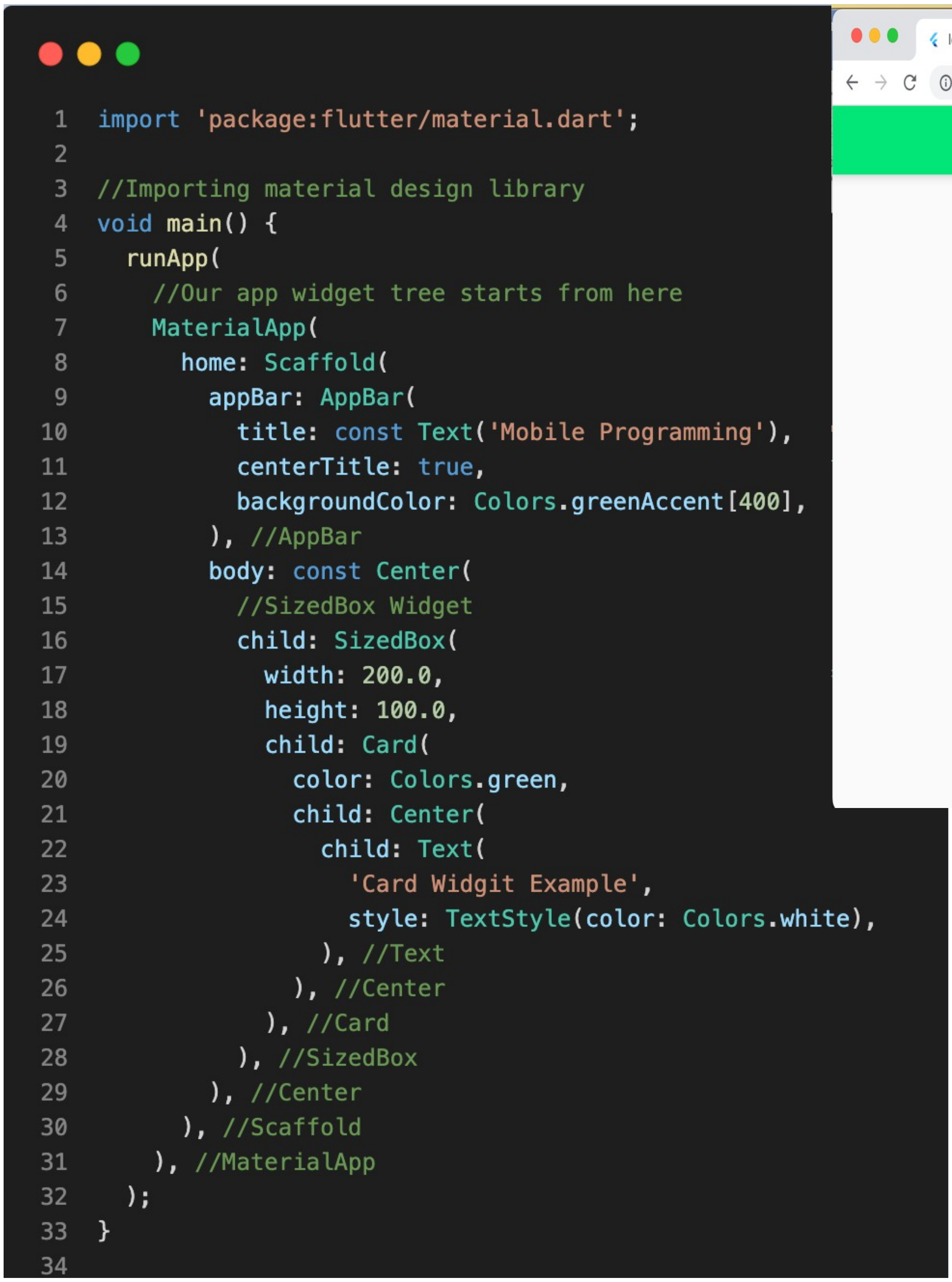
```
Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Mobile Programming',
```



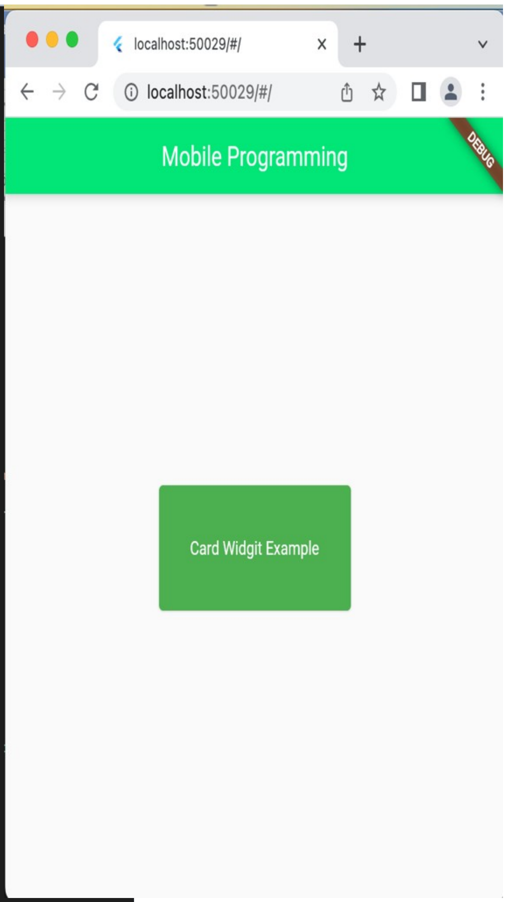
```
    theme: ThemeData(primarySwatch: Colors.lightBlue),
    darkTheme: ThemeData(primarySwatch: Colors.grey),
    color: Color.fromARGB(255, 242, 3, 31),
    home: Scaffold(
      appBar: AppBar(title: const Text('Wolkite University')),
      body: Center(
        child: Container(
          height: 300,
          width: 300,
          child: Card(
            color: Colors.blueAccent,
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.center,
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                const Text(
                  'Flutter is Cool',
                  style: TextStyle(color: Colors.white, fontSize: 30),
                ),
                SizedBox(height: 20),
                Container(height: 100, width: 100, child: FlutterLogo()),
              ],
            ),
            shape: RoundedRectangleBorder(
              side: BorderSide.merge(
                  const BorderSide(
                      width: 1.5,
                      color: Colors.grey,
                      style: BorderStyle.solid),
                  const BorderSide(
                      width: 1.5,
                      color: Colors.grey,
                      style: BorderStyle.solid)),
              borderRadius: BorderRadius.circular(20),
            ),
          ),
        ),
      ),
    ),
  );
}
```



Example: Card widget



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

//Importing material design library
void main() {
  runApp(
    //Our app widget tree starts from here
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Mobile Programming'),
          centerTitle: true,
          backgroundColor: Colors.greenAccent[400],
        ), //AppBar
        body: const Center(
          //SizedBox Widget
          child: SizedBox(
            width: 200.0,
            height: 100.0,
            child: Card(
              color: Colors.green,
              child: Center(
                child: Text(
                  'Card Widgit Example',
                  style: TextStyle(color: Colors.white),
                ), //Text
              ), //Center
            ), //Card
          ), //SizedBox
        ), //Center
      ), //Scaffold
    ), //MaterialApp
  );
}

```



Mobile Programming

Card Widgit Example

3.5. Column & Row widget

Row and Column are the two most important and powerful widgets in Flutter. These widgets let you align children horizontally and vertically as per the requirement. As we know that when we design any UI(User Interface) in a flutter, we need to arrange its content in the Row and Column manner so these Row and Column widgets are required when designing UI.

Constructor Of Column Class:

```
Column(
{Key key,
MainAxisAlignment mainAxisAlignment: MainAxisAlignment.start,
MainAxisSize mainAxisSize: MainAxisSize.max,
CrossAxisAlignment crossAxisAlignment: 
CrossAxisAlignment.center,
TextDirection textDirection,
VerticalDirection verticalDirection: VerticalDirection.down,
TextBaseline textBaseline,
List<Widget> children: const <Widget>[]}
)
```

Constructor Of Row Class:

```
Row(
{Key key,
MainAxisAlignment mainAxisAlignment: MainAxisAlignment.start,
MainAxisSize mainAxisSize: MainAxisSize.max,
CrossAxisAlignment crossAxisAlignment: 
CrossAxisAlignment.center,
TextDirection textDirection,
VerticalDirection verticalDirection: VerticalDirection.down,
TextBaseline textBaseline: TextBaseline.alphabetic,
List<Widget> children: const <Widget>[]}
)
```

Properties of Row and Column Widgets:

- **children:** This property takes in *List<Widget>*, that is a list of widgets to display inside the *Row* or the *Column* widget.
- **clipBehaviour:** This property holds Clip class as the object to decide whether the content on the *Row* or *Column* should be clipped or not.
- **crossAxisAlignment:** The *crossAxisAlignment* takes in *CrossAxisAlignment enum* as the object to how the children's widgets should be places in *crossAxisAlignment*. For *Row* it is vertical and for *Column* it is horizontal.
- **direction:** This property holds as the *Axis enum* object to decide the direction used in the main axis. For *Row* and *Column*, it is fixed.
- **mainAxisAlignment:** This property takes in *MainAxisAlignment enum* as the object to decide how the children widgets should be place in *mainAxisAlignment*. For *Row* it is horizontal and for *Column* it is vertical.
- **mainAxisSize:** This property decides the size of main-axis by taking in *MainAxisSize enum* as the object.
- **runtimeType:** This property tells the run-time type of the *Row* or *Column* widget.
- **textBaseline:** This property is responsible for the alignment of the text in the *Row* or *Column* widget with respect to a baseline.
- **textDirection:** This property controls the text direction of the *Row* or *Column* widget, which can either be from left-to-right (by default) or right-to-left.
- **verticalDirection:** This property takes in *VerticalDirection enum* as the object to determine the order in which the children should be layered.

Row:

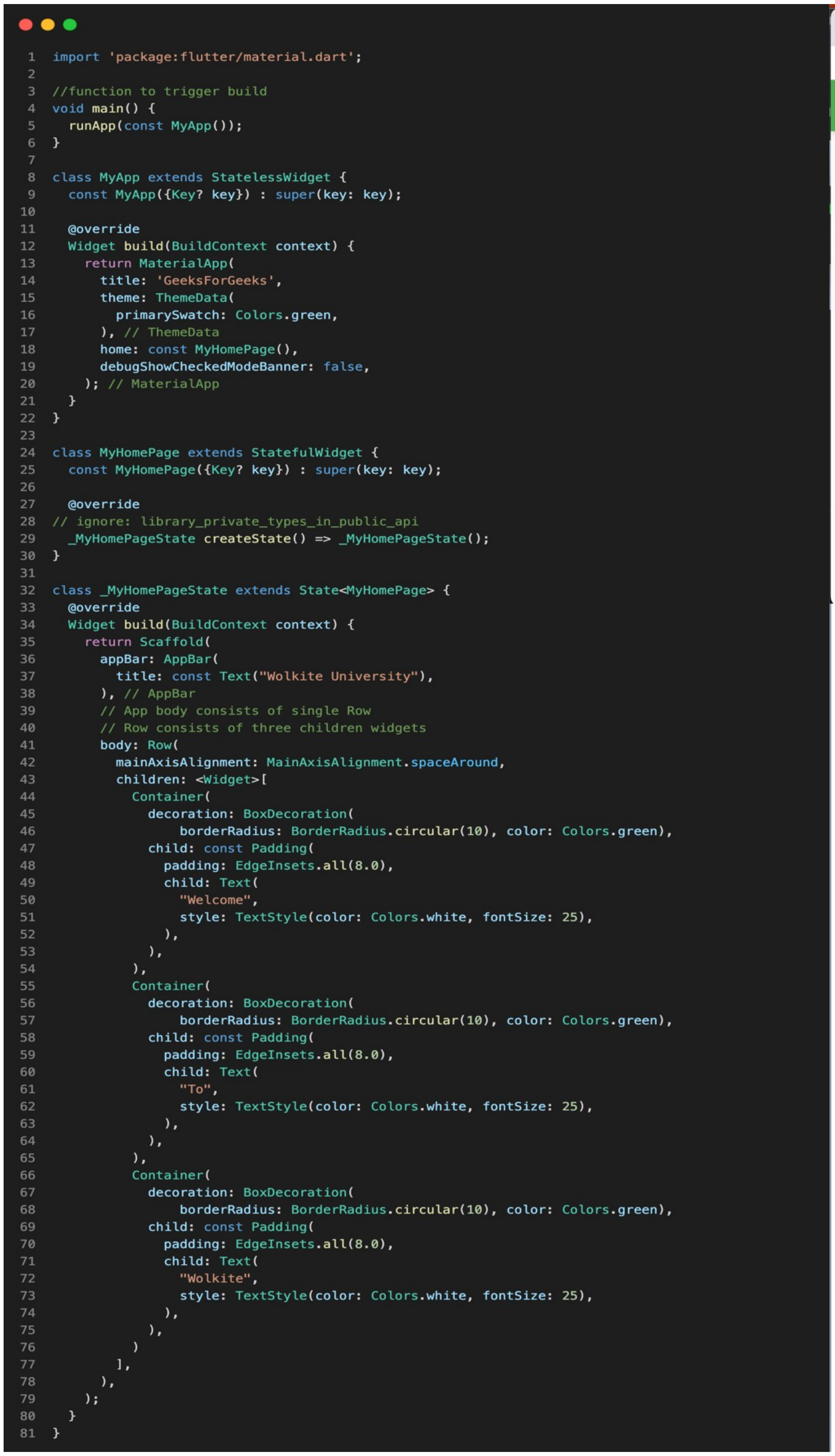
It creates a **horizontal** array of children.

Alignment Properties: We can align content as per our choice by using *mainAxisAlignment* and *crossAxisAlignment*. Row's mainAxis is horizontal and cross Axis to Row's main Axis is vertical. We can align children horizontally using MainAxisAlignment and vertically using CrossAxisAlignment in that row.

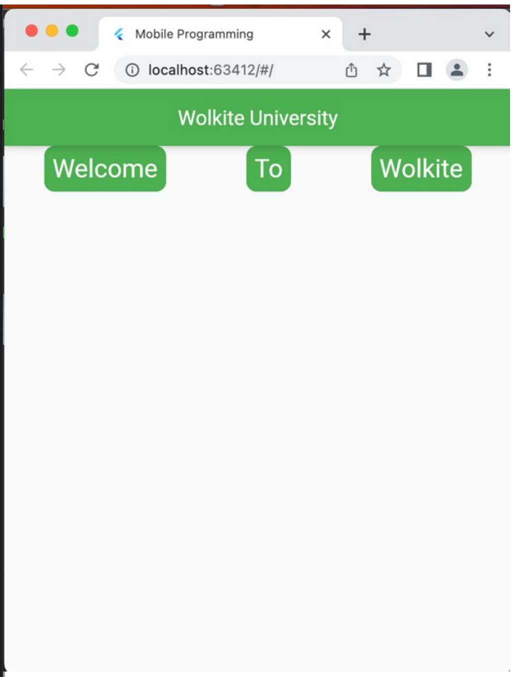
We can align the content by using the following properties:

- *start*: Place the children from the starting of the row.
- *end*: Place the children at the end of the row.
- center: Place the children at the center of the row.
- *spaceBetween*: Place the space evenly between the children.
- *spaceAround*: Place the space evenly between the children and also half of that space before and after the first and last child.
- *spaceEvenly*: Place the space evenly between the children and also before and after the first and last child.

We will see the difference with the help of examples. Let's suppose we want to align content such that there is space around the children in a row :



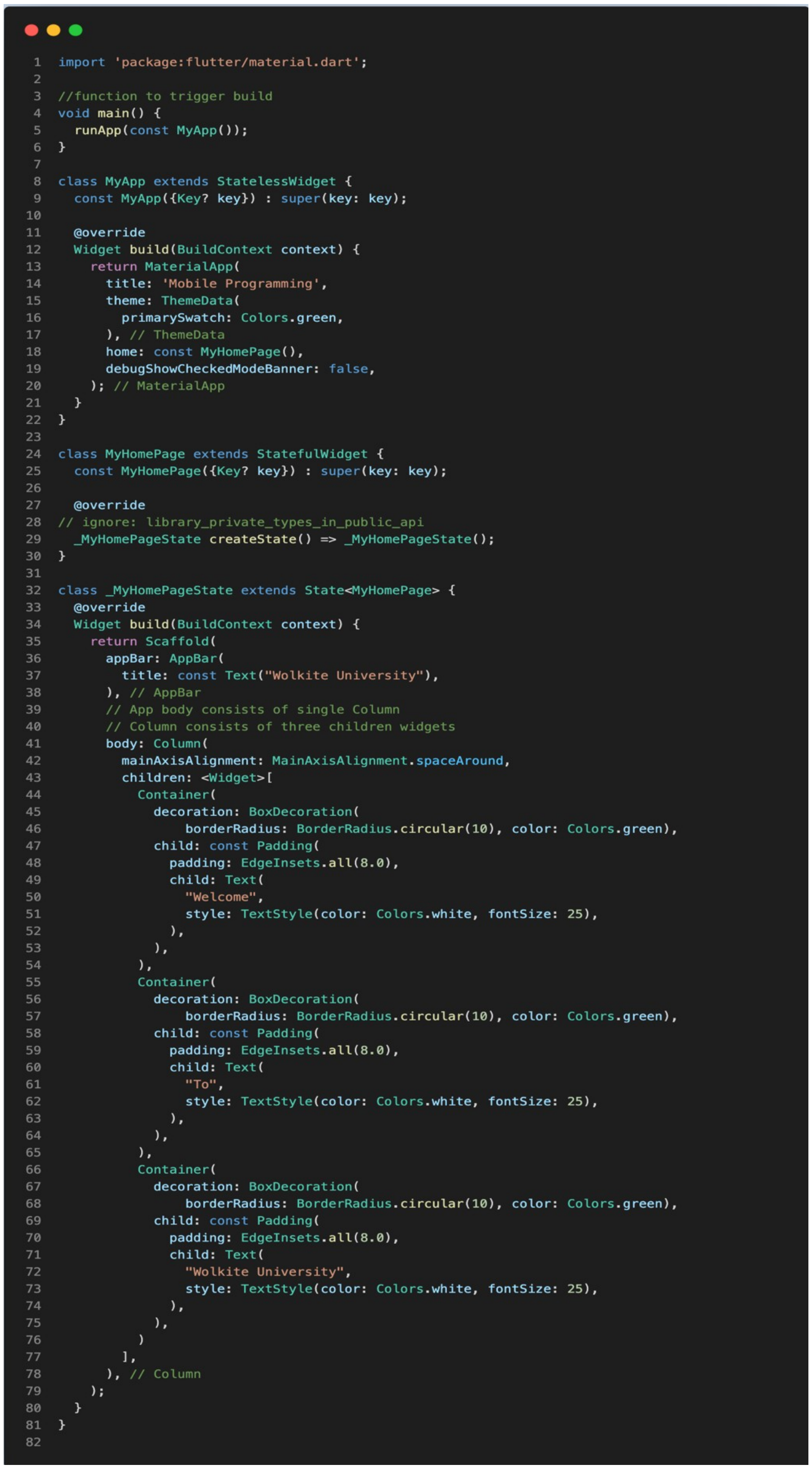
```dart
import 'package:flutter/material.dart';

//function to trigger build
void main() {
  runApp(const MyApp());
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'GeeksForGeeks',
      theme: ThemeData(
        primarySwatch: Colors.green,
      ), // ThemeData
      home: const MyHomePage(),
      debugShowCheckedModeBanner: false,
    ); // MaterialApp
  }
}

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

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

class _MyHomePageState extends State<MyHomePage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Wolkite University"),
      ), // AppBar
      // App body consists of single Row
      // Row consists of three children widgets
      body: Row(
        mainAxisAlignment: MainAxisAlignment.spaceAround,
        children: <Widget>[
          Container(
            decoration: BoxDecoration(
                borderRadius: BorderRadius.circular(10), color: Colors.green),
            child: const Padding(
              padding: EdgeInsets.all(8.0),
              child: Text(
                "Welcome",
                style: TextStyle(color: Colors.white, fontSize: 25),
              ),
            ),
          ),
          Container(
            decoration: BoxDecoration(
                borderRadius: BorderRadius.circular(10), color: Colors.green),
            child: const Padding(
              padding: EdgeInsets.all(8.0),
              child: Text(
                "To",
                style: TextStyle(color: Colors.white, fontSize: 25),
              ),
            ),
          ),
          Container(
            decoration: BoxDecoration(
                borderRadius: BorderRadius.circular(10), color: Colors.green),
            child: const Padding(
              padding: EdgeInsets.all(8.0),
              child: Text(
                "Wolkite",
                style: TextStyle(color: Colors.white, fontSize: 25),
              ),
            ),
          )
        ],
      ),
    );
  }
}
```



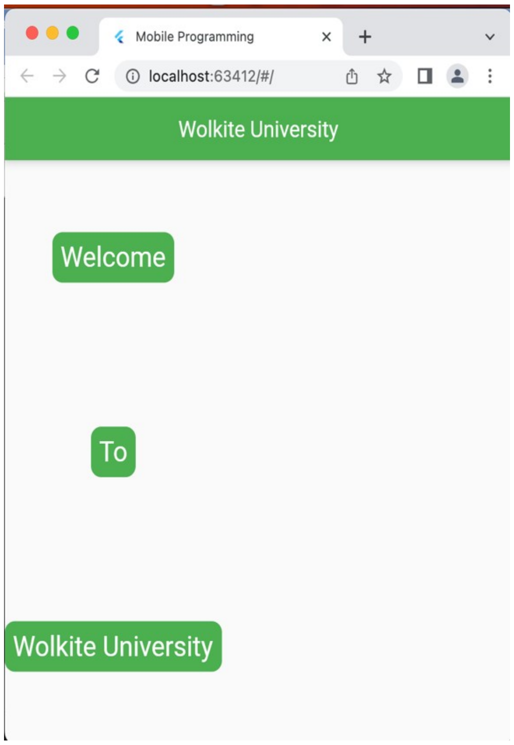
Column:
It creates a **vertical** array of children.
Alignment Properties:
In this also we have *mainAxisAlignment* and *crossAxisAlignment*. In column, children are aligned from top to bottom. Main Axis is vertical and the Cross Axis is horizontal. MainAxisAlignment aligns its children vertically and CrossAxisAlignment aligns horizontally in that Column.

We can align the content by using the same properties as discussed above in Row (start, end,spaceBetween,spaceAround,spaceEvenly)

We will see the difference with the help of examples. Let’s suppose we want to align content so that we have space around the children . Assign mainAxisAlignment as spaceAround as shown below:



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

//function to trigger build
void main() {
  runApp(const MyApp());
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Mobile Programming',
      theme: ThemeData(
        primarySwatch: Colors.green,
      ), // ThemeData
      home: const MyHomePage(),
      debugShowCheckedModeBanner: false,
    ); // MaterialApp
  }
}

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

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

class _MyHomePageState extends State<MyHomePage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Wolkite University"),
      ), // AppBar
      // App body consists of single Column
      // Column consists of three children widgets
      body: Column(
        mainAxisAlignment: MainAxisAlignment.spaceAround,
        children: <Widget>[
          Container(
            decoration: BoxDecoration(
                borderRadius: BorderRadius.circular(10), color: Colors.green),
            child: const Padding(
              padding: EdgeInsets.all(8.0),
              child: Text(
                "Welcome",
                style: TextStyle(color: Colors.white, fontSize: 25),
              ),
            ),
          ),
          Container(
            decoration: BoxDecoration(
                borderRadius: BorderRadius.circular(10), color: Colors.green),
            child: const Padding(
              padding: EdgeInsets.all(8.0),
              child: Text(
                "To",
                style: TextStyle(color: Colors.white, fontSize: 25),
              ),
            ),
          ),
          Container(
            decoration: BoxDecoration(
                borderRadius: BorderRadius.circular(10), color: Colors.green),
            child: const Padding(
              padding: EdgeInsets.all(8.0),
              child: Text(
                "Wolkite University",
                style: TextStyle(color: Colors.white, fontSize: 25),
              ),
            ),
          )
        ],
      ), // Column
    );
  }
}
```



3.6. Stack Widget

Stack widget is a built-in widget in flutter SDK which allows us to make a layer of widgets by putting them on top of each other. Many of the times a simple row and column layout is not enough, we need a way to overlay one widget on top of the other, for example, we might want to show some text over an image, so to tackle such a situation we have Stack widget. The Stack widget has two types of child one is positioned which are wrapped in the *Positioned* widget and the other one is non–positioned which is not wrapped in the *Positioned* widget. For all the non-positioned widgets the alignment property is set to the top-left corner. The positioned child widgets are positioned through the *top*, *right*, left, and *bottom* properties. The child widgets are of Stack are printed in order the top-most widget becomes the bottom-most on the screen and vice-versa. We can use the *key* property of the Stack widget to change that order or to assign a different order.

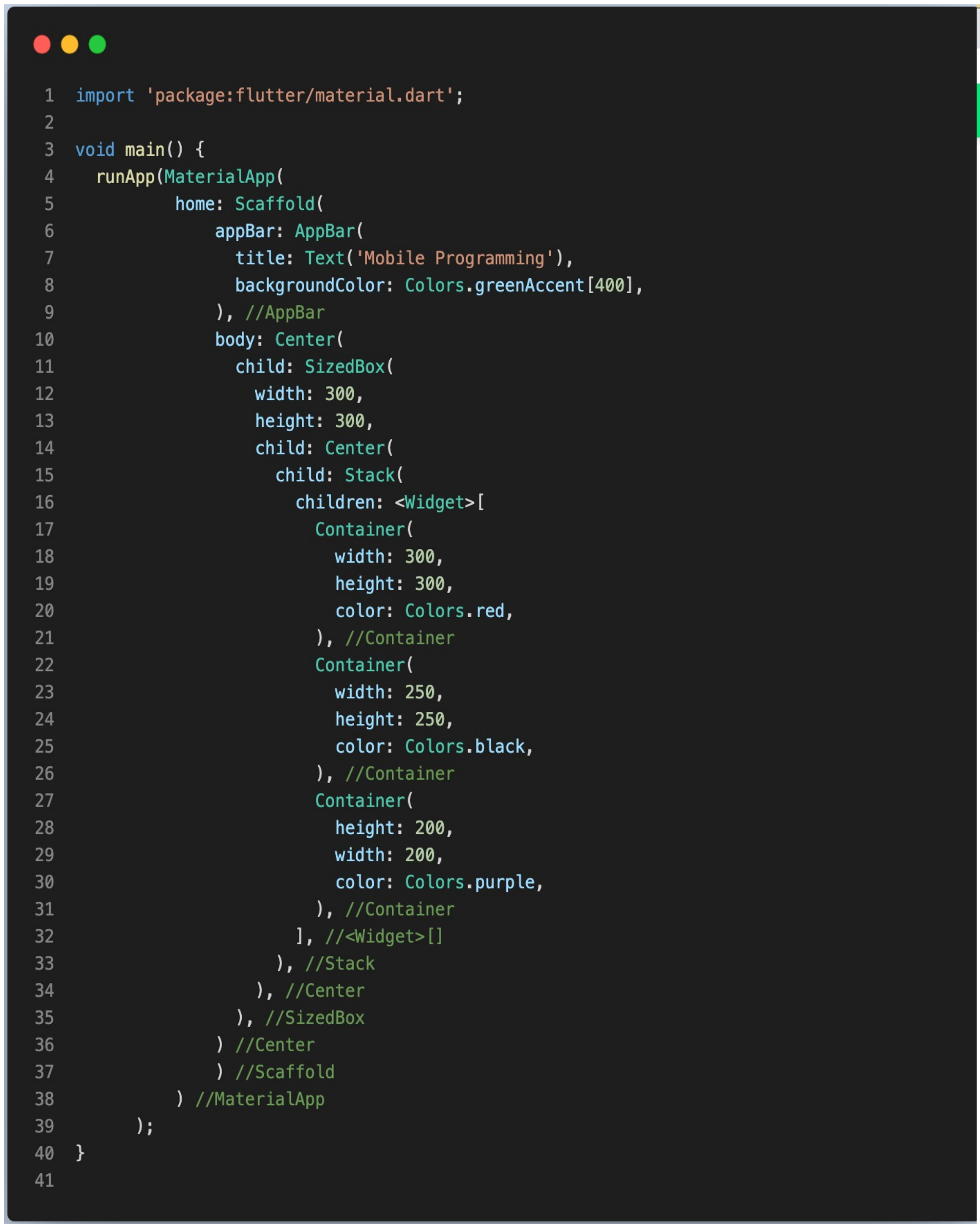
Constructor of Stack Class:

```
Stack(
{Key key,
AlignmentGeometry alignment: AlignmentDirectional.topStart,
TextDirection textDirection,
StackFit fit: StackFit.loose,
Overflow overflow: Overflow.clip,
Clip clipBehavior: Clip.hardEdge,
List<Widget> children: const <Widget>[]}
)
```

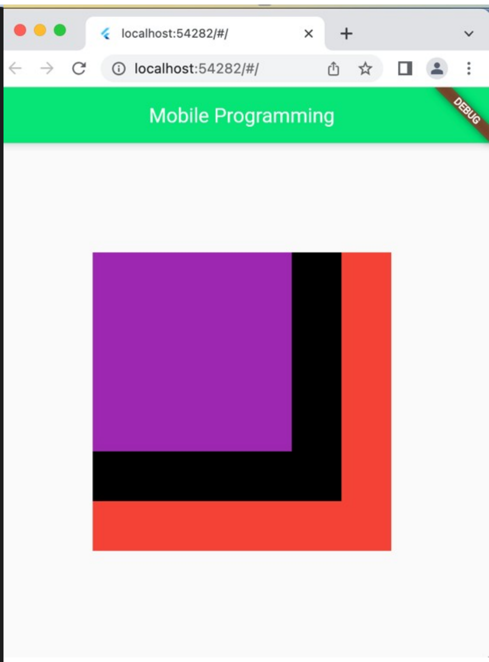
Properties of Stack Widget:

- **alignment:** This property takes a parameter of Alignment Geometry, and controls how a child widget which is non-positioned or partially-positioned will be aligned in the *Stack*.
- **clipBehaviour:** This property decided whether the content will be clipped or not.
- **fit:** This property decided how the non-positioned children in the *Stack* will fill the space available to it.
- **overflow:** This property controls whether the overflow part of the content will be visible or not,
- **textDirection:** With this property, we can choose the text direction from right to left. or left to right.

Example 1:



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

void main() {
  runApp(MaterialApp(
          home: Scaffold(
              appBar: AppBar(
                title: Text('Mobile Programming'),
                backgroundColor: Colors.greenAccent[400],
              ), //AppBar
              body: Center(
                child: SizedBox(
                  width: 300,
                  height: 300,
                  child: Center(
                    child: Stack(
                      children: <Widget>[
                        Container(
                          width: 300,
                          height: 300,
                          color: Colors.red,
                        ), //Container
                        Container(
                          width: 250,
                          height: 250,
                          color: Colors.black,
                        ), //Container
                        Container(
                          height: 200,
                          width: 200,
                          color: Colors.purple,
                        ), //Container
                      ], //<Widget>[]
                    ), //Stack
                  ), //Center
                ), //SizedBox
              ) //Center
              ) //Scaffold
          ) //MaterialApp
      );
}
```



Taking a look at the code of this flutter app, we can see that the parent widget for this app is Scaffold. On the top of the widget tree, we have *AppBar* widget with *title Text* widget reading '*Mobile Programming*' and the background color of the app bar is *greenAccent*[400]. In the body of the app, the parent widget is *Center* followed by the *SizedBox* of *height* 300 and *width* 300. *SizedBox* is also having a child *Center* which in turn is holding the *Stack* widget. In the *Stack* widget, we have a list of children widgets holding three *Container* widgets. The first *Container* widget is having a height and width of 300, the same as the *SizedBox,* with a red color. The Second *Container* is having a width and height of 250 with black color. The third *Container* is having a width and height of 200 with a purple color. Now, looking at the app we can see that all three containers that are children to *Stack* are stacked on top of each other with the red containers at the bottom of the purple at the top, and the black in the middle. All these three containers are non-positioned widget in *Stack* so the alignment property for them is set to Alignment.topRight, therefore we can see all of them aligned to the top right corner.

3.7. Buttons(ElevatedButton, TextButton and IconButtton) Widget

TextButton Widget is a built-in widget in Flutter which derives its design from Google's Material Design Library. It is a simple Button without any border that listens for onPressed and onLongPress gestures. It has a style property that accepts ButtonStyle as value, using this style property developers can customize the TextButton however they want.

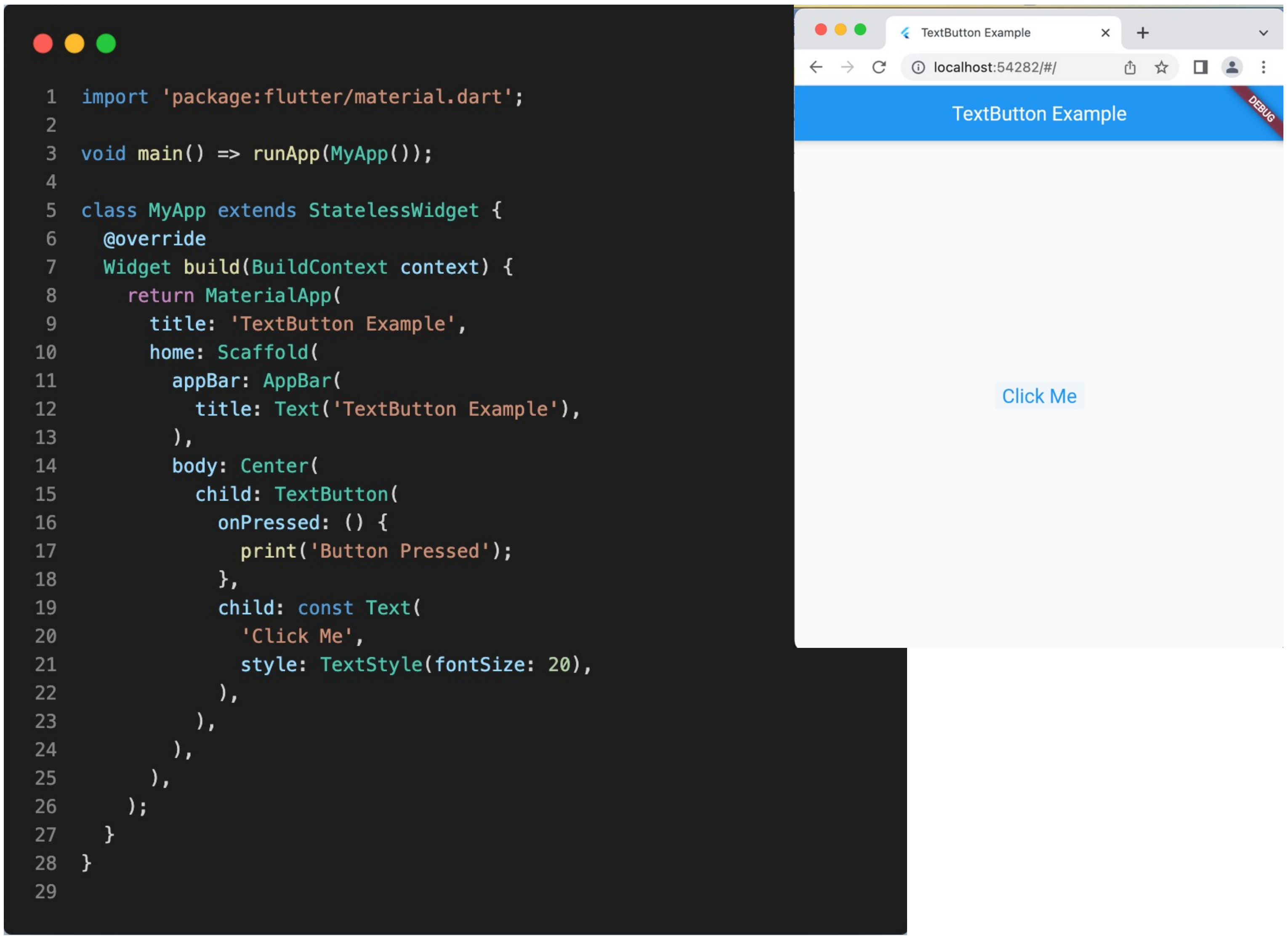
The constructor of TextButton Class:

```
const TextButton({
  Key? key,
  required void Function()? onPressed,
  void Function()? onLongPress,
  void Function(bool)? onHover,
  void Function(bool)? onFocusChange,
  ButtonStyle? style,
  FocusNode? focusNode,
  bool autofocus = false,
  Clip clipBehavior = Clip.none,
  required Widget child,
})
```

Properties of TextButton Widget:

- **onPressed:** *(required)* This property takes a Function that returns void. That function will be invoked whenever the TextButton is pressed.
- **onLongPress:** This property also takes a Function that returns void. That function will be invoked whenever the TextButton is long pressed.
- **onHover:** *onHover* property takes a void Function which takes a boolean value as its only parameter. The passed parameter is *true* if a pointer entered the button response area and is *false* when a pointer leaves the button response area.
- **onFocusChange:** *onFocusChange* property also takes a void Function which takes a boolean value as its only parameter. The passed parameter is *true* if the button gains focus and is *false* if the button loses focus.
- **style:** This optional property accepts ButtonStyle as an object. It is used for customizing the look of the TextButton.
- **focusNode:** An optional focus node to use as the focus node for TextButton.
- **autofocus**: Pass *true* if you want this widget as the initial focus when no other node in its scope is currently focused. The default value is *false*.
- **clipBehaviour:** Accept value of type Clip enum. The default value is *Clip.none*.
- **child**: *(required)* The child widget of TextButton Widget.

Example1: TextButton widget in Flutter:



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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'TextButton Example',
      home: Scaffold(
        appBar: AppBar(
          title: Text('TextButton Example'),
        ),
        body: Center(
          child: TextButton(
            onPressed: () {
              print('Button Pressed');
            },
            child: const Text(
              'Click Me',
              style: TextStyle(fontSize: 20),
            ),
          ),
        ),
      ),
    );
  }
}
```

ElevatedButton Widget: is a flutter component included inside the material package i.e. “package:flutter/material.dart“. The main characteristic these buttons hold is the slight elevation in their surface towards the screen on getting tapped by the user. In simple language, elevated buttons are un-deprecated raised buttons with no explicitly defined button styling. Elevated Buttons cannot be styled i.e. you cannot modify the color of the button, font size, text style, etc explicitly like raised buttons. This class was launched in version 1.22 of flutter. You can pass text or icons as a child to them. To handle the styling of the Elevated Button, a ButtonStyle class is used which allows the styling of a button according to requirements.

Constructor for Elevated Button

```
ElevatedButton.icon({
  Key? key,
  required VoidCallback? onPressed,
  VoidCallback? onLongPress,
  ValueChanged<bool>? onHover,
  ValueChanged<bool>? onFocusChange,
  ButtonStyle? style,
  FocusNode? focusNode,
  bool? autofocus,
  Clip? clipBehavior,
  required Widget icon,
  required Widget label
})
```

Parameters

Elevated Button offers two important parameters:

1. child: this represents the button’s label.

```
ElevatedButton(

        child: const Text('Raised Button'),

      ),
```

2. onPressed: this represents the action to be executed when the button is tapped

```
onPressed: () => Navigator.of(context)

            .push(MaterialPageRoute(builder: (context) => const NewScreen())),
```

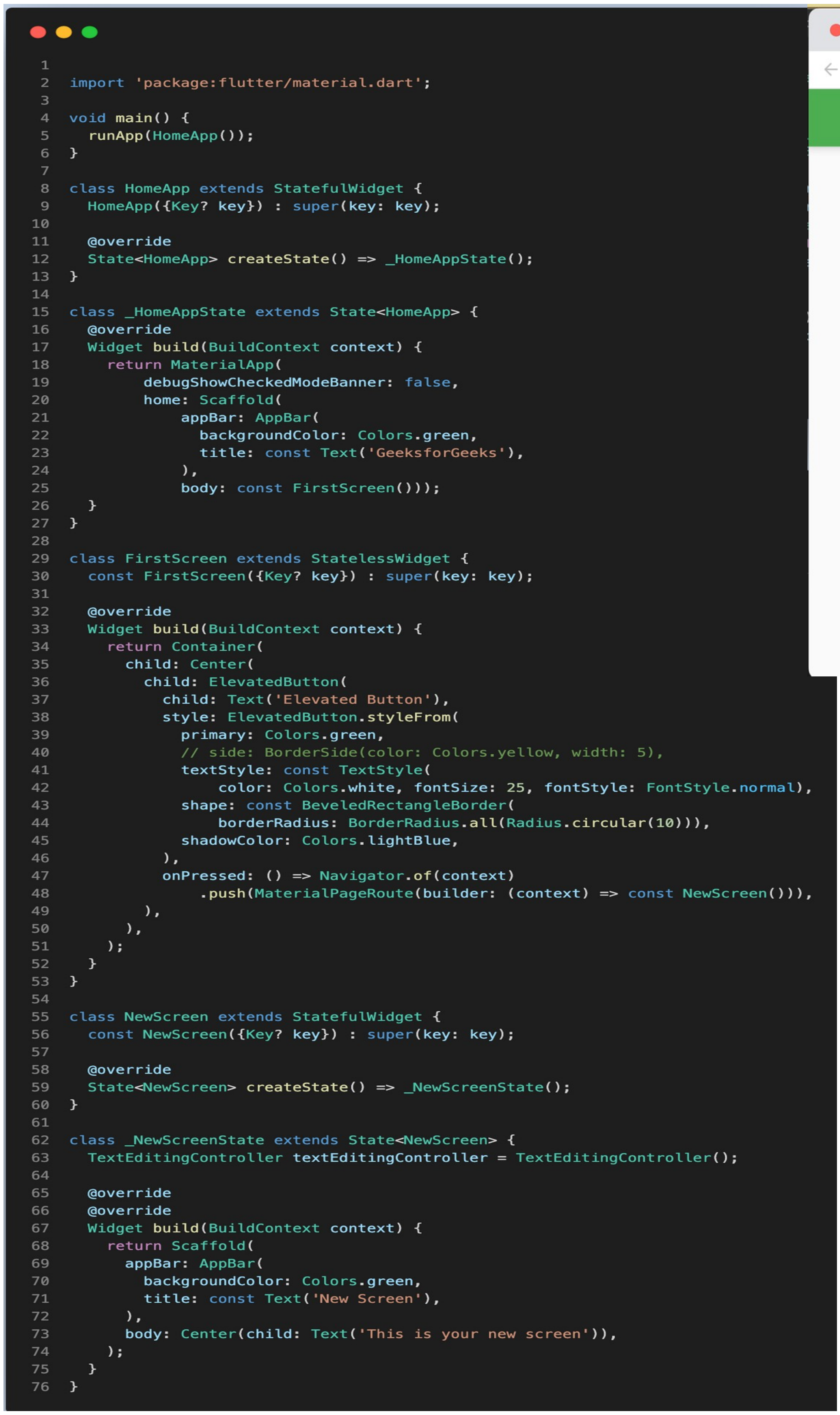
Here, when the button is tapped a navigation action to NewScreen is executed.

Properties of Elevated Button

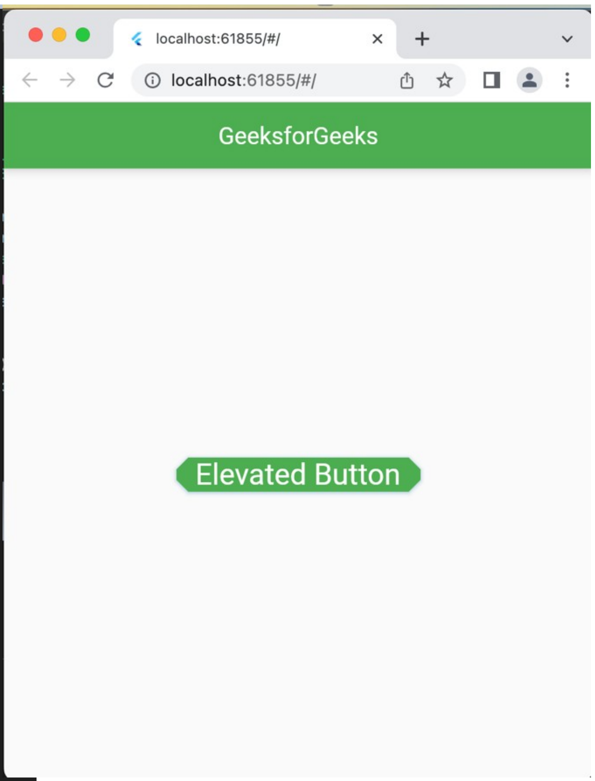
- **autofocus**: gives a boolean value corresponding to the initial focus of the button.
- **clipBehaviour**: decides whether the content is clipped or not.
- **focusNode**: represents the focus node of the widget
- **ButtonStyle** style: decides the styling of the button
- **onLongPress**: the action to be executed when the button is long pressed by the user
- **enabled**: gives a boolean value for the activity of the button
- **hashcode**: decides the hashcode of the button
- **Key**: Controls how one widget replaces another widget in the tree.
- **onFocusChanged**: a method to be executed on changing the focus of the button
- **onHover**: actin to be executed when the user hovers the button

Methods Provided in an Elevated Button

- **createElement()**: create a StatefulElement to manage button’s location in the widget tree.
- **createState()**: Creates the mutable state for this widget at a given location in the tree.
- **build(BuildContext context)**: Describes the part of the user interface
- **createElement()**: creates a StatelessElement to manage this widget’s location in the tree.
- **debugDescribeChildren()**: Returns a list of DiagnosticsNode objects describing this node’s children
- **debugFillProperties()**: Add additional properties associated with the node
- **noSuchMethod()**: Invoked when a non-existent method or property is accessed
- **toDiagnosticsNode()**: Returns a debug representation of the object that is used by debugging tools
- **toString()**: A string representation of this object
- **toStringDeep()**: Returns a string representation of this node and its descendants.
- **toStringShallow()**: Returns a one-line detailed description of the object
- **toStringShort()**: A short, textual description of this widget.



```dart

import 'package:flutter/material.dart';

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

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

  @override
  State<HomeApp> createState() => _HomeAppState();
}

class _HomeAppState extends State<HomeApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        debugShowCheckedModeBanner: false,
        home: Scaffold(
            appBar: AppBar(
              backgroundColor: Colors.green,
              title: const Text('GeeksforGeeks'),
            ),
            body: const FirstScreen()));
  }
}

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

  @override
  Widget build(BuildContext context) {
    return Container(
      child: Center(
        child: ElevatedButton(
          child: Text('Elevated Button'),
          style: ElevatedButton.styleFrom(
            primary: Colors.green,
            // side: BorderSide(color: Colors.yellow, width: 5),
            textStyle: const TextStyle(
                color: Colors.white, fontSize: 25, fontStyle: FontStyle.normal),
            shape: const BeveledRectangleBorder(
                borderRadius: BorderRadius.all(Radius.circular(10))),
            shadowColor: Colors.lightBlue,
          ),
          onPressed: () => Navigator.of(context)
              .push(MaterialPageRoute(builder: (context) => const NewScreen())),
        ),
      ),
    );
  }
}

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

  @override
  State<NewScreen> createState() => _NewScreenState();
}

class _NewScreenState extends State<NewScreen> {
  TextEditingController textEditingController = TextEditingController();

  @override
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.green,
        title: const Text('New Screen'),
      ),
      body: Center(child: Text('This is your new screen')),
    );
  }
}
```



IconButton Widget It is one of the most widely used buttons in the flutter library. First, as the name suggests, the icon button is the button having an icon, and ontap it does something. A sample video is given below to get an idea about what we are going to do in this article.

Constructor

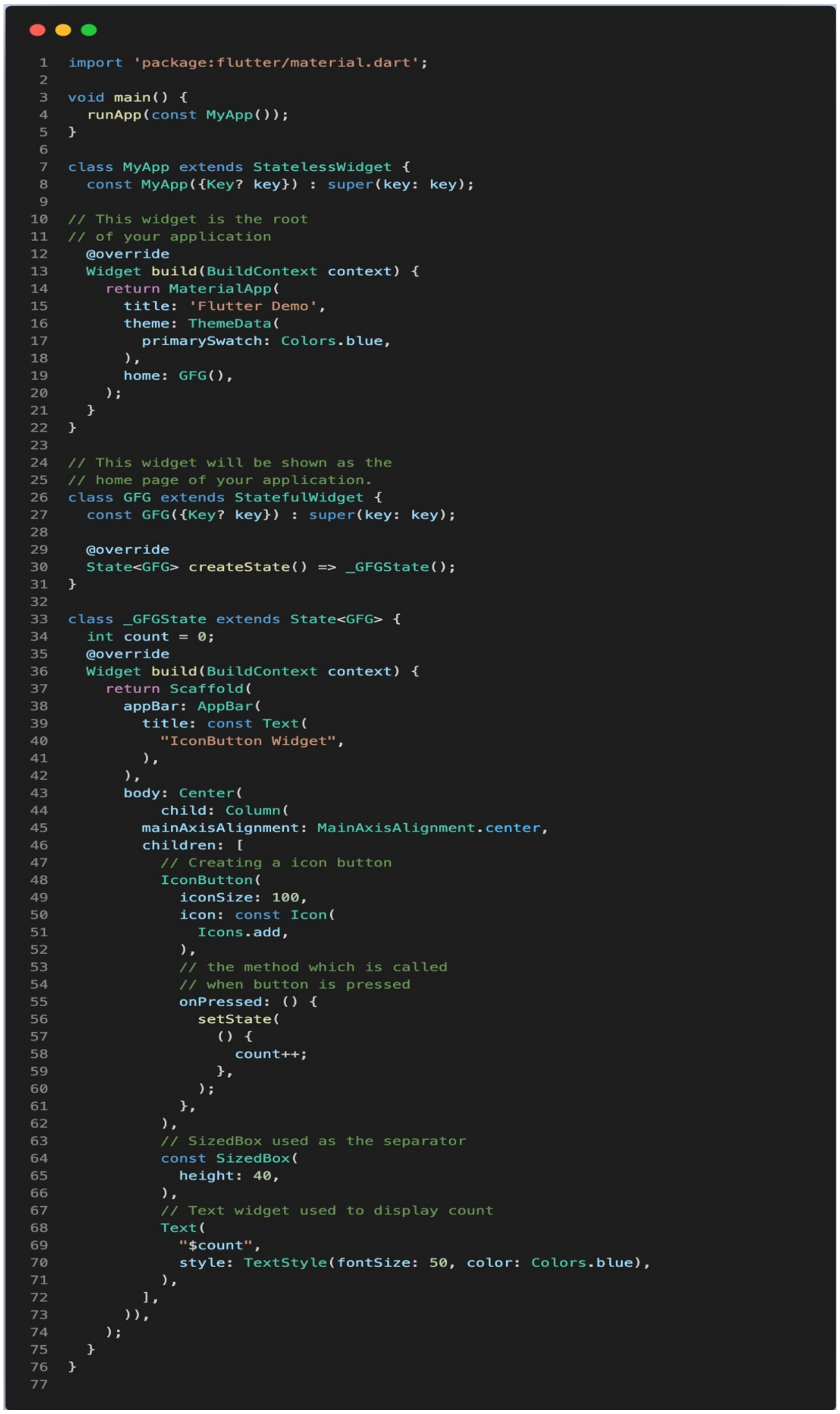
```
IconButton
(
   {
    Key? key,
    double? iconSize,
    VisualDensity? visualDensity,
    EdgeInsetsGeometry padding = const EdgeInsets.all(8.0),
    AlignmentGeometry alignment = Alignment.center,
    double? splashRadius,
    Color? color,
    Color? focusColor,
    Color? hoverColor,
    Color? highlightColor,
    Color? splashColor,
    Color? disabledColor,
    required VoidCallback? onPressed,
    MouseCursor? mouseCursor,
    FocusNode? focusNode,
    bool autofocus = false,
    String? tooltip,
    bool enableFeedback = true,
    BoxConstraints? constraints,
    required Widget icon
  }
)
```

Properties

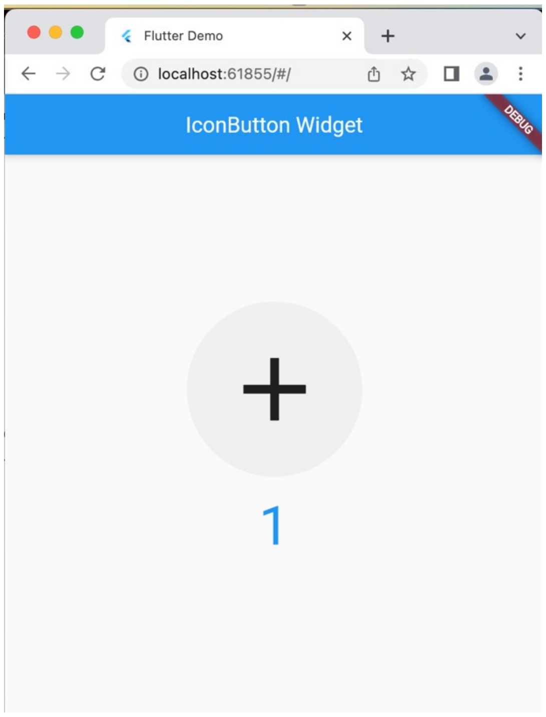
- **alignment:** Defines how to place the button on the widget tree.
- **autofocus:** True if other widgets are not in focus, instead this widget is.
- **color:** Defines the color of the Icon inside the button.
- **constraints:** Optional size constraints for the button.
- **disabledColor:** The color to show when the widget is disabled.
- **enableFeedback:** Whether detected gestures should provide acoustic and/or haptic feedback.
- **focusColor:** The color of the button when it is in focus.
- **hashCode:** The hash code for this object.
- **highlightColor:** The secondary color (optional) of the button when it is pressed.
- **hoverColor:** The Icon color while hovering over the Icon.

- **icon:** The icon to display inside the button.
- **iconSize:** Icon's size Inside the button.
- **key:** Controls how one widget replaces another widget in the tree.
- **mouseCursor:** The type of cursor to show when it hovers over the widget.
- **onPressed:** The action to perform when the button is pressed.
- **padding:** The padding to provide to the Icon.
- **runtimeType:** A representation of the runtime type of the object.
- **splashColor:** The color of the ripple effect produced when the user presses the button.
- **splashRadius:** The splash radius.
- **tooltip:** Text to represent the action when the button is pressed.
- **visualDensity:** Defines how compact the icon button's layout will be.

Example Project: The ***main.dart*** file:



```dart
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: GFG(),
    );
  }
}

// This widget will be shown as the
// home page of your application.
class GFG extends StatefulWidget {
  const GFG({Key? key}) : super(key: key);

  @override
  State<GFG> createState() => _GFGState();
}

class _GFGState extends State<GFG> {
  int count = 0;
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(
          "IconButton Widget",
        ),
      ),
      body: Center(
          child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          // Creating a icon button
          IconButton(
            iconSize: 100,
            icon: const Icon(
              Icons.add,
            ),
            // the method which is called
            // when button is pressed
            onPressed: () {
              setState(
                () {
                  count++;
                },
              );
            },
          ),
          // SizedBox used as the separator
          const SizedBox(
            height: 40,
          ),
          // Text widget used to display count
          Text(
            "$count",
            style: TextStyle(fontSize: 50, color: Colors.blue),
          ),
        ],
      )),
    );
  }
}
```



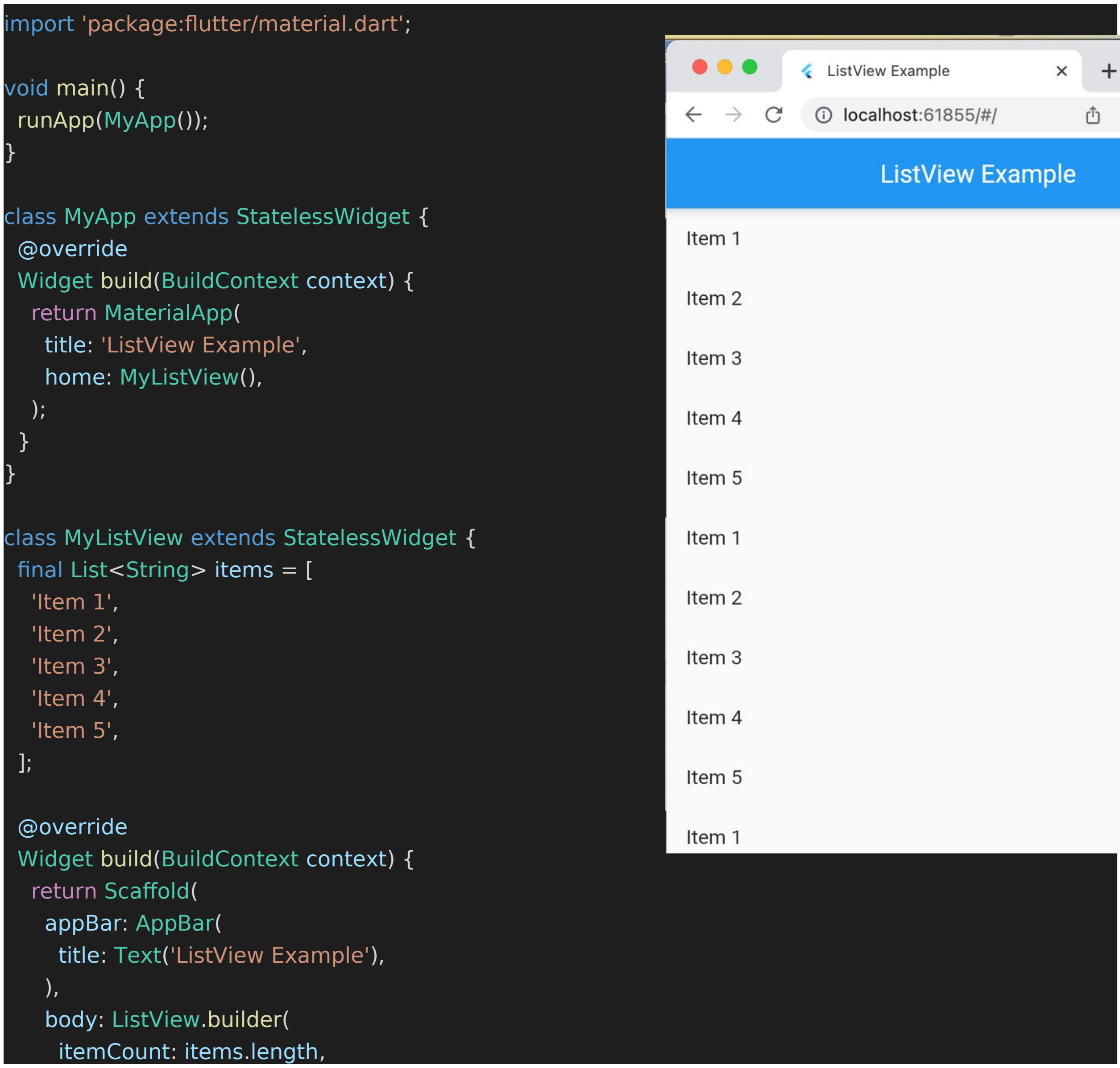
3.8. ListView and ListTile widget

ListView Class is a scrollable list of widgets arranged linearly. It displays its children one after another in the scroll direction i.e, vertical or horizontal.

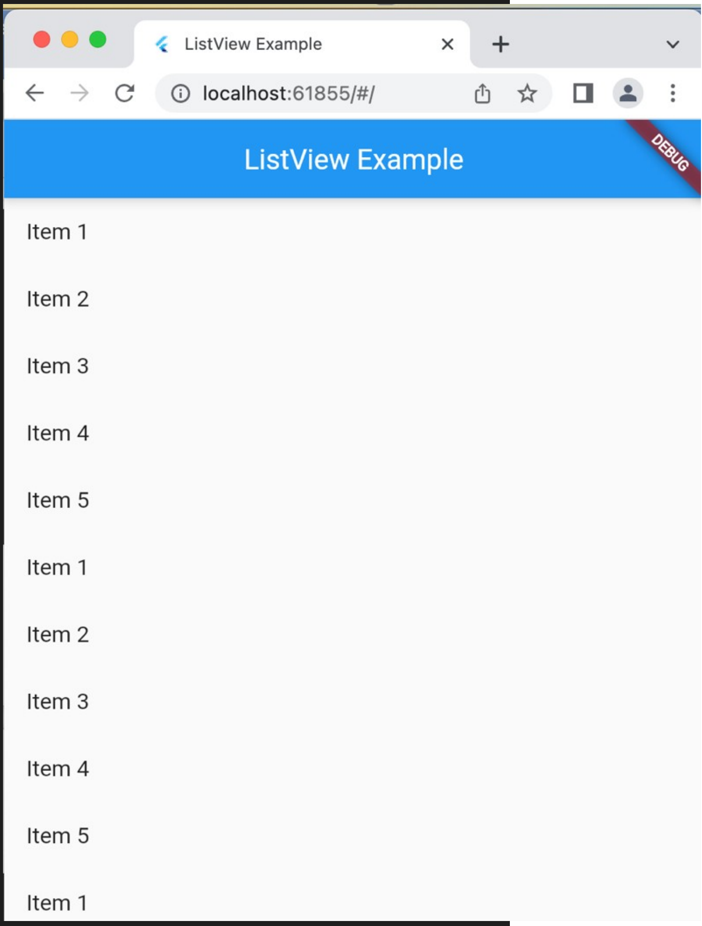
There are different types of ListViews :

- **ListView**
- **ListView.builder**
- **ListView.separated**
- **ListView.custom**

ListView()
This is the default constructor of the ListView class. A ListView simply takes a list of widgets and makes it scrollable. Usually, this is used with a few children as the List will also construct invisible elements in the list, so numerous widgets may render this inefficiently.



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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'ListView Example',
      home: MyListView(),
    );
  }
}

class MyListView extends StatelessWidget {
  final List<String> items = [
    'Item 1',
    'Item 2',
    'Item 3',
    'Item 4',
    'Item 5',
  ];

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('ListView Example'),
      ),
      body: ListView.builder(
        itemCount: items.length,
```



```
ListView Example

Item 1
Item 2
Item 3
Item 4
Item 5
Item 1
Item 2
Item 3
Item 4
Item 5
Item 1
```

```
    itemBuilder: (BuildContext context, int index) {
      return ListTile(
        title: Text(items[index]),
      );
    },
   ),
  );
 }
}
```

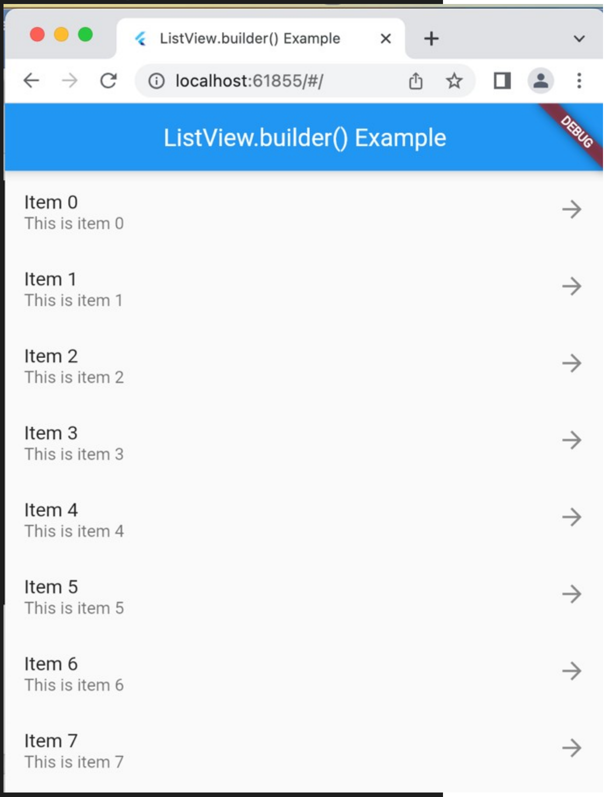
ListView.builder()

The **builder()** constructor constructs a repeating list of widgets. The constructor takes two main parameters:

- An ***itemCount*** for the number of repetitions for the widget to be constructed (not compulsory).
- An ***itemBuilder*** for constructing the widget which will be generated '**itemCount**' times (compulsory).

If the *itemCount* is not specified, **infinite** widgets will be constructed by default.

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

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

class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   title: 'ListView.builder() Example',
   home: Scaffold(
    appBar: AppBar(
     title: Text('ListView.builder() Example'),
    ),
    body: ListView.builder(
     itemCount: 10,
     itemBuilder: (context, index) {
      return ListTile(
       title: Text('Item $index'),
       subtitle: Text('This is item $index'),
       trailing: Icon(Icons.arrow_forward),
      );
     },
    ),
   ),
  );
 }
}
```



ListView.builder() Example

Item 0
This is item 0

Item 1
This is item 1

Item 2
This is item 2

Item 3
This is item 3

Item 4
This is item 4

Item 5
This is item 5

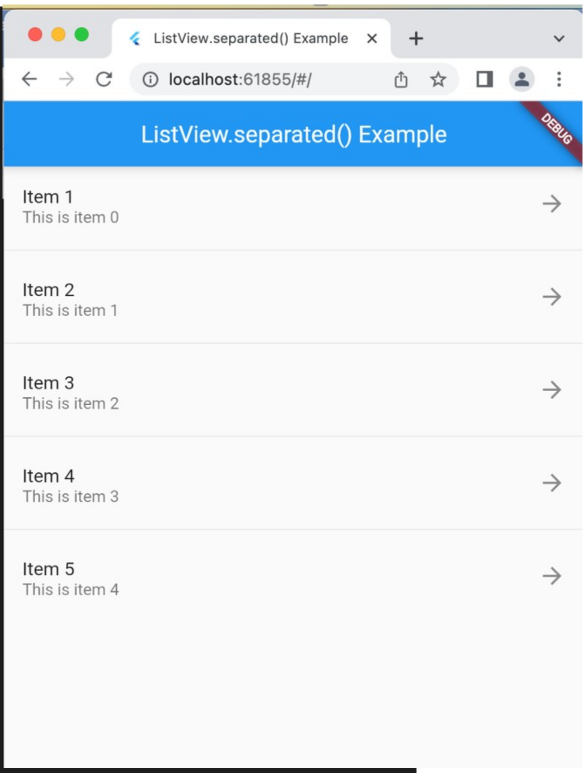
Item 6
This is item 6

Item 7
This is item 7

ListView.separated ()

The **ListView.separated()** constructor is used to generate a list of widgets, but in addition, a **separator widget** can also be generated to separate the widgets. In short, these are two intertwined list of widgets: the main list and the separator list. Unlike the builder() constructor, the ***itemCount*** parameter is compulsory here.

```dart
// ListView.separated ()
import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {
  final List<String> items = [
    'Item 1',
    'Item 2',
    'Item 3',
    'Item 4',
    'Item 5',
  ];

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'ListView.separated() Example',
      home: Scaffold(
        appBar: AppBar(
          title: Text('ListView.separated() Example'),
        ),
        body: ListView.separated(
          itemCount: items.length,
          separatorBuilder: (BuildContext context, int index) => Divider(),
          itemBuilder: (BuildContext context, int index) {
            return ListTile(
              title: Text(items[index]),
              subtitle: Text('This is item $index'),
              trailing: Icon(Icons.arrow_forward),
            );
          },
        ),
      ),
    );
  }
}
```



ListView.separated() Example

Item 1
This is item 0

Item 2
This is item 1

Item 3
This is item 2

Item 4
This is item 3

Item 5
This is item 4

ListTile Widget

ListTile widget is used to populate a ListView in Flutter. It contains *title* as well as *leading* or *trailing* icons. Let's understand this with the help of an example.

Constructor of ListTile class

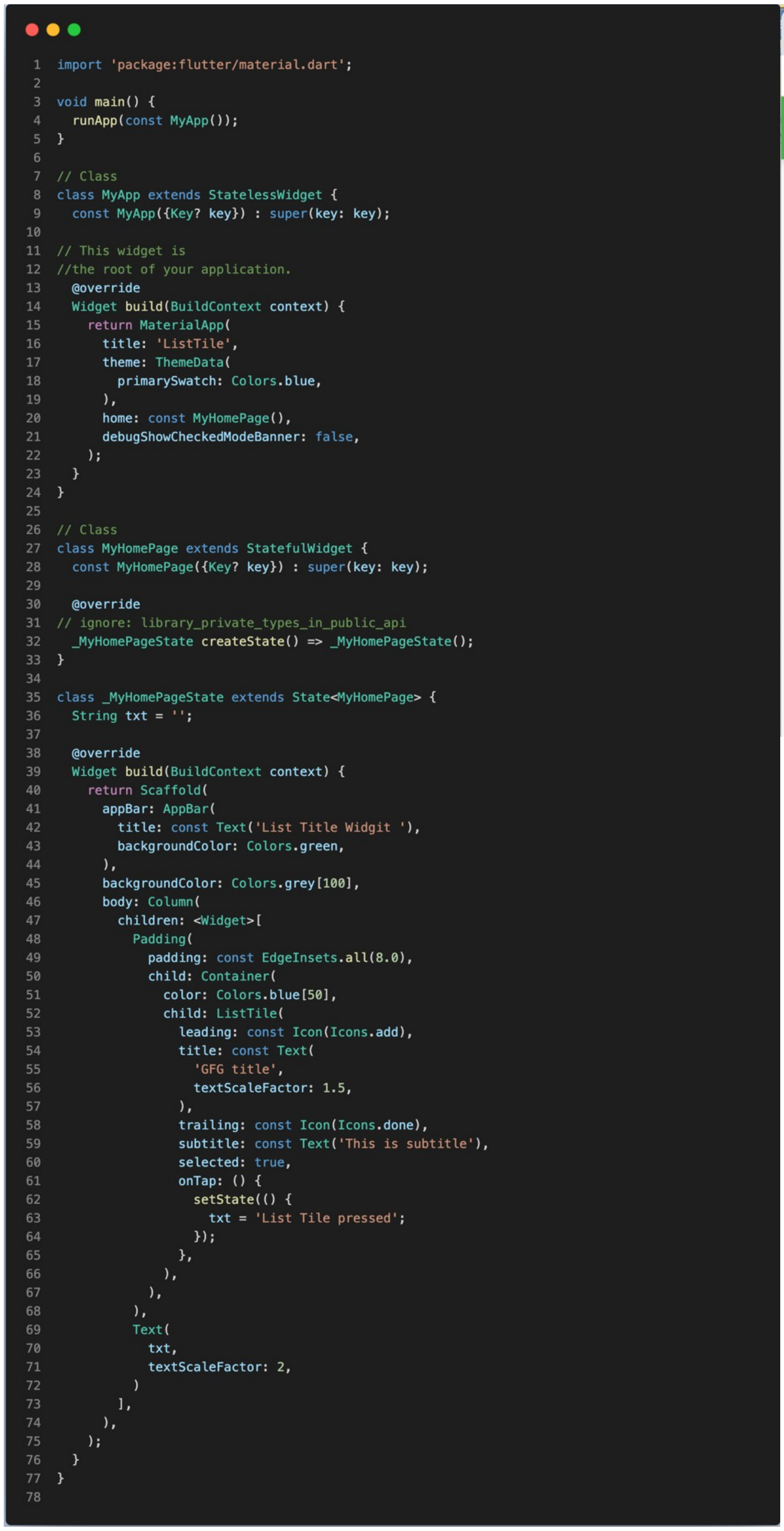
```
ListTile({Key key,
        Widget leading,
        Widget title,
        Widget subtitle,
        Widget trailing,

bool isThreeLine: false,
bool dense,
VisualDensity visualDensity,
ShapeBorder shape,
EdgeInsetsGeometry contentPadding,

bool enabled: true,
GestureTapCallback onTap,
GestureLongPressCallback onLongPress,
MouseCursor mouseCursor,

bool selected: false,
Color focusColor,
Color hoverColor,
FocusNode focusNode,
bool autofocus: false})
```

Properties

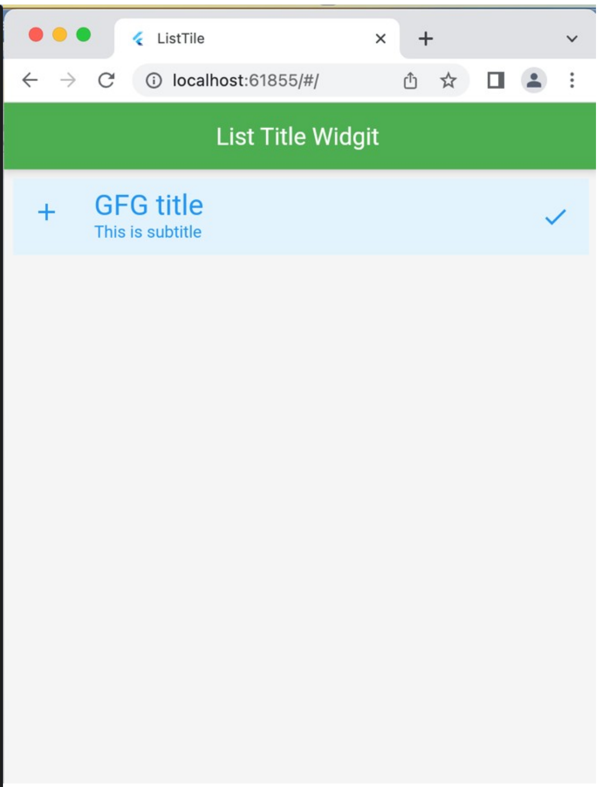
- **autofocus:** This property takes in a *boolean* as the object to decide whether this widget will be selected on the initial focus or not.

- **contentPadding:** By taking *EdgeInsetsGeometry* as the object this property controls the padding.
- **dense:** This property decides whether the ListTile will be a part of a vertically dense list or not by taking in a *boolean* as the object.
- **enable:** This property controls whether the ListTile will be interactive or not by taking in a *boolean* as the object.
- **focusColor:** This property holds *Color* class as the object to control the color of the widget at the time of input focus.
- **focusNode:** This property provides an additional node.
- **hoverColor:** This property takes in *Color* class as the object to decide the color of the tile at the time of hover.
- **isThreeLine:** whether this list item should display three lines of text or not.
- **leading:** leading widget of the ListTile.
- **mouseCursor:** The *mouseCursor* property holds *MouseCursor* class as the object to decide the cursor for the mouse pointer at the time of pointer event.
- **onLongPress:** This holds *GestureLongPressCallback typedef* as the object
- **onTap:** function to be called when the list tile is pressed.
- **selected:** This property holds a *boolean* as the object. If set to true then the text and icon will be painted with the same color.
- **selectedTileColor:** This property controls the background color of the *ListTile* when it is selected.
- **shape:** the shape of the title's InkWell.
- **subtitle:** additional content displayed below the title.
- **titleColor:** This property defines the background color of the *ListTile* when it is not selected, by taking in *Color* class as the object.
- **tile:** title to be given to *ListTile* widget.
- **trailing:** trailing widget of the *ListTile*.
- **visualDensity.** This property takes in *VisualDensity class* as the object. It defines the compactness in the layout of the *ListTile.*

Example:



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

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

// Class
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: 'ListTile',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: const MyHomePage(),
      debugShowCheckedModeBanner: false,
    );
  }
}

// Class
class MyHomePage extends StatefulWidget {
  const MyHomePage({Key? key}) : super(key: key);

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

class _MyHomePageState extends State<MyHomePage> {
  String txt = '';

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('List Title Widgit '),
        backgroundColor: Colors.green,
      ),
      backgroundColor: Colors.grey[100],
      body: Column(
        children: <Widget>[
          Padding(
            padding: const EdgeInsets.all(8.0),
            child: Container(
              color: Colors.blue[50],
              child: ListTile(
                leading: const Icon(Icons.add),
                title: const Text(
                  'GFG title',
                  textScaleFactor: 1.5,
                ),
                trailing: const Icon(Icons.done),
                subtitle: const Text('This is subtitle'),
                selected: true,
                onTap: () {
                  setState(() {
                    txt = 'List Tile pressed';
                  });
                },
              ),
            ),
          ),
          Text(
            txt,
            textScaleFactor: 2,
          )
        ],
      ),
    );
  }
}
```


ListTile
localhost:61855/#/
List Title Widgit
GFG title
This is subtitle