

Creating your first Flutter project

On the startup screen, there should be a new option called **Start a new Flutter project**. Select that button and select **Flutter application** as your template. After clicking **Next**, you should be taken to a screen where you can configure your Flutter application.

Under Project Name, put **ProjectName** and make sure that the Flutter SDK path matches the path where you installed the Flutter SDK. Android Studio should automatically detect this but it's good to verify twice. Select a Project Location to save your project and you may optionally set a Description. When you're satisfied with your configuration, press **Next**.

Finally, you will be asked to set the **Package Name**. If you're coming from iOS development, the package name is similar to the bundle identifier. I am setting mine to **com.yeasirarefin**. Making sure that all the AndroidX and Platform channel language checkboxes are selected, click on **Finish** to have Android Studio create your Flutter project

Exploring the Sample Application

When the project opens up, you'll see that you are in a file called **main.dart**. On the lefthand side, you will see a sidebar. This sidebar will contain all of the folders and libraries associated with your Flutter project. Opening the **project** folder, you'll notice that there are many folders underneath it. The **ios** folder indicates the backend Flutter code that helps bridge the Dart code to iOS, the **android** folder does the same but for Android, and the **web** folder is used to port our code over to making a web app. However, we will be working out of the **lib** folder. This is where most of our code will go. If you open this folder, you can see that there is one file in there: **main.dart**. This file is also the file that the IDE is currently displaying to you.

Typically, Flutter convention states that **main.dart** should be the file containing the **main()** method, but this is not a hard and fast rule. The **main()** method is the first method Flutter looks for when running an app. Within this method, you'll see that Flutter is calling another function called **runApp** which takes the class **MyApp** as an input.

Without going into too much detail, you'll see that **MyApp** is a class that conforms to a type of **StatelessWidget**. If you want to see what this sample application does right now, at the top of our IDE, you'll see a dropdown menu which is currently set to **<no device selected>**. Switch this to **Open iOS Simulator** and then when the iOS Simulator boots up, press the green **play** button. This will build and run the app on the simulator.

Build Your First Widget

Getting Started

Open the project we created in the last section. We'll need to remove the template project so change the main.dart file to look like such.

```
void main() {  
    runApp(MyApp());  
}  
  
class MyApp extends StatelessWidget {  
    @override  
    Widget build(BuildContext context) {  
        return Container();  
    }  
}
```

Material App

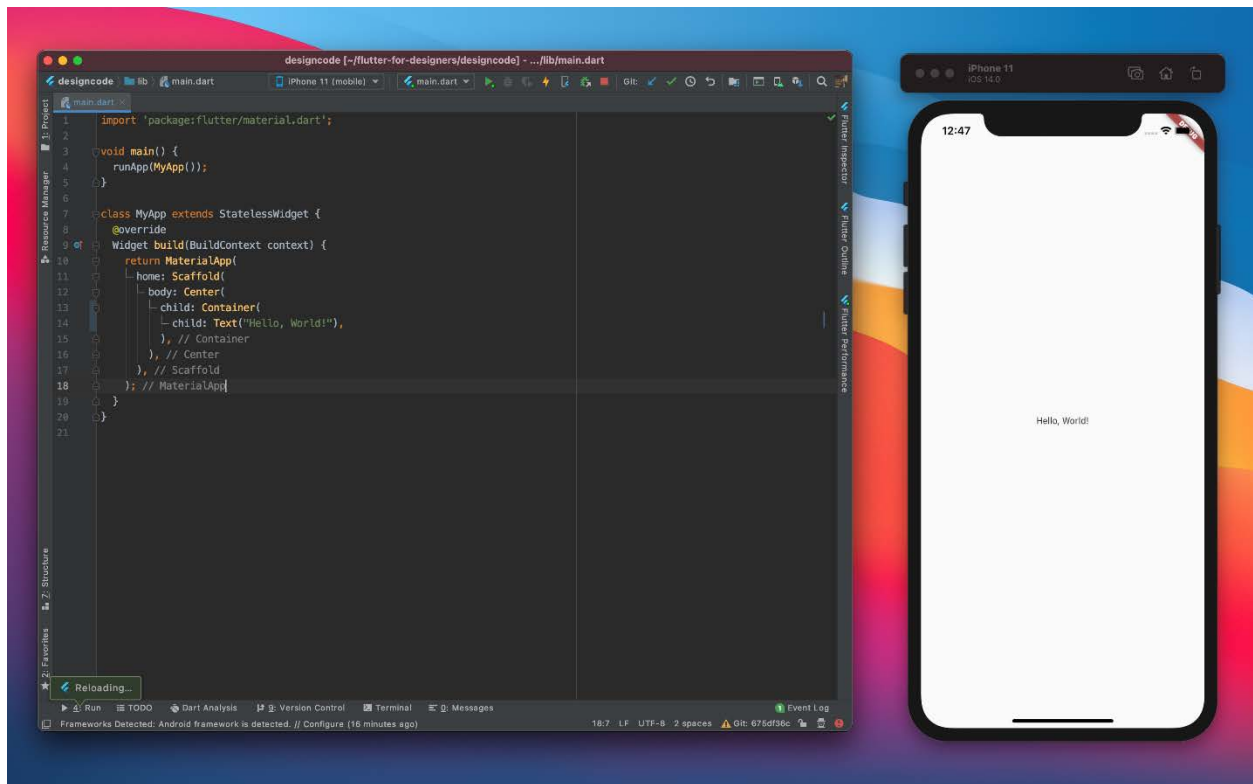
We now have a black screen so to change that, delete the `Container` widget and replace it with the following

```
return MaterialApp(  
    home: Scaffold(  
        // ...
```

```
body: Center(  
  child: Container(  
    child: Text("Hello World")  
  )  
)  
)  
)  
)
```

These are some of the common widgets you will see in Flutter apps.

- **MaterialApp:** The MaterialApp widget provides a number of widgets at the root of your app for helping with navigation, localization, debugging tools, and more.
- **Scaffold:** A Scaffold widget is also another widget that provides many sub-widgets related to the UI of your app like a background color, a navigation bar, etc.
- **Center:** The Center widget is a layout widget that centers its child in the middle of the parent widget.
- **Text:** A Text widget displays a text label. The string value should be provided and the text styles can be customized.



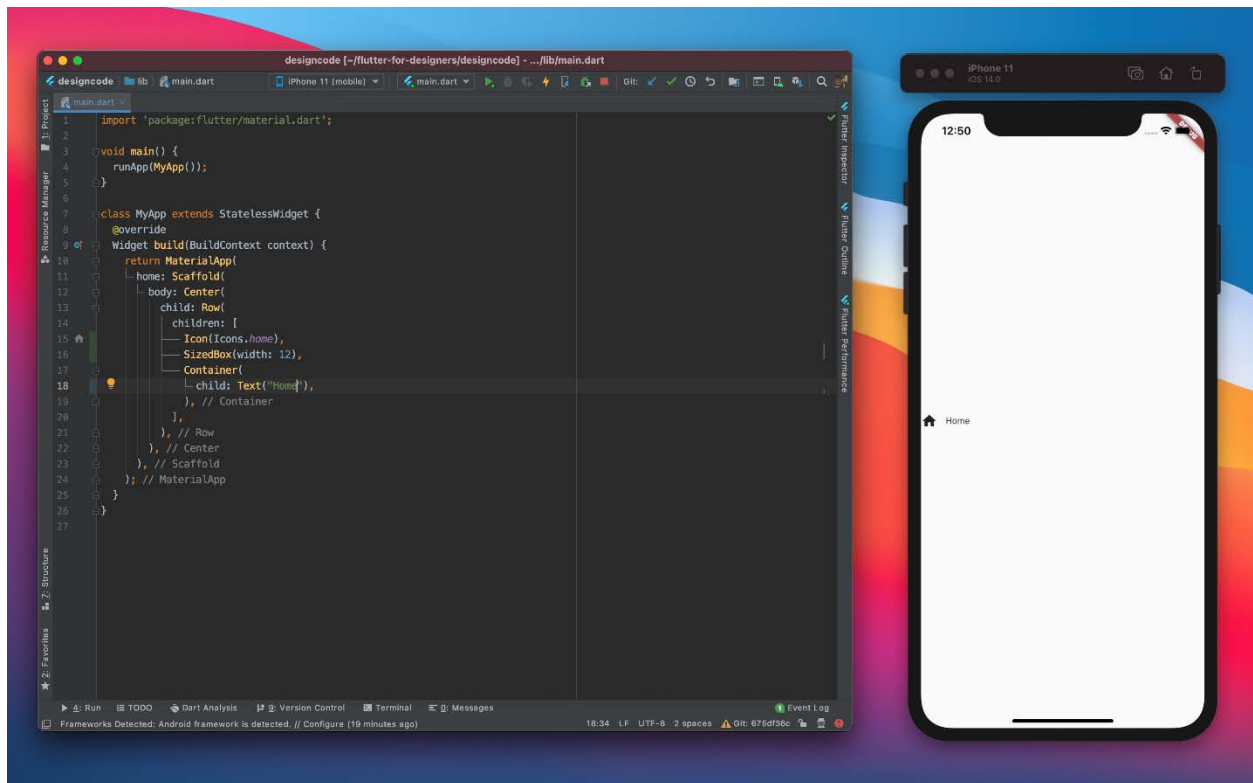
Sidebar Item

Creating the sidebar item is just a matter of a few keyboard taps and mouse clicks.

- First, change the **Hello World** string to **Home**.
- Next, place your cursor on **Container** keyword and press **ctrl+.** and select **Wrap with Row** in the menu that appears
- Before the **Container** widget in the new **children** argument of the **Row** widget, add the **Icon** widget for the icon and the **SizedBox** widget for padding

Row(

```
children: [  
    Icon(Icons.home),  
    SizedBox(width: 12),  
    Container(  
        child: Text("Home"),  
    ),  
],  
)
```



Styling the Sidebar Item

To style the sidebar item, we need to add a **style** property to the **Text** widget, color the **Icon** widget, and finally wrap our **Icon** widget in a **Container** widget and set the properties of this container.

- For the **Text** widget, we'll set it to a font size of 16.0 pixels, a font weight of 800, and a color of #242629.
- We'll set the color of our **Icon** widget to white.
- For the **Container** widget that is wrapped around our **Icon** widget, we'll set the width and height to 42.0 pixels, add a padding of 10.0 pixels, add a border radius of 14.0 pixels, and add a linear gradient with colors of 0xFF00AEFF, 0xFF0076FF

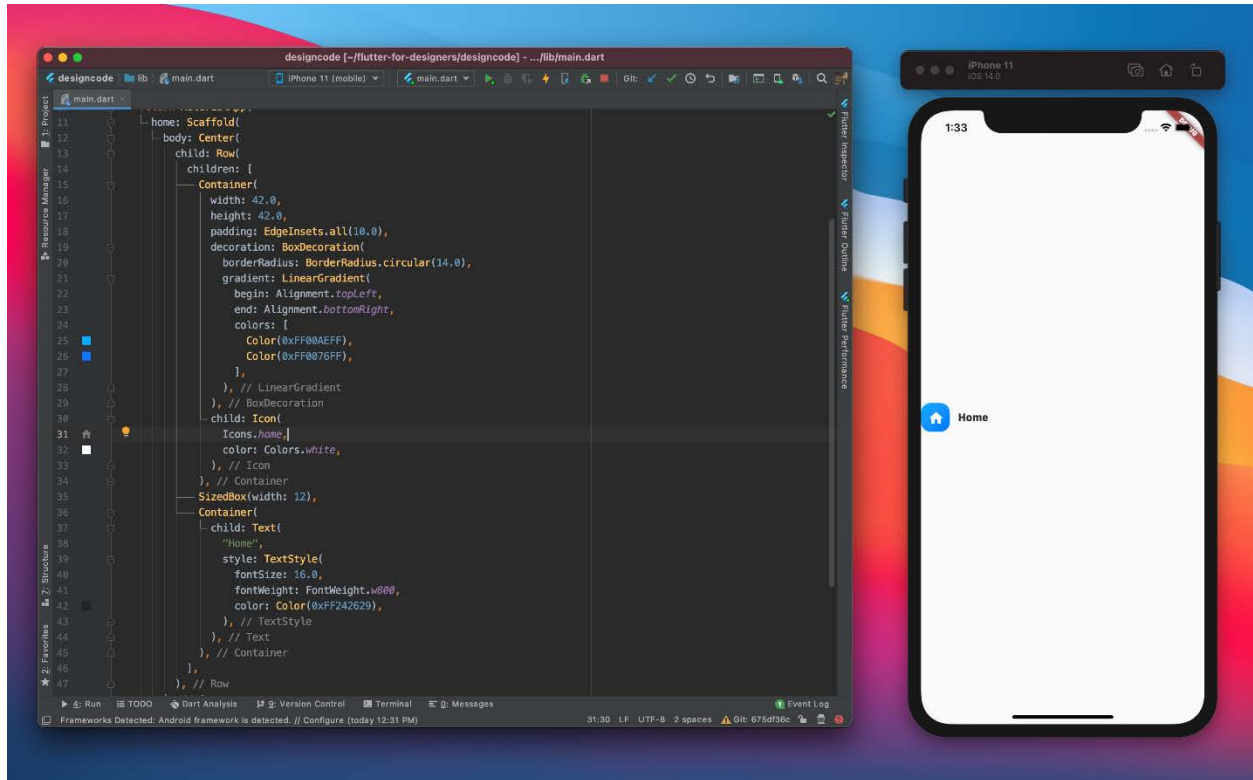
```
Row(  
  children: [  
    Container(  
      width: 42.0,  
      height: 42.0,  
      padding: EdgeInsets.all(10.0),  
      decoration: BoxDecoration(  
        borderRadius: BorderRadius.circular(14.0),  
        gradient: LinearGradient(  
          begin: Alignment.topLeft,  
          end: Alignment.bottomRight,  
          colors: [  
            Color(0xFF00AEFF),  
            Color(0xFF0076FF),  
          ],  
        ),  
      ),  
    ],  
  ),  
)
```

```

        ),

        child: Icon(
            Icons.home,
            color: Colors.white,
        ),
    ),
    SizedBox(width: 12),
    Container(
        child: Text(
            "Home",
            style: TextStyle(
                fontSize: 16.0,
                fontWeight: FontWeight.w800,
                color: Color(0xFF242629),
            ),
        ),
    ),
],
)

```



Import Assets and Project Files

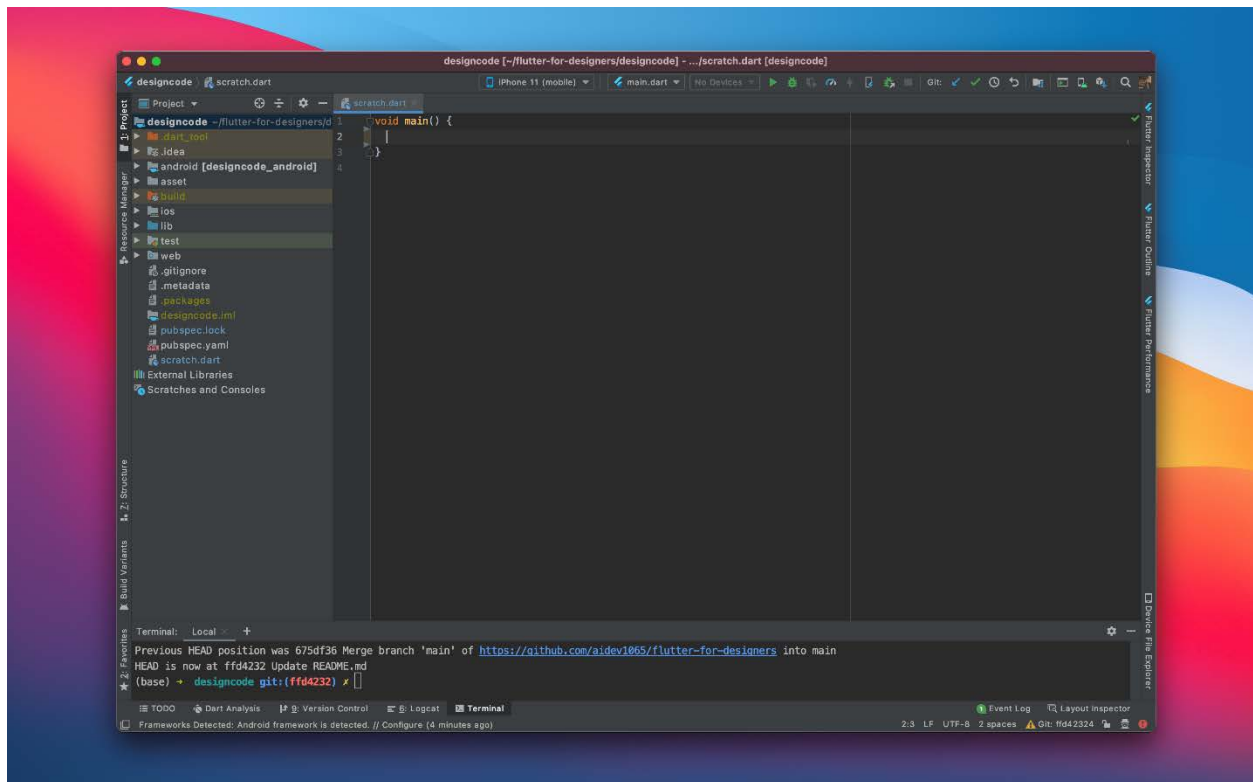
Before moving forward, we need to add the assets and project files. Open the **Project Files** folder provided at the top. Within the folder, drag and drop the **Assets** folder at the very top of the file hierarchy. The folder will contain all the images and fonts we will be using inside our application.

Short Tour To Dart

Getting Started

```
void main() {  
  
}
```

This **main** function will let Dart know that this is the entry point function to run. The **void** keyword means that this function will return nothing.



Variables

Variables can be thought of as boxes that hold a particular value. Below, I'll define a variable called **name** that will hold a String value of my name. Then, I will print it to the console.

```
void main() {
```

```
var name = 'Yeasir Arefin Tusher';
```

```
print(name);
```

```
}
```

Keywords

- **Dynamic:** This keyword lets Flutter know that even though name contains a value of type String, it can be allowed to change to a value of a different data type.
- **Final:** This keyword lets Flutter know that this variable cannot be changed later on making the variable immutable. Final is initialized at run-time meaning that every time the state of the app reloads, so do the variables marked final.
- **Const:** This is similar to the final keyword. Const is initialized at compile-time so when the code compiles, the variable is initialized and is never reinitialized again.
- **Note:** You can also mark variables by the data type they hold. For example, the name variable can be marked String to let the code know this variable should always hold a value of type String.

```
final String name = 'Yeasir Arefin Tusher';
```

```
print(name);
```

Integer Data Type

Setting a variable to hold a value of type **integer** is very simple. All it takes is marking a variable **int** and setting its value.

```
int age = 19;
```

String Interpolation

String Interpolation refers to creating string objects which can dynamically be changed by including variables within the string.

```
print('My name is $name and my age is $age!')
```

Collections

Many programming languages include some form of collections. While Swift has arrays, Dart, similar to Python, has lists. To create a list, set a variable equal to a multiple objects of the same data type wrapped in square brackets and separated by commas.

```
var hobbies = <String>['coding', 'biking', 'reading'];
```

Functions

Functions can be handy when you want to perform a particular operation repeatedly without having to write the same lines of codes over and over again. Let's delete everything inside the **main** function. We'll create a **showInfo** function that simply prints a statement to the console.

```
void main() {  
  
    void showInfo() {  
  
        print('My name is Sai and I am 19 years old');  
  
    }  
  
    showInfo();  
}
```

If we run the **Terminal** command `dart run scratch.dart`, we should see the statement printed to the console.

Functions with Arguments

We can customize our functions by providing arguments. This can customize small portions of our function to change based on the arguments provided. We'll add an argument called **name** that will be used to change the **print** statement in our function.

```
void showInfo({String name}) {  
    print("My name is $name and I am 19 years old");  
}
```

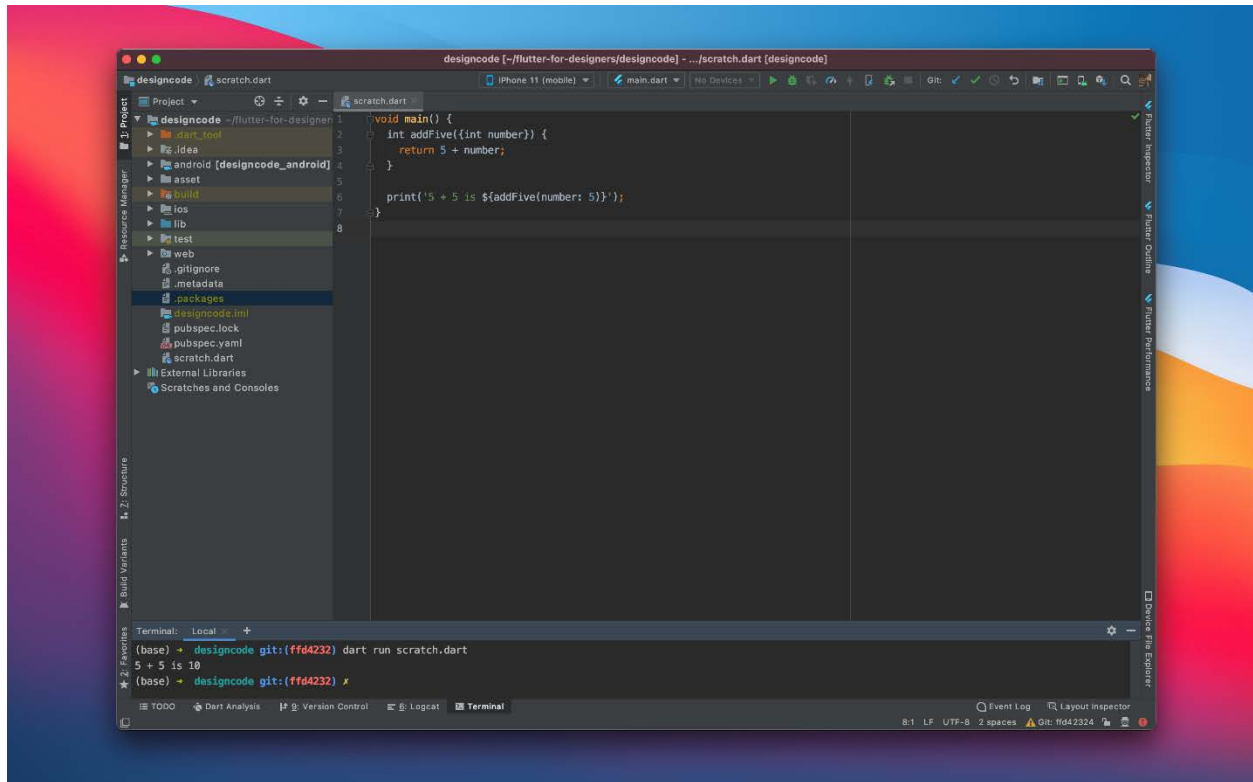
```
showInfo(name: 'Tusher');
```

Functions that return values

Let's remove everything inside the **main** function for the last time. We'll now create a function that will take a number as an argument and return the value of that number plus 5. To make a function return a value, we have to replace the **void** keyword with the data type we expect the function to return.

```
void main() {  
    int addFive({int number}) {  
        return 5 + number;  
    }  
  
    print('5 + 5 is ${addFive(number: 5)}');  
}
```

We used the **int** keyword since we want our functions to return an integer value. Now if we run the Dart code, the arithmetic operation will be printed to the console.



Classes in Dart

Develop sidebar items with the help of classes

In this section, you will learn how to create classes in Dart. Classes are important to any object oriented programming language. With the help of classes, we will build the `SidebarItem` class that can be used to dynamically change our sidebar item. We'll also learn about refactoring our code for easier readability and how to import and use the assets inside our `Project Files` folder.

Getting Started

Select the **lib** folder and press CMD+N on your keyboard. Create a new Dart file and name it **model/sidebar**. This will create a new **sidebar.dart** file inside a newly created **model** folder. We will use this file to create the sidebar class.

Creating the SidebarItem class

At the top of the file, we will need to import the Material library. This will give us access to all the widgets and data types, most notably the **LinearGradient** widget.

Taking a look at our current sidebar item, we have 3 customizable properties: the title, the icon, and the background gradient. So we'll create a class to take those three properties as input.

Getting Started

Select the **lib** folder and press CMD+N on your keyboard. Create a new Dart file and name it **model/sidebar**. This will create a new **sidebar.dart** file inside a newly created **model** folder. We will use this file to create the sidebar class.

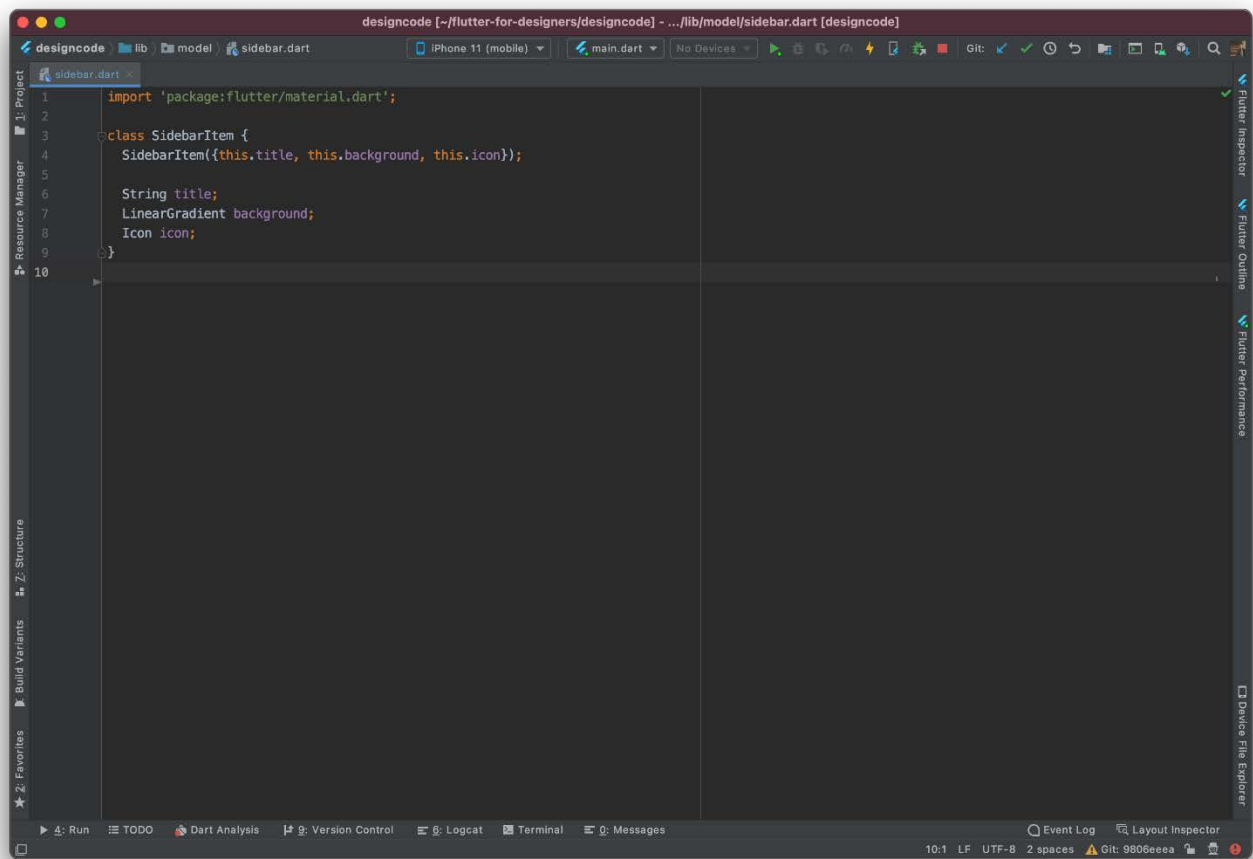
Creating the SidebarItem class

At the top of the file, we will need to import the Material library. This will give us access to all the widgets and data types, most notably the **LinearGradient** widget.

Taking a look at our current sidebar item, we have 3 customizable properties: the title, the icon, and the background gradient. So we'll create a class to take those three properties as input.

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

```
class SidebarItem {  
  
  SidebarItem({this.title, this.background, this.icon});  
  
  String title;  
  
  LinearGradient background;  
  
  Icon icon;  
  
}
```



Adding the Sample Data

Now before we go and use this class, we'll need to actually create some sample data to use. You can copy the sample data I have below and paste it right under where we defined the **SidebarItem** class.

```
var sidebarItem = [  
  SidebarItem(  
    title: "Home",  
    background: LinearGradient(  
      begin: Alignment.topLeft,  
      end: Alignment.bottomRight,  
      colors: [  
        Color(0xFF00AEFF),  
        Color(0xFF0076FF),  
      ],  
    ),  
    icon: Icon(  
      Icons.home,  
      color: Colors.white,  
    ),  
  ),  
  SidebarItem(  
    title: "Courses",  
    background: LinearGradient(  
      begin: Alignment.topLeft,  
      end: Alignment.bottomRight,  
      colors: [Color(0xFFFA7d75), Color(0xFFC23D61)]),  
    icon: Icon(  
      Icons.library_books,  
      color: Colors.white,
```



```
    ),  
    ),  
    SidebarItem(  
      title: "Billing",  
      background: LinearGradient(  
        begin: Alignment.topLeft,  
        end: Alignment.bottomRight,  
        colors: [Color(0xFFFFAD64A), Color(0xFFEA880F)]),  
      icon: Icon(  
        Icons.credit_card,  
        color: Colors.white,  
      ),  
    ),  
    SidebarItem(  
      title: "Settings",  
      background: LinearGradient(  
        begin: Alignment.topLeft,  
        end: Alignment.bottomRight,  
        colors: [Color(0xFF4E62CC), Color(0xFF202A78)]),  
      icon: Icon(  
        Icons.settings,  
        color: Colors.white,  
      ),  
    ),  
  ];
```

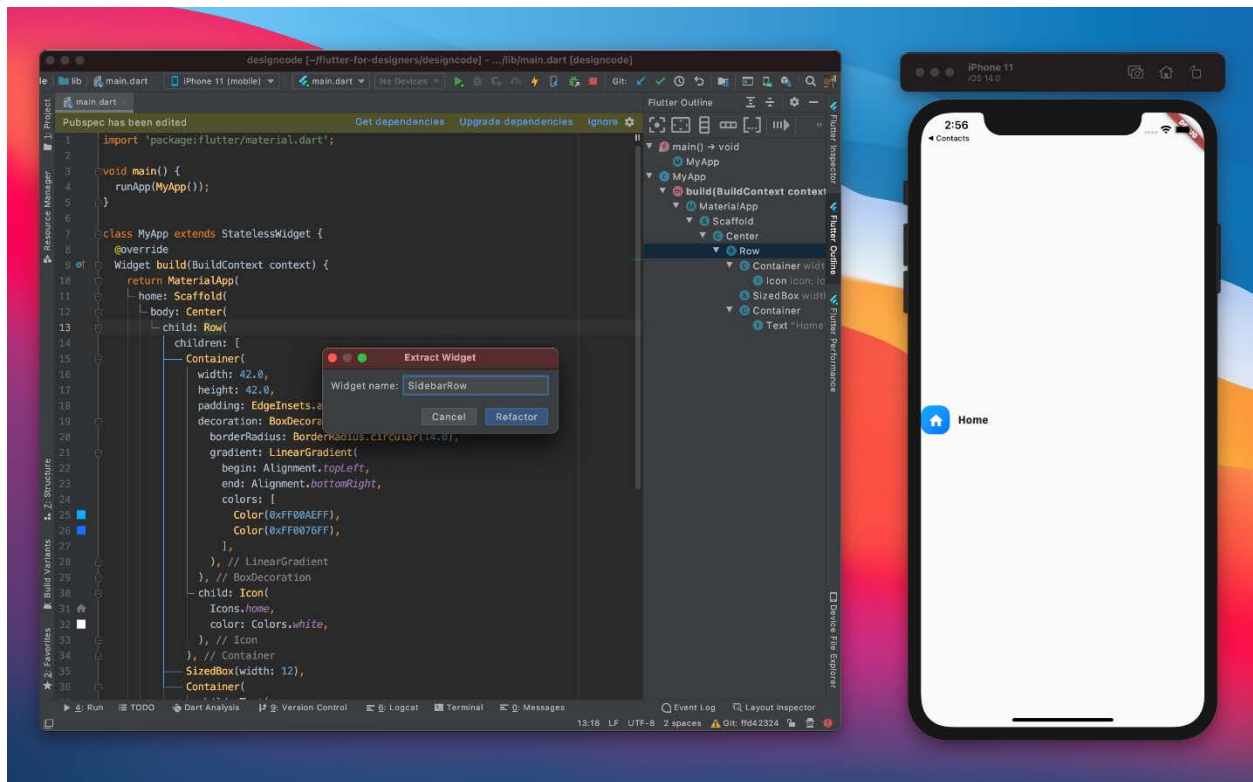
Extracting a Widget

Go back to **main.dart**. We now need to create four instances of the **Row** widget. However, if we copied and pasted that code four times, it wouldn't be good design practice. So what we need to do is extract the **Row** widget and all its sub-children to name it something else. The follow steps should be followed to extract a widget

- Place your cursor on the **Row** widget
- Select **Flutter Outline** from the right hand side pane
- CNTRL+Select the **Row** widget in the hierarchy tree and select Extract Widget
- Name this new widget. We'll call this **SidebarRow**

A new stateless widget is created called **SidebarRow** that returns the **Row** widget. and our whole code containing the **Row** widget is replaced with just **SidebarRow**.

Note: If your Flutter Outline menu is empty, CMD+Click on any keyword inside your code. This will force Flutter Outline to refresh its menu. Then, go back and you should see the hierarchy tree inside Flutter Outline



Passing the SidebarItem to the SidebarRow

While our SidebarRow is created, it is using static values like the blue gradient background or the home icon. To pass a SidebarItem, we need to change SidebarRow to the following.

```
class SidebarRow extends StatelessWidget {
```

```
  SidebarRow({@required this.item});
```

```
  final SidebarItem item;
```

```
  @override
```

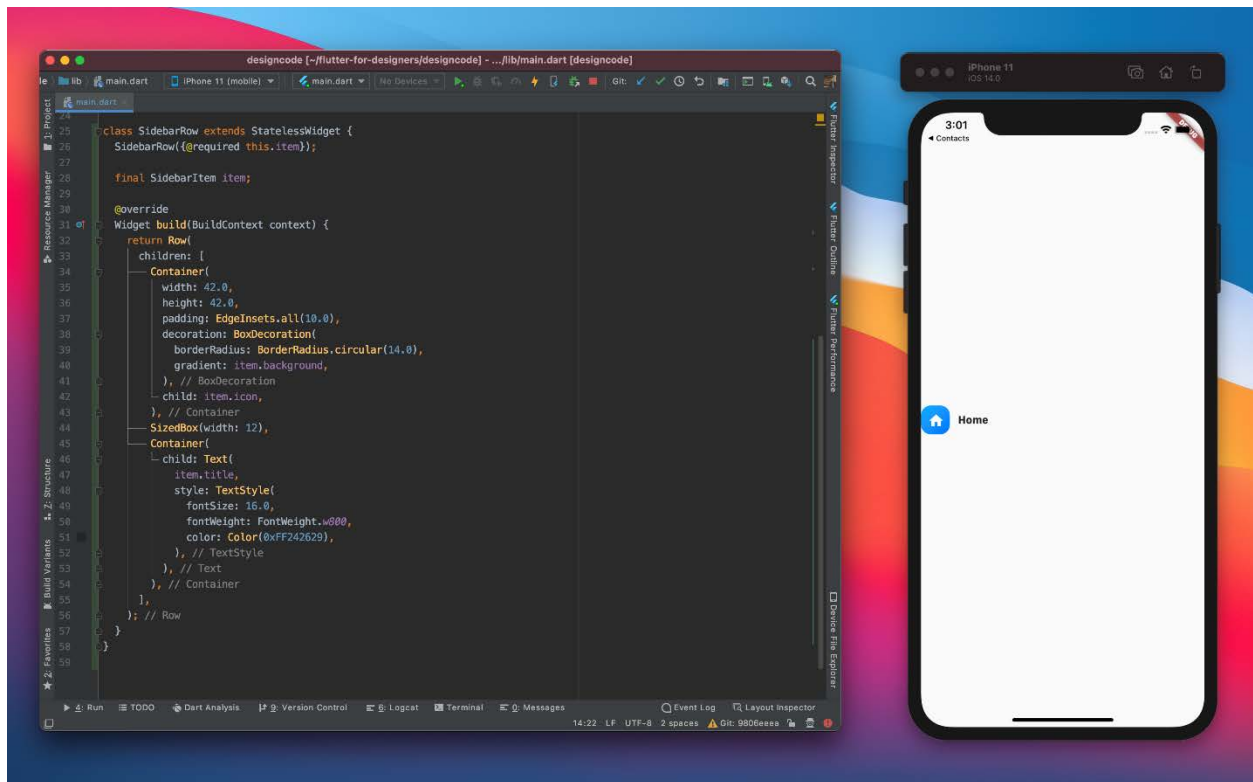
```
  Widget build(BuildContext context) {
```

```
    return Row(
```

```
      children: [
```

```
        Container(
```

```
width: 42.0,
height: 42.0,
padding: EdgeInsets.all(10.0),
decoration: BoxDecoration(
  borderRadius: BorderRadius.circular(14.0),
  gradient: item.background,
),
child: item.icon,
),
SizedBox(width: 12),
Container(
  child: Text(
    item.title,
    style: TextStyle(
      fontSize: 16.0,
      fontWeight: FontWeight.w800,
      color: Color(0xFF242629),
    ),
  ),
),
],
);
}
```



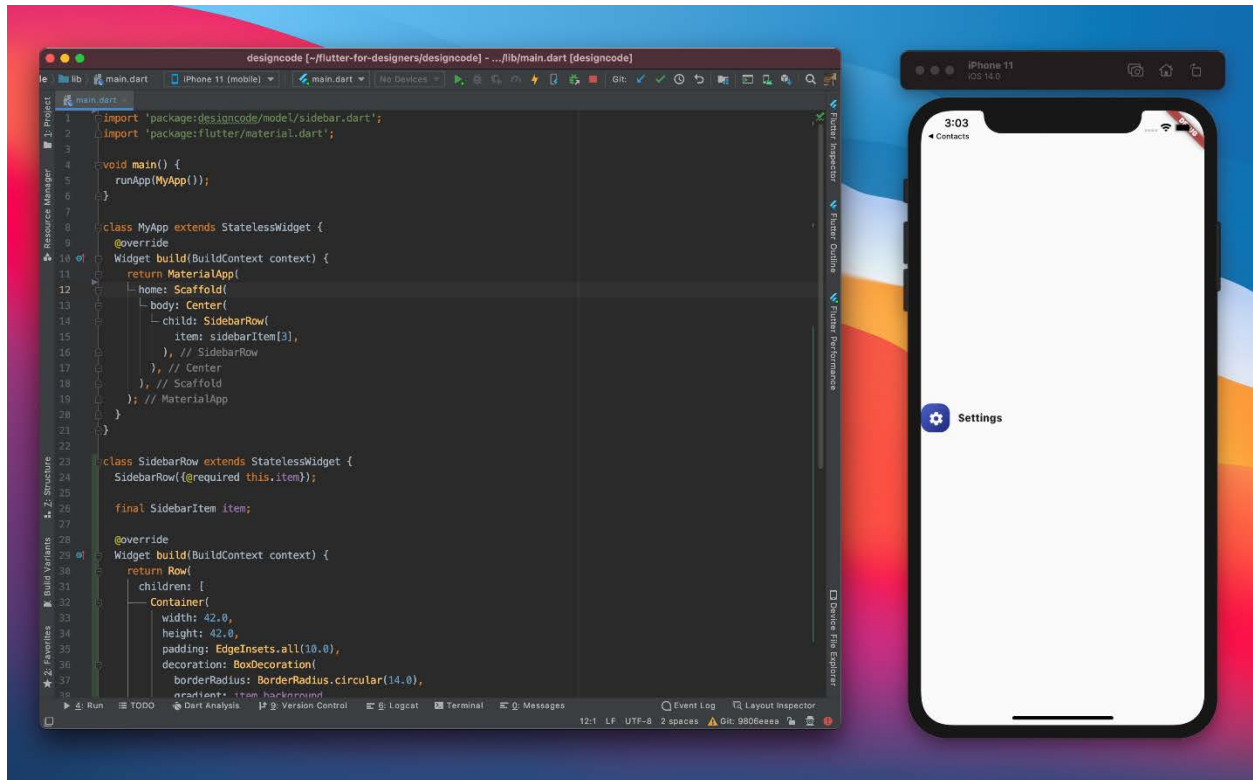
Passing the Data

Inside our `MyApp` widget, we need to add the `item` argument to our `SidebarRow`. This is a simple fix and requires us to change `SidebarRow` to:

```
SidebarRow(item: sidebarItem[3])
```

Here we are using the fourth item from the sample data we created earlier.

Note: Although Android Studio should do this automatically, if you get an error saying `sidebarItem` is not defined, then the `sidebar.dart` file has not been imported at the top. Make sure this is imported to access the `SidebarItem` class and sample data.



Refactor Code

Inside the `lib` folder, create a new folder called `components`, and inside this folder, create a new Dart file called `sidebar_row.dart`. At the top of the file, import the Material library by typing:

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

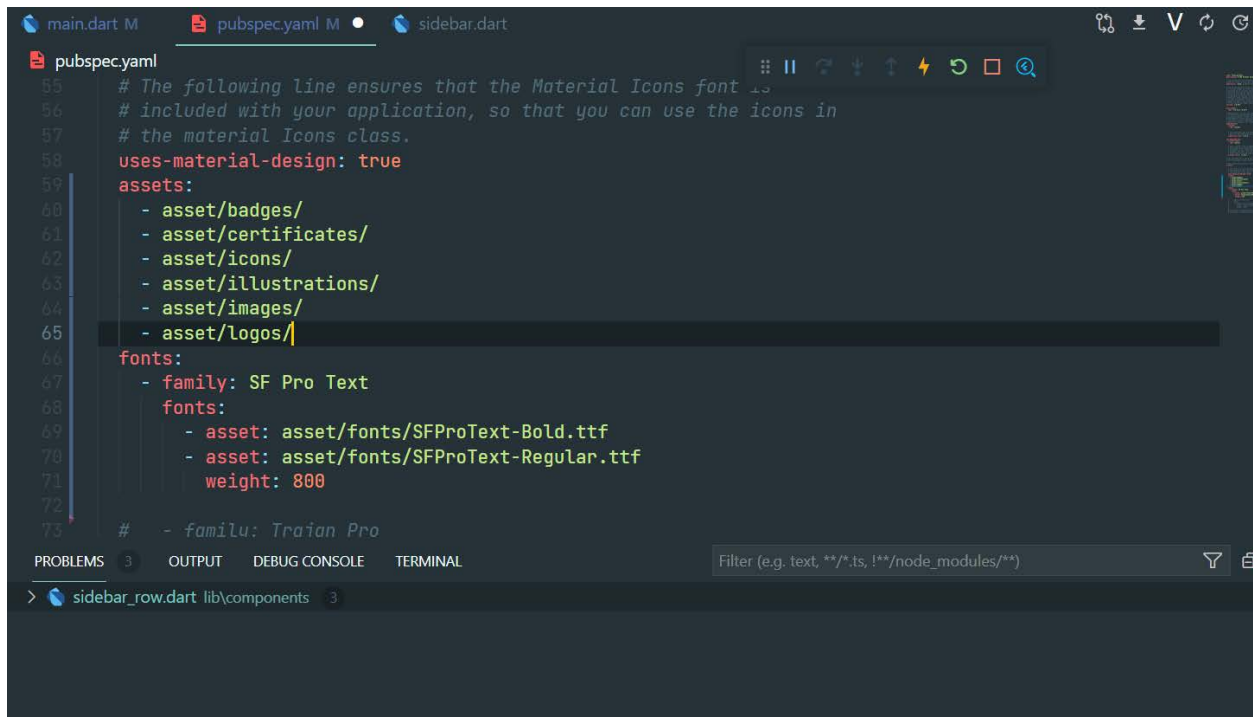
Go back to `main.dart` and cut the entire code defining our `SidebarRow` widget. Paste it into this new file right under where we imported the Material package.

You may get an red squiggly line right underneath the `SidebarItem` keyword. This indicates that Flutter does not know what this keyword is since we have not imported the file referencing this. To fix this:

- Hover over the error keyword
- Select the red lightbulb
- From the menu that shows up, select the option that begins with **Import library**

YAML File

The pubspec.yaml can be used to declare what dependencies and packages our project needs. We'll modify this to add our assets and fonts. At the bottom of the file, where we have the flutter section, change it to look like the following.



```
pubspec.yaml
55 # The following line ensures that the Material Icons font is
56 # included with your application, so that you can use the icons in
57 # the material Icons class.
58 uses-material-design: true
59 assets:
60   - asset/badges/
61   - asset/certificates/
62   - asset/icons/
63   - asset/illustrations/
64   - asset/images/
65   - asset/logos/
66 fonts:
67   - family: SF Pro Text
68     fonts:
69       - asset: asset/fonts/SFProText-Bold.ttf
70       - asset: asset/fonts/SFProText-Regular.ttf
71         weight: 800
72   # - family: Traian Pro
```

Building Full Screen Layout

Getting Started

Start by going to the **sidebar_row.dart** file. We'll need to change the text style here.

At the bottom of the **SideBarRow** stateless widget, we'll change the style of our **Text** widget.

- Remove the style argument

```
Text(
```

```
    item.title,
```

```
)
```

- Replace it with a predefined text style

```
Text(
```

```
    item.title,
```

```
    style: kCalloutLabelStyle,
```

```
)
```

Creating the SidebarScreen

Go to main.dart and extract the Center widget. Name this new widget SidebarScreen and remove the predefined initialization. It should now look like this.

```
class SidebarScreen extends StatelessWidget {
```

```
    @override
```

```
    Widget build(BuildContext context) {
```

```
        return Center(
```

```
            child: SidebarRow(
```

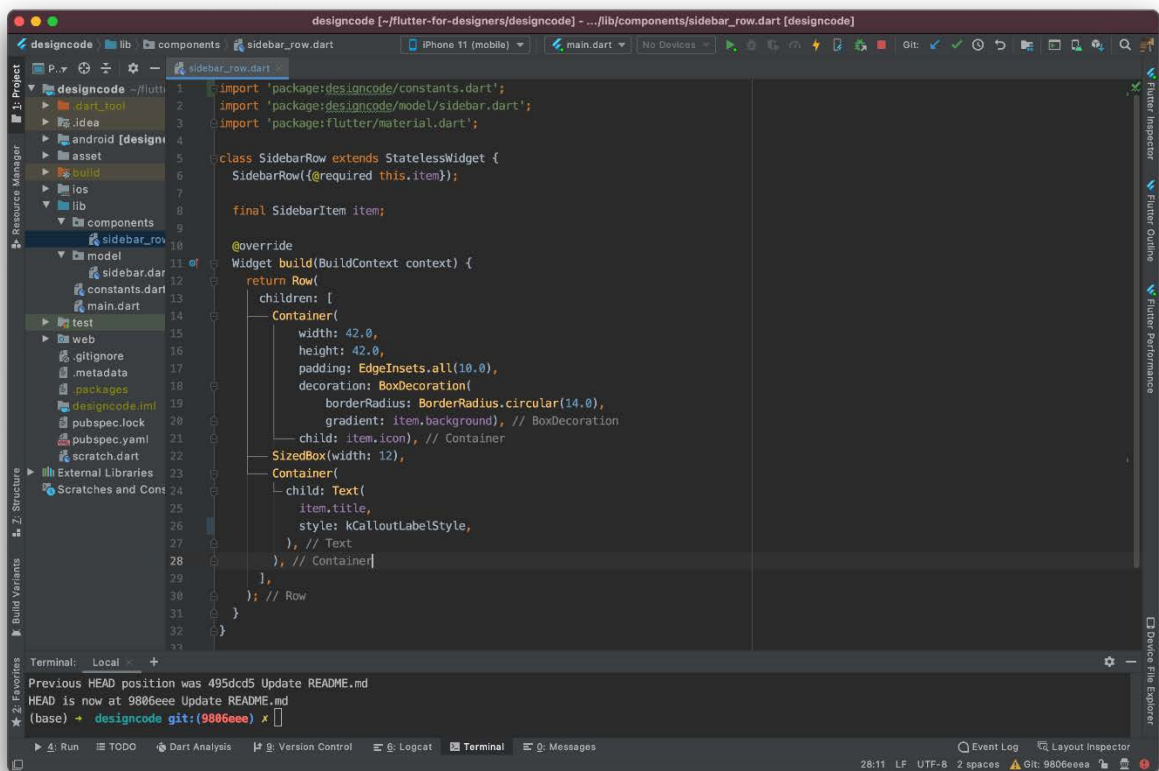
```
                item: sidebarItem[3],
```



```

    ),
  );
}

```



Designing the SidebarScreen

To make our sidebar item button more visible, we'll need to change the **Center** widget to a **Container** widget and decorate our container. This includes setting the background color, border radius, height, width, and padding.

```
class SidebarScreen extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return Container(

      decoration: BoxDecoration(

        color: kSidebarBackgroundColor,

        borderRadius: BorderRadius.only(

          topRight: Radius.circular(34.0),

        ),

      ),

      height: MediaQuery.of(context).size.height,

      width: MediaQuery.of(context).size.width * 0.85,

      padding: EdgeInsets.symmetric(

        vertical: 35.0,

        horizontal: 20.0,

      ),

      child: SafeArea(

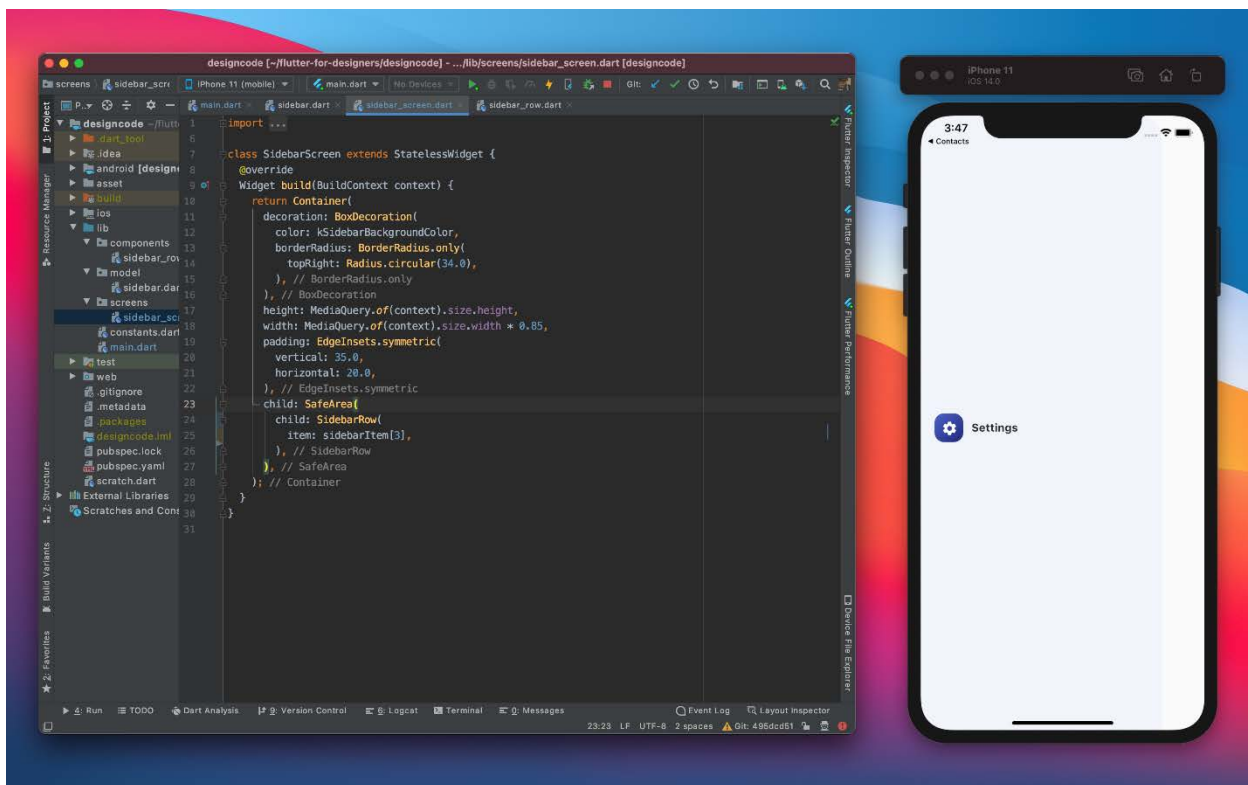
        child: SidebarRow(

          item: sidebarItem[3],
```

```

    ),
  ),
);
}
}

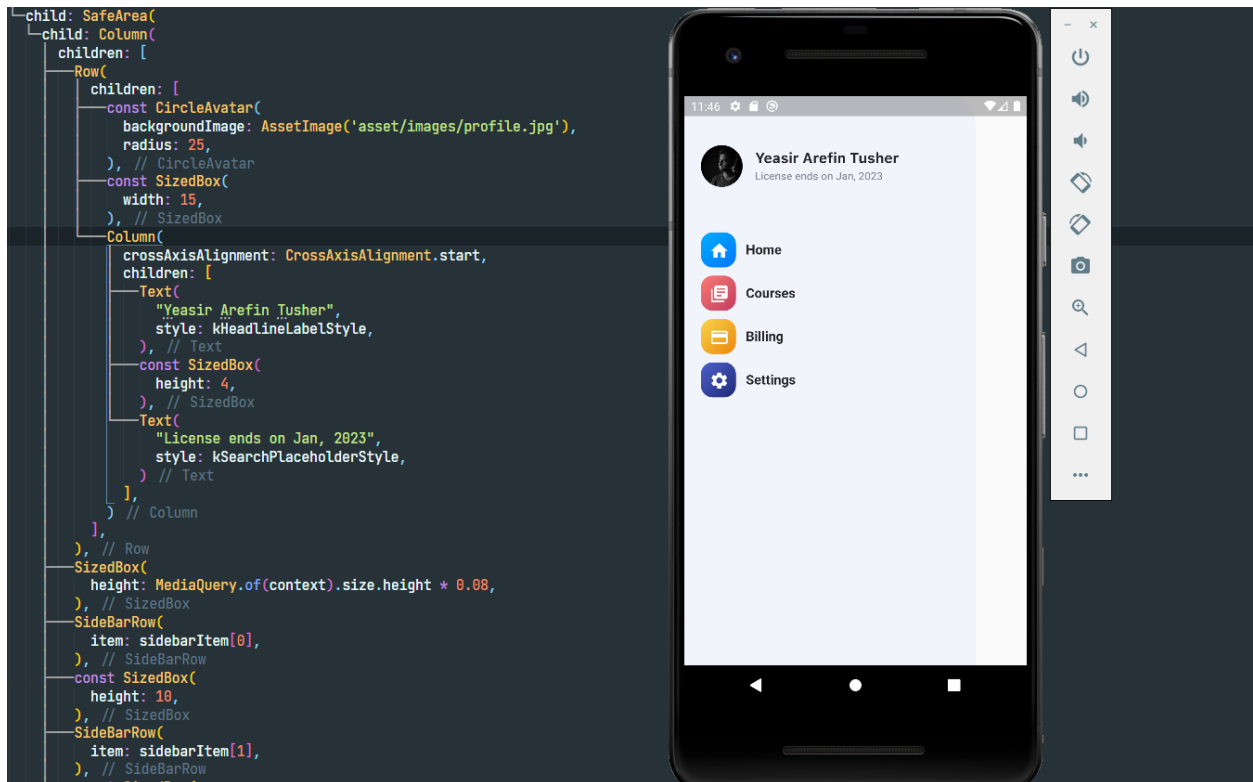
```



Sidebar Header

The sidebar header will contain our profile picture, two labels containing our name and when our license expires. Then we will add adequate spacing in between the header and our sidebar item button.

First, I'll wrap the **SidebarRow** widget inside a **Column** widget. Then, I can add the following code.



Remaining sidebar items

Next, we need to add the remaining sidebar items. Lucky for us, because of refactoring, we are able to add the remaining items quite easily.

- Change the index value from 3 to 0

```
SideBarRow(  
  
    item: sidebarItem[0],  
  
) ,
```

Add a **SizedBox** widget with a height of 32.0 pixels underneath the above **SideBarRow** widget. This will create the spacing between all the **SideBarRow** widgets.

```
    SizedBox(  
  
        height: 32.0,  
  
    ),
```

- Select both the **SidebarRow** and **SizedBox** widgets and press CMD+D 3 more times. This will duplicate the selected code. Now change the indices of **SidebarRow** for the second, third, and fourth **SidebarRow** widgets.

```
SidebarRow(  
  
    item: sidebarItem[1],  
  
),  
  
SizedBox(  
  
    height: 32.0,  
  
),  
  
SidebarRow(  
  
    item: sidebarItem[2],  
  
),  
  
SizedBox(  
  
    height: 32.0,  
  
),  
  
SidebarRow(  
  
    item: sidebarItem[3],
```

```

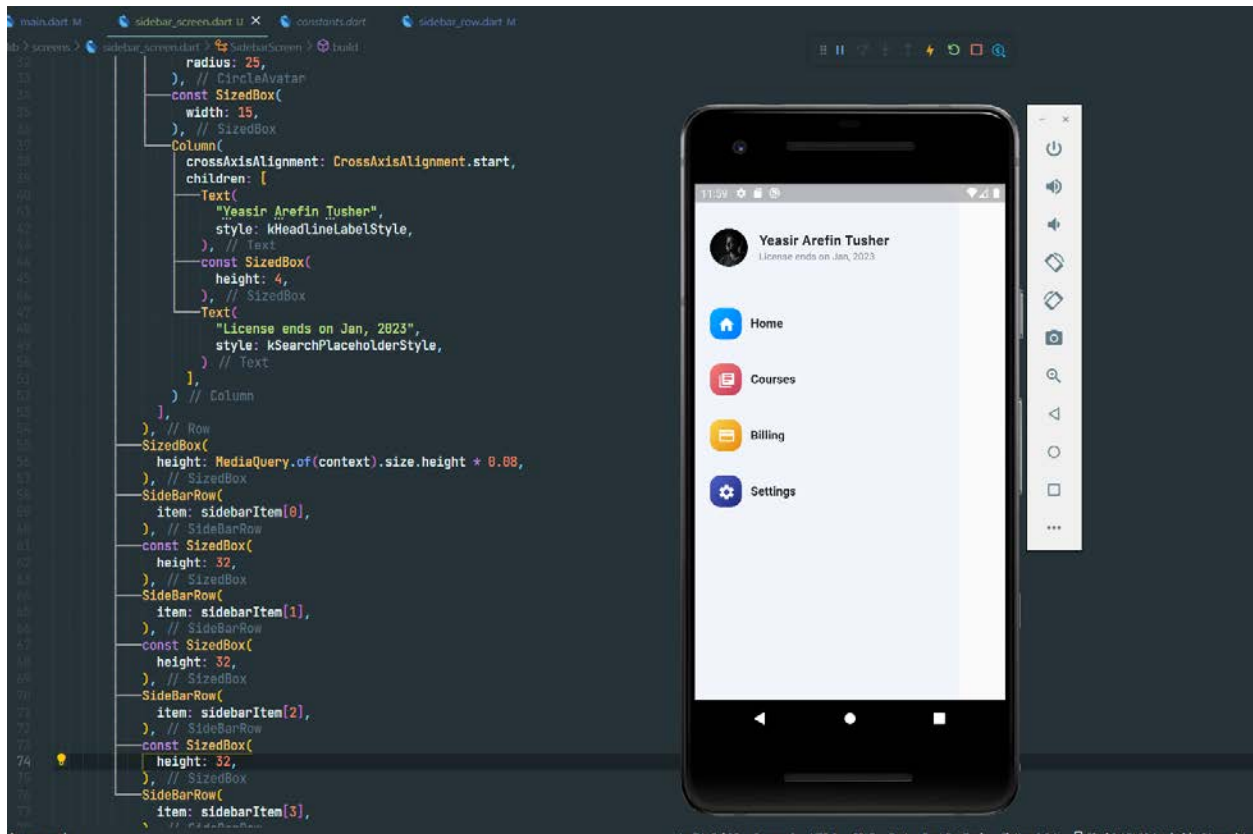
),

  SizedBox(

    height: 32.0,

  ),

```



Logout Button

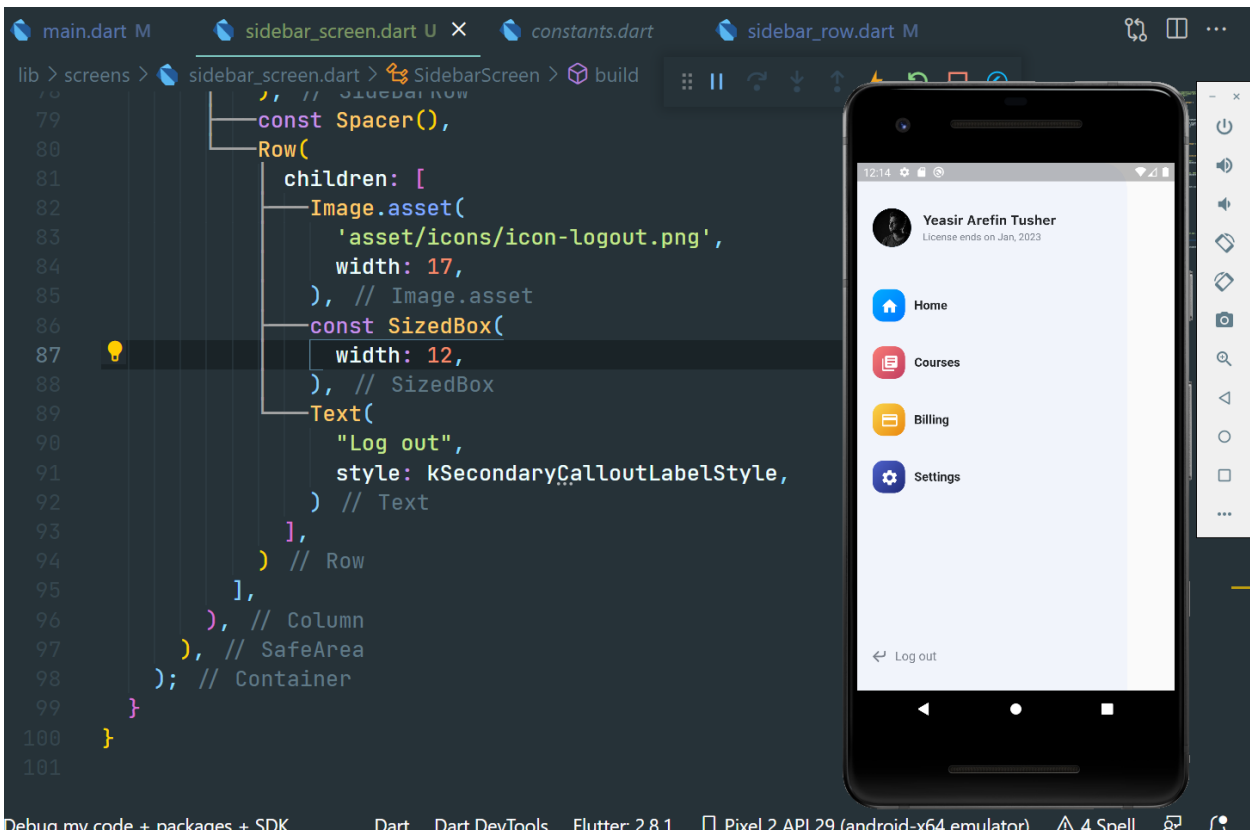
After the sidebar items, we want the logout button to be place at the very bottom of our **SideBarScreen**. So after the last **SizedBox** widget, we'll have to add two widgets.

- The first widget is a **Spacer** widget. This will push all the content after to the bottom.

```
Spacer() ,
```

- The next widget is a **Row** widget that will contain an **Image** widget, a **SizedBox** widget for spacing, and a **Text** widget

```
children: [  
    Image.asset(  
        'asset/icons/icon-logout.png',  
        width: 17.0,  
    ),  
    SizedBox(  
        width: 12.0,  
    ),  
    Text(  
        "Log out",  
        style: kSecondaryCalloutLabelStyle,  
    ),  
],  
)
```



Refactoring

Now that we have our **SidebarScreen** widget complete, let's move it out of the **main.dart** file. Cut the entire code for **SidebarScreen** out of **main.dart**. Create a new folder inside of the **lib** folder titled **screens**. Inside this folder, create a new file called **sidebar_screen.dart**.

At the top of this file, import the **Material** package.

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

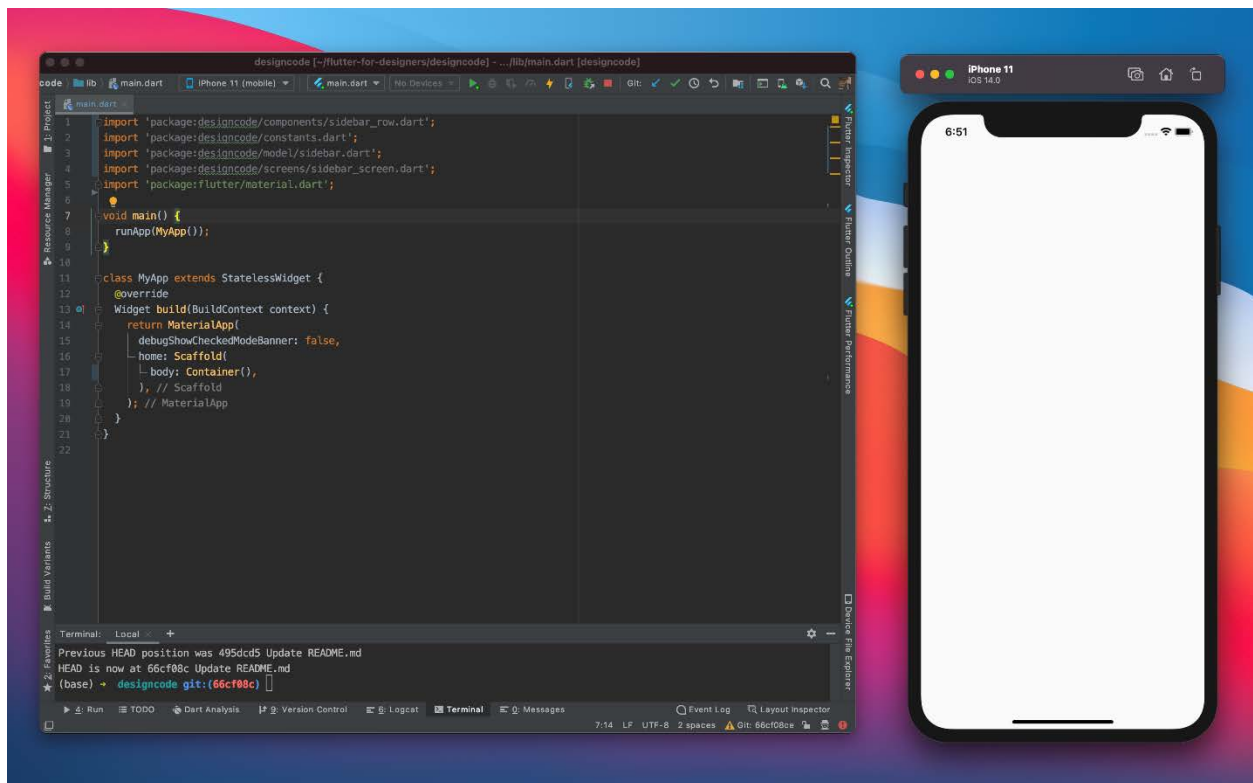
Then, paste the code below. You may have to import the extra libraries by hovering over the keywords with the error lines, clicking on the red lightbulb, and selecting the first item in the popup menu that shows up. You may need to do the same in **main.dart** for the **SidebarScreen** widget.

Building Card Widget

Getting Started

Inside **main.dart**, we'll replace the **AppBar** widget with the **Container** widget as a child for the **Scaffold** widget.

```
return MaterialApp(  
  
  debugShowCheckedModeBanner: false,  
  
  home: Scaffold(  
  
    child: Container(),  
  
  ),  
  
);
```



Creating the RecentCourseCard widget

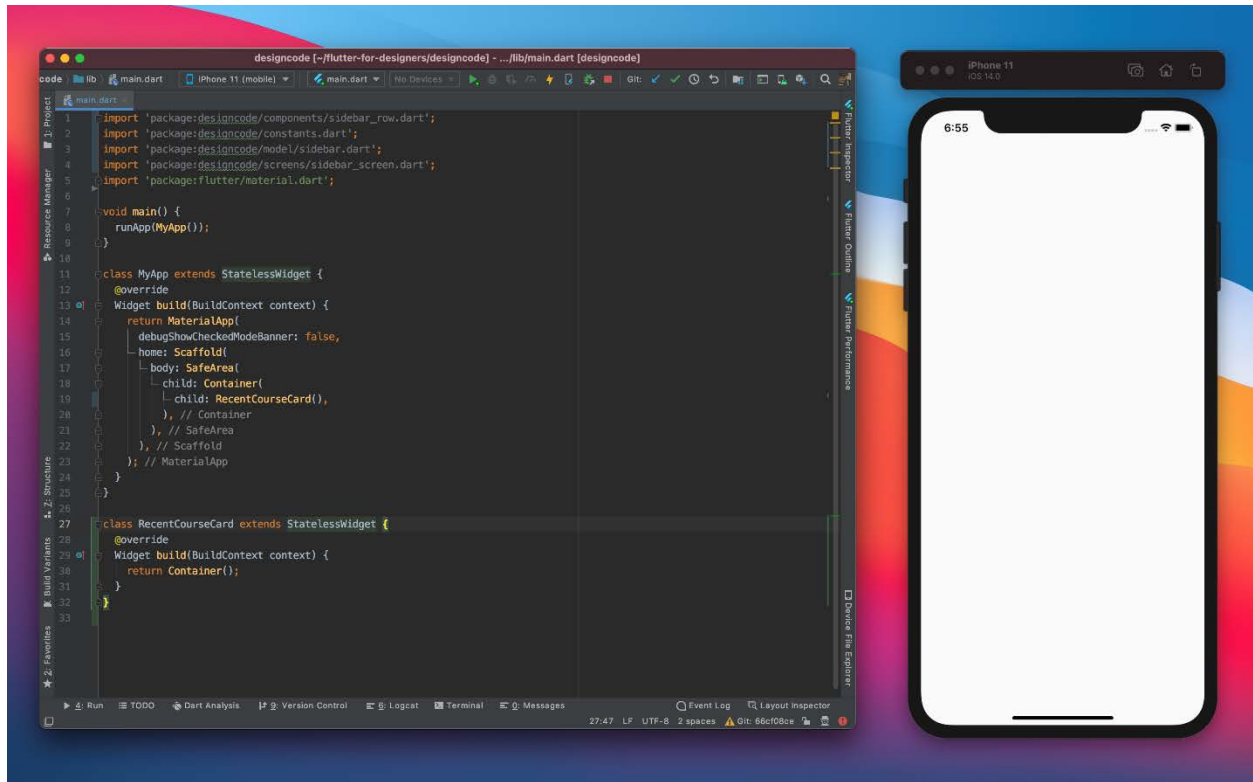
At the bottom of **main.dart**, type the shortcut `stless` to create a new stateless widget. Set the name to **RecentCourseCard**. It should look like this.

```
class RecentCourseCard extends StatelessWidget {  
  
  @override  
  
  Widget build(BuildContext context) {  
  
    return Container();  
  
  }  
  
}
```

We'll also make a few modification to the **Scaffold** widget inside the **MyApp** widget.

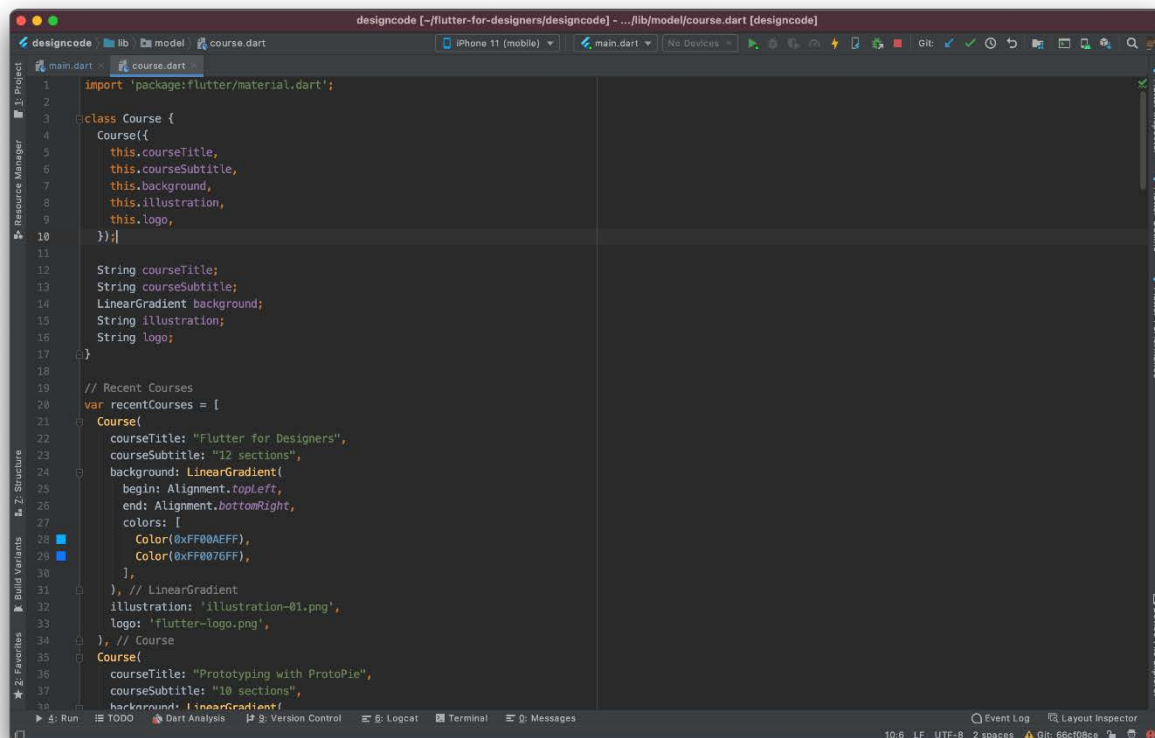
- Wrap the **Container** widget in a **SafeArea** widget

```
Scaffold(  
  
  body: SafeArea(  
  
    child: Container(),  
  
  ),  
  
),
```



Importing the Course class

Inside our project files, we have a file called **course.dart**. This file contains the **Course** class and sample data. Drag and drop it into the **model** folder.



Pass the Course Data to RecentCourseCard

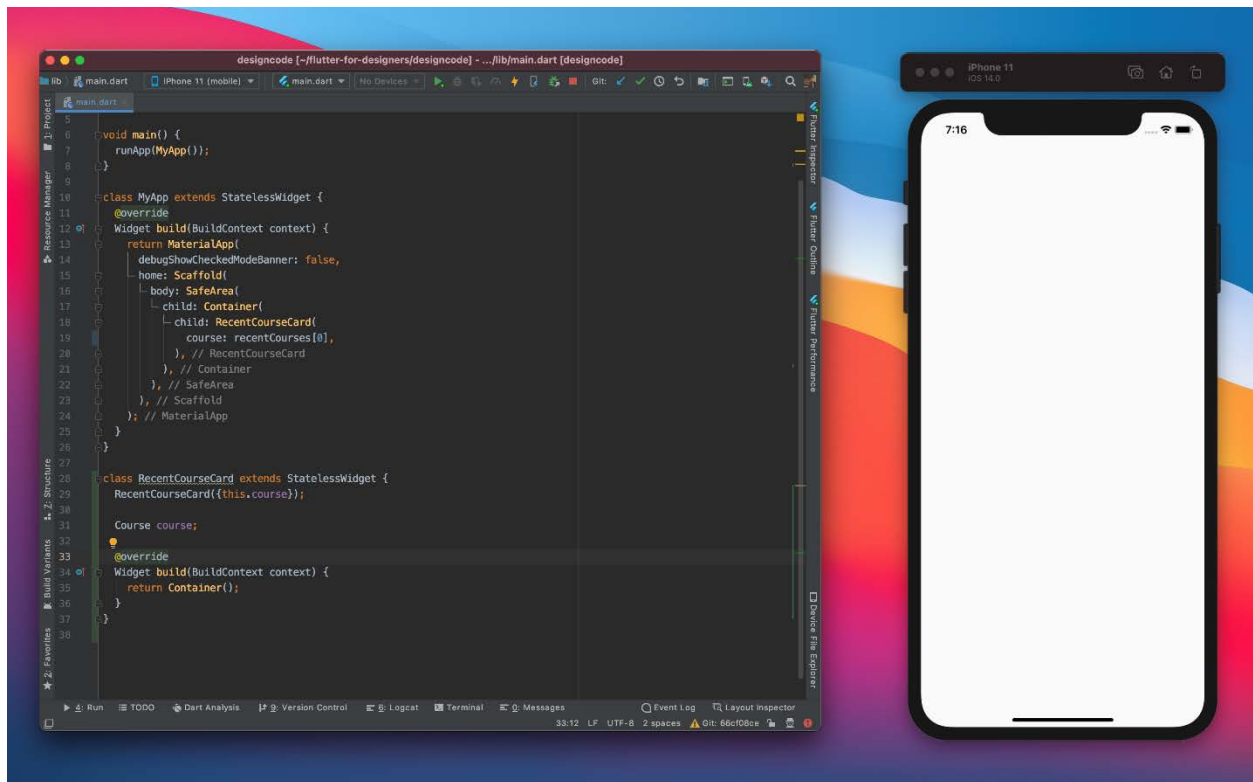
Just like with all class initializations, we'll need to pass the course argument to **RecentCourseCard** in order to pass data.

- Inside **main.dart**, add the course argument of class type **Course** to **RecentCourseCard**

```
class RecentCourseCard extends StatelessWidget {  
  
  RecentCourseCard({this.course});  
  
  Course course;  
  
  @override  
  
  Widget build(BuildContext context) {  
  
    return Container();  
  
  }  
  
}
```

- Pass the sample data to our initialization of **RecentCourseCard** in **MyApp**.

```
RecentCourseCard(  
  
  course: recentCourses[0],  
  
),
```



Set the Card Background

First we'll create the background of our card. This will involve wrapping our **Container** widget in a **Padding** widget, setting the **decoration** property of our container, and setting some of the constraints.

```
return Padding(  
  
  padding: EdgeInsets.only(top: 20.0),  
  
  child: Container(  
  
    width: 240,  
  
    height: 240,  
  
    decoration: BoxDecoration(  

```

```
gradient: course.background,

borderRadius: BorderRadius.circular(41.0),

boxShadow: [

    BoxShadow(

        color: course.background.colors[0].withOpacity(0.3),

        offset: Offset(0, 20),

        blurRadius: 30.0),

    BoxShadow(

        color: course.background.colors[1].withOpacity(0.3),

        offset: Offset(0, 20),

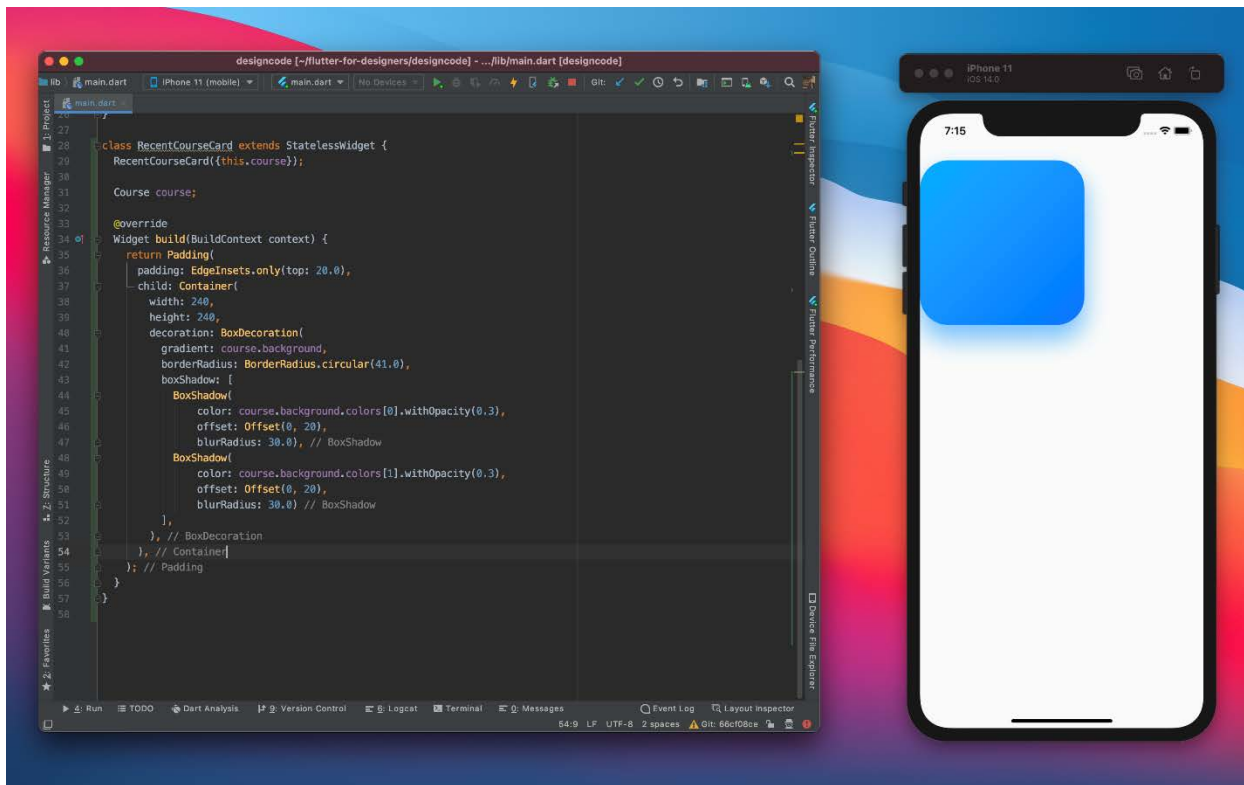
        blurRadius: 30.0)

    ],

),

),

);
```



Add the Card Content

Next we need to set the content of our card. This will be the course title, course subtitle, and course illustration. We will use a **Column** widget to vertically align our content and be the parent widget as a child for our container. So add the `child` argument to our **Container** and add the following **Column** widget.

```
Column(  
  
  children: [  
  
    Padding(  
  
      padding: EdgeInsets.only(  
  
        top: 32.0,  
  
        left: 32.0,
```

```
        right: 32.0,

    ),

    child: Column(

        crossAxisAlignment: CrossAxisAlignment.start,

        children: [

            Text(

                course.courseSubtitle,

                style: kCardSubtitleStyle,

            ),

            SizedBox(

                height: 6.0,

            ),

            Text(

                course.courseTitle,

                style: kCardTitleStyle,

            ),

        ],

    ),
```



```

    ),

    Expanded(

      child: Image.asset(

        'asset/illustrations/${course.illustration}',

        fit: BoxFit.cover,

      ),

    ),

  ],

```

```

),

```

